Adaptive Asynchronous Control and Consistency in Distributed Data Exploration Systems

by

Benjamin A. Ring

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

August, 2017

Abstract

Advances in machine learning and streaming systems provide a backbone to transform vast arrays of raw data into valuable information. Leveraging distributed execution, analysis engines can process this information effectively within an iterative data exploration workflow to solve problems at unprecedented rates. However, with increased input dimensionality, a desire to simultaneously share and isolate information, as well as overlapping and dependent tasks, this process is becoming increasingly difficult to maintain. User interaction derails exploratory progress due to manual oversight on lower level tasks such as tuning parameters, adjusting filters, and monitoring queries. We identify human-in-the-loop management of data generation and distributed analysis as an inhibiting problem precluding efficient online, iterative data exploration which causes delays in knowledge discovery and decision making. The flexible and scalable systems implementing the exploration workflow require semi-autonomous methods integrated as architectural support to reduce human involvement. We, thus, argue that an abstraction layer providing adaptive asynchronous control and consistency management over a series of individual tasks coordinated to achieve a global objective can significantly improve data exploration effectiveness and efficiency. This thesis introduces methodologies which autonomously coordinate distributed execution at a lower level in order to synchronize multiple efforts as part of a common goal. We demonstrate the impact on data exploration through serverless simulation ensemble management and multi-model machine learning by showing improved performance and reduced resource utilization enabling a more productive semi-autonomous exploration workflow. We focus on the specific genres of molecular dynamics and personalized healthcare, however, the contributions are applicable to a wide variety of domains.

Primary Reader: Yanif Ahmad, PhD
Secondary Readers: Thomas Woolf, PhD and Randal Burns, PhD

Acknowledgments

I would like to thank the members of my committee for their advice and insight throughout my tenure at Johns Hopkins University. Their wisdom and constructive criticism have helped to ensure I achieve the highest quality work possible.

I want to recognize the Maryland Advanced Research Computing Center (MARCC) for supporting my research in Molecular Dynamics and providing tremendous assistance with supercomputing resources.

I want to acknowledge the support of the Office of the Chief Information Officer (G6), U.S. Department of Defense, the U.S. Army, and the National Security Agency for enabling me to pursue this academic endeavor.

I also want thank the fellow students and research assistants with the Johns Hopkins Data Management Systems Lab (DaMSL) and the Saria Lab for your advice, perspective, and camaraderie. Without the daily interaction with my fellow students some of this would never have been possible. I can take with me that some of the most memorable and beneficial aspects of my time have been the various conversations on all topics, be it technical, philosophical, or even political.

I am especially grateful to my advisor, Dr. Yanif Ahmad. His support, guidance, and mentoring have been invaluable to my efforts. He demanded the highest standards in my work and pushed me to achieve the very best. Thank you!

I want to thank and recognize my parents, Steven and Susan Ring and my sister Jennifer. Throughout my life they have always been there to support me in everything I have achieved. My father, especially, has been an inspiration to start me down the path of computer science nearly 40 years ago.

Finally, I am truly grateful for the support and encouragement from my wife, Ramit, and children, Sara, Jack, and Sam. I cannot understate how grateful I am for the incredible sacrifices which they have endured not only as part of this PhD process but also as military spouse and children. Ramit has been the rock and foundation which keeps me going every day. I love you and thank you for all that you have gone through in making my lifelong academic dream a reality.

Dedication

This thesis is dedicated to my father, Steven J. Ring. In the late 1970's, my dad, a software engineering working on defense systems, first brought home a Wyse terminal and showed me how to connect to a main frame and how to navigate a command line prompt. Nearly 40 years later, I have finally achieved my lifelong goal of attaining a PhD in Computer Science - a goal that grew out of that inspiration he sparked in me when I was just a little boy. A former Soldier who has dedicated his life to supporting the U.S. Armed Forces, I am truly thankful for all he has given me and know that there will always be a part of him in everything I achieve. I love you, Dad!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Increased networking speeds and steady growth in computational power combined with a global proliferation of Internet connectivity has spurned a "Big Data" revolution in computing science. Both commercial and researching communities are trying to keep pace with this expansion as technological capabilities quickly evolve. Organizations who once managed internal server rooms to process gigabytes of data locally have shed the administrative burden and now yield to cloud providers overseeing petabytes of data with computation moving toward the exascale. The aggregation of data from social media, near ubiquitous injection points through mobile technology, and a cultural psychological shift in the willingness to share information provides tremendous opportunities in data analytics. Advances in machine learning algorithms and streaming data management systems provide a backbone to transform vast arrays of raw data into valuable information, creating a means to accelerate the synthesis and discovery of new knowledge. This process, data exploration, is increasingly becoming more difficult with the vast expansion of information systems. This is a problem which we must address and one which is at the forefront of many research efforts. Any solution must be both effective and efficient given the increasing pace of technology to process raw data. In this research effort, we examine the challenges of enabling data exploration in the contemporary "Big Data" Information Age and provide an adaptive framework to overcome them.

1.1 Big Data

Over time, as Moore's Law has dictated the advances in computational power, we have simultaneously seen a migration from single server to multi-host processing which can integrate a diversity of technologies. From locally managed servers to globally deployed cloud resources, contemporary computing environments are becoming increasingly reliant on distributed system technology. File systems, such as Hadoop (HDFS), Gluster, and Lustre have replaced parallel network systems and are now mainstays in many server rooms with cloud vendors providing their own internally optimized solutions. Physical servers, once individually configured with specific operating systems, are now commonly deployed with virtualization hypervisors and containerization engines such as VMWare or Docker. This capability facilitates the sharing of resources for diversified applications and allows users to quickly migrate data processing across a myriad of platforms.

Data management has seen a new breed of systems in support of both transactional and analytical processing needs. Many modern Big Data applications are built on top of a more flexible and scalable NoSQL, key-value based data organization concept, shedding the structured relational database model. Dynammo and Cassandra, for example, have served as the backbone systems which propelled Amazon and Facebook to become multi-billion dollar companies. Simple, structured and serializable text based formats, such as XML, YAML or JSON found in systems such as MongoDB, facilitate cross-platform integration and communication. Furthermore, many systems capitalize on decreased memory costs to store and process information entirely in RAM, such as Redis and memcached. While these technologies do not replace existing SQL based, online transactional processing systems (OLTP) which still remain as the dominant data management tool, they serve an essential role, enabling a rich set of data science capabilities. In fact, many OLTP systems, such as Vertica and Oracle, have embraced these changes, providing in-memory storage and column based access.

Data analytical engines leverage the distributed and unstructured nature of these storage and management technologies to effectively process mass amounts of information and solve problems

at unprecedented rates. Leveraging a map-reduce programming model, applications such as Spark or Dryad, can now efficiently perform tasks which had been computationally unfeasible in the past.[1,2] In a map-reduce design, a system distributes the same executable program to many nodes, each of which maps one or more operations to locally stored data, organizes and shuffles the output by keys to connected nodes who subsequently reduce the collected data through an aggregation method. Applying this iterative programming model, streaming systems, such as Flink or Twitter's Storm, perform these tasks in an online fashion, capable of analyzing data in real time. Many of these frameworks provide a relatively simply user application programming interface (API) for languages, such as Java, R, and Python

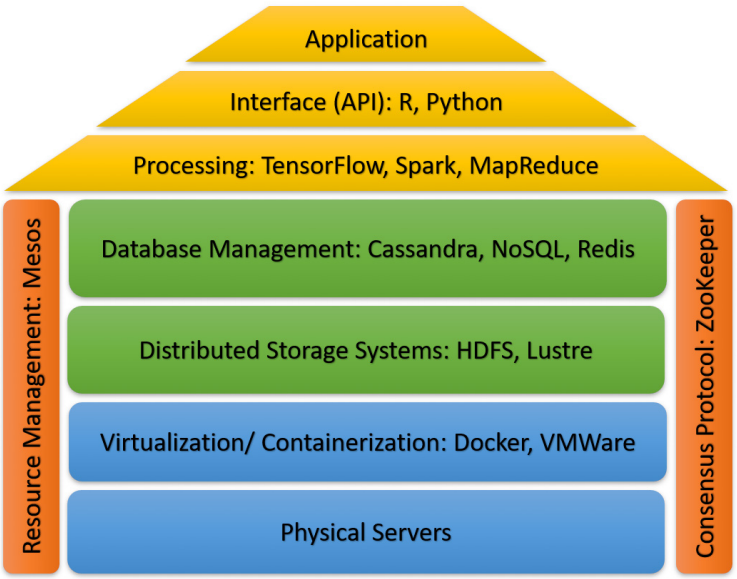


Figure 1.1: Layered stack of "Big Data" Systems

Other supporting software in the Big Data ecosystem include management, monitoring, as well as higher level applications for visualization and web interface. To provide a fault tolerance capability and to ensure eventual consistency, many of these systems rely on a distributed consensus protocol (e.g. Paxos) and deploy services such as Apache ZooKeeper alongside their systems. The workload management tools, Mesos and YARN, are also interwoven in a distributed environment to dispatch, monitor, and balance computational tasks and resources.

This description of the Big Data revolution provides a backdrop and sets the scene in what

we describe as the contemporary operating environment for computer science. Given this setting, how can we find opportunities to capitalize on these advances and better improve our society and the world around us? We, thus focus our attention on the direct correlation between the raw data and the knowledge we seek to gain though data exploration.

1.2 Data Exploration

Data exploration is a process by which we extract information from raw data and transform it into knowledge. Its origin dates to the mathematician, John Tukey, who professed exploratory data analysis as a goal for statisticians to identify unseen correlations, associations, and meaning from data.[3] He believed that we should focus more scientific efforts on identifying new hypotheses to test as opposed to only confirming or denying existing ones. Beyond the exploratory data analysis professed by Tukey, we extend the term data exploration to include the search for an objective, which may be known or unknown, within a given set of data. In executing data exploration, we may not know specifically what we are looking for, however, we can conceptually grasp that we are looking for *something*. An abstract concept, we attribute this as the process of acquiring new knowledge. With rare events, for example, we may not be able to describe or identify the anomaly, but we want to detect it. This is a common struggle in fighting disease and Cyber security: how do we detect an new pathogen or zero-day attack?

To best conceptualize data exploration, we can envision it as traversing up the knowledge pyramid, depicted in Figure 1.2, a common model used in data science and information management domains.[4] This image shows the tiers of the data, information and knowledge with wisdom at the peak. At the lowest level, we find the raw input which, in a computational setting, can be collected from sensors or input devices, or generated as output from another process. The next tier is the transformation of data into a useful and organized form, answering the basic questions of who, what, when, and where associated with the raw data. Above that rests

the knowledge tier whereby we apply the data and information from the tiers below. Analytics, which encompasses the process by which we implement these techniques to "move up the pyramid," takes as its input either of the lower two tiers and produces either information or knowledge. In the context of this research, we scope data exploration into these three tiers and view the peak, wisdom, as the integration of exploratory output into a larger context. We, thus, define data exploration as the process of extracting knowledge from data. By specifically stating knowledge from data, we imply that information is the bridging step to enable the exploratory process. We add a dashed line connecting the knowledge back to the data tier reflecting the adaptive control focus in this research effort.

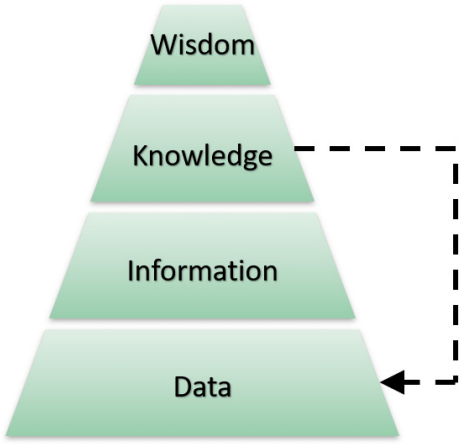


Figure 1.2: The DIKW pyramid linking raw data at the bottom with knowledge and wisdom at the top. The feedback control (dashed arrow) highlights the contemporary online nature of the process.

1.2.1 Raw Data Input

We can characterize data domains in a variety of ways. The recent rise in Big Data capabilities has only added depth to these characteristics. The first and intuitive description is the sheer volume of the dataset. This is characterized by its dimensionality, defining the number of input features or attributes, and the number of individual elements. Although we organize data using a multitude of techniques, in all cases, data can be either structured or unstructured. Traditionally, we associate structured data with a MxN matrix or entity table in a database containing N columns representing the features and M rows representing the data items. In contrast, key-value organization, which has become increasingly popular in contemporary systems, is well suited with unstructured data as well as sparse datasets. This latter classification denotes the density of feature values for individual elements in a dataset; for example, rows in a wide table with only a few values entered would be considered sparse. Beyond merely

sparse, datasets may be incomplete with elements or features missing from portions of the data.

Another classification delineates between online and offline data processing. Static, or offline, datasets are immutable. We can process and analyze them with the *a prior* understanding that the input will not change. In contrast, online data is subject to change, such as real time streaming information or a live, transactional database which may update during the exploration process. Interestingly, we can always capture and store an online dataset as an offline one and process an offline dataset in an online, streaming fashion.

We also characterize data based on the associated genres from which it derives. Some traits include temporal, where elements are organized by time such as streaming medical data, or spatial with data organized by location such as input associated with geography or cosmology. Other categories may include graphic or image based and text or language based data. Within these domains, we can categorize the elements as independent observations or as data whose values are dependent on one or more other elements. The underlying datatypes, which can be continuous, discreet or boolean, or any combination thereof, provide a deeper means of describing a domain.

1.2.2 Data Exploration Objectives

In viewing the end state of the exploration process, we typically define the target goal within the context of an application, such as detecting cancerous cells or associating correlations in retail purchases. Common among all the applications is the underlying purpose of acquiring knowledge. To provide a generalized framework quantifying data exploration, we align the objectives into the following three categories: (1) uncovering unknown patterns, correlations or others meanings in existing data, (2) searching for anomalous outliers, and (3) gaining a better understanding of the input domain. The first two directly address the desire to acquire new knowledge and they align with the fields of study data mining and rare event detection. The last of these, with which we associate the term, *situational awareness*, applies to the inverse of reducing the level of uncertainly of our data and information or to use a military concept: reducing the Fog of War. We explain these

categories in more detail below.

Data Mining

The first of these three objectives is the classical focus of exploratory data analysis. In this context, the goal is to identify unique patterns within a dataset can reveal correlations among input elements or features within a domain. Many of the techniques applied include graphical or visual aids which can assist the human analyst in the process along with other less human interactive algorithms and machine learning models.

Contemporary interactive media, queries, and tools can drive an exploration session and lead to more in-depth knowledge discovery. Common examples include applying a principal component analysis (PCA) algorithm and clustering data in a lower dimension to identify the groupings of elements within a dataset. Another example is commonly found in marketing where retailers perform frequent item analysis to identify purchasing patterns among their customers. These tools may discover unknown correlations which can helps steer advertising decisions, such as the revelation that a significant portion of customers who purchase beer at a certain time of the day tend purchase diapers, leading the retailer to move diapers and beer closer together in the store.[5]

Rare Event Detection

The second objective we identify is the detection of rare events. Also described as rare class mining, outlier or anomaly detection, this goal refers to the set of algorithms, issues, and concepts of classifying and sampling events which are extremely unlikely. Defined in a statistical manner: a 99.9% accurate systems which can properly classify an event will, in contrast, incorrectly classify 0.1% of its input data. If rare events occur at a scale below 0.1%, then such systems cannot reliably identify the anomalies. By using the term *rare*, we refer to those events which are really *interesting*, defined as occurring at a rate below that which a reasonable person would either expect to observe in a given timeframe or the likelihood of identifying an element from among a population.[6]

We discuss this objective in more depth in Chapter 2 as the motivation behind this research

effort is directly linked to rare event detection. Detecting a threat, in the scope of Cyber security, for example, is a rare event detection problem. The threat's online actions are the anomalies among the billions of operations occurring within a computer system every second. However, the problem of rare event detection is common throughout society. Examples include the manufacturing industry of trying to identify a product defect before is it packaged and delivered or in the health industry in trying to detect a disease before it can spread.

Situational Awareness

The term situational awareness (SA) has historically referred to a human's level of understanding of a surrounding environment. It applies in both a spatial and temporal sense in that as the environment changes around a person, in both time and space, that individual's level of comprehension of her surrounding also changes. While its roots can trace back for centuries in warfare, the term situational awareness is more recent, dating to the U.S. Air Force whose crews employed this term in both the US-Korean and US-Vietnam Wars claiming that situational awareness of their surroundings was a decisive advantage in air combat operations.[7] The concept, however, is applicable to many genres outside the military with extensive research studies and models. Target use cases include fields such as air traffic control and vehicle navigation units, which employ these concepts to make operational decisions.[8] We now see situational awareness as a critical factor extending beyond the scope of military application and beyond the perspective of one individual: organizations seek to attain common SA among its members and even computing systems must maintain SA in order to drive decisions. This latter extension is especially important as our society becomes more dependant on autonomous capabilities, such as self-driving cars.

We highlight situational awareness within the context of data exploration because they share the same workflow and underlying objective. In both cases, the system, person or team, must acquire new information to gain knowledge. In the case of SA, that knowledge is incorporated into an existing picture. Uncertainty, represented as holes or gaps in the comprehension of the surrounding

environment, must be reduced. This process, equivalent to the military concept of reducing the Fog of War, a term developed by Carl von Clausewitz in the late 1800's, describes uncertainly in battle. While the literal metaphor implies a Soldier's lack of awareness of his surroundings due to the extensive fog of smoke on the battlefield, the actual concept pertains to the military commander's awareness of the combat environment, enemy, and friendly forces.[9] Reducing this fog requires the commander to explore the accessible information at his disposal and reduce any gaps by adjusting collection assets to gather more data. The underlying goal is to improve the understanding - or new knowledge - of the situation in order to make a more informative decision. Maintaining a high level of SA, as with acquiring new knowledge, is predicated on the ability of the individual, team, or system to process and update new information. We thus view situational awareness as a data exploration task whose objective is to acquire knowledge in order to reduce uncertainty (or the Fog of War) and improve the underlying comprehension of an environment.

1.3 Applications

We highlight these objectives and present opportunities through five use cases: scientific simulation, healthcare, Cybersecurity, and military intelligence and autonomous vehicle navigation.

Scientific and Engineering Simulations. Computer simulations have become an integrated component within the scientific and engineering research and development community. With sensors, such as high powered telescopes and spectroscopy devices costing in the hundreds of thousands to millions - or even billions - of dollars the advantages of conducting computer simulations are clear and apparent. Simulations are also a critical risk mitigating factor in areas such as biomedicine where research on infectious diseases carries additional possibility of exposure to harmful organisms. Costs aside, in some scientific fields, simulation is the only viable option for research: studying and understanding aircraft failure or natural disasters, for example, is only otherwise possible in the aftermath of actual events. While computer simulation cannot perfectly replicate a live experiment,

it is a tremendous asset to the research effort. Any improvements in simulation execution and implementation can have a compounding influence on the underlying science.

Cybersecurity. Rare event detection, data exploration, and situational awareness are all important tasks in the daily effort of computer administration and Cyber security. Network and system logs are often referred as haystacks of data for which administrators seek the proverbial needle to build a common description of a system's health. Popular distributed solutions, such as Spark[10] or Dryad[2] provide efficient and effective tools to process and analyze massive amounts of data; however, they treat input sources as external injection points. Such solutions lack a feedback loop mechanism to systematically steer log collection parameters and rely on the human system administrator to configure parameters. Other tools, such as intrusion detection and prevention systems (IDPS) analyze streaming network data and provide responsive actions; however, they have several limitations. Chief among them is the extensive human involved parameter tuning.[11] Furthermore, while these systems can defend and respond to threats, they cannot proactively steer heterogeneous network and system settings to provide a more encompassing internal Cyber picture, or *Cyber Situational Awareness.* This latter capability is completely lacking in the field. We envision a capability whereby systems proactively focus resources on identifying internal vulnerabilities and weaknesses and combine a current operating picture with a threat assessment to better secure a system before it is targeted for an attack.

Healthcare. More so than ever, we are collecting a surplus of personal health information with advanced sensors and even wearable devices. Combining this raw data with an efficient means to collect, store, and analyze the information creates the means to better understand health trends or fight diseases. With records migrating from a paper-based filing system to completely digital medium, doctors can more easily fuse multiple analytical techniques to drive health recommendations and decisions. Integrated hospital equipment which can automatically store and process raw data for not only one patient, but for a entire hospital or even series of hospitals provides a input means

to collect information for downstream analysis. Combined with wearable technologies feeding not only a patient's health profile, but also social networks and research databases, we are now seeing unprecedented aggregation of new statistics on a global scale. Each person's localized, or *personalized* dataset may be a small piece; however, when placed within the context of the larger population, we can now create tailored analytics or personalized models suited to each individual. This gives rise to the challenge of balance between sharing data and securing private information. Systems which can manage this duality can bring about considerable gains in data exploration, providing applications in areas such as personalized health monitoring or disease warning. Such capabilities provide the means to detect the onset of a fatal condition, such as septic shock or cardiac arrest, with the ultimate goal of saving a life. Although risks to personal privacy and security are inherent in the process and should be carefully mitigated, the opportunities to benefit society are tremendous.

Military Intelligence. A challenging application, Military Intelligence Analysis represents the integration of modern technology into a relatively monolithic structure. A continuous cycle of collection and analysis, the methodology involves creating a holistic view of an operating environment to predict an adversary's future actions.[12] By focusing a vast array of assets to gather information on physical and information environment factors within a determined area of interest, military analysts gather raw data and synthesize intelligence in order to develop likely scenarios, known as courses of action. Throughout the process, analysts identify intelligence shortcomings, known as intelligence gaps, in order prioritize reconnaissance and surveillance resources. One could classify the military intelligence process as a massive streaming system which is time tested and refined over centuries. Its major shortcoming is the significant reliance on human interaction. While recent development in automated data management collection have improved capabilities, they have only exacerbated the resultant Big Data challenges. Today's military forces can overcome such challenges by seeking to integrate automated and adaptive data support systems into its analysis cycle.

Autonomous Vehicles. Another direct application we foresee for this research rests within

the decision making process of an autonomous vehicle. Many autonomous decisions, such as whether to stop at an oncoming yellow light or when to make a left turn across traffic, require continuously updated SA. Furthermore, complex, even morally impacting choices necessitate an efficient and effective data analysis and decision making workflow which can quickly process new data to make decisions: a car may some day have to choose between hitting a deer trying to jump across the street or swerving into oncoming traffic - with either choice potentially resulting in a major accident or fatality. We believe that the interjection of a mediation control loop to prioritize data collection and analysis tasks localized to the vehicle is a prime use which can benefit from this research.

1.4 Challenges

Traditionally, data exploration has been a single scoped effort with static input and a terminating objective. Statisticians have previously spent days, weeks, or years combing through offline datasets in the hopes of identifying new knowledge. While this workflow still has a place in modern society, the **demand for a more expedient process is the prevailing norm**, presenting a challenge for data exploration systems to overcome. In part, and somewhat ironically, this situation has arisen out of the exact technological era in which we live. More capabilities has led to a society characterized by instantaneous information gratification with high expectations in computing systems to provide immediate responses to data requests. Projected into the next decade and beyond, this demand will only increase as autonomous vehicle take over roads and personalized healthcare becomes the norm. Our modern, on-demand culture has put the onus on computer scientists to rise up and develop systems which can efficiently deliver knowledge in order to keep pace.

Even more challenging, internally within the field, we are seeing an **imbalance of advances within computer science**. Some areas, such as machine learning, are making dramatic steps forward with new algorithms and methodologies. The integration of their evolving capabilities and requirements has led to a clash with older, existing architectures. Computational resources,

algorithms, and storage technologies originally designed for smaller, local deployments are unable to maintain pace with the blossoming of data science. The underpinning of traditional data management technologies, relational database management systems (RDBMS), simply cannot scale to the meet the sheer volume of data or the analytical demands. Reliant on global visibility of a data domain to performing tuning and other optimization techniques, these system are tied a structured data schema with limited flexibility. Relaxing the stringent requirements of ACID (atomicity, consistency, integrity, durability), many Big Data systems instead seek to provide an eventually consistent guarantee while maintaining availability and partition-tolerance in accordance with the proven CAP Theorem.[13] This has established a paradigm shift in software architecture and development allowing organizations to capitalize on the benefits gained through more dispersed and asynchronous solutions brought about with integrated distributed systems.

As we have stated, contemporary computing environments supporting Big Data analytics are characterized with flexible, scalable, and distributed systems. With applications built on top of a NoSQL, key-value based organization, data analytical engines leverage the unstructured nature of the storage and management technologies to effectively process mass amounts of information and solve problems at unprecedented rates. They require persistent computing nodes with tightly integrated protocols and services which depend on dedicated hardware with long running processes to manage and dispatch individual executable tasks. However, although they are designed to scale, **at some point, adding more resources does not improve performance**. In addition, the same challenge also arises when trying to bring analytics into a diverse set of environments. This includes supercomputing, to improve data exploration over scientific simulations as well as non-traditional computing environments, such as the modern battlefield.

1.4.1 Problem Statement

In the examples listed above, we described data exploration within a cyclic and iterative context. Knowledge generated from one step is not only the output, but also a means to help drive

input decisions for subsequent data exploration steps by steering the input of raw data generation (see Figure 1.3). As analytic systems become more evolved and dataset sizes continue to grow at a record pace, the overhead and ancillary requirements to manage and drive exploration incurs a tremendous burden on the overall system. As this is still a semi-autonomous process, the key and central component within this loop, the human, becomes a bottleneck. Instead of focusing on the more important tasks associated with exploration, namely making decisions, determining what information is interesting, and driving the knowledge synthesis at a higher level, the human-in-the-loop finds himself delving into lower level tasks. These tasks, such as tuning parameters, adjusting filters, monitoring queries, and scanning lower level patterns are more amenable to computer oversight. While one could argue that data exploration should be completely independent of human interaction, we focus on the process as a semi-autonomous one. We perceive autonomous exploration as a philosophical viewpoint, one which is outside the scope of this research effort.

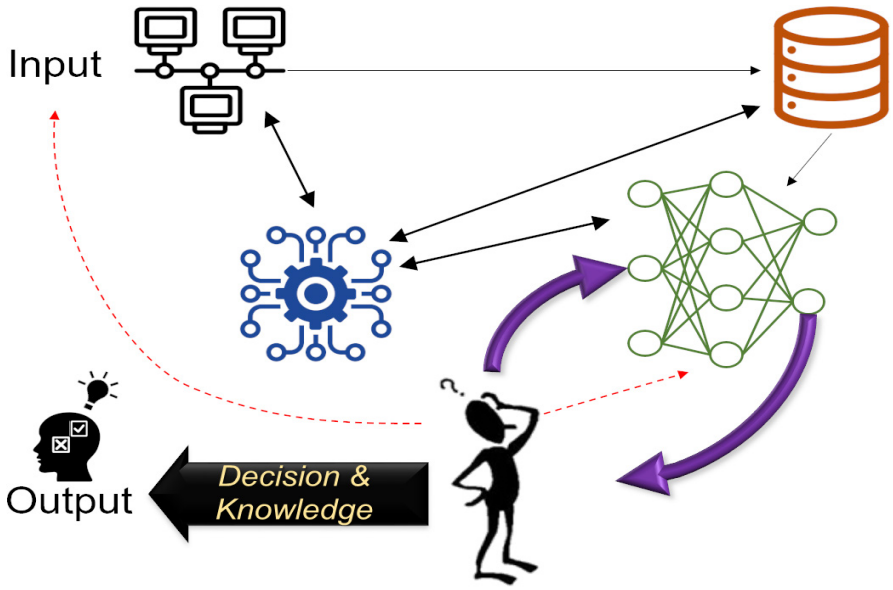


Figure 1.3: *The Data Exploration Loop.* Input is the raw data from source, such as network, sensor, or other stream. The data tier (orange) addresses storage and organization and feeds the analysis tier (green) to transform data into information and knowledge. User interfaces (purple) provide human interaction while the systems tier (blue) integrates the whole process. The output (in black) also helps to drive subsequent data exploration tasks by steering raw data generation. The dashed red lines represent the human-in-the-loop problem we foresee as the human becomes overly involved in lower level tasks which delays outcomes and results.

Given that the human will always be a central figure in data exploration, we identify the human-in-the-loop as a major setback in executing data exploration within the contemporary computing landscape. Because humans partake in lower-level tasks, the entire exploration process is withheld from reaching its performance potential. This results in delayed decisions and knowledge discoveries; outcomes which can be extremely problematic and catastrophic. We, thus, identify the source problem which drives this thesis as follows:

Problem Statement

Human-in-the-Loop management of data generation and distributed analysis tasks inhibits efficient online, iterative data exploration causing delays in knowledge discovery and decision making.

These challenges in data exploration require new solutions and methodologies. We believe the current computer science field lacks architectural support for semi-autonomous methods to efficiently implement the process. This support, enabled through abstraction layers and lower level coordination of autonomous tasks, can help reduce the manual tasks to manage the system allowing the human to focus on the higher level efforts, such as making decisions and synthesizing knowledge. We foresee opportunities through better synchronizing efforts of dispersed resources combining to progress a data exploration goal. Applying an adaptive control methodology with asynchronous consistency management, we aim to improve the computer science field along this direction. We first offer some background on this topic to pinpoint where our work best fits in the computing community.

1.5 Adaptive Control Background

The concept of adaptive control has been studied for decades. Commonly associated with nonlinear control, research has focused on the means with which to continuously adapt a closed-loop system containing at least one non-linear component to remain withing a prescribed specification.[14] A common use case is a home thermostat which controls whether a furnace should turn on or off

based on sensors placed throughout the home. Adaptive control for systems dates to the 1960's with its origin based in NASA technology used as part of the Apollo program. One of its derivations and popular algorithms, the Kalman filter,[15] is a methodology which continuously gathers sensory data and updates a systems active state. Accounting for noise from internal erroneous readings or external factors, the Kalman filter reduces uncertainty via a covariance calculation to update a new state based on previous values. Systems can subsequently adapt execution, such as by changing speed or direction.

Since the turn of the 21st Century, we have seen the rise of combining streaming data with historical data into an autonomous system. Darema proposed the Dynamic data driven application systems (DDDAS)[16] paradigm as a means to collect data online and leverage its value to steer the measurement and modeling process through a synergistic feedback control-loop. We note that our work falls under the DDDAS classification in that we process data during execution to dynamically control performance. During this time, we have also seen a rise of data management advances to traditional systems providing an enabling capability to support adaptive control and integration of analytical tasks. These include column oriented storage for databases,[17] human interactive query capabilities through systems such as CONTROL (or continuous output and navigation technology with refinement),[18] and query approximation techniques which provide users with a less accurate but more timely response aimed at establishing a more interactive, exploratory data session.[19, 20]

Integration of adaptive exploration techniques into data management systems has recently seen a rise in development and research. Kersten, Idreos, Manegold, and Liarou presented the visionary concept of an interactive data exploration engine to enhance the scientific processes.[21] One notable example of adaptive control in database management systems is Automated Query Steering with integrates automated data exploration with database systems.[22] The system features an iterative process which builds a user profile to classify database objects as relevant or irrelevant. Subsequently, it automates query formulation, query processing, and result reviewing to strategically sample data items for user interaction. Together it combines machine learning, data exploration,

and sampling techniques to steer user queries toward more *interesting* results. Specifically, AIDE is demonstrated on evidence-based medicine where a domain expert can interactively work in tandem with the automated exploration engine.[22] It employs decision tree classifiers as an ideal means to model both numerical and categorical data.

Although recent advances have allowed adaptive control to integrate with traditional data systems, the capabilities can more effectively integrate into un-structured, key-value based technologies, such as MongoDB, Cassandra, Dynamo or Redis. With recent algorithmic advances, to include map reduce data processing,[23] we are now seeing a path forward to automate the exploration process. Building a feedback capability on top of the MapReduce framework, HaLoop,[24] for example, provides interface support for iterative applications. An extension of the Hadoop API, it contains a loop control mechanism to continually dispatch map-reduce operations until a user defined termination condition is met. As a key contribution, HaLoop integrated dynamic control of task scheduling into the map-reduce jobs thereby significantly improving the key bottleneck in the framework, the shuffle. Inspired by this dynamic scheduling capability, we integrated loop control within the time delayed scheduling HPC, but target more general purpose programming.

1.6 Thesis Statement

We have outlined how contemporary advances in computational processing and technological capabilities have brought about a surge in data availability. This "Big Data" phenomena has rendered existing architectures insufficient to handle the data demand. This doctoral thesis addresses the data exploration problem in the Big Data era focusing on data and system integration approaches to improve this field. We argue that an abstraction layer providing adaptive asynchronous control and consistency management over a series of smaller tasks coordinated as part of a global effort can significantly improve data exploration effectiveness and efficiency. We support this though a demonstration of systems executing scientific simulations and machine learning training

whose tasks integrate into a global objective coordinated with asynchronous controls designed to improve performance while reducing resource utilization. The thesis statement is summarized into the following:

Thesis Statement

An abstraction layer providing adaptive asynchronous control and consistency management over iterative, distributed data exploration can significantly improve exploration performance

We focus our efforts on a part of this larger problem, applying our methodology to the specific genres of scientific simulations and personalized healthcare. However, our contributions are applicable to a wide variety domains specifically targeting incomplete datasets represented as temporal and online. These methods are agnostic to the volume of data in terms of both dimensionality and sheer size and can apply to both independent and dependant data. We highlight applications in both the medical and scientific domains, but these techniques can easily extend to include image data, network and other input sets. While some fields, such as natural language processing, are not an ideal target to apply these techniques, we believe our research is extendable to a variety of fields. To further frame the problem, we categorize the issues pertaining to data exploration into the following components which correspond to the data exploration loop in Figure 1.3:

1. *Data* (in orange). Data related efforts concern the selection and organization of the input and intermediary storage of the exploratory problem. Example efforts include feature engineering and storage management.

2. *Analysis* (in green). Analysis includes the methods and models to execute the exploratory process representing the tasks of creating information from raw data and knowledge from information. A significant focus of contemporary research in computer science and applied mathematics, this includes algorithms, statistical models, and machine learning.

3. *User* (in purple). User related issues include both the visualization of data and the declarative means with which to interact with a system.

4. *Systems* (in blue). Systems components focus on the abstraction of data exploration processes across all domains and applications. Systems solutions are an enabling effort to provide efficient capabilities for the other components.

We specifically address the data and systems components listed above by integrating adaptive techniques into simulation scheduling and machine learning. In the first part of this research we show a novel means of data organization to more effectively carry out the data exploration process using a feature lattice. In the second part, we focus on adaptive control through a mediation control loop using knowledge acquired to improve data collection. Finally, in the third effort, we extend the exploration process to a multi-domain setting to improve machine learning of localized, or personalized data. In all cases, we seek to provide a more efficient and effective process for attaining a given exploratory goal with ultimate objective of reducing the lower level, burdensome tasks from the human-in-the-loop to make exploration more semi-autonomous. Our main takeaways are:

- We demonstrate a significant reduction in computing resource utilization allowing increased productivity in data exploration. Specifically, our adaptive control methodology for simulation ensemble management improves starting points through advanced sampling techniques. With each independent task producing more productive outcomes, as shown through increased rare event detection, the collective efforts aggregate to provide greater contribution to a common, synchronized objective. The end result is a system which converges faster to a global data exploration goal using less cost. This, in turn, frees resources to run more simulation, exploration, or other analytical tasks and enable the system to be less human dependent.

- We justify the need to develop synchronization primitives as an abstraction for distributed machine learning training. Similar to adaptive control in simulations, any system which aims to autonomously steer multiple, isolated tasks necessitates synchronization mechanisms. Applied to the complex, distributed training environments of modern machine learning applications, such as that found within personalized health, this need is further in demand. With multiple

training tasks sharing dependent, overlapping efforts, which we describe as semi-isolated model training, the field requires more sophisticated and developed abstraction primitives. These controls enable the user to better coordinate the efforts of the asynchronous, dependent tasks with the goal of improving training efficiency and effectiveness as well as helping to reduce the human involvement in lower level tasks of monitoring progress and tuning parameters.

In chapter 2-4, we also provide more background and related work on the current research status pertaining to the technologies on which we focus as well as specific open directions we foresee. As an overview, the remainder of this thesis divides as follows.

In Chapter 2, we focus on data organization as a means to bridge raw data and information. Using this structure, we lay out a data exploration solution to enable adaptive control by examining how it applies to rare event detection. Our novel approach to data exploration centers around a lattice structure which organizes data features to drive input sampling as part of the exploratory process. We target the objective of identifying and classifying rare events, specifically, the rare states and transitions among them within molecular dynamic simulations. Critical knowledge which affects drug design and disease prevention, understanding how biomolecular systems function in these rare events is an elusive and challenging task consuming a significant portion of supercomputing clusters worldwide. We demonstrate how a sampling technique can identify and select more optimal starting coordinates for each simulation resulting in a 22.5x improvement over naive techniques.

In Chapter 3, we build on these data concepts to provide a systems solution implementing an adaptive control mediation loop. This solution contains two abstraction components: one to drive the exploration process and another to manage data as part of a shared global state. We focus on high performance computing to demonstrate how a serverless framework can execute adaptive data exploration. We developed it into a data-driven *in-situ* analysis and control (DINSAC) simulation ensemble manager. Utilizing in-memory data services executing locally inside the computing cluster, DINSAC couples simulation execution with *in-situ* analysis supporting query based sam-

pling and dynamic control. This chapter addresses the systems challenges presented above solving them through a middleware technology. It integrates the data organization and sampling techniques presented in Chapter 2 alongside simulation execution. Capitalizing on the improved exploration outcomes from lattice based sampling, we demonstrate how the serverless framework synchronizes the efforts of multiple, isolated tasks to reduce resource utilization.

In Chapter 4, we re-frame the exploration process by transitioning from the independent tasks of simulations to the dependent tasks found in multi-model machine learning. In contrast to the previous two chapters, which treated all input as one, unified global domain, we, instead, show how to improve exploration with increased data representation through many, semi-isolated sub-domains. This process enables a more localized exploration solution tailored to a specific - or personalized - effort. Specifically, we address multi-model training in distributed machine learning as applied to healthcare. Addressing the challenges associated with providing personalized solutions, namely striking a balance between data sharing and data isolation, we show how synchronization and adaptive controls are necessary to improve machine learning outcomes. Extending the lessons learned from synchronizing the efforts of dispersed, independent simulations, we address the need to control asynchronous execution among many, dependent models. Our work shows how the field should establish abstract synchronization primitive mechanisms to provide users more control over training in order to improve the data exploration process.

Finally in chapter 5, we provide open directions for future research. We expand on the serverless framework presented in Chapter 3, showing how it can serve as a nucleus for a large scale multi-user simulation ensemble management system. As an orthogonal effort, we also describe how explainability can help improve adaptive control in data exploration by opening the black box of analysis tasks. We conclude by describing the integration of the main contributions presented herein. Specifically, we present the case for using a feature lattice approach to organize multiple, semi-isolated models for both training and testing in machine learning and argue how this framework should be considered for data exploration systems.

Chapter 2

A Lattice Based Approach to Rare Event Detection in Molecular Dynamics

Rare event detection is a challenging problem. It entails distinguishing a proportionally small subset of data from among a very large one and is a task we routinely encounter throughout society. Juxtaposed alongside the technological surge to acquire and process information within the Big Data era, this problem is increasingly becoming more difficult. Addressing this problem from a data centric perspective, we present a solution aimed at improving overall system effectiveness. As part of an adaptive *execution-analysis-control* feedback loop, our methodology autonomously steers data exploration toward the identification and execution of rare events. To drive this process, we introduce a novel data analytics technique by exploiting a feature lattice to organize raw data and identify optimal system input parameters. We evaluate our adaptive approach on molecular dynamic simulations and demonstrate a rate of rare event observation increase on average by a factor of 22.5x over comparable naive techniques. We show how the value gained in improving data exploration not

only out-weighs the costs in computing resources but it substantially reduces time spent to meet scientific goals.

2.1 Introduction

Expansions in analytics have created a surge in new technologies and solutions to meet today's big data challenges. Advances in distributed processing, streaming data management systems, and machine learning provide a backbone to transform vast arrays of raw data into valuable information, creating a means to accelerate the synthesis and discovery of new knowledge. The past decade of exponential growth in computational processing power and network connectivity speeds combined with expanding Internet connectivity have led to a significant rise in stream processing systems. This presents both opportunities and challenges when integrating and applying new technologies and solutions to existing processes. Such applications take advantage of multiple hardware resources and software solutions to accomplish more computational tasks in a faster time span. This ensues a data deluge dilemma with humans as the central bottleneck. In this chapter we address exploration at the data layer and present a solution utilizing a feature lattice to drive the exploration process.

2.1.1 Challenges

As outlined in the opening chapter, rare event detection is a challenging problem. It entails distinguishing a proportionally small subset of data from among a very large one and is a task we routinely encounter throughout society. Examples include identifying a network vulnerability from among billions of computing operations per second or detect malignant cells in a human before they can become cancerous or life threatening. In all cases, these events, items, or other phenomena are intertwined into a large population whereby the proportionality is on the order of magnitude of millions or more to one. A brute force solution to simply explore all possible choices (e.g. scan

every cell in a human body) is impractical.

Adding complexity to this problem, rare events can easily be confused with common ones. A simple comparison operation between two input data may not suffice to distinguish an outlier from among a larger pool. We can also consider a third layer of complexity if we aim to improve accuracy of a system by categorizing different anomalies. Trying to identify and distinguish a single entity as a boolean function is one challenges task, but this problem becomes even more difficult if the exploratory task is to identify and quantify multiple rare events. In this chapter, we view these challenges within the scope of molecular dynamic simulations.

To overcome these challenges and enable adaptive control for simulation management, we identify that improving the selection process for input data is a critical area to target. Our sampling methodology and framework helps to dynamically steer individual simulations as part of a larger ensemble. We stress, however, that techniques and framework presented herein are applicable to other genres. They are suited for domains where generated data is prevalent and continual modifications to input domain can drive the system toward a user-defined goal. The primary contributions laid out in this chapter are listed below:

1. We utilize a data feature lattice to perform analytics on generated scientific data. This novel approach, which is inspired from lattices used in traditional OLAP applications in database management systems, organizes raw output to correlate high dimensional data with scientific events.

2. An adaptive sampling technique combining multiple analytical methods which steer the system towards either a data exploratory goal to gain new knowledge or a data convergence goal to more comprehensively understand a structure.

3. We apply our methodology and framework to molecular dynamics simulations to demonstrate a more efficient scientific exploration pipeline. Our results show improved simulation outcomes by producing 22.5x more rare events in substantially less time as compared to traditional, naive

exploration techniques.

We break this chapter into three sections. The first lays out the broad foundation linking input and output in data exploration as well as identifying the key considerations when building an adaptive exploratory framework. The next section discusses our exploratory methodology as applied to scientific and engineering simulations, and specifically to biophysics and molecular dynamics. It includes rare event detection using an OLAP-centric technique and a lattice based approach. The final section covers specific implementation details and several experiments which highlight the methodologies and adaptive steering presented herein.

2.2 Background

We traditionally view data exploration as a human involved process whereby a user employs one or more techniques to find patterns or correlations in an existing dataset or database. We include the caveat that in order to perform data exploration, it must be executed in an efficient manner.[25] We provide a short background to lay the foundation on how we can integrate adaptive control into the data exploration process. This includes data exploration objectives and techniques which we link with input selection through sampling as a means to adapatively steer a system of tasks toward a common goal.

2.2.1 Exploratory Techniques

We begin with the understanding that the exploration process is not a single effort task. We view the process as a repetitive one with individual efforts, components, and executable operations occurring in sequence or in a more disjoint and distributed manner. This is in contrast to a one-shot effort, such as an explorer trying to cross the Ocean to discover a new world or traveling to Mars. By framing the process in this manner, any analytical technique we apply, which can include statistical methods, graphical tools or machine learning, can benefit from knowledge gained between iterations

of the exploration process. This is how we extend simple exploration to be an adaptive exploration approach.

The premise of an adaptive approach is derived from Bayesian inference. First formalized by the mathematician, Thomas Bayes, Bayesian inference is a statistical method in which the probability of a hypothesis is updated as new information is attained.[26] It is often applied to decision making where we use the knowledge of a past event to better predict future events. A prime example is a game of chess: a player retains a running estimate of the current chess board, as the game progresses and the chess pieces update at each turn, she assesses the likelihood of her adversary's next move. Based on that assessment, the player decides on her best possible move. Bayesian inference is employed in our daily lives and found in many applications.

Extending Bayesian inference, we can also approach data exploration as a series of choices. In each iteration, a system or user must select a subset of tasks to execute. For each one that completes, we observe the outcome and decide whether to (1) re-run, or *exploit*, the task again or (2) *explore* a new task. This is the premise behind the multi-armed bandit problem which was first developed by Herbert Robbins in 1952 as a design methodology for sequential experiments.[27] It was later formalized by John Gittens in 1979 as a metaphor using slot machines.[28] The goal of the bandit is to select a set of levers in each turn which can best maximize the monetary payout. After the turn of each lever, the bandit must decide whether to exploit a winning machine or explore a new one. We project this explore/exploit paradigm onto rare event detection as a methodology to perform adaptive control. In our case, a system can choose to either exploit a promising set of input parameters to achieve a desired outcome (i.e. produce or detect a rare event) or it can explore a new parameter space.

Now that we have outlined a basic methodology, our solution must identify the best way to produce tasks which we aim to exploit. To implement this, we examine how to steer this process toward this goal. Given that we are developing a system with a large set of subtasks, which can be distributed in either time or space or both (we could employ this on a single host with tasks executed

serially over time), can we *a priori* affect the outcome of a task without biasing the results?

We examine this question with the assumption that each task is a black box function where we cannot alter the inner logic of the task. This enables our system to generalize as an abstraction layer and apply to a myriad of fields. Thus, to affect the outcome and steer a task to improve progress toward the overarching exploration goal, we focus our attention on the input. These can be tunable parameters, starting coordinates, or any other source data. Given that our process is a cyclic one, we seek ways to link output back to input as a way to adaptively improve our system.

2.2.2 Input Selection

As we stated previously, one possible solution to data exploration entails a brute force method. If we selected every possible input combination for all parameters, we could compile the results and eventually achieve our goal, which, in the case of rare event detection, means that this is a guaranteed solution, given the event exists. Computationally, this implies the if we add enough resources to the problem, then we can solve it. However, this course of action is not only infeasible, but in some genres impossible due to the sheer magnitude of the input parameter set. Thus, we must find alternative solutions aside from a brute force one. To develop our solution, we draw from concepts used in database management and statistical methods and provide a short overview of these concepts below.

Sampling Methods

Sampling in a statistical context refers to selecting a subset of elements from a population. The methods and techniques from which one samples vary as do the underlying reasons for selecting a sample. Sampling is employed when processing a large data set is either infeasible or impossible; thus, the sample is meant to serve as a proxy for the general population. In such cases, one can infer an error value and bound the sampled result within an upper/lower tolerance. Certain issues among sampling include what value(s) to sample, the sample size, how to select samples, and whether to

sample with or without replacement. We provide a brief overview of some sampling technique which are incorporated into our framework.

Basic Sampling. The most common, and simplistic method of sampling is a random uniform sample. For a given domain, a single value is selected among all the possible values. This is considered an unbiased sampling. The sample distribution may fall within an infinite range of potential values (e.g. any real number between 0 and 1) or within a discrete set of values (0 or 1). We also consider that multiple selections may be considered independent whereby one choice has no bearing on future choices or the inverse, dependent where such a relationship exists. We can also introduce bias into the sampler by applying an adjusted or skewed distribution. The most common of these, a Gaussian normal, is a symmetric, bell-curve shaped probability.

Rejection Sampling. Used in Monte Carlo (MC) simulations, rejection sampling iteratively makes a boolean choice over a sample of data. For each iteration, a sample is chosen and it is either accepted or rejected according to an applied acceptance function.[29] Because sampling from the actual distribution function, P, is either impossible or too complex, a secondary distribution function is provided such that it is used to determine acceptance. One classic example is the solution for pi, π, where a random value (x, y) is chosen on a uniform distribution, $x, y \in \{-1, 1\}$, and accepted if $x^2 + y^2 \leq 1$.

Umbrella Sampling. Umbrella sampling employs output from a function or an existing dataset to create an inverse distribution from which to sample. The technique is often applied in an exploratory process to ensure coverage over a domain: given an input of previous choices, we create an inverse or umbrella distribution over the regions in our exploratory space which are sparsely covered. Within the context of rare event detection, and specifically applied to scientific data exploration, we note that Torrie and Valleau first suggested an umbrella sampling approach to molecular dynamic systems.[30] They showed how one can more efficiently study the complete energy landscape of a system by taking a biased sampling from a collection of simulations (as opposed to a random, uniform sampling). To ensure full coverage, the bias should sample proportional to the

inverse of the systems energy probability distribution (hence an *umbrella*). In addition, we note that Palmer and Faloutsos showed how density based sampling improves over uniform sampling technique[31] by simply inverting samples based on density: regions with fewer data points are over-sampled and highly clusters regions are under-sampled.

Stratified Sampling. With stratified sampling, we group the input dataset into district categories, or clusters. We subsequently, create a distribution over each cluster, or strata, and either reweight the sampling distribution of all input choices by factoring in a weight associated with each strata or we could sample from among the clusters. This second technique implies a two-step approach: one to select the strata and a second to select data from within the cluster, noting that we could apply any of these sampling techniques at both of these steps. Stratified sampling is often used to ensure inclusion of undersampled populations in a target distribution.

Importance Sampling. One of the drawbacks with basic sampling techniques is the lack of specificity or an ability to steer (or affect) the results for a particular outcome. This often results in wasted computation resources, such as outcomes producing redundant data, if we assume that the same input produces the same output. In the exploratory process, we want to learn more about unexplored input space, which has a higher likelihood of achieving this goal, given our assumption correlating input to output. Importance sampling aims to overcome this limitation by factoring in a target distribution into a source distribution when sampling. Often, this technique is necessary when the target distribution is either too complex or impossible to sample directly. To implement this technique, we representing the importance aspect of the problem (e.g. a rare event) through a target distribution or an importance sampling density $g(X)$, and calculate a weight with respect to the source distribution, $f(X)$. This provides a re-focusing or re-weighting with a reduced variance resulting in a sampling that favors important samples.[32] The likelihood ratio is given as:

$$w(X) = \frac{f(X)}{g(X)}$$

Using this weight, it is possible to establish the estimate with respect to the importance sampling density, given N samples:

$$E(f) = \frac{1}{N} \sum f(x_i) w(x_i)$$

Approximate Query Processing

We reference the literature behind this database technique as we include these concepts in our solution. Approximate query processing is a database management technique to overcome the delay incurred when responding to queries over large data sets.[33] This is a technique which trades space for time. By *approximation*, a query plan allows for less than 100% accuracy in order to provide more timely and efficient responses to user queries.[19] The challenge is in determining how to provide an approximation on a full data set and how to associate that approximation with an error band or confidence interval. Various techniques to perform the approximation include, sampling, which we highlighted in the previous section, caching which is a data-centric approach, as well as other more analytical techniques which employ historical queries to predict future query patterns.

Online Aggregation. Hellerstein, at el introduced online aggregation (OLA) as a means to provide users with timely feedback of partial results while a DBMS continued to process the remainder of a given query.[34] They provided a user with information on approximate time to completion and a confidence interval for accuracy as quantifiable measure to balance accuracy for speed. Users could interject and make the computation much more effective (and efficient) by identifying thresholds for accuracy, setting constraints for time, and prioritizing tasks among multiple queries.

Core Sets. A coreset is a minimal set of points which approximates a large data set while still maintaining the original properties of the source data. Operations performed on the coreset provide near similar results as if performed on the full data set, within a tolerable error, ϵ, but at

a much cheaper cost.[35] Coresets are applicable to certain datasets, to include spatiotemporal data, but identifying which points to retain is also a challenge.

Meta-Data Statistics. if we can identify secondary or summary statistics over our dataset, then we can utilize them as part of a global sampling strategy to help focus the exploration process. In a database this implying retaining specific information over (a) data distribution within a data set, such as knowing that a certain percentage of queries will always *select* or *group by* a specific data value, or (b) data values included in future queries, such as knowing that most queries will always include a specific set of columns. Extending this concept, systems such as BlinkDB,[36] employ an adaptive framework centered around statistical stratified sampling of query column sets (QCS). This allows the system to maintain an accessible sample set of data which can best meet future queries.

Bootstrapping

One of the critical aspects of providing an approximate sample for a general population is ensuring an error component associated with the result. Systems often provide an upper/lower bounds on the error as part of an aggregate functional computation (e.g. the approximated average over a sampled set of numbers is $\hat{\mu}$ and it is accurate within 3% where $0.97(\hat{\mu}) \ll \mu \ll 1.03(\hat{\mu})$. A major limitation in confidence bounds is the specificity associated to the aggregate function. Each function requires its own separate confidence calculation. A more general approach is the bootstrapping technique developed by Efron in 1979.[37] The basic premise of bootstrapping is to (a) select a sample data set, (b) generate a set of *bootstraps* from the sample, (c) apply an operator to the bootstraps and then compare the aggregate results to the sample set. The distribution of the bootstraps reflect the original data set due to the law of large numbers: as the sample size grows, the approximation will more closely resemble the actual data. Furthermore, the distribution of all bootstrap results provides a confidence interval on the as the error bounds of the operator over the full data set.[38]

2.3 Scientific Computing

2.3.1 Simulations

Molecular Dynamics (MD), one such scientific genre, employs computer simulation to trace the energy and physical properties of biochemical structures. The process employs computing intensive applications to iteratively calculate spatial locations and velocities for individual atoms and determine the overall equilibrium for biomolecule, polypeptide protein or nucleic acid.[39] Simulation steps typically occur at the femtosecond time scale with each step calculating the energy and new physical configuration of the bio-system; the collective set of timesteps define the biomolecule's spatiotemporal trajectory which represent anywhere from a few picoseconds to milliseconds of real-time biophysical interactions.

The individual positions and velocities over time provide a partial means to describe the characteristics of the underlying biophysical system. With additional factors, such as tensor angles, temperature, atoms bonds, the comprehensive feature set defines high-dimensional energy landscape. Temporal data and the changes over time compliment atemporal structure to provide a more comprehensive energy landscape description. In a more abstract classification, the system transitions through varying states of activity. Among the most critical tasks for scientists is to better explore and understand these various states and the transitions among them. Such information is critical to fighting infectious diseases, developing drugs, and screening for cancer. Conducting simulations facilitate exploring a protein's global landscape to discover new features and to converge on how the proteins act during different states of its existence.

With a recent rise in computational power, the MD community has capitalized on the increased efficiency, parallelism, and processing speeds to execute simulations. Despite advances to integrate graphical processing (GPU), parallel programming models (MPI), and remote memory access through Infiniband networking (RDMA), scientific computing is inherently still reliant on a supercomputing architecture dating back to the 1980's with programs typically written in C, C++,

or Fortran. We highlight the systems integration challenges in Chapter 3 and discuss the data centric challenges below.

2.3.2 Data Challenges

Contemporary biophysics and biochemical computing in MD maintains a segregation between simulation execution and offline analysis. Users allocate resources on high performance computing (HPC) clusters to run simulated experiments which can run from hours to days or even weeks of total wall clock time. Carefully selecting critical start points can help to maximize the value of generated data and the time spent performing simulation; however, due to the stochastic nature of the experiment, one can never ascertain the simulation's value until after completion. While some efforts demonstrate how event detection can be more autonomous,[40] it is still viewed an an offline process. Due to resource limitations in a shared computing environment, users are faced with strict storage ceilings. This results in a common Big Data dilemma whereby storage is insufficient to meet streaming data demands to support down stream analysis tasks. Thus, users must either prioritize and retain selective data or provide external resources to off-load overage. Scientists typically prefer to retain everything, which leads to a follow on issue of data migration management.

Once completed, a system must move large amounts of data from clustered to remove servers (either cloud or locally hosted) for offline analysis and processing. Control over data movement is recognized as a major constraint within scientific computing[41] entailing an extract, transform, load (ETL) operation for each output item. Because users cannot pinpoint critical data to retain as simulations execute, they must move it to remote locations in order to conduct offline analysis. If a given scientific objective is not met, analysis results feed a refinement step to derive a new set of input parameters for a next cycle of experiments. Although the data input requirements are minimal in MD, targeting an optimal set of parameters can expend unnecessary and potentially costly computing resources. It should also be noted that other scientific or engineering fields may have a high input data requirement which would necessitate a secondary data migration task to

move data back into the HPC. We argue that coupling online data management and analysis with simulation execution inside the HPC environment can reduce the data transfer rate - essentially eliminating the 'E' and 'L' from the ETL operation - and provide a means to reduce data retention requirements by distinguishing critical and valuable data immediately after output generation.

A common input selection technique, parameter sweep (PS), organizes input data sets with one or more open parameters and subsequently schedule workflow tasks to execute and analyze results for the domain of input values. PS is a means for scientists to explore a large and potentially unknown input space; however, PS can consume unnecessary resources in trying to explore biosystem states which yield no significant output. Dynamic workflows which aim to iteratively influence parameter sweep selection have been shown to improve overall system efficiency.[42] However, a lack of *in situ* analysis coupled with the parameter selection process limits their ability to be both adaptive and accelerating during experiment execution. Such systems must either process a complete set of experiment output en mass in an offline manner, whereby large amounts of data are potentially moved across computing resources, or a human must interactively steer parameter selection toward a desired end state. We argue that despite the data management overhead, integrating online analysis process (OLAP) with simulation execution can be more efficient than an offline workflow management process which relies on conventional processes, such as PS.

Coupling analysis with simulation would provide tremendous gains in not only steering scientific input but also in prioritizing data for retention within the clustered computing environment. Some solutions provide a supporting effort to facilitate experimentation, such as scientific workflow management systems (SWfMS), which integrate various processes during the experimentation lifecycle. Unfortunately, they lack a data management integration capability to make adaptive decisions. This leaves the current state of the field lacking a data staging capability, in situ analysis of big data, or integrated dynamic steering of scientific goals[43] – all of which could tremendously benefit the field. We recognize these shortcomings as potential research opportunities to improve scientific computing.

2.4 Methodology

Given the current state of MD computational simulations, we identify a set of objectives upon which our framework is aligned. Specifically, we target users who have one of two primary goals when conducting simulation experiments: (1) a discovery goal whereby users seek rare event sampling to not only find the rare events, but also understand the relative frequency of occurrences and the likelihood of states leading up to them; and, (2) a convergence goal whereby users desire a more comprehensive picture of the feature space exhibited by a biomolecular structure. To meet these goals, we leverage the research on bovine pancreatic trypsin inhibitor (BPTI). This 58-residue polypeptide protein is well-studied since 2010 due to the published 1.03 millisecond labeled dataset.[44] Consisting of 892 atoms, BPTI is known to exhibit five slow motion states and numerous fast-motion sub-states. We aim to better understand the rates of transitions among these states and sub-states and more comprehensively understand the global BPTI feature landscape. We, therefore, present the following model for query based sampling with scientific simulation data exploration and subsequently present a more novel approach using a feature lattice.

2.4.1 Data Representation

Simulation data in MD consists of a relatively small number of input parameters while output consists of very large datasets, measured in gigabytes, terabytes or petabytes. Specifically, our system input consists of initial starting coordinates for all atom positions in Cartesian Space with input domains restricted to a bounded set of values as defined by the physics of the simulation (e.g. atoms cannot physically overlap).

Incorporating high dimensional data for analysis is among the primary challenges within the scientific computing field. Generating data down to the femtosecond timescale could produce up to a gigabyte of output coordinate data every second of computing time; however, only a sampling set is retained. Each stored frame, known as a conformation, represents a single atemporal data

point for which we apply data labeling techniques. Aggregated data over time combines into a temporal data point over which we apply a covariance matrix calculation.

The total number of primitives for a data element defines the total dimensionality of the program in high-dimensional space ($\mathbb{R}^N$). Additional values, such as temperature and force field, are also potential input parameters along with alternative representations of data (e.g. ϕ, ψ angles). Conveniently, the output schema is the same data as the input schema stored as a temporal collection of Cartesian points. In order to manage this vast amount of data, we employ common data reduction techniques, to include root mean square distance calculation, as well as linear and kernel principle component analysis. Using reduce dimensionality data, further clustering, processing and classification is employed via KD Tree and K-dimensional grid structures to derive a hypercube, a spatially defined structure which is discussed in more detail below.

Classification

Classification of data requires either a prelabeled comparative dataset or a scientific means with which to categorize observations for unlabeled data. We leave the latter of these options as future work to implement this framework for further expansion on data exploration and instead focus on leveraging pre-labeled data. We employ the published 1ms D.E.Shaw labeled data set for BPTI[44] as the basis for data labeling. Utilizing this data set, we define a feature classification strategy integrating a small fraction of the total MD dimensionality. It is noted that accurately describing the complete free energy landscape of BPTI in its entirety requires hundreds or even thousands of dimensions which is outside the scope of this effort. Thus, we define a simplified feature classification strategy, and from this, we detail both a single and multivariate labeling technique:

1. **Single Variate Labeling**. This technique ascribes a single label to each observed conformation. For this experiment, we classify each observation with a state based on its observed root mean square distance to a predefined set of centroid locations. RMSD is widely employed within the biophysics community and has been demonstrated to show data correlations.[45]

Each of the five known BPTI centroid positions are derived from the prelabeled dataset based on several locally run simulations in implicit water solvent. We classify as a point as either in a *well* or a transition out of one of the well states based on its aggregate proximity to each of the centroids. We further specify both a primary label, the nearest centroid, and a secondary label: the next closest centroid if the observed conformation is close to equidistant between the two centroids. For the five states of BPTI, this provides a set of 25 difference labels or *bins* into which we group atemporal data.

We observed that single value classification is very sensitive and difficult to clearly articulate. This subtle point is illustrated in the center section of fig 2.1. In a single variate manner, this image classifies as State 1; however, observe how the difference between its primary state and States 0 and 2 are not very pronounced, especially given the variance as shown in the red error bar. This is indicative of the challenges within MD when classifying and labeling data.

2. **Multivariate Labeling**. Instead of classifying data elements as existing in one specific state or sub-state, we define an alternative data representation, which we call a *feature landscape* consisting of a distribution of several features. With in the genre of MD, feature selection is nearly endless: Cartesian Coordinates are one; however, torsion angles, velocities, and any subset of data elements could be factored in a feature landscape. For purposes of this study, we limit the scope of the feature landscape data items: the relative distances to a common set of pre-defined state centroids. Using this metric and the total count of observation, we establish three sets of features comprising a 20-dimensional data point. Figure 2.1 shows these 20 dimensions organized into three groups. The first represents the normalized single variate count for the nearest neighbor for each observation in temporal or aggregated data (or even a single observation). The second group describes the proximity to each centroid. To capture

proximity, we employ a logistic function:

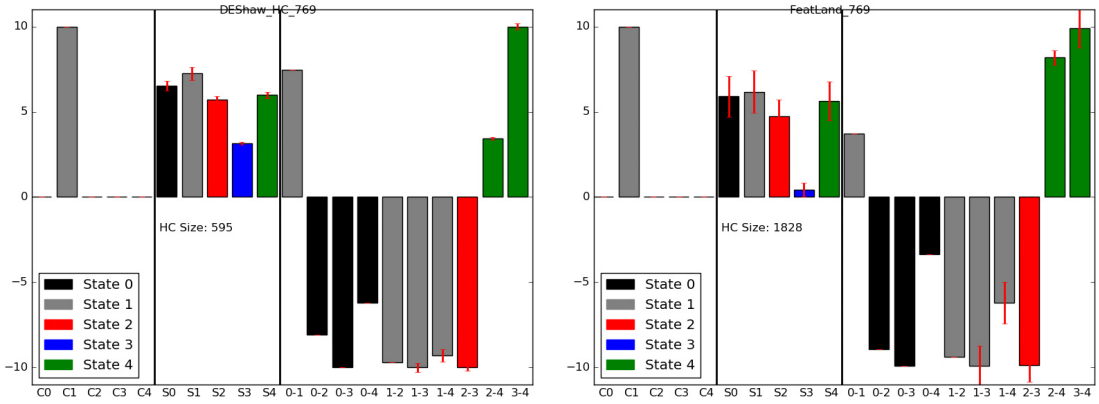
$$\frac{1}{1 + e^{-k(-d_{ic})}}$$

Where d_{ic} is the distance to centroid c for observation i. The constant, k, is a scale factor which we have set to 10. The third group is a second-order, pairwise relationship of distances to two centroids for each pair of centroids. Derived from landmark multidimensional scaling (LMDS),[46] this method uses only the five predetermined centroids as landmark points and then calculates the distance between each preserving polarity which is highlighted on the graph for positive and negative distance values. For example, the column for 2-3 is approximately -9 and is closer to centroid 2 than 3, hence it is red while the column for 2-4 is positive and closer to state 4. For the five centroids of BPTI, this represents 10 pairs (non-directional), but this feature space could easily become more complex by adding a third-order component.

Hypercubes

We employ a spatially defined hypercube data structure as the basis for our fist data organization solution. The concept is derived from data cubes in database systems which has been used in a variety of ways.[47,48] Algorithmically, we create a KD Tree over the raw data set to derive a clustered set of data points in either high dimensional space or in a reduced subspace dimensionality. The resultant leaves of the tree cluster into k-dimensional hypercubes where points within the hypercube represent data input values from which a sampling technique can draw. We chose to implement a hypercube model to prepare data for an analysis and subsequent sampling process due to the natural correlation between hypercube operations and query processing. Such a technique has been recently shown to improve stream processing efficiency and effectiveness[49] and is related to techniques incorporating data cube and map-reduce processing.[50,51]

Associated with each hypercube is a weight value which can include number of points,



(a) Source Data: Pre-labeled, from long running 1.03ms experiment (D.E. Shaw)

(b) Source Data: Generated locally from a 200ns adaptively controlled simulation

Figure 2.1: Graphical Representation of the MultiVariate Feature Landscape employed to describe MD data. Each of the three sections in the two images defines a unified feature set: the left five columns, C0..C4, define the normalized number of observed items with closest RMSD to the given centroid. Next 5 columns, S0..S4, are absolute average distances to each state centroid. The rightmost 10 columns pairwise relative distance measurements for RMSD to respective centroids. Number of observations for this Hypercube is also indicate on the graph. Also note: vertical scale only applies to the S0..S4 labels. Both of these images represent the same selected hypercube of data after applying the same spatial clustering, K-D Grid, algorithm to different data sets. The left image is from pre-labeled data and the right is from a generated generated data using our adaptive framework.

volume, or a mass/density value. We can aggregate data within a hypercube using the previously described feature landscape distribution as depicted in figure 2.1. Some of the operations over hypercubes include: filter and mapping, set operations (union, intersection, difference), splitting along one or more dimensions, projection or reduction to few dimensions, and a reweight operation which we describe below. It is also important to note that these operations can be chained together to create a more complex query over data with the end goal of producing a subset of hypercubes with a sampling distribution in order to select subsequent input coordinates.

2.4.2 Query Based Control

Adaptive control ultimately requires a decision on which set of subsequent input values. As with any decision support system, a controller needs a global assessment of the current state and a list of potential decision choices. We represent the state as a status toward a user defined

convergence goal and the latter as a resultant set of output from a sampling algorithm.

Bootstrapping for Convergence

We employ a bootstrapping technique to provide a general purpose metric on determining system convergence. We use the size of this confidence interval (CI) as the premise for determining convergence: data with a tighter CI have a more converged state. Specifically, we employ a non-overlapping, blocking bootstrap algorithm which groups sets of observed conformation into a single bootstrap. Over time, convergence is determined by calculated the bootstrap over all previous blocks:

Algorithm 1 Confidence Interval

1: $[\hat{x}_i] \leftarrow X$
2: $\hat{\mu}_i = f(\hat{x}_i), \forall\ \hat{x}_i \in X :$
3: $\hat{\mu} = \dfrac{\sum \hat{\mu}_i}{|X|}$
4: $\delta = sort(\hat{\mu}_i - \hat{\mu} \forall \hat{\mu}_i \in \{B\})$
5: $CI = (\hat{\mu} + \delta_{low}, \hat{\mu} + \delta_{hi})$, where $low = \dfrac{1-I}{2}, hi = I + \dfrac{1-I}{2}$

Sampling

We provide multiple sampling techniques as part of this framework. As stated above, sampling in a statistical sense refers to selecting a subset of elements from a population. In our context the population is a stratified set of hypercubes for which we can attribute a weight, such as number of observations, spatial size, or total convergence. For a given domain, a single value is selected among all the possible values. As a naive approach, we provide a baseline comparison of applying a uniform sampler over the set of grouped, single variate sets of *bins* as described above.

In contrast, we also incorporate an umbrella sampler which inverts a observed distribution over a data set. To ensure full coverage, the bias should sample proportional to the inverse of the systems energy probability distribution. We implement this umbrella sampling over the same set of *bins* using the calculated convergence over each as the input distribution. Thus, states and sub-

states which are least converged will be more likely selected as candidates for subsequent execution. For both of these, when a *bin* is sampled for a subsequent control decision, a secondary uniform sampling is performed to find a new starting coordinate.

We have also implement a multivariate nearest neighbor (MVNN) data sampling algorithm. Based on nearest neighbor solutions in data analytics,[52,53] this approach leverages the feature distribution as an input vector to a standard nearest neighbor search. For input, we use the umbrella over system's aggregate mean feature distribution among all previous data points. The resultant target distribution is probed against each hypercube (or leaf) from a KDTree constructed among all data points. Once probed, we retain the ten nearest hypercubes and use the normalized euclidean distance among all 10 as a weight for the discrete probability distribution from which to sample. As with single variate sampling, a secondary uniform sampling is performed to find a new starting coordinate.

Reweight Operator

As a final sampling technique to support a data exploration task, we introduce the reweight operator. Comparable to an ensemble learning techniques which leverages different machine learning techniques, the reweight operation applies two different clustering algorithm on the source data. Using the hypercube data structure, we demonstrate how combining clustered atemporal data consisting of single frame observations with temporal time windows of trajectories can highlight areas of the data space that an objective convergence function can either explore or exploit in a active learning fashion. This operation selects data points in one dimension from a set of clustered spatial hypercubes, back-projects each sampled point to higher dimension, and then projects it into a target subspace. Weight attributed to hypercubes from each source subspace, such as size, are combined a part of a join operation using a user defined gamma function. The end result is an aggregated set of hypercubes which can be sampled for select input parameters for subsequent simulation execution.

Defining how data aggregates among the different subspaces through the gamma function, γ

Algorithm 2 Reweight Operator: $\oplus_R$

Input: Sets of hypercubes and weights in two different subspaces, a lambda function (for data projection) and a gamma function (or data aggregation):$\{H_a, W_a\}, \{H_b, W_b\}, \lambda_a, \gamma$

Output: Set of hypercubes with weights: (H', W') *s.t.* $\sum w_i = 1,\ \forall w_i \in W'$

1: Back-project H_b to *N-Space* $\{H_{(b\perp N)}, w_b\} : \mathbb{R}^b \rightarrow \mathbb{R}^N$

2: Project $H_{(b\perp a)}$ to *a-Space* $\left\{H_{(b\perp a)} : w_b\right\}$: $\mathbb{R}^N \rightarrow \mathbb{R}^a$
$\forall h_b \in H_{(b\perp N)}$: $h_{(b\perp a)} = \lambda_a(x_b)\ \forall x_b \in h_b$

3: Overlay: $\{H_a, w_a\}, \{H_{(b\perp a)}, w_b\}$ on same sub-space

4: Join correlated data: $H_a \bowtie H_{(b\perp a)}$
$\forall H_a, H_{(b\perp a)}$: $h'_i = h_a \bigcup h_{(b\perp a)}, w'_i = \gamma(w_a, w_b)$

5: Normalize, W': $W' = \dfrac{w'_i}{\sum w'_i}$

reveals how the adaptive sampling process can be finely tuned to steer simulation execution toward potentially new *interesting* starting points. These starting points represent ideal positions which ideally lead to rare event occurrences in the simulation (or events leading up to such occurrences which can also provide insightful knowledge). The ultimate objective with this operation is to support the data scientist in her data exploration tasks. We believe we have only covered a small portion of the potential for this type of operation and leave additional research for future work.

Back Projection

To tie in the explainability lineage of output to input, we define a the concept of a back-projection function. Its purpose is to serve as the inverse of the executing task and can be a significant resource consuming operation. We implement it as the same operation employed to perform sampling on data when transforming targeted output hypercubes into starting input parameters. This operation relies on historical provenance of data as maintained in the online catalog through the overlay service. In some cases, data retrieval back to higher dimension may require a file transfer from an off-site or archived location. For our experimentation, we retained data locally within the distributed file tier and still found that this operator back projected upwards of 100K points from thousands of trajectory files consuming over 10 minutes. For this reason, we integrated an in-memory caching overlay service to significantly improve data management support by a factor of 10. While back projection can retrieve original source data for reweighting, it can implement any

aggregation function (e.g. return the mean Cartesian coordinates for a trajectory of atom positions). We also recognize that one could project and retain data for all subspaces of reduce dimension data locally and completely bypass a back projection requirement; however, even reduced dimension subspaces can grow over time and require offloading and approximation due to local memory or storage limitations.

2.5 Feature Lattice

We finally discuss our novel and contributing technique for data exploration. Building on the sampling concepts listed above, the lattice is a different approach from the hypercube based methodology. Drawing inspiration from formal concept analysis[54] and online analytical processing (OLAP), the lattice is based on data cube concepts representing a multidimensional array structure for maintaining and computing summary statistics. Originally formulated by Gray,[55] they enable N-dimensional aggregations and relations as well as hierarchical tree-based organization. Typical data cube operations include the *roll-up* for merging and the *drill-down* for refining aggregated data. The lattice is constructed and maintained as part of the global control decision macrothread. We first describe the background and then show how it integrates into the system pipeline to adaptively steer data exploration.

2.5.1 Control Data Model

We represent the simulation ensemble output as a set of K-dimensional feature vectors for N timesteps, defining an $N \times K$ matrix. This input matrix can be an set of raw trajectory data or it could be the result of a series of transformations; our design is flexible to support a variety of data manipulation operations to produce this input matrix. Given this dataset, we define the feature lattice as a graph model consisting of nodes and edges based on frequent itemsets from the input matrix. Each node in the lattice, $C_i = \{K_i, N_i\}$, represents a unique subset of features, $K_i \subseteq K$,

collectively observed in a set of timesteps (tuples), $N_i \subseteq N$. Using Figure 2.2 as reference, node C_6 has the feature set, $K_6 = \{A, B\}$. It also contains a set of tuples, N_6, where both k_A and k_B were simultaneously observed.

Lattice construction is predicated on a boolean function, F_a, which accepts a feature and measurement, k_i, and determines if the given feature is observed in the input event. A common example function is the threshold calculation as shown below, however, any other boolean function could be applied. Given a cutoff parameter, λ:

$$F_a(k_n) = \begin{cases} 1, \text{if } k_n < \lambda \\ \\ 0, \text{otherwise} \end{cases}$$

We apply F_a to all features for all events, resulting in the incidence which is a 0-1 matrix of dimension $N \times K$

We reduce the scope of the feature set, K, to K' by removing all trivial features, k_m^*. A feature is deemed trivial if it is observed for every event or never observed at all. This can be relaxed to adjust for potential noisy events by changing the threshold to meet or exceed a pre-determined user-defined noise parameter, θ_n:

$$K' = \{k_n | F_a(k_n^*) < \theta_n \text{ and } F_a(k_n^*) > (1 - \theta_n)\}$$

Traversing up the lattice, two nodes combine into a parent node by performing a union operation on their feature sets and an intersection on their tuple sets. We formally state that a parent-child relationship exists between two nodes, C_i, C_j, *s.t.* C_j is a parent of C_i iif $K_j = K_i \cup \{k_x\}$ where $k_x \notin K_i$. Extending our example, C_7 is the parent node of C_6, where $K_7 = \{A, B\} \cup \{C\}$

We conceptually create the complete power set lattice consisting of all combinations of all features for all dimensions. Although fast algorithms exist,[56, 57] constructing the complete lattice is

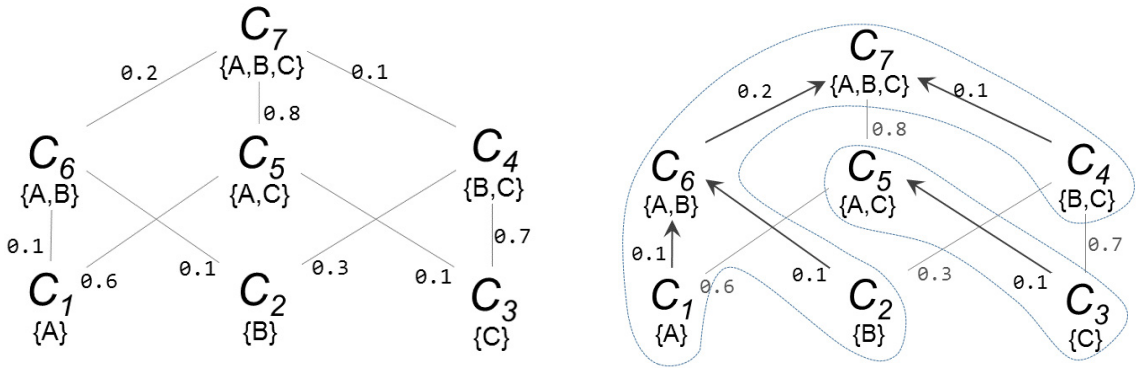
computationally expensive with complexity of $O(n^K)$ as it entails a union and intersection operation on all adjacent nodes. For efficiency, we employ a max-miner algorithm[58] to materialize only the nodes whose item count exceeds a minimal support threshold, θ. We discuss the impact of this θ parameter on sampling performance in Section 2.7.5. With respect to materialization, we define two classical methods commonly found in system processing which answer the question: do we materialize results immediately or defer computation until the results are needed? We highlight these two techniques below:

Lazy. In a lazy approach, computation is performed only when needed. This method provides more resources to the main task at head (e.g. answering user queries or performing analysis tasks) and it may eliminate unnecessary processing if the materialized results, e.g. the lattice clusters, are never used in subsequent efforts. This technique is the underpinning of functional programming as functions are only evaluated when needed.[59] The main drawback in lazy processing is that the system must ensure it retains enough information to execute the computation when necessary. In memory constrained systems, this could be problematic or require paging to secondary storage.

Greedy. In a greedy fashion, materialization is computed immediately when data is received. In a classical data systems setting, this technique may be necessary since information may not be available after a process executes. As noted, however, greedy processing incurs additional overhead. Because we employ the lattice as part of an adaptive system, we defer computation in a lazy fashion and forego the greedy approach, only executing the algorithm when we need to make a control decision.

Once materialized, we add weighted edges in the lattice between all parent and child nodes. We project a parent node's distribution onto each of its child's feature spaces by removing the one extra feature, k_x, and comparing distributions over the common feature set, K_i. Specifically, we employ a modified earth mover's distance (EMD) algorithm to calculate the distribution difference between two nodes and set the resultant value as the corresponding edge weight in the feature lattice. Referencing our example, the parent-child edge between nodes C_6 and C_7 has a weight of 0.2. This

represents the relative EMD work to convert the distribution of values for node, C_6, to node, C_7 along the common feature set, $K_i = \{A, B\}$.



(a) Lattice Construction

(b) Rolling up Nodes into Cuboids

Figure 2.2: Left: Feature Lattice is constructed with edge weights calculated from the distribution difference of each child-parent node. Right: Two cuboids are identified by rolling up child nodes into the parent with which it is most similar.

We trade accuracy for efficiency in order to enable lattice construction and support in situ analysis and control inside the HPC which we will cover in Chapter 3. The modification we apply to the standard EMD algorithm calculates an approximate value using corresponding differences for each pair of single dimensional distributions over the two input data. Conceptually, the EMD algorithm, is a linear programming model which finds the set of flow values, f_{ij} with a given cost model, c_{ij} that minimize the work to transform one distribution into another.[60] Our modification calculates the flow from one distribution to a second for each feature, $k_n \in K_i$ and subsequently aggregates the total flow for all features required to transform the set of one-dimensional distributions. Histogram data is incrementally maintained using a predefined size and width of bins over the data set. By fixing these parameters, the EMD cost model is reduced in complexity, helping to speed up both individual operator execution and long-term lattice maintenance. Concepts derived from incremental view maintenance approaches;[61,62] these two factors are critical in allowing DINSAC to perform in situ analysis and control in the HPC.

The key takeaway is the ability to compare adjoining parent-child nodes for similarity in

high dimensionality enabling the system to aggregate nodes in the lattice. Parent and child nodes with similar data distribution can collapse together: we formally state that a child *rolls-up* into a parent node to form a *cuboid*. Iteratively, we roll-up all nodes to at most one parent node by selecting the parent with the smallest weight until either no parent node exists or all parents nodes have high edge weights, above a pre-determined threshold. This operation is shown in Figure 2.2 (b) where nodes, C_1, C_2 roll up into C_6 which subsequently rolls up into C_7 to form one cuboid. A second cuboid is formed by rolling up C_3 into C_5.

2.5.2 Clustering and Sampling for Control

We cluster the lattice to identify high dimensional data patterns in the feature sets which correspond to lower dimensional patterns or states in the scientific model. We first explain the methodology employed, which derives from traditional correlation clustering techniques, and then show how it relates to the data exploration process.

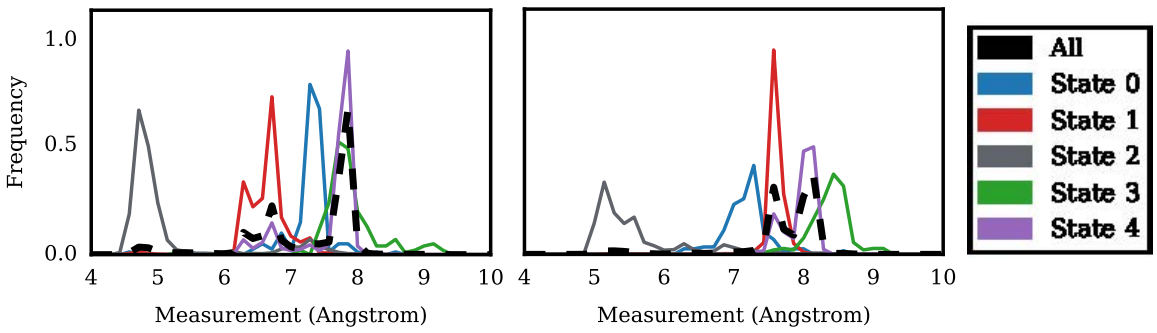
We establish clusters by assigning raw data points to one or more lattice cuboids. For each row in the input matrix, we identify the corresponding cuboid containing the node with the same feature set as the data point, if the node materialized during construction. If not, we traverse down the lattice, iteratively reducing the data point's feature set by one attribute to match it with a node in a lower tier. If one or more matching nodes are found, we assign the data point to all associated clusters with matching nodes in that tier. The algorithm is provided below:

As an additional step to improve downstream sampling, we plot the clusters in euclidean space using the raw data from the input feature domain. We calculate cluster centroid coordinates and employ a hierarchical, correlation cluster graphic based methodology to reduce the number of clusters. This merging step iteratively identifies the two closest clusters and merges them together, stopping when a pre-determined number of clusters is reached. We explore the impact of this hierarchical step on sampling accuracy in Section 2.7.5.

The outcome of clustering produces groups of data which correlate in both high and low di-

Algorithm 3 Lattice Clustering

```
Construct Lattice using Max-Miner Algorithm
Calculate EMD for every parent-child relationship
Collapse Lattice nodes to cuboids: {C} → {C'}
Initialize all D_i = { } for all C_i ∈ C'
for e_n in E do
    Assign e_n → D_i based on best matching feature set
Prune D: Let D' = {D_i | LEN(D_i) > 1}
Calculate centroids for all D'
G ← pairwise distance for all centroids
while LEN(D') > numcluster do
    Select D_i, D_j with smallest G_(i,j)
    Merge D_i → D_j: D_j = D_j ∩ D_i
    D' = D' − D_i
return D'
```



(a) Feature A (b) Feature B

Figure 2.3: Correlation clustering. Distribution of measured values (in Angstroms) for two of the input features, A and B. Dashed line is the aggregate distribution for all events where each corresponding feature was observed in the output. Solid lines are distributions for clustered data colored to show correlation with scientifically labeled states.

mensions. Applied to spatial-temporal data for a scientific simulations, the clusters identify a unique state that the underlying simulated model exhibits. Specifically, we demonstrate this relationship in 1D using molecular dynamics trajectory data from our experiment section which we graphically represent in Figure 2.3. Here we depict two features, k_A and k_B, whose raw data is the distribution of distances over time between two atoms within the biophysical protein structure, measured in in Angstroms (10^{-10} meters). The dashed black lines are the distribution of values from the tuple sets for the corresponding leaf nodes. The solid lines are the distributions of values for five resul-

tant clusters. They plot as compact, well-defined, higher-dimensional distributions corresponding to lower-dimensional states in the source scientific model: each of the five clusters identifies with a unique state, labeled 0-4, as shown in the graph.

One additional advantage of a lattice is follow-on hierarchical clustering by aggregating rolled-up cuboids. The natural structure of the lattice enables grouping sub-states in a taxonomic manner to further improve data exploration. This can help with not only data classification, but also with sampling for control by improving the input domain for a subsequent sampling strategy. We provide the pseudo-code for the lattice clustering is provided in Algorithm 3.

We exploit the feature lattice structure to devise a sampling strategy and show how it can meet a user-defined data exploration goal. In an ideal setting with unlimited time and resources, optimal data exploration entails selecting every lattice node and subsequently choosing every possible starting input parameter combination. Realistically, we revert to a sampling strategy to select the best set of nodes with a limited resource budget of both time and space.

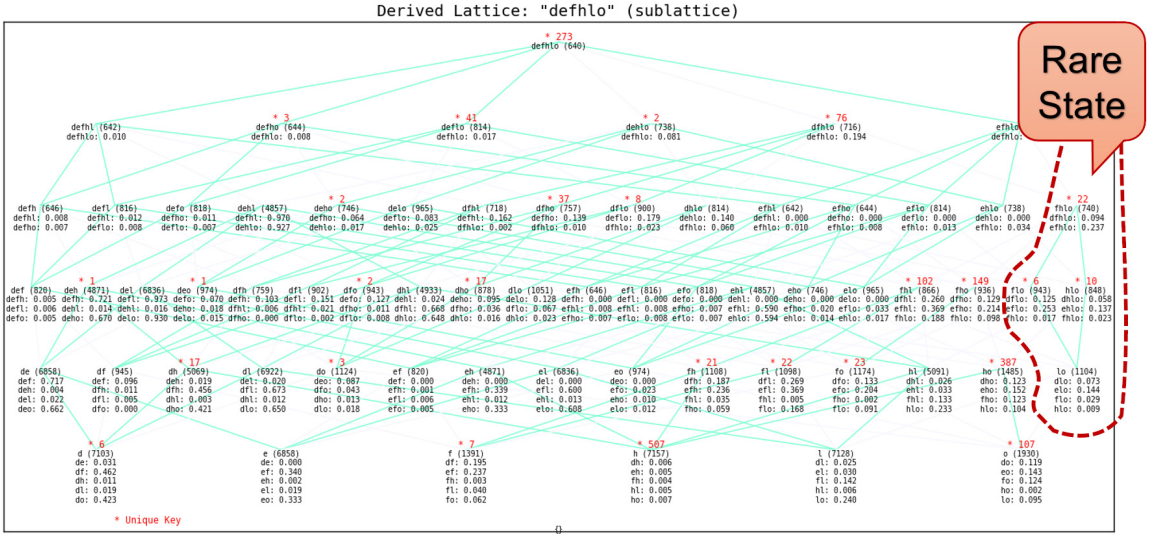


Figure 2.4: Output from the lattice clustering using the techniques described in Section 2.5.2. This depicts a sub-lattice of 6 features with a majority of the nodes *rolled-up* into one large cluster. On the right a smaller cluster is circled in red denoting a potentially ideal set of points to exploit for running new simulation tasks.

For our work, we define the exploration goal as producing rare events; the sampling objective is, thus, to select input parameters which will best meet this goal. To further complicate the selection process, potential noise is introduced in the system due to the stochastic nature of simulation execution resulting in erroneous or failed models. We, thus, factor in noise when selecting new input parameters. Given that clusters in the lattice correspond to patterns or states in the scientific model, we apply a scoring metric to each cluster, based on size and variance, to meet the data exploration objective of finding rare events. The metric, which defines the probability distribution function used to select new input parameters, is how the system selects starting coordinates from the smallest clusters (rarity) with the lowest variance (less noise) as ideal candidates to run a new set of simulations. Changing this scoring metric is how the system can apply to different user-defined sampling strategies. The sampling result is shown in Figure 2.4 where the output from a sub-lattice of BPTI is depicted with the smaller, detached lattice cluster.

2.6 Implementation

We implemented five adaptive sampling techniques: two supervised and three unsupervised. Our supervised approaches assume a classification technique is provided whereby an autonomous labeling algorithm processes streaming output and assigns a declarative label to each observed event. The first technique, a simple *uniform* sampler, chooses an equal number of observed events from which to restart. The second, a more targeted exploration technique, is a *biased* sampling approach. It applies an adjusted, umbrella probability distribution and randomly samples from a set of observed events using a weighted distribution. The three unsupervised techniques based on (1) the multivariate nearest neighbor (MVNN) algorithm, (2) the reweight operator and (3) the lattice-based data structure are described in more detail below.

For much of the analytics tasks, we leveraged the numpy, SciPy, and SciKit Learn packages for Python. We felt integrating these were a natural fit for the data science community. As much as

possible, we implemented machine learning and clustering algorithms as provided by these packages. In certain cases, we developed our own implementation of a KDTree and K-Dimensional Grid used for spatial clustering and organization of data into hypercubes.

Hypercubes

Our recursive KD-Tree contains three primary data elements: the leaf size, the backing data handler, and a root node. Inner nodes of the tree contain the middle or "split" value along with left and right children. Leaves contain a list of indices into the data handler and data value upper and lower for a specific dimension and we note that a leaf is a hypercube. We construct the KDTrees using read-optimized data arrays (`ndarray` which can be memmapped) and for adding data to the node, we leverage a secondary, write-optimized deque. The implementation is designed for offline storage using a compressed binary data encoding as a key to define a single hypercubes. This way, the index lists within all the hypercube can be stored separately and retrieve lazily (only when needed) allowing an efficient retrieval and recreation of the KDTree using only these compressed binary keys. The one hyperparameter, the leafsize, tunes performance: a larger leaf size is a smaller tree with less footprint, but will entail more I/O requests for insertions.

```
kd = KDTree(100, 15, np.array(feallist), 'median')
nn, hc = dict(), kd.getleaves()
for key, hcube in hc.items():
    hc_ftlndscp = [featlist[i] for i in hcube['elm']]
    hc[key]['feat_landscape'] = np.mean(hc_ftlndscp, axis=0)
    nn[key]=LA.norm(global_landscape - v['feat_landscape'])}
```

Listing 2.1: Hypercube Construction using KDTree

Listing 2.1 shows the implementation for the hypercube construction. Parameters passed into its construction in the first line include `leafsize`, `maxdepth` and the underlying data. The data is the the raw, spatio-temporal trajectory output divided into windows. The final parameter, in this

case `median`, dictates the space partitioning function used to perform a binary segregation of data along a single dimension. We have implemented `median`, `mean`, `midpoint`, and `max_gap`. The last one is akin to a one-dimensional support vector machine which identifies the two points that are farthest apart and splits the dimension at their midpoint. Once constructed, we collect the leaves as the hypercubes, `hc`, and then create the feature landscape as we defined in 2.1. We employ a separate feature listing data structure which is calculated in situ with the simulation output. The final line in this listing creates the nearest neighbor index, `nn`, which is used in the two step sampling approach:

```
neighbors = sorted(nn.items(), key=lambda x: x[1])[:N]
nn_keys, nn_wgts = zip(*neighbors)
nn_wgts /= np.sum(nn_wgts)
while numresources > 0:
    selected_hc = np.random.choice(nn_keys, p=nn_wgts)
    index = np.random.choice(hc[selected_hc]['elm'])
    selected_index_list.append(index)
    numresources -= 1
```

Listing 2.2: Two-Step sampling

The sampling algorithm, shown in Listing 2.2 starts by generating a list of the top-N neighbors for each hypercube. After sorting and normalizing, the actual sampler first performs a biased selection using the probability distribution of the weights; note that the weights are derived from the feature landscape. This highlights how we can bring together knowledge derived from recently executed tasks using dynamic analysis, or in situ as we will explore in the next chapter, to help drive the selection of future tasks. This points to the core component of adaptive control where we intelligently steer global progress by driving the sampling of individual task execution. The second sampling technique, in this case uniform, could be any alternative sampling approach as well.

Feature Lattice

The Lattice based implementation is also a customized python developed codebase. The data structure is a standard python dictionary object, although we employ an `OrderedDict` where necessary for efficiency in processing. Our basic construction algorithm is shown in Listing 2.3.

```
H = histograms(Ik, D, nbins, brange)
for node, parentlist in dlat.items():
    for parent in parentlist.keys():
        delta = H[node] - H[parent][[parent.find(i) for i in node]]
        flow = np.zeros(len(node))
        for k in range(len(delta)):
            flow_k = 0
            for i in range(nbins-1):
                flow_k     += delta[k][i]
                delta[k][i+1] += delta[k][i]
            flow[k] = flow_k
        dlat[node][parent] = np.sqrt(np.sum(flow**2))
```

Listing 2.3: Feature Lattice Construction

During the initial few iterations of the adaptive control mediation loop, we construct the complete lattice since the data size is small in scope. As the dataset grows, we transition to storing the lattice indices in the catalog and only making updates to this structure using an alternate algorithm. However, we show the full construction here for more clarity. Initial lattice structure, as well as any new data items used to update the lattice, is implemented via the max-miner algorithm. Individual lattice nodes are materialized using a selection based query via numpy:

```
Ik[key] = np.where(np.logical_and(CM[:,list(node)], True).all(1))[0]
```

In this case, the 0-1 Matrix, `CM` is the molecular dynamic contact matrix. Raw data input to the feature matrix is the filtered spatio-temporal positional coordinates for selected atoms within each

residue. This matrix contains a 1 for two residues considered in contact and a 0 otherwise. It is the input source for the basic lattice construction using the maxminer algorithm. We retain the euclidean distances for all pair in all windows, D, for use in constructing the derived, feature lattice, as described in Section 2.5.1 and depicted in Figure 2.2.

Reflecting on Listing 2.3, we note that this is a 3-layer nested loop. However, the inner most layer is not an order of magnitude equivalent to N, meaning this runs in $O(N^2)$ time. Due to the computational complexity of the EMD algorithm, we sought an alternative solution using the approximated method for EMD which we mentioned previously and show in the second part of this listing. This logic, which adds a weights to each edge in the lattice, finds the component-wise difference for corresponding distributions. The algorithm subsequently finds flow, F_n for each dimension, k_n, shown as `flow_k` by applying a fold operation: it accumulates a rolling sum of the change of flow for each adjacent histogram bin along each dimension. Total approximate EMD work to transform one distribution into another is solved by taking the square root of the sum of squares of each F_k.

Lattice sampling follows a similar 2-step approach we outlined with hypercubes, however, in this next code sample we show a deterministic selection for the inner item on the second selection:

```
cuboids  = clusterlattice(dlat, CM, D, Ik)
cluster_weights, clust_items = [], []
for c in cuboids:
    cluster_weights.append(score_funct_1(c))
    cluster_items = sorted([score_funct_2(i) for i in c])
pdf = cluster_weights / np.sum(cluster_weights)
candidates = []
for i in range(num_tasks):
    index = np.random.choice(len(pdf), p=pdf)
    candidates.append(cluster_items[index][count[index]])
    count[index] += 1
```

Listing 2.4: Feature Lattice Sampling

The lattice is rolled up to organize and clustered into cuboids, as defined above. For each cuboid, we apply two scoring functions. The first provides a single score for an entire cuboid; it could be size, volume, or even variance using an eigen decomposition. The second calculates a score for each item within the cuboid. For simplicity, we used the euclidean distance from the data point to cuboid center in the distance space, `D`. When all combined into the controller's logic, this demonstrates how the organization of the data into a lattice helps to drive the sampling to select the coordinates which are designed to steer the system to improve the data exploration objective.

2.7 Experiments

We conducted experiments utilizing the Maryland Advanced Research Computing Center,[63] an HPC shared computing cluster. Each node consists of two Intel Xeon E5-2680v3 processors with 24 cores, and 128 GB of RAM. Nodes are interconnected with FDR-14 Infiniband providing 56 gbps throughput. The cluster employs an underlying Lustre file system with up to 2 PB of storage. We wrote all overlay and macrothread implementations using Python v3.4. The catalog service is built on top of the Redis NoSQL (v3.0) data store and we leveraged the Slurm workload manager to schedule and queue jobs for execution. Scientific MD simulations are executed with the NAMD v2.10 simulation engine[64] and we leverage the MDTraj API[65] to perform analysis tasks for transforming and processing simulation output trajectories. We achieve ensemble parallelism by running many independent simulations on separate compute nodes. We note, however, that our system is capable of leveraging MPI-based parallel simulation execution across multiple compute nodes.

Molecular dynamics (MD) employs simulations to trace energy and physical properties of biochemical structures. Simulations iteratively calculate spatial locations for atoms at the femtosecond (10^{-15}) time scale. Specifically, we leverage the research on bovine pancreatic trypsin inhibitor (BPTI), a well-studied polypeptide protein with a published 1.03 millisecond labeled dataset.[44] Trajectory data consists of sequential atemporal conformation positions of 892 atoms organized into 58

residue proteins. We associate a Cartesian coordinate with each residue and define the feature set, K as the set of all pairs. Raw input is the euclidean distances for all pairs for each timestamp. We reduce our dimensionality to 25 for efficiency, group spatial-temporal output into collective windows, and calculate the median distance vector for each one. Each window, is thus, a single input tuple for which we apply a threshold function to identify observed events and transform it into the input matrix. Subsequent lattice construction is performed as described in Section 2.5. Experiment Highlights:

- Data collected from two naive techniques and three adaptive ones
- Adaptive Simulations produce rare events
- Adaptively controlled simulations provide a better convergence strategy
- Experimental results finish up 60x faster

2.7.1 Adaptive Sampling for Rare Events

We demonstrate the improvements gained through iterative execution of shorter, controlled simulations by comparing three adaptive sampling techniques utilizing the DINSAC framework against two naive, long running sets of simulations. For this set of experiments, we define the data exploration goal as executing simulations to produce physical structure conformations in rare event states. Historical data on the BPTI polypeptide has shown the protein to spend 82% its time in two of its common states, labeled 0 and 1.[44] Translated into monetary costs, this equates to 18 cents of value for every dollar spent on computing resources; we quantitatively define value as the amount of time a simulation produces rare events.

The first of the two baseline techniques employed consisted of a single, long running serial simulation taking place over one week of computation time. The second naive technique mirrored a traditional parameter sweep of input starting coordinates. To emulate this approach, we ran 10 simulations in a 36 hour period with each having different initial coordinates. For our adaptive

sampling techniques, we used a consistent model of scheduling simulations lasting 4 ns in duration with 25 jobs per control cycle.

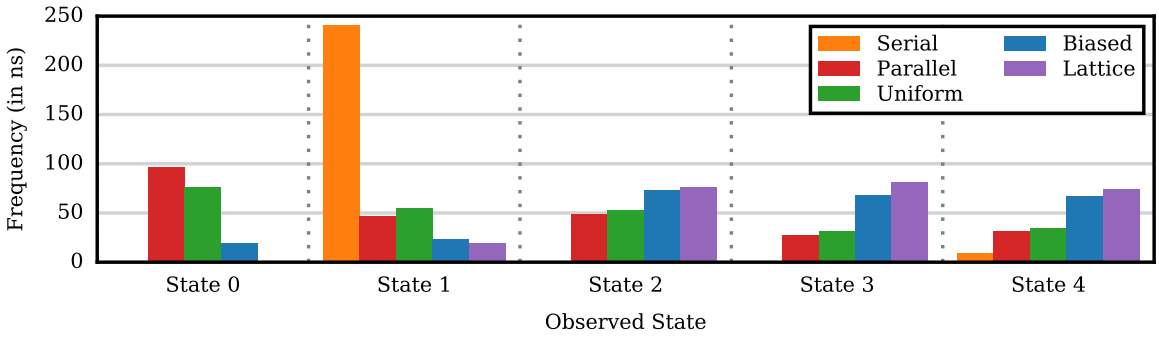


Figure 2.5: Frequency of observed stables states (in ns) of MD motion for all trajectories. Note that States 2,3 and 4 are rare. States 0 and 1 are common.

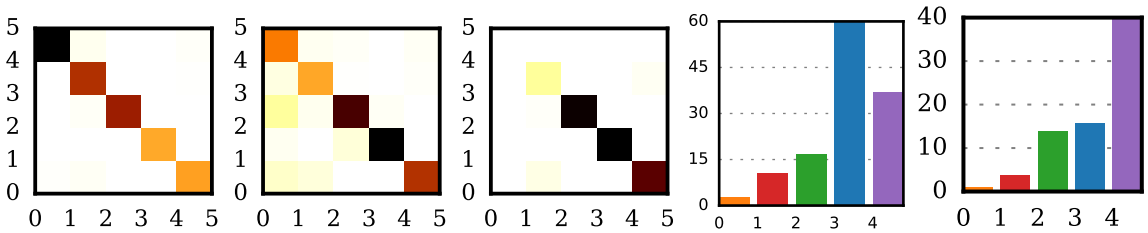


Figure 2.6: On Left: Heatmaps showing the sampling accuracy (a,b,c) for the three adaptive strategies (uniform, biased, and lattice). Each row in the heatmap shows the output distribution for sampling from the given start state. Right: Count of the number of observed transition events using historical data (d) and using no prior data, *de novo* (e) for each of the five experiments.

As shown in Figure 2.5, the adaptive sampling strategies for biased and lattice based clearly produce more rare events. The baseline naive serial approach, a typical method employed in scientific computing, highlights the data discovery complexity. Despite running for one week straight, this simulation only produced 4.5ns worth of rare event data out of 250ns of motion. This amounts to 1.8% of total time and 187.5 hours of unproductive computation resources expended.

Applying an offline, human-involved parameter sweep technique in the naive parallel experiment to select targeted starting parameters helps to improve overall performance. This demonstrates how starting in a rare state produces more rare observations. However, once a transition changes the MD state to a common one, the observation space remains unchanged producing only common

events (e.g. State 0). For both of these naive approaches, a lack of control reveals how simulations can get *stuck in a well* and rarely visit more *interesting* states.

In contrast, adaptive sampling overcomes this drawback. Uniform sampling, as expected, produces an evenly distributed set of events. The biased algorithm, which applies an umbrella sampling approach, improves on the overall number of rare events. Adjusting sampling distributions can further influence the biasing rate and allow the system to steer output to meet specific user query demands. The drawback is that these approaches are supervised and require a user-provided classification methodology. Applying a lattice based approach shows how the system can be tuned to explicitly produce rare events in a unsupervised manner. The histogram results (in purple) show that 92.5% of all observations are rare (states 2, 3, and 4), a drastic improvement over traditional approaches which have historically shown 18% efficiency.

We further highlight the sampling improvement Figures 2.6 (a-c). These heat-maps depict the sampling accuracy for the three adaptive techniques. In these graphs, the Y-Axis represents the actual starting starting states for all simulations. Each row shows the final distributions with darker boxes indicating more observations; white regions indicate no or very few observations for the labeled state. While uniform sampling targeted an equal number of starting parameters, it mostly started and finished in State 0. Biased sampling improved over uniform with a more dispersed distribution of outcomes. Lattice clustering is shown to predominately sample and output rare events (States 2, 3 and 4).

2.7.2 Transitions

While identifying and observing rare event states is one scientific goal, capturing the critical moment of transition among the states can be even more valuable in the data exploration process. We show that by simply coupling analysis to track and process these events as data is generated helps to steer future simulation execution. Given a labeling methodology to classify output in situ, Figure 2.6 (d) shows how adaptive sampling can steer the data exploration goal.

Among the three, biased sampling strategy was the best option using historical data. This makes sense, given an online, supervised labeling technique: local analysis on trajectories identify window(s) within the output as critical events; a subsequent global controller macrothread task aggregates all critical event data and makes a control decision designed to increase the occurrence of critical events in future simulations.

While a biased sampling strategy produced the most transitions among the given techniques - 22.5x more than the comparable parallel naive technique and 34x over serial - it relies on an externally defined labeling strategy. This classifier replicates a domain expert's knowledge on how to determine the low dimensional state based on observed high dimensional features. As an alternative when such a labeling technique is not viable, we show how an unsupervised lattice based approach using only the raw output can also meet the data exploration goal.

Sampling selects the clusters exhibiting the most collective activity based on a user-defined criteria. We note that domain expertise can greatly improve the quality of sampling; providing a declarative query engine on top of the lattice based structure would enable domain users to directly influence the scientific exploration process, however, this is outside the scope of our research.

2.7.3 De Novo Data Exploration

The two previously discussed sets of experiments relied on historical data to drive the sampling process. We now demonstrate how the control feedback loop coupled with in situ analysis can perform in a *de novo* setting, i.e. without any historical data. Since we are assuming no prior history, these experiments take, as input, pre-determined conformational positions and mirror the scientific exploration of a newly discovered virus (e.g. Zika virus). To set up this experiment, we use coordinates which reflect observed states in the BPTI protein. These observations represent measurements which may be deduced from physical observations of the polypeptide using data collected from molecular biology stereoscopy imaging. We ran the *de novo* experiment starting from each of the five states. As a baseline, we also ran five simulations for 100ns worth of MD motions.

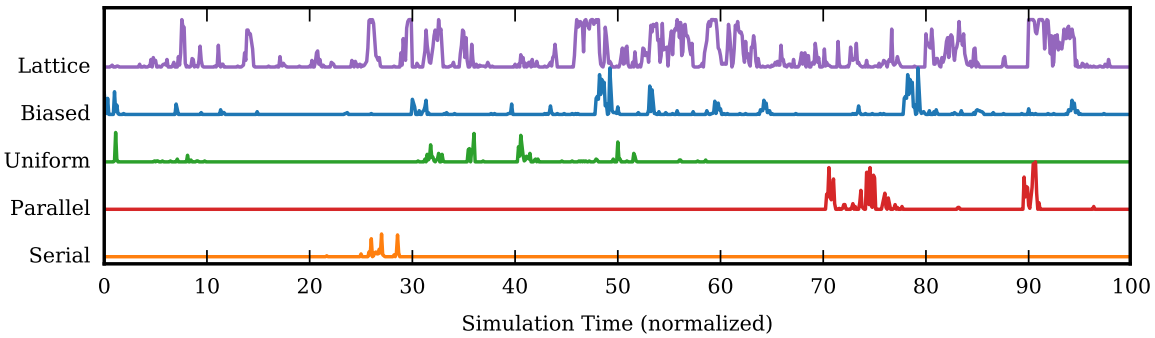


Figure 2.7: Time series of molecular dynamic activity from experiments start *de novo* (without prior data). Periods of low activity, known as a *well state*, are flat. High activity regions are indicative of rare state transitions.

For comparison, we include the output from the naive serial simulation.

Summary results for this experiment are found in Figure 2.6 (e). We stress how naive simulations are prone to remain in a stable state which explains the extremely low number of transitions. The uniform and bias sampling techniques each exhibited transitions, but subsequent re-sampling did not necessarily replay them. This attests to the stochastic nature of MD whereby starting a new simulation from a previously identified rare event does not guarantee subsequent rare events. Providing a lattice based approach demonstrates a more data centric solution to steer the system toward rare event discovery. This is seen in the increased number of transitions with the lattice technique showing a 21.5x improvement over a serial one and a 10x factor increase over the parallel method.

We have also graphically presented the trajectory data in a time series in Figure 2.7. Each of the lines represents activity in the MD protein based on the aggregate change in atom coordinates. Periods of flat lines, such as the longer stretches in the serial and parallel methods, show little to no change. This is a common output from traditional computer simulation executions over an extended time period of days or even weeks. Occasionally the simulation exhibits a transition, but shortly after it regresses into a stable position. Using a lattice based approach we show how DINSAC can improve sampling and steer the system toward its scientific goal.

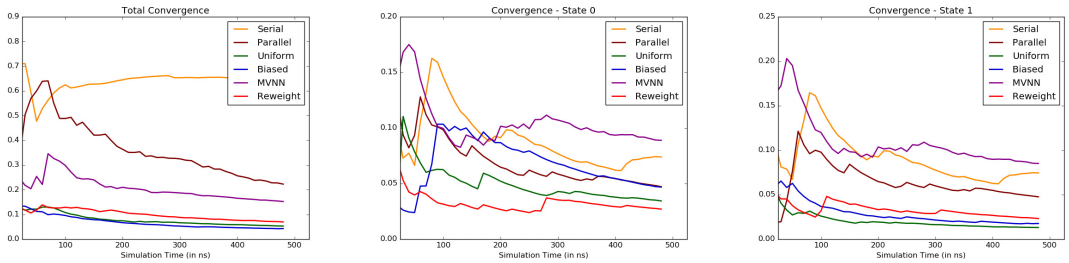
2.7.4 Non-Lattice Exploration Techniques

We provide two experiment results focusing on raw feature exploration techniques describes above. These experiments do not include the lattice based approach, which applies exploration analysis over a derived feature set. These results are included as comparison and background efforts as part of this thesis. They include the label-based approaches using root mean square deviation (RMSD) labelling, hypercubes, multi-variate nearest neighbor (MVNN), and the reweight operator.

Accelerating Convergence

The next scientific objective we demonstrate is how dynamically sampled control accelerates a convergence goal toward a more comprehensive understanding of the energy landscape. In this scenario, scientists desire a more comprehensive understanding of the energy landscape exhibited by a biomolecular protein. Using simulated BPTI and the statistical bootstrapping metric described previously, we compare performance for several sampling techniques. We evaluate how each strategy convergences on a stable set of values over time for each of the five states of BPTI using a multivariate description. Aggregate convergence on the distances and pairwise relative distances for the entire system in accordance with the multivariate landscape classification is shown in figure 2.8 (a). Subsets of features from this technique are extracted to show how the system converges on features pertaining to each of the five centroids in Figure 2.8 (b)-(f). To ensure consistent comparison we aligned experiment results to a common timeline based on total analyzed simulation nanoseconds. For insight into the impacts on wall clock time and resource utilization, see the resource management and cost section below.

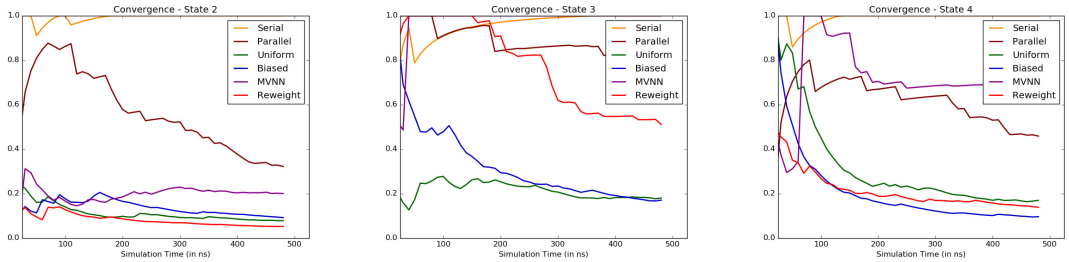
As shown in figure 2.8, our methodology provides an adaptive, controlling mechanism to steer the system toward a more converged status in accordance with a specified strategy. Each of the adaptive techniques performed better as compared to a naive approach. The graphs reveal how a long running, serial execution (in yellow) takes longer to exhibit a protein state space. The parallel technique (brown) provides decent results early since it employs a parameter sweep technique; how-



(a) Average Convergence for all features

(b) State 0 Features

(c) State 1 Features



(d) State 2 Features

(e) State 3 Features

(f) State 4 Features

Figure 2.8: Convergence results on feature descriptions. Top left shows the aggregate convergence for all ten features measures. Each of the other five show average convergence of features pertains to the respective state.

ever, each of these Uniform sampling achieves a slighter lower convergence in longer time while biased sampling shows how it can quickly achiever better results for the rarer, state 4 values. MVNN, which is a non-probabilistic version, has a steady, predictable curve. The reweight operator demonstrates its ability to exploit (an shown in the first 100ns) and explore as the simulation progresses. While this does not show the actual convergence data for the BPTI protein due to the limited feature dimensions, it reveals how an adaptive sampling technique can steer scientific experiment toward a convergence goal. We believe this demonstration can extend to a much richer feature landscape for more descriptive results on MD energy states.

Adaptive Data Exploration

In this section we show the advantages gained from the reweight operator. It is designed to highlight areas for either exploitation or exploration in an adaptive learning fashion enabling data

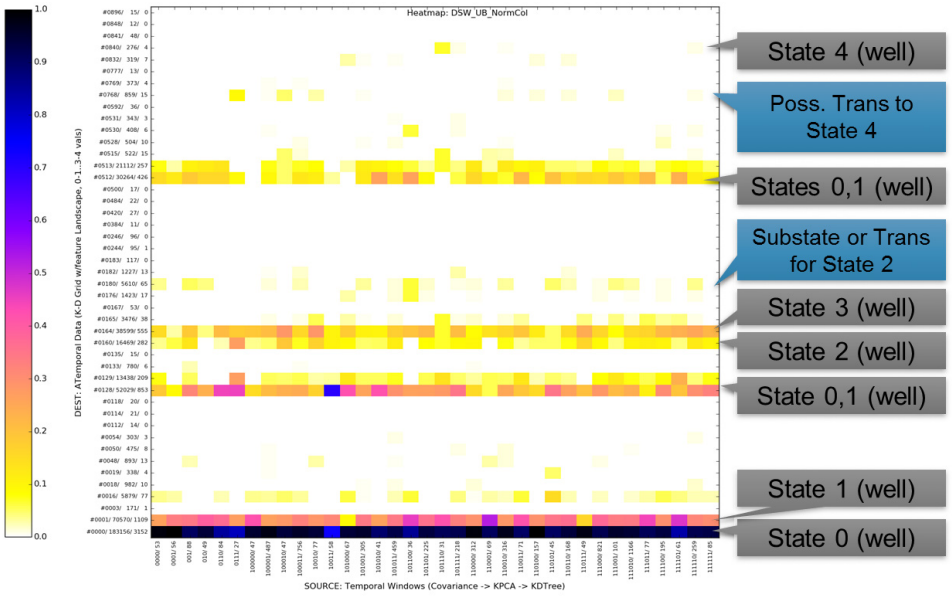


Figure 2.9: Heatmap from the reweight operation shown with projected number of points normalized by column (top). Dark bands reveal possible states, which could be either in a well or transitions. Select bands are labeled

exploration to occur dynamically as an experiment is progressing. Figure 2.9 shows the reweight operator in action. The source data, hypercubes representing clustered temporal data, are projected into hypercubes of clustered atemporal data. With no reliance on a prelabeled dataset, the aggregation reveals correlated trends: darker bands are more commonly exhibited feature spaces, while lighter colors are relatively unexplored characteristics of the MD protein. For each horizontal row of clustered points, we apply the aggregate feature landscape using the mean value of each point. We ascertain potential well states and transition points among the bands of points and have labeled some selected ones as indicated. The larger clusters represent potential well states with labels in grey with the unknown and relatively unexplored regions on blue labels.

Figure 2.10 shows the results with data normalized by row provided a description on how the projections from temporal space are distributed. In a semi-autonomous manner, user can ingest queries based on this projection to target potentially *interesting* regions (e.g. dark yellow or pink boxes). As a reference we provide the two feature landscape graphs in the lower right representing

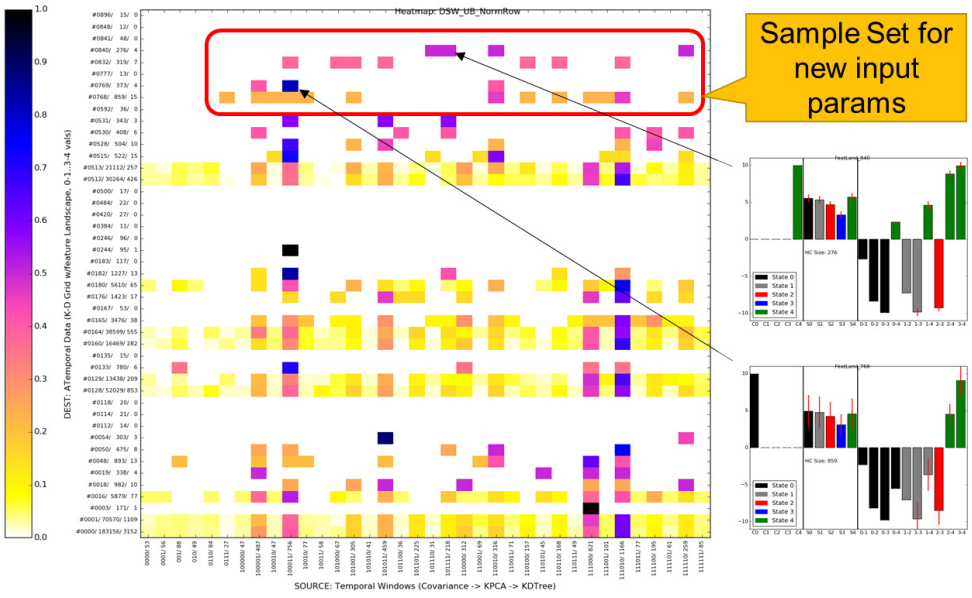


Figure 2.10: Heatmap from the reweight operation normalized by row (bottom). This highlights the distribution from temporal subspace. Two sample feature landscape graphs are provided as reference to show how the reweight can identify potential rare events to explore: Lower graph is indicative of state 0, upper graph is state 4, however variance indicates it is not a stable state and thus a potential transition.

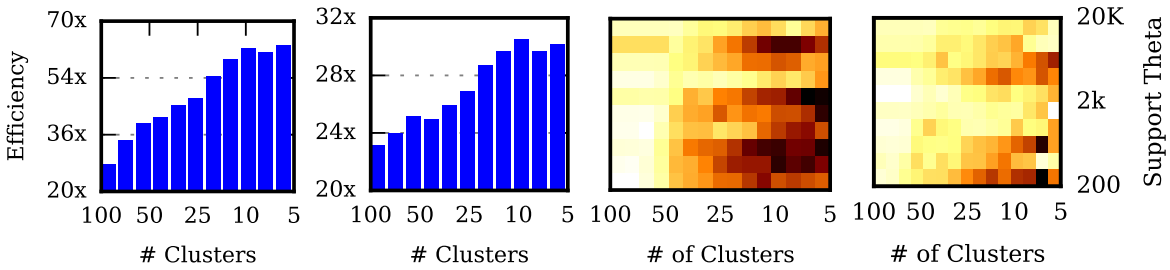
target exploration region. The lower graphs show a state 0 substate while the graph above show a state 4 substate – these could be a well or a transition; however the variance from these graphs indicates potential transition. Given a goal to understand rare events and transitions, we thus, aggregate the collective group of hypecubes (circled in red) and filter the remainder to produce the input distribution for a subsequent control decision. Sampling technique can be further chained to provide either an umbrella, uniform, or even a nearest neighbor sampler. We believe this is an area worth researching in more detail on how to best tune this operation (see section 2.9).

With no reliance on a prelabeled dataset, the aggregation reveals correlated trends: darker bands are more commonly exhibited feature spaces, while lighter colors are relatively unexplored characteristics of the MD protein. In a semi-autonomous manner, user can ingest queries based on this projection to target potentially *interesting* regions (e.g. dark yellow or pink boxes). Alternatively, we implemented an automated version which aggregates the projection horizontally and then

creates an umbrella sampling on the atemporal subspace for subsequent execution. We believe this is an area worth researching in more detail on how to best tune this operation.

2.7.5 Lattice Resourcing: Findings and Analysis

To optimize performance and enable execution we identified critical implementation decisions affecting both efficiency and accuracy. To further analyze this dichotomy, we conducted micro-benchmark experiments on hierarchical correlation clustering. The algorithm, which initially identified 150-2000 starting clusters, merged clusters until it reached a final set. For each, we calculated the aggregate probability distribution for two scientific exploration goals, finding rare events and triggering transitions. The results, shown in Figure 2.11 (a) and (b) highlight the improved probability of selecting starting parameters likely to meet these goals compared to non-adaptive techniques. For targeting rare events, we found sampling efficiency improved to as much as 67x for finding rare events and 31x for triggering transitions.



(a) Rare Events (b) Transitions (c) Rare Events (d) Transitions

Figure 2.11: Stochastic sampling efficiency of the lattice correlation clustering. The bar charts on the left show sampling efficiency by varying the number of merged clusters. Efficiency is calculated as compared to naive sampling. Two heatmap images on the right show the efficiency gained by varying both the number of clusters (X-Axis) and the frequent item support threshold, θ (Y-Axis), during lattice construction. Darker regions reflect combinations of variables with higher probabilities for sampling both rare event and transitions.

We also studied the lattice construction impact on the sampling efficiency. Given a historical dataset of 100,000 observations, we varied both the support threshold value, θ, and the number of final clusters. The colored heatmaps in Figure 2.11 (c) and (d), depict the probability of selecting

input parameters likely to produce significant scientific events. Darker regions reflect combinations of variables with higher probabilities. In general, lower support thresholds combined with fewer clusters are more likely to produce better outcomes. From this study, we determined that over-clustering can occur and that lower support threshold for lattice construction is more critical in identifying extremely rare events.

2.8 Related Works

This research is a follow on effort to previously implemented Molecular Dynamics Database at Johns Hopkins University.[66] Comparably, MDDB sought to adaptively steer simulation execution by providing a control abstraction layer to iteratively perform sampling over analyzed data. The end result was to optimally execute only the most interesting simulations. In contrast, MDDB employed a reinforcement learning technique aimed at maximizing a reward function for selecting techniques which could better describe the full Markovian state model of the underlying system. While we employ machine learning, it is not a fundamental component of this effort, but rather we focus on the computer systems, data management, and sampling framework.

One of the earlier adaptations to improve database frameworks, MonetDB provided an alternative viewpoint on data management systems through a redesign of the DBMS kernel. A pioneer in the field, MonetDB[67] recognized that contemporary applications, and mostly a growing number of analytical engines, required access to many rows of data, but only a few select columns. Fundamentally, it enabled column based data organization to drastically improve read optimization for query processing – tasks vital to support big data analytics. Open source and widely used today in many scientific research endeavors, MonetDB sparked a whole community of column based storage engines.[68,69]

A derived implementation is the OSCAR system or Overload-Sensitive Management of Archived Streams.[70] In it, Chandrasekaran and Franklin note that combining both streaming (live)

data with tradition DBMS (historical) data requires a novel approach to handle the continuous flow of streaming data to address the challenges of overloading a system. While traditional DMBS can offload, page, and delay processing, a streaming system cannot; such a technique would only delay and further exacerbate an already overloaded system. While other potential solutions such load shedding offer a potential solution this has the unfortunate side effect of losing data - an option generally not preferred in scientific or engineer applications. OSCAR employed a logical multi-resolution view of data at varying levels, such as 25%, 50%, and 75%, which is achievable in one, two, or more physical copies of data. While OSCAR demonstrated a capability to integrate DBMS concepts with streaming management systems, it did not address adaptive control. In addition while maintaining multiple views of data for analytical processing is intriguing, we did not pursue this technique, however, we feel this should be considered for certain applications to support OLAP and leave this as an potential area for future research.

One interesting trend is to forego the database tier altogether. The data management philosophy known as NoDB, professes this notion whereby user queries execute directly on raw files.[71] These systems incorporating an adaptive indexing concept to parse and tokenize data in line with the query processing. This technique completely eliminates the data loading bottleneck and drastically improves data-to-query time. This contrasts with the unstructured key-value storage techniques found in systems such as Cassandra, Dynamo, and BigTable[72–74]

Among the more widely employed databases for the science community, SciDB[75] uses an array data model. To meet the unique demands of scientific data and overcome inefficiencies of describing a scientific schema using a traditional RDBMS model, SciDB leverages multi-dimensional arrays to not only organize data for processing, but also to seamlessly integrate data into processing. This has allowed the system to enable math operations and lower level operators, such as those found in LAPACK or BLAS packages to process data directly on its internal format. Using this framework, SciDB is massively scalable while still capable of maintaining all the principles of ACID. It has interfaces for Python and R programming languages, thus helping to bridge the gap between the

database and analytics and is used by prominent organizations, to include NASA and the National Institute for Health.

An extension on query approximation is to focus on building viable sample sets of data from which queries can run to generate quick query responses within a given error range. BlinkDB[36] employs an adaptive framework centered around statistical stratified sampling of query column sets (QCS) to maintain an accessible sample set of data which can best meet future queries. Architecturally, BlinkDB contains an offline sampler which continuously updates the dataset and a runtime selection module which manages an error-latency profile on queries. The offline module focuses on identifying an optimal set of columns to include in a QCS on which to stratify and generate sample quantities. The optimization takes into account (1) sparsity of a column – whereby a more sparse column provides a more stratified sample (ergo, a non-sparse column is best sampled via uniform sampling, and hence, does need not to be includes in a QCS); (2) workload prediction uses historical queries to determine which columns are most likely to be included in future queries; and (3) storage costs which are used to maximize the coverage of the columns included.

JetStream[49] is a system designed for real-time analysis of distributed systems with changing data sets. It uses OLAP data cubes to couple analysis when data is generated along with adaptive filtering to meet bandwidth limitations. It employs a degradation technique to data analysis by using a *lossy* compression algorithm to trade accuracy for availability while monitor available bandwidth to keep the *lossy* component to a minimum. One of the design choices it implemented was to allow users to decide query results needed in real time and what level of accuracy is needed in order to maintain performance. Although JetStream focuses on bandwidth limitations across a wide-area network with multiple data centers, the underlying data structure concepts are applicable in virtually any venue to support an adaptive control system.

Combining active learning with human intensive tasks, such as image annotation, Mozafari, et al[76] demonstrated how machine learning can help to focus human involvement enabling an application to process more data at a faster rate. Implementing a nonparametric bootstrapping

technique, they employed active learning to steer a global application toward a convergence goal at an accelerated pace through an uncertainty classification algorithm. The computer iteratively selects input datasets for crowd sourced annotation which are more optimally suited for human actions. While our work is targets a separate field of research, we incorporate similar techniques as part of our data driven control framework.

2.9 Open Directions and Takeaways

Expansion of reweighted sampling. This research effort merely introduced the potential capabilities in combining multiple techniques to conduct sampling through a reweight operation. Follow on research will expand on this operator and, more opportunistically, apply it to unlabeled data sets. One novel approach is to apply view-centric method of correlating data across multiple dimensions. The system would identify unique correlations among data by aggregating all views in a collapsing join operation (e.g. perform all two-way joins on adjoining subspace hybercubes, followed by all three-way joins, the four way joins, etc...). Although computationally expensive, optimization would derive from incremental view maintenance techniques as developed through systems such as Borealis, DBToaster, LinView, or K3.[77–80]

Another potential implementation for this sampling technique is align it with multiview learning (MVL). MVL applies two different algorithms (or the same algorithm under different parameters) on the same data set and then combines the views in a subsequent step to either correlate or better approximate the high-dimensional representation.[81] By combining multiple views in a complimentary fashion, an algorithm can more comprehensively describe or classify the high-dimensional data set. For example, Shon, et al, applied this concept to implement robotic motion synthesis from images by projecting multiple observation spaces into a single unified lower dimensionality latent subspace.[82] By creating a single shared subspace, they showed how views projected into different dimensional spaces can be unified and correlated. Multiview learning techniques are well applied

and studied in many applications, to include natural language processing, uncertainty quantification, and image retrieval and synthesis. A future direction would be to further expand the reweighting operation along these lines.

Heterogeneous Environments. One area of future research is to expand adaptive control and data management integration across various computing platforms and locations. This effort coalesces resources from HPC, cloud environments, locally manages hosts, as well as individual work stations and mobile devices. A diversity of resources allows a dynamically adaptive system to capitalize on technological benefits in real time and allow the scientific application to elastically allocate resources to either reduce overall costs or to meet user time demands for accelerated convergence. This includes a trigger to recognize on demand reduced cloud pricing opportunity, such as Amazon's spot pricing and Google's preemptive instances, or to dispatch simulation tasks to external resources when an HPC management queue is backlogged.

This endeavor would also expand the requirements and demands of a serverless support system which we cover in the next chapter. The assumptions that failure impacts within the HPC computing environment were minimal and that data storage access was universally available do not apply when computing bridges across different medium. Implementing a more robust and resilient serverless overlay model would be paramount to heterogeneous deployments. Furthermore, such an approach may allow the system to leverage some persistent computing resources in locally controlled assets for asynchronous communication and consensus management (e.g. to run a minimal paxos state machine) while each homogeneous cluster operates is own distributed serverless protocol for internal data management requirements. From a systems perspective managing data consistency to maximize computing resources and steer an integrated application toward a unifies goal is a welcoming challenge and open research opportunity as the dividing lines erode between cloud, HPC, remove servers, and local devices.

Large scale, multi-user adaptive sampling. A long range goal of this system is to providing a basic framework to maintain a long running system supporting multiple user analytical requests.

For example, such a system is designed to simulate and explore a newly discovered micro-virus while, multiple scientists provide exploratory and tailored queries into the system. This example presents an opportunity to capitalize on the synergy of combining many user demands into one unifying system: one, global analytical system should be preferable over individual users executing separate analytical tasks. Although integrated, adaptive systems present additional overhead requirements, such as scheduling and fairness policies, we believe the benefits would outweigh the costs. A future direction would aim to develop a fair scheduling algorithm to ensure that each user's scientific experiment completes in less time than it would if she had conducted it in isolation. We exand on this in more detail in Chapter 5.

Scientific and engineering fields rely on highly powered computational resources to conduct modeling in support of research, development, and testing efforts. Simulation engines, residing in supercomputing infrastructures, are highly optimized to perform their computation tasks. However, they lack an integration with follow on data analytics. The offline cycle of simulation execution, data migration, and analysis is an area ripe for modernization. We have developed a framework which aims to recreate the traditional, scientific process of conducting experiments, collecting data, analyzing results, and isolating specific data parameters for subsequent experiments in order to seek a scientific objective. The significant contributing difference is the ability to perform this scientific process through an online, semi-autonomous, query based sampling approach to dynamically steer the experimentation process and improve science and engineering fields.

The methodologies described within this chapter are all designed to provide a data centric solution to facilitate with the exploration process. They enable the system to assume a more autonomous role in data exploration by performing some of the lower level pattern matching tasks, such as identifying correlations in the raw and derived data. This specifically addresses the human-in-the-loop problem which we highlighted in Chapter 1. While we do not attempt to completely automate data exploration, we aim shift the burden of some of the data tasks to the computer in

order to free the human to focus on higher level objectives. To make this happen in the Big Data Era, systems require new data-centric methodologies and techniques, which we have outlined in this chapter. However, they also need a systems framework to bring together the data organization and sampling requirements. We define this framework in a serverless fashion as a means to couple execution and analysis in situ in a computing environment.

Chapter 3

A Serverless Framework for HPC in situ Data Analytics

The Big Data revolution has brought about major steps forward in information processing and resource efficiency. With flexible, scalable distributed systems now capable of storing and processing terabyte and even petabytes of data, we are now able to perform tasks once deemed infeasible. Despite these advances, scientific experimentation involving high performance computing (HPC) lacks the ability to capitalize on this trend due to stringent architectures and a reliance on offline workflows to process generated data; analysis and exploratory requirements occur outside the supercomputing infrastructure, often on less powerful systems. We argue for the integration of online analytical processing (OLAP) tightly coupled with simulation execution and present two abstraction frameworks to enable this integration: one is a design pattern we call *macrothread* and the other is an overlay framework to provide data management in a serverless fashion. In our system, we leverage in-memory data services tightly coupled with simulation execution inside the HPC cluster to enable in situ data analytics supporting query based sampling and dynamic control. We evaluate using a variety of sampling techniques to demonstrate how adaptive control can steer a system in

support of data exploration and reduce costs associated with simulation execution.

3.1 Introduction

Computer simulations have become a critical tool in the scientific and engineering research and development community. They drive down experimentation costs and serve a critical risk mitigating role to study infectious diseases and other harmful pathogens. Executing in a high performance computing (HPC) cluster, simulations are tightly coupled with underlying hardware to maximize performance. Unfortunately the process has two major shortcomings. One is an inefficient use of resources: many simulated experiments in HPC clusters often last days or weeks, however, they only produce a small number of significant scientific phenomena. The second is a missed opportunity to capitalize on HPC capabilities due to an inefficient analytical pipeline: simulation output migrates to remote locations, typically to less capable hardware, for offline analysis.

We aim to overcome these two drawbacks by coupling in situ analysis with an execution control methodology to enhance the quality of simulation output and better support the scientific exploration objective. Our solution of coupling analysis with execution is enabled through an integrated controlling loop framework. Here, a scientific analysis workflow is continuously tuned with a decoupled, semi-autonomous control task to maximize domain-specific data exploration objectives. Integrating these solutions within a high-performance computing (HPC) environment would allow the community to capitalize on a high performance infrastructure, mitigate data migration challenges, and seize opportunities in data exploration. Thus, we strive to bridge HPC and analytics through an adaptive control framework aimed at improving data exploration in scientific simulations.

3.1.1 Applications

Although we focus this research efforts on scientific simulations in high performance computing, we note that the abstraction layers and framework presented are generalizable to any com-

puting infrastructure. We offer an alternative application for this research as related to austere environments which, like HPC execution, see tasks distributed in time and space. We also address how this framework applies to multi-model machine learning which we will explore in depth in chapter 4 as we project this abstraction from a set of isolated tasks to semi-isolated ones.

Forward Relief in Austere Environments. Austere environments are typically associated with high latency and limited bandwidth. We often correlate them with challenging physical settings, such as a desert or mountain range. Computationally, these environments are characterized with sparse collections of resources having a variety of non-homogenous capabilities. A prime example is a forward relief agency who brings with them some technical capabilities and can integrate into existing, limited computing infrastructure in a desert, arctic, jungle, or other extreme setting. An applicable exploratory tasks may necessitate the organization to perform some forward data analysis, such as discovering an underlying source behind a rising epidemic. Another example, such as in a naval setting, operational demands may require updated analysis of data collected from disconnected submarines or other vessels, each possibly containing different capabilities and databases. Without missing a beat Navy applications must continuously adapt with each ship needing to adjust course settings and speeds to ensure maximum coverage of Seaborn objectives. In both of these cases, gathered data cannot easily migrate to off-site locations for further processing. While they may have some high performing and capable tools to gather data, they may not provide the same computational resources for analytics. Any in situ analysis capability which they can employ would be beneficial.

Multi-Model Machine Learning. Machine learning applied to mutliple models is a technique which we employ to accomplish many different tasks. The straightforward approach of transfer learning uses information, such as model structure, hyperparameters or initialization values, from one training session to improve performance in a secondary model. We may have many orthogonal tasks to train concurrently on a source domain and assign one model to each task, such as classifying images using many sets of target labels. Models and data domains may not necessarily overlap, but

could be indirectly related. A global exploratory task, in this case, is to maximize accuracy across all models while minimizing aggregate training time. As a secondary effort, acquiring and processing data about the learning process, known as meta-learning data, can combine with an adaptive control framework to improve the overall objective. Thus, any effort to not only co-locate the meta-learning and analysis with the execution of training, but to also integrate into the underlying training is application use case for our research.

We highlight that we apply this framework to multiple tasks which are isolated from each other. With meta-learning or transfer learning, for example, input can derive from non-overlapping datasets or training on each model can occur as completely separate task independent of all others. This compares to the computer simulations executed in the HPC environment whereby each executed job is independent of one another. We will contrast this relationship with a semi-isolated approach as we extend the concept of adaptive control to dependant tasks in Chapter 4.

From this point forward, we focus this chapter on computer simulations in high performance computing clusters. We note that the mediation loop and abstraction layers described herein are extendable to these other computing environments, as we have outlined above. Specifically, these techniques can apply anywhere we seek to co-locate the data analysis tools in situ with the executing tasks.

3.1.2 Implementation Challenges

The Big Data operating environment is a stark contrast to legacy and seemingly monolithic high performance computing infrastructures. We highlight these difference in Table 3.1 with a side by side comparison. Contemporary data and analytic systems take a data centric approach to processing by asynchronously distributing computational tasks where needed. This includes the popular general processing systems of Spark, Kafka, Flink and Dryad.[1,2,83,84] This leads to applications and system architecture which tend to be less hardware dependent, providing an abstraction layer for user application interfaces in languages such as Python and R. They sit on top of a distributed data

management tier, such as HDFS or HadoopDB,[85, 86] or even distributed globally with systems such as Spanner[87] which abstract the storage and facilitate couple of the processing with data shards. Using consistency management tools, such as ZooKeeper,[88] they can maintain shared state and execute tasks asynchronously. In addition, they integrate with a resource management abstraction layer to efficiently match task to resource through tools, such as Mesos or Yarn.[89, 90] While supercomputing integrates resource management, the applications, however, depend more on hardware optimization with code bases strictly in C or C++ making interoperability with analytical tools more challenging. Simulation tools, in particular, are still reliant on these structured systems which integrate tightly with underlying hardware. Such coupling, however, is paramount to leverage low level optimizations and is the reason why languages, such as Fortran, are still prevalent in today's modern programming environment.

	HPC	**Big Data**
Mental Model	CPU Centric: "PetaFLOPs"	Data Centric: "PetaBytes"
Resourcing Strategy	Data → Processing	Processing → Data
Execution Allocation	Ephemeral	Persistent
Workflows	Discrete, Serial	Pipelined, DAG
Multi-Processing	Synchronous, Parallel (MPI)	Asynchronous, Distributed
Underlying Architecture	Physical Hardware	Virtualization/Containerization
Failure	Ignored and/or Rescheduled	Expected, Tolerated, Recovered
Languages	C, C++, Fortran	Python, R, Java

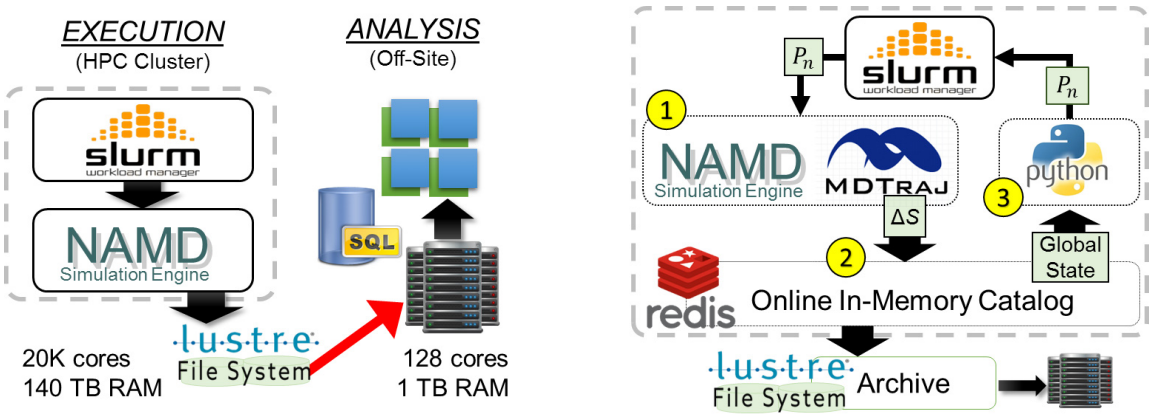
Table 3.1: Contrast between the high performance computing environment and Big Data.

Implementing a solution for in situ processing is met with the challenges of co-designing both analysis and execution workflows. Traditionally segregated in time and space, analytical tools are designed to operate after simulations complete and in off-site locations. in situ processing, where analysis and simulation tasks co-locate on the same compute node, and in-transit processing, where data is off-loaded to a secondary resource, provide a means to more efficiently meet scientific goals. They can potentially reduce the end-to-end time for the scientific exploration pipeline from weeks to hours.

This direct, integrated coupling, however, is no easy task. It requires either a direct interjection in the simulation or analytical engine codebase or the development of a middleware tier. Extensions to simulation code bases, while achievable, force programmers to specify analytical objectives during development and, thus, limit exploration to single, focused tasks. Some simulation engines, such as LAMMPS,[91] provide a capability to directly interject analytical operations during simulation execution, but they are very limited in scope and cannot take advantage of advanced analytical methods.

The inverse, interjecting simulation code directly into the analytical tools, is also a challenging hurdle. As we highlighted in Table 3.1, the contrasting architecture make this integration extremely difficult resulting in a complete redesign seem as a more appealing choice. The popular engine, CHARMM, for example, was originally released in 1983 and has not undergone any major large-scale restructuring of its original code base.[92] While it has remained flexible and enduring, capable of integrating new methodologies and ideas throughout the years, incorporating it and comparable applications, such as GROMACS or NAMD, into contemporary analytical processing remains difficult.

In contrast, middleware solutions provide flexibility, but they must coordinate asynchronous efforts among multiple computing nodes. Such solutions necessitate a data management capability to efficiently store and process simulation output. Integrating existing analytical tools with synchronized data management solutions, such as Postgres or SQLite, can serve this role, however, their implementation leads to data inconsistency and deadlock when developed on top of the underlying, distributed file layer.[93] Comprehensive, distributed solutions, such as Spark on HDFS, require additional overhead to directly manage the storage tier.[94] While some efforts in in situ analysis have made strides forward, they are relatively single focused on a single analysis task.[95] Any solution that aims to implement in situ processing and drastically improve simulation workflow efficiency must address these difficulties.



(a) Shortcomings

(b) DINSAC Architecture

Figure 3.1: Left: Current scientific exploration methodology segregates execution and analysis both logically and physically. Right: DINSAC in situ analysis and control architecture with an adaptive feedback loop

3.1.3 Simulation Ensemble Management

One alternative to proving a query sampler with a scientific application is to completely integrate the two concepts. This entails developing a simulation engine from the ground up with a sampling strategy in mind. Such a solution allows a user to manipulate the execution on the fly to model any known state. Unfortunately, this requires *a priori* knowledge of the system which is rarely achieved in scientific genres and especially in MD. Thus, frameworks must develop data models and processes to capture the essence of these system both at the macro and micro level.[21]

To overcome these shortcomings and challenges, we present the Data-driven in situ Analysis and Control simulation (DINSAC) ensemble manager. Our system combines local in situ analysis with in-transit processing as part of a global control methodology operating within the HPC environment. It integrates historical and streaming data to more optimally - and efficiently - meet a user-defined data exploration objective. We employ a data feature lattice inherently designed to organize raw output and correlate observations in a high dimension with scientific exploration objectives (e.g rare phenomena) in a lower dimension. This approach allows our system to link these objectives with a task execution strategy aimed at selecting ideal starting conditions or parameters.

Underpinning the system is an abstract framework designed to execute a scientific exploration objective unifying the results of many, shorter tasks distributed in both space and time. To synchronize efforts, an online, in-memory catalog service mediates data flow between the nodes enabling a cyclic *execution-analysis-control* feedback loop. The system collects and batches output from individual tasks and merges it with historical data to make control decisions. New tasks are scheduled and dispatched within the computing environment with updated input parameters designed to produce improved results. Over several iterations, the system adaptively steers simulation results to meet a user-defined exploration goal, such as the production of rare events.

Our system leverages the data feature lattice presented in the previous chapter to link an analytical objective with a global exploration strategy by controlling task scheduling. It dispatches many simulation jobs, each of which executes with locally coupled, in situ analysis operations, and aggregates output into a unified scientific exploration effort. Running in-transit within the HPC environment, a controller schedules subsequent tasks by selecting ideal input parameters derived from the lattice to meet a user-defined, global exploration objective. We overcome the challenges associated with in situ and in-transit processing by reducing dependency on the storage layer through an overlay framework built on top of an in-memory data tier. Evaluated on molecular dynamics, we demonstrate increased rare event observations by a factor of 22.5x while substantially reducing time and costs to meet scientific goals.

3.1.4 Contributions

1. We have developed an abstract programming model to support ensemble management. Both distributed and serverless, the design allows the sytem to implement the underlying details of scheduling and data processing. It coordinates simulations, in situ analysis, and periodically executed control jobs to operate as a unified, logical effort within the HPC. We demonstrate improve acceleration of scientific experiments using a controlled and adaptive iterative process.

2. Addressing the challenges associated with in situ analytics, our system employs an overlay

framework to provide an in-memory data management solution. Operating on top of existing capabilities, our framework serves as a middleware layer to efficiently mediate data flow between simulation and analysis tasks and control jobs. It significantly reduces dependency on the storage tier by capitalizing on in-memory stores and high-speed network interconnections among compute nodes.

3. We demonstrate efficient resource utilization using an elastic allocation policy. Adaptive capabilities can enable a system to converge on an goal at a either a more cost efficient rate or at a faster pace depending upon user resource priorities.

4. We provide findings, recommendations, and feedback on employing an in-memory, serverless data service for online analytical processing in HPC. Using an abstract framework for providing services to OLAP, we highlight architectural decision considerations and factors when deploying in an HPC cluster.

The remainder of this chapter is divided into the following sections: Section 3.2 covers the system control design considerations where outline the mediation control feedback loop. Section 3.3 and 3.4 provide the detailed design and architecture for the DINSAC system. We demonstrate how it impacts data exploration and adaptive control in Section 3.6.

3.2 System Control

Traditional data exploration processes require significant human interaction. Data scientists collect results and output from sensors, simulations, or other streaming systems into a single location to perform offline, decoupled analysis. Although much effort in research focuses on improving data analysis techniques through machine learning and analytical engines, little effort is diverted to understanding and improving the systems framework to build a more supportive infrastructure.

While there are many factors to consider when designing a system for adaptive data exploration, we focus on some of the principle architectural choices. We first layout the general framework

as a single system consisting of many, smaller tasks, each of which contributes to a global objective. In the case of data exploration, each task could be an analytic job or scientific simulation whose output contributes a small part to the overarching goal. We could also generalize this application to larger deployments of sensors dispatched across a vast geographic region trying to detect the onset of a natural disaster or to network and system administration services trying to prevent an intrusion. To build an adaptive control framework, we identify the following considerations as part of our design process mediation control, global data management, task and resource management, and level of user interaction.

3.2.1 Mediation Control Loop

Performing continuous and dynamic updates to drive global optimization requires a global control structure. We, thus, iteratively divide training into sessions whereby the system updates a shared global state and resource allocations are divided among model training requests from clients which applies a series of local optimization techniques in support of a global optimization strategy. Each iteration of the loop performs a framework which applies a modified OODA model[96] which was defined by Colonel John Boyd in 1976. We describe each of these:

1. **Observe**. The observation phase of the control loop entails receiving, updating, aggregating data. In includes data transformation and calculating any secondary or derived information. Data can come from a variety of sources and in this phase it must be consolidated and organized for processing. Observation should also account for meta-data statistics such as execution costs, resource costs, future estimations. In addition, source data, of both new and old, should be evaluated (or reevaluated) using a quality based metric, such as accuracy, correctness, staleness, or validity. From a computational and systems perspective, we classify the aggregate collection of observed data as the a global state. We also want to identify any information gaps where additional collection may be required.

2. **Orient**: The second phase, orientation, is what Boyd argues to be the most important. It is in this phase whereby we consolidate all of the information and interpret it into an assessment. As humans, we apply our own cultural, genetic, and experience biases to the data. This affects how we understand the situation and can have a drastic impact on the follow on decision. In a computational setting, we view this phase as the analysis processing phase and could replace the term, orient, with analyze (and re-coin this loop as OADA). In this analysis phase, we process information though a variety of statistical or machine learning techniques. We also want to include the local bias which may occur in a distributed processing setting: when nodes each have a local view of a shared global state, they may provide different analysis results which equates to a local bias in analysis processing.

3. **Decide**. The decision is the key action driving the loop process. Every decision should move the overall system toward the common goal. While the single exploratory goal may be the driving factor, any decision made must balance its effectiveness with efficiency. It must account for resource availability and other scheduling considerations. In some cases a less goal-progressing option in one step of the loop may provide significant resource efficiency which can enable larger goal-progressing leaps in subsequent control steps.

4. **Act**. The action is execution control overseeing tasks and performance. It includes dispatching and scheduling tasks to meet objectives. In this phase, we may employ event triggers to identify completion or when to initiate the next decision cycle. Computationally, this includes responses to external pressures, such as faults, latency, or data imbalance.

3.2.2 Resource Management

Scheduling which tasks to to execute in a given sequence or at a given time as well as where to execute them is a resource management task. Ultimately, a scheduler aims to maximize task execution (productivity) with a finite amount of computational resources. In some cases, an

objective function strive to execute minimal acceptable productivity by utilizing the minimal amount of resources; in other cases, the system may aim to always maximize 100% of available resources. Much of the background research on workload management can be found in Krompass, et al.[97] It is best to divide the decision into three different time-based question:

1. *Admission control:* Decision to determine if a job or query should be accepted to run. Based on an estimated cost, a system can accept or reject the request; alternate options may be to hold or warn the user as well.

2. *Scheduling:* Proactive, preventative mechanism aimed at precluding a system overload (or unstable or undesirable) state. At its core, scheduling is queue management to determine when to launch a specific job or query. Management can take the form of a priority queue, FIFO, sorted (e.g. largest first), or none (e.g. all queries are equal).

3. *Execution control:* Reactive response to a runtime event, such as an overloaded system, starved query, or a resource-hogging task. This can vary in types of events triggered and how to respond. When managing tasks, potential actions include: stop, kill, kill and requeue, suspend and resume, as well as none, warn, and reprioritize. We also note that execution control includes other real-time tasks, such as load balancing and fault tolerance.

With the HPC environment for which our simulation ensemble manager is targeted, we note that scheduling is generally provided through an established system, such as Slurm. Providing an abstraction layer on top of this is necessary and we include it as part of the system framework herein. We present a serverless design below to support data exploration within the HPC environment. For data management, we define an overlay framework providing a serverless, in-memory infrastructure locally within a computing cluster. In order to support local in situ processing and couple it with a global analysis task for simulation control, we also introduce an abstract programming model, which we call, *macrothread*. The macrothread is the abstraction primitive to execute the mediation control loop overseeing execution decisions and enabling adaptive control in support of the data exploration

objective. We describe these two components below.

3.3 In-Memory Analytics Overlay

To support data management requirements, we define an abstract protocol providing a serverless data support capability locally within the computing cluster. This solution is a vital and integral component to couple analysis and simulation execution. Although alternatives exist, such as persistent, remotely hosted servers in off-site or cloud based locations, the latency incurred in data throughput cannot sufficiently support analytical tasks. Within the HPC environment, attached networking though DRMA via Infiniband not only outperforms off-site networking connectivity, but also disk I/O transaction. In fact, data transfer rates across nodes have been shown to meet speeds as fast as a CPU's capability to read data for processing.[98] Thus, local networking should no longer be considered a bottleneck with system architectural design for HPC; scientific computing should, instead, incorporate on-line, in-memory data management services for experiment input or in-line analysis. We demonstrate this capability by implementing both a data catalog and a cache on top of a Redis key-value store as well as a garbage collection and data movement service, although the abstract protocol is not limited to these services.

3.3.1 Catalog Design

The catalog bridges macrothreads' data dependencies. Worker threads notify the catalog upon completion and register all newly generated output data. A controlling function, which could integrate into the catalog, subsequently decides when to schedule tasks to drive overall system convergence by adjusting parameter(s) affecting adaptability, acceleration, and elasticity. As a result, the catalog provides much of the functionality found in traditional DMBS as outlined by Hellerstein, Stonebraker, and Hamilton.[99] This includes the transaction storage manager and specifically, the log and buffer manager. Additionally, the collection/fusion component mimics the shared component

and utilities of a typical DBMS, to include catalog management, local file storage management (akin to cache management), and garbage collection. Finally, the collection/fusion layer is responsible for updating the current state of system. This specifically is calculating the objective function, but could include other related tasks such as tracking user query results. When viewing this system design, we consider all possible operating environments, not just stable and dedicates server resources. Thus, we consider providing such capability in a disjoint, time delayed, and distributed computing environment. Although we focus on HPC, these are applicable in other, austere settings such as geo-spatially disperse and military battlefield systems.

We explored several techniques to meet data analytical and control requirements. They range from a complete serverless implementation, such as using a file based data management protocol, to a dedicated, always-available transactional database. Consideration for these alternatives is provided below.

1. **File-based**. This technique is void of any separate catalog. Tasks within the adaptive framework update transaction logs on stable storage and rely on underlying operating system mechanisms for locking and transaction consistency. While this solution is simple to implement and eliminates resource overhead, it may only be feasible where data throughput is very minimal due to the significant cost of disk I/O. A serverless implementation, such as SQLite, relies on the underlying operating system to provide consistency management with distributed file systems, such as Lustre, as a primary data storage tier. An application must rely on the operating system to manage data consistency. In a shared computing environment, attaining such control is not feasible to provide optimization at a higher application level which can lead to data inconsistency and deadlock. SQLite, in particular, warns users to explicitly refrain from using its API for multiple connections with networked or distributed file storage. As increased number of users attempt to update the same distributed file, data operations can become improperly interleaved leading to unresolved transactional ordering.[93] This ultimately leaves the data system in an unstable state with a lack of common consensus resulting in concurrency

conflict and data inconsistencies .

2. **Persistent Service**. Following the classic master-client model, a persistent or logically persistent process executes and handles data requests. For stable environments, this is a trivial server with client worker tasks accessing the master on demand to get/receive data. For all other environments, to support limited execution (in time) within a computing node, the catalog server may not be truly persistent; thus, it must be capable of shutting down and storing all state to stable storage. The latency between server instances represents a service gap; executing tasks reliant on the catalog would need to either block and wait for a catalog service or access a file based transaction log. Implementing a master service ensures the most data consistent and data available solution since all data messaging must pass through the same online service. Likewise, this represents a single point of failure in the system, which can be mitigated via a fault tolerant or consensus based replication service, such as Zookeeper. Increased allocation of resources, especially when incorporating fault tolerance could make the service overhead extraneous; however, this is application and user specific prioritization.

3. **Data Store**. A data store solution implements the catalog as only a data structure. In this scenario, a running process loads system state into memory from shared storage and exists as long as necessary to meet the executing task data flow demands. This technique supports a serverless framework, since any task can be programmed to assumed the duties of the catalog. In an HPC setting, this allows processes running on a computing node within the operating cluster to endure as either a separate scheduled task or as a persistent worker thread. Upon initialization, a worker task would first detect the presence of a catalog service on the network and, if not found, it assumes the role as the catalog service. Absent any discovery service, the worker would subsequently need to advertise itself (e.g. through the shared file via a persistent file) as the active node to receive data input. While the worker incurs initial overhead to load the current state, it eliminates the transactional disk I/O overhead associated with a completely

file-based, serverless solution by retaining all data in memory. Limitations occur when the state reaches available memory for a given node forcing data to swap to local disk - an additional overhead cost although less than that of a serverless solution which reads and writes to shared storage. This also reduces the dependency on the underlying storage infrastructure to handle transactional file locks by restricting the file locking to the initial loading and final saving of state when the service is terminating.

One could aim to implement all three of these solutions in a flexible manner, allowing the system to maintain a low overhead, but capable of providing a persistent catalog service to meet demand. Normal operations should target a data store approach where a process runs within the computing environment to handle notification requests and schedule tasks. Continually, the process monitors the amount of overhead it consumes within the system and when its projected costs exceed a threshold, the system establishes itself as a persistent service. Ultimately, we decided on a technology capability through a serverless means with a data store approach for a catalog.

3.3.2 Overlay Model

We developed the overlay protocol (see Figure 3.2) to support in situ analysis data management and analytical requirements. This protocol is the means to provide a reliable service, such as Redis or even Spark, inside the HPC environment. Is it designed such that any executing task can assume the role as an overlay node, if needed. The primary advantage of a serverless approach is to allow ephemeral allocation of physical resources while maintaining a global state for a long running, logical ensemble effort. This protocol is the means to provide a reliable service inside the HPC environment. It leverages in-memory data storage and avoids complicated distributed file storage architecture. By serverless, we specifically imply that any executing task can assume the role as an overlay node, providing data services to the system. It is not implemented to replace other data management solutions, but rather to work on top of and in conjunction with these capabilities.

A single instance of the service executes as a running process on a computing node within

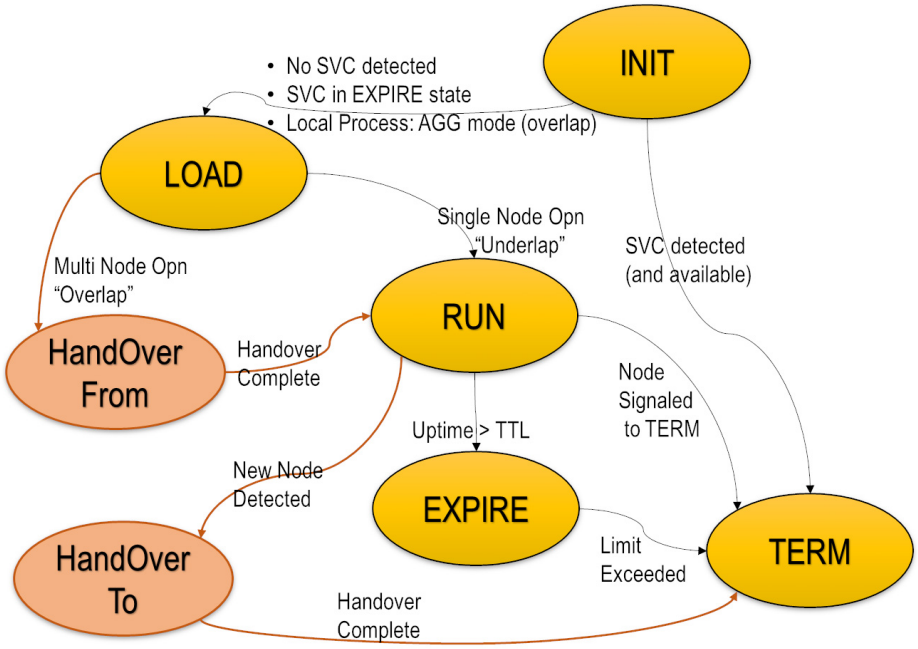


Figure 3.2: Overlay Protocol State Machine. Basic *underlap* version is shown in yellow with the added *overlap* version in orange

the operating environment. Upon initialization, it loads global system state into memory from shared storage and advertises itself to other processes asynchronously through the shared file system. We rely on the assumption of a dedicated and dependable file tier: distributed and network storage in HPC provides a resilient means to retain system state when the overlay service is not needed and to serve a reliable means of asynchronous communication. However, because one can never assume a complete lack of failure, the protocol is designed to recover from non-responsive nodes and problems with file-based communication. For any catastrophic failures or intrusion based attack we rely on externally provided resources. We also note that distributed, consensus based protocols, such as Paxos,[100] would be an ideal alternative to provide asynchronous communication; however, providing such a service within the computer cluster would, itself, present the same challenges as running any other service. We therefore would recommend to HPC administrators that such a service should be provided to customers as a means to better integrate analytics with supercomputing.

While the overlay service process incurs initial overhead to load the global state, it eliminates the transactional disk I/O overhead associated with a serverless, file-based solution. Individual task longevity is designed such that it can exist only as long as necessary to meet the system data flow demands. It can detect an idle timeout and shutdown to save on unnecessary CPU costs; however, this decision is tempered with a balance between reducing idle CPU resource allocation and retaining an in-memory store to preclude service unavailability due to data loading or other initialization demands. To provide users with a choice, we developed an alternative, overlapping protocol to maintain the service when demand requires it while also abiding by resource time limitation, such as a computing task getting preempted to shutdown or a job which has reached its scheduled or allowable time limit.

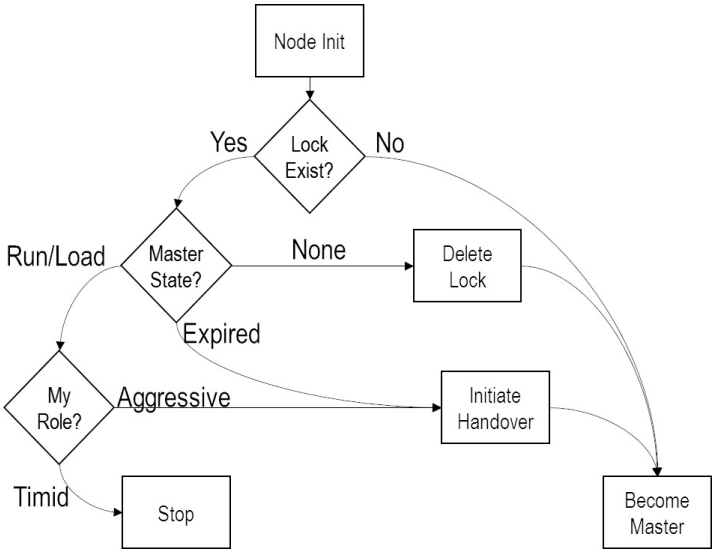


Figure 3.3: Overlap Protocol Flow Diagram

As shown in Figure 3.3, the process first detects the presence of an active service on the network and, if not found, assumes the role as the catalog service as a master by creating a lock file. If the service is detected, the process ensures the master is active and responsive, thus, providing a fault tolerant capability. Given that all tasks within an HPC have a time limit threshold and to ensure continuous overlapping coverage, the service can initiate a handover procedure to assume control of the service from the current master. This handover process allows the service to logically persist in an environment where physical persistence of tasks is not feasible. During the handover process, both nodes revert to a read only status in order to transfer state. To support fault tolerance,

we included both a timid and aggressive mode for initiating a handover: at most only one aggressive node is either active or scheduled in the queue. Any other tasks can fulfill a timid role whereby they only assume control when a master is either failed or expiring. This separation within the protocol prevents processes from continually replacing each other as the master.

To help steer cost decisions, we identified the critical factors as measuring the amount of continuous idle time for the overlay service process (Overlap cost) against the aggregate delay incurred on clients who may request data from an unavailable service (Underlap cost). The follow two equations solve for the worst case scenarios in each model whereby all active nodes try to request data concurrently.

Given the following:

```
T = Total Time
N = Number of clients in the system
V = Number of overlay nodes in the system
Cc = Unit cost per client
Cn = Unit cost per overlay node
L = Loading time
H = Handover time
D = Duration of a single executing task, such as a simulation.
S = Scheduled TTL for each overlay instance.
```

Cost decisions should meet the following:

$$\frac{TLC_cN}{D} < Underlap < \frac{TLC_cN^2}{D} \quad Overlap < \frac{THC_cN}{S} + C_nV$$

Since the scheduled handover time is fixed, the overlapping handover time is predictable; however, the underlap is bounded between a best case, where the service only need to load once per control decision and a worse case, where it loads and shutdown once for every client node, N. If the overlay service idle time for one node is less than this aggregate costs, then it should implement a logically persistent overlapping protocol. If the service is a catalog or cache, for example, L is directly related to the size of the dataset: as a simulation system progresses over time, the catalog grows in size and, thus, retaining an active service may provide more cost effective services. Depending

upon dynamic changes in the experiments (e.g. changes to the number of resources dispatched per control job or varying the length of an simulation), some of these parameters change over time. In addition, determining the relationship between handover time, H, and loading time, L, impacts this as well. We note this relationship in our findings (see Section 3.6.2). This is presented as a general consideration for costs when deciding on a systematic approach to supporting analytics in HPC. Other factors, such as price changes or queuing priorities should also be considered when deciding on persistence in data management support.

3.4 Macrothreads

While the overlay service protocol provides data management support, we introduce the workflow process framework, macrothread, to support computing execution and analysis tasks. Akin to a processing thread defined in operating systems, a macrothread represents a single executable task. It fits within a larger streaming framework concept whereby macrothreads initiate with upstream data as input and produce downstream data as output. We define two components which comprise the macrothread: a *worker* responsible for executing the macrothread's associated task, and a *manager* which monitors progress and controls execution by scheduling the workers. We first provide a background on decision control considerations to enable adaptive steering:

3.4.1 Decision Control Consideration

In any genre, adaptive steering of a system toward a common objective requires a decision methodology. This is the realization and implementation for the OODA mediation control loop described in Section 3.2.1. One possible solution is to isolate this to a few, critical decision points made by a single source, such as an Admiral deciding to steer a ship in a different direction. This provides simplicity and consistency, but it is also limits the quality of the decision to the amount of information and data collected at the sole source, noting that a lack of any knowledge is equivalent

to a random guess. In contrast, we could enable smaller more localized decisions as part of a larger group. Each individual decision is smaller in scope, requiring less information to make an intelligent choice; however, to be effective globally, each of the local decisions requires coordination and synchronization.

1. *Centralized.* A centralized decision methodology is simply a single point of authority. It follows the traditional master/worker pattern whereby data is processed and submitted in one part of the system (the workers) and collected to support a decision at another (the master). Computationally, this translates to the system maintaining a continuously running process to monitor both execution and analysis results in order to make control decisions in real time. The master controller dispatches job requirements to the workers, scheduling any tasks on resources, as necessary. From a data perspective, this reduces the consistency requirement to one single process, which maintains a current view of a global state of information. The controller would be responsible for scheduling and dispatching all tasks. While this is a simple solution to implement it may not be feasible in HPC or austere environments. In addition, the master can be a bottleneck or a single point of failure.

2. *Locally Scoped.* De-centralized in real time where each task executes a local control decision. Decision would include the scope of analysis/control and could be made either before the actual analysis (e.g. how much scope to analyze) or after (e.g. does a archive/garbage collect) or both. It would also include a control decision of what job(s) to launch next. Each task would have to deconflict with other running nodes. Nodes could be capable of canceling scheduled jobs and re-scheduling them based on the updates convergence toward the global objective. Each control decision should include *staleness* of data and estimate of a current global state given a local view of the shared state. Based on this estimate, the task must make a globally impacting decision (e.g. whether to launch a simulation or an analysis task in the HPC cluster) based on the current local view of the global state. While this eliminates any potential single

point of failure, it adds additional coordination and consistency control between executing tasks.

3. *Plan-based.* Instead of making a real time decision, the control decision is a plan of action which tries to predict the best set of tasks execute which will optimize resources and make the best possible progress toward global convergence. It can either run at set intervals or after some pre-determined time or event trigger. This option requires less coordination, but the effectiveness of a scheduling decision is dependant upon accuracy of estimation. This is an alternative approach

Ultimately, we opted for the serverless approach following a locally scoped decision methodology. This overcomes the limitations of HPC whereby persistent scheduling may not be available. It also provides a basis for alternative environments where stable computing platforms may not be reliable.

3.4.2 Architecture

Akin to a processing thread defined in operating systems, a macrothread represents a single executable task. It fits within a larger pipelined framework whereby macrothreads initiate with upstream data as input and produce downstream data as output. In applying the DINSAC system framework to scientific simulations, we define two logical macro-threads with a data management overlay service. Referencing Figure 3.1 (b), at the beginning of the pipeline, the (1) simulation and analysis tasks accept starting coordinates, P_n, and run an externally provided scientific simulation engine, NAMD. Leveraging local, shared memory, coupled in situ analysis tasks, built on top of MDTraj for our molecular dynamics application, process output using the same computing resource nodes. Downstream output, ΔS, which is reduced and transformed, merges into the (2) online overlay data catalog, which we have implemented on top of a Redis NoSQL database. A secondary data movement task serves as an online, in-memory cache to lazily copy output to the Lustre file

storage tier for archiving.

Downstream, the (3) controller macrothread collects batches of output from the catalog. It aggregates and merges locally generated simulation output with historical data and using the updated, global data model, the controller selects new parameters, P_n. The controller is also responsible for running a user-defined sampling strategy to drive exploration. Finally, it schedules new simulations via the workload manager and restarts the feedback loop. The complete design for the macrothread and overlay service working in tandum is depicted in Figure 3.4. As a core component in the serverless architectural design, any macrothread can assume a role as an overlay service provider. Upon initialization, every worker and manager task performs an active check on the in-memory data service. This ensures service availability in support of the global exploration effort and allows the data service to persist inside the HPC cluster.

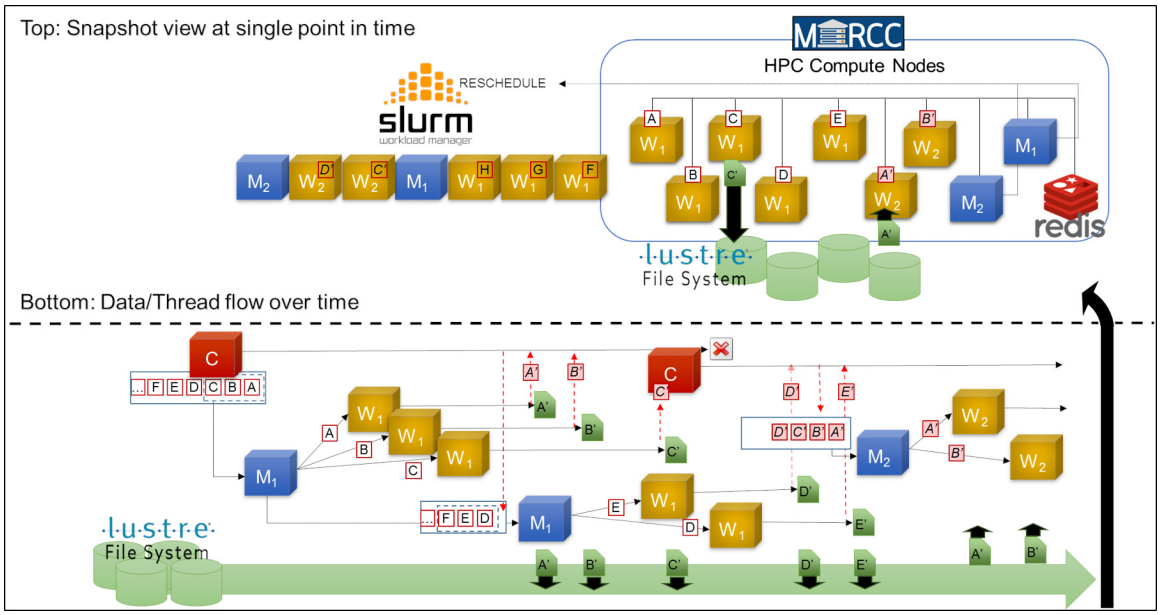


Figure 3.4: Design for macrothread and overlay framework as envisioned inside the HPC cluster. At top is a snapshot in time showing a view of the HPC with the framework operations. The bottom graph shows how manager thread (in blue) schedule worker threads (in gold) over time with data input / output transferring between online catalog overlay (in red) and the underlying Lustre file system. Note that the top graph is the snap shot in time at the right end of the bottom graph time horizon

Manager

```
MANAGER ():
  IF NOT catalog.available():
    catalog = EXEC OVERLAY_SERVICE()
  S = catalog.load(state)
  I_immed, I_defer = f_split(S)
  IF I_immed == 0:
    DELAY_SCHEDULE(MANAGER)
  ELSE:
    FOR i IN I_immed:
      SCHEDULE(WORKER, i)
    SCHEDULE(MANAGER)
```

Worker

```
WORKER (jobid):
  IF NOT catalog.available():
    catalog = EXEC OVERLAY_SERVICE()
  input = f_fetch(jobid)
  S = catalog.load(state)
  output = f_exec(input, S)
  registry = CATALOG(catalogFile)
  IF NOT catalog.available():
    catalog = EXEC OVERLAY_SERVICE()
  catalog.append(output)
```

Figure 3.5: Pseudo Code for the Macrothread's Manager and Worker abstract algorithms

We provide the pseudo-code for both the manager and worker algorithms in Figure 3.5 Managers are the component which implement adaptive and elastic resource allocation. The manager reads in a global state and determines the number of worker tasks to schedule on the HPC queue. Workers perform the underlying objective, such as executing scientific simulations: they retrieve data upon initialization, execute their assigned task, and update the catalog with new data. As this is a distributed system, as much as possible, the system performs append-only transactions into the data management service.

Workers perform the underlying scientific objective tasks, such as executing simulations or implementing machine learning algorithms. Using a user-defined fetch function, f_{fetch}, workers retrieve data on demand prior to execution. Such a capability integrated into the framework allows the system to implement additional optimizations, if necessary, such as a pre-fetch operation. In this scenario, when a manager schedules workers, it can initiate a request to retrieve large, off-site data files for local caching in-memory inside the HPC cluster in a distributed overlay caching service. Workers subsequently execute their tasks and update the catalog with new data. Finally, in both algorithms shown in Figure 3.5, the macrothreads perform an active check on catalog availability. Integral to the implementation, the catalog service provides the metadata and state for the system. This DBMS based concept is a critical piece which allows the system to perform in situ analysis

with execution. Furthermore, it provides the medium through which control structures can update input parameters to adaptively steer execution. As shown, either a manager or a worker can assume the role as an overlay service node in accordance with the protocol specified above. We show this algorithm implementation in Listing 3.1 below.

```
1  while retry < MAX_RETRY:
2      try:
3          self.catalog = RedisClient(settings.name)
4          if self.catalog.isconnected and self.catalog.ping():
5              break
6          self.start_local_catalog()
7      except (redis.RedisError, OverlayNotAvailable) as e:
8          self.start_local_catalog()
9  self.catalog.loadSchema()
10 self.load(self._mut, self._immut, self._append)
11 ...
12 self.save(self._mut)
13 if self.localcatalogserver:
14     self.localcatalogserver.join()
```

Listing 3.1: The macrothread execution code

This code provides a detailed look at how macrothreads comes "online" in the serverless framework. It first attempts to connect to an existing Redis catalog in the overlay service as a client (a `RedisClient`) in lines 1-5. If necessary, it starts one locally in the event one is not detected in line 6 or its unavailable and caught as an exception in lines 7-8. Once connected, the macrothread immediately loads the schema for further processing followed by the shared global state organized as mutable, immutable, or append only data structures. These latter two are particularly important in implementing lattice construction or in managing the hypercubes we described in Chapter 2. Once loaded, the macrothread executes its underlying task. Upon completion, the macrothread pushes the results downstream and updates any mutable global state which this marcothread may have affected. Finally, if the macrothread had stared a local catalog service, it joins that background

thread before terminating.

As we have shown, each macrothread contains a set of user defined functions and state. Combined, they determine the exact tasks that a worker thread should execute and the manner in which input data is divided for scheduling. This framework enables the underlying scientific or engineering system to adapt in response to analyzed data, elastically grow or shrink resources in order to capitalize on opportune costs, or accelerate more quickly toward an desired end state.

Adaptability

By introducing an iterative scheduling approach, our system allows for incremental changes to job priorities whereby data analysis results directly influence subsequent input parameters. By running several smaller simulations, the system adapts it long term course of execution to improve the scientific experimentation process. This can either ensure system convergence toward stability, integrate an adaptive learning approach of exploiting opportunities found during analytics, or explore undiscovered regions of interest to gain new knowledge. If necessary, the system can recall scheduled tasks and preempt running ones to replace them with input selected to have a better contribution toward overall system goals. By interweaving an iterative simulation-analysis-control loop, the system adapts as output is generated enabling it to complete faster than a naive serial-based approach.

Acceleration

We define acceleration as the increased rate at which the system moves towards a converging state. With a user defined objective function, we demonstrate acceleration toward a system goal by scheduling tasks which probabilistically have the highest proportion of contribution toward this goal. By introducing a cyclic control mechanism, the framework operates in a feedback loop which inherently drives the overall system towards a faster rate of convergence: output from upstream analysis threads feed a downstream controller process which, in turn, samples input data for sub-

sequent execution. While maximizing overall acceleration is a natural goal, it comes at a cost of resource utilization. We argue that factoring resource costs into the acceleration of the system is necessary in order to account for fluctuations or variance in resources. This is well apparent in cloud base cost models where pricing may vary at different utilization times: ideally the system should maximize acceleration when utilization costs are minimal and reduce acceleration when costs are more expensive. This, naturally ties closely with elasticity.

Elasticity

Adaptive steering of simulation convergence enables the system to optimize resource scheduling to meet user demands. Specifically, we explore how the system can save cost or time through elastic allocation of resources. A system which recognized that it is accelerating toward its goal can increase resource allocation dynamically to achieve its overall objective in a quicker time span. Likewise, the system can also consider fluctuations in resource costs, such as changes in cloud instance prices, and shrink or grow resource allocations in order to meet a user's budget.

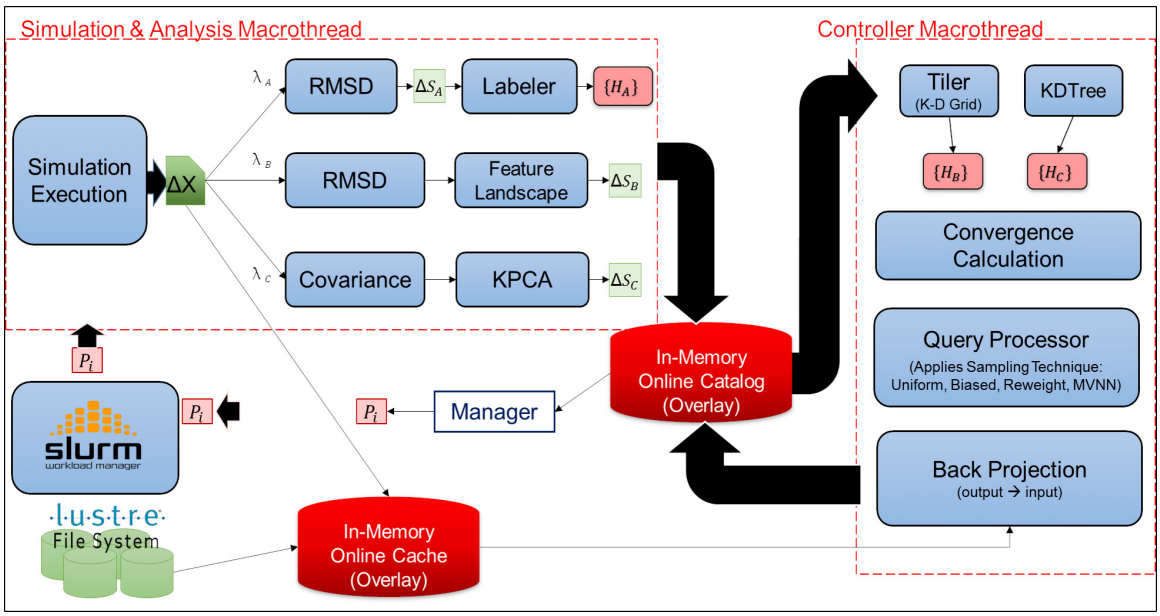


Figure 3.6: The Adaptive Control framework with macrothreads for both Simulation-Analysis and Control. In-Memory locally hosted Overlay services for a data catalog and cache are shown as well.

3.5 Implementation

Given the macrothread and overlay framework outlined above, we apply these system concepts to the MD computational simulation experimentation process. This includes representing the data, implementing data reduction and machine learning algorithms, and applying various sampling techniques. We implement two marcothreads: The first executes and perform in situ analysis on a single simulation. The second aggregates output from a collection of simulations, applies a specific sampling technique, and allocates resources for subsequent simulation execution.

The framework is a Python code base designed as a prototype to provide abstract programming interface. We designed the macrothread class an abstract data type with the intention of having data scientists implement this framework by defining their own split, f_{split}, execution, f_{exec}, and fetch, f_{fetch} functions. Additional implementations can also define a termination function to allow the adaptive simulation execute without any human supervision and cease once a convergence goal or other hard limitation is triggered, such as restricting total number of CPU hours consumed to affix a strict budget.

The abstract class for the macrothread framework is shown in Listing 3.2. This first portion of this class shows methods used to interact with the overlay catalog service. Data elements are registered with the catalog as key-value pairs. This mirrors the underlying unstructured, key-value storage executing in-memory inside the HPC cluster. We separate data into mutable, immutable, and append only, proving separate implementations for each. Specifically, upon completion of the `execute` function, the macrothread will append any new data acquired registered under with `addAppend` and it will merge changes with the catalog for keys registered with `addMut`. This optimizes overall data management by minimizing the complex operations of updating existing, mutable data. The `load` and `save` methods interact with the overlay catalog to retrieve and store data reliably. These functions check for the existence of the overlay service in the network and, if not found, initiate the overlay service protocol as described in Section 3.3. We highlight that the

macrothread manager and worker algorithms, shown in 3.5, will always call these methods before and after execution; however, users can also connect with the catalog during execution calls.

```
class macrothread(object):
    def __init__(self, filename, mt_name):

    #  Methods to register data elements with catalog
    def addMut(self, key, value=None):
    def addImmut(self, key, value=None):
    def addAppend(self, key, value=None):

    # Load/Save state to/from catalog
    def load(self, *keylist):
    def save(self, *keylist):

    # Registers a downstream macrothread
    def register_downnode(self, mthread):

    # Implementations should define these methods
    @abc.abstractmethod
    def term(self):

    @abc.abstractmethod
    def split(self):

    @abc.abstractmethod
    def execute(self, item):

    def configElasPolicy(self):
        """Default to re-run manager every minute"""
        self.delay = 60

    def fetch(self, item):
        """Default to identity function"""
        return item

    # Main execution entrypoint
    def run(self):
```

Listing 3.2: The Macrothread Abstract Class

The $f_{term}, f_{split}, f_{execute}$, functions corresponding to `term()`, `split()` and `execute(item)` are defined as abstract methods for user-defined implementation. Note that the execute function accepts a single item parameter which is designed as a lightweight element passed via the scheduler. When implemented with Slurm, we pass this parameter as a command line option with the execution call. It is designed as an index to retrieve a larger input, such as simulation parameters, using the f_{fetch} function, `fetch(item)`. The method, `configElasPolicy`, serves as the controlling mechanism to shrink or grow execution. As shown, we have a simple delay defaulting to 60 seconds, a functionality mirroring a common event loop; however, this method is designed for flexibility and can dynamically adjust resource allocation. A manager process invokes this call when it executes in order to set resource allocations (e.g. time in this case) for the subsequent manager's execution. Finally, the `run` method serves as the entrypoint into the program executing the design methodology described in Section 3.4. It detects whether the marcothread is executing as a manager or worker based on the presence of input parameters and executes the appropriate algorithm.

We provide a simplified example of a macrothread implementation in Listing 3.3. This shows how to extend the abstract macrothread class to execute a garbage collection process within the HPC to clean up intermediate files which may have been left from simulations or analysis tasks and are no longer needed. We present this as a toy example capable of running multiple, distributed collection tasks in order to highlight the asynchronous, serverless implementation.

This sample garbage collector executes on a time based policy reading in a list of jobs in the global state and removing any whose reference count is at zero. The three immutable data elements allow external control of the collector: while the macrothread, itself, does not change these values, another process or even a user, can directly update the state in the online catalog affecting elasticity and termination. These, along with a list of all jobs, represent the shared global state.

```
class garbageCollect(macrothread):
    def __init__(self, fname):
        macrothread.__init__(self, fname, 'gc')
        self.addImmut('haltFlag')
        self.addImmut('gcDelay')
        self.addImmut('nTasks')
        self.addMut('jobList')

    def term(self):
        return self.data['haltFlag']

    def split(self):
        return [self.data['nTasks']], None

    def configElasPolicy(self):
        self.delay = self.data['gcDelay']

    def execute(self, idx):
        jobConfigList = {}
        for i, job in enumerate(self.data['jobList']):
            if i % idx == 0:
                jobConfigList[job] = self.load(job)
        for job, config in jobConfigList.items():
            if config['gcRefCount'] == 0:
                if os.path.exists(config['workdir']):
                    shutil.rmtree(config['workdir'])
                self.data['jobList'].remove(job)
        return None

if __name__ == '__main__':
    mt = garbageCollect(__file__)
    mt.run()
```

Listing 3.3: MacroThread Use Case: Garbage Collection

When the marcothread is scheduled via a Slurm queue and executed, this state is loaded into local memory via the `run()` method. Subsequently, a manager tasks schedules worker jobs and when the workers are allocated resources, they implement the underlying garbage collection algorithm, defined in the `execute` method. Note how the local view of the global state is available

through the local dictionary, `self.data`. In this case, the worker loads all job configurations for which it is assigned (deterministically allocated via the modulo function). This occurs immediately upon execution when the catalog is available. Asynchronously, the workers perform the collection task to remove unused files and delete the job from the active job list.

For the underlying catalog and caching service, we chose to build our overlay service framework on top of a Redis key-value store. Although we considered a more traditional SQL data management tool, such as Postgres, we felt that the flexibility of a key-value store was more beneficial. In addition, we considered running Redis directly without an overlay framework within the HPC environment. This option would have necessitated running an additional discovery protocol within the HPC. As previously noted a persistent discovery service through an consensus protocol implementation, such as Zookeeper would provide tremendous benefit to bridge analytics and HPC. We leave this as a recommendation to HPC administrators for consideration. Finally, one additional benefit with Redis is the single-threaded nature of the managing client requests. This significantly reduced data consistency problems, although it did not completely eliminate them and for those we leveraged the overlay framework to implement a simple key locking mechanism on top of Redis.

All simulations executed in NAMD, a highly scalable molecular dynamics engine developed by the Theoretical and Computational Biophysics Group in the Beckman Institute for Advanced Science and Technology at the University of Illinois at Urbana-Champaign.[101] Although it provides an MPI-enabled parallel processing capability to scale up to hundreds of computing nodes, we chose to run all simulations on a single node, leveraging all available cores to ensure consistency across all simulations. Simulation configuration utilized the Amber 99 force fields in implicit water with a standard temperature of 310 kelvin. We found simulation rates of execution fluctuated greatly which we have identified as a potential opportunity to further employ an data driven adaptive control framework. A macrothread can constantly read its local progress and if a simulation is not completely as fast as other simulations on the network, it can a make a local decision to preempt and reschedule or abandon it current task in order to reduce operating costs.

Although we have a tolerance for failure in the overlay framework as previously mentioned, the macrothread component does not have a separate fault tolerance capability. By implementing append only data structures throughout much of the system, a failed worker thread is easily recovered by ensuing processes: if a control decision fails, the manager will simply reschedule another one. Granted the loss of a manager is a critical loss and one which we leave for follow on research.

With regard to data loss, we adopted the assumption that a loss of a small amount of output from simulations is tolerable given the vast amounts of data generated. This is acceptable within MD where a molecule can become unstable due to the stochastic nature of the system: a randomization process inherent in the modeling engines may place atoms in an untenable locations causing the system to lose equilibrium and fail. We stress a small amount, as when we first ran our experiments, we observed over 15% simulation failures. To overcome this, we integrated a retry capability. The simulation macrothread assesses the expected amount of output and checks to see if a certain percentage was produced in a decreasing amount for up to three chances (e.g. 75%, 50%, 25%) after which it merely processes whatever output it has.

3.6 Experiments

We conducted experiments utilizing the Maryland Advanced Research Computing Center consisting of 750 compute nodes. Each node's standard configuration consists of two Intel Xeon E5-2680v3 processors with 12 hyperthreaded cores, 128 GB of RAM, and 500 GB of local hard disk storage. Larger high memory nodes are utilized for database and in-memory caching which consists of four Intel Xeon E7-8857v2 processors and 1 TB of RAM. Nodes are interconnected with FDR-14 Infiniband networking providing 56 gbps throughput. The cluster employs an underlying Lustre file system with up to 2 PB of storage. To ensure consistency, each individual simulation was executed on one dual-processor computing node. The experiment highlights include:

- Dynamic control reduces overall costs by 50.4%

- Overhead costs account for 6.2% of total allocation
- Elasticity can save time or CPU costs

3.6.1 Efficiency in Lattice Based Exploration

Among the goals in coupling data analytics and control in situ with execution is ensuring the price paid in excessive processing and resource usage provides value added to scientific productivity. Over several experiments, we found the overhead for running an in-memory service and in situ analysis and control accounted for on average 6.8% of total operating costs. The excess costs translates to an average of 22.5x improvement in data exploration value.

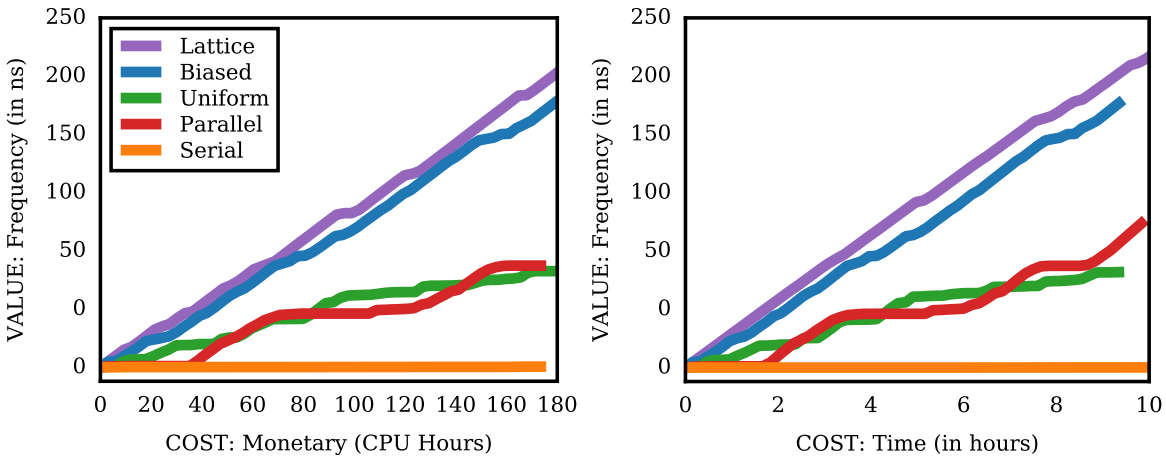
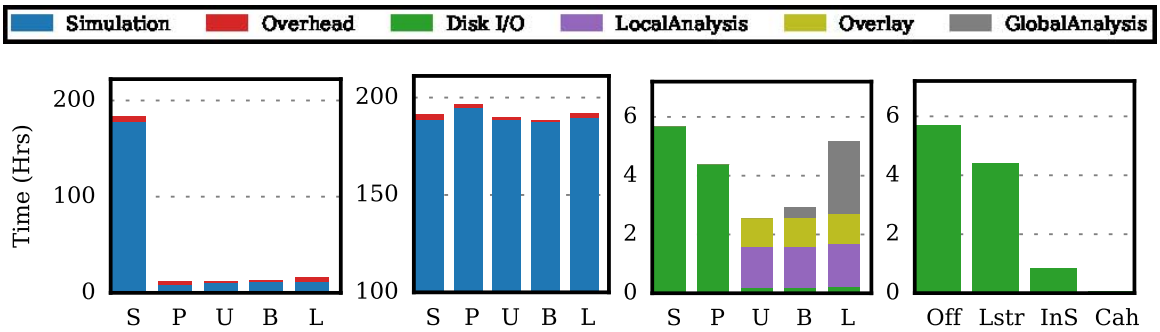


Figure 3.7: Adding costs by coupling analysis with execution improves overall performance. Both graphs highlight increased value over time (Y-Axis), as measured in CPU Hours (left) and Wall Clock Time (right)

Figure 3.7 highlights the value gained by executing many, shorter, controlled simulation jobs while adding control overhead to steer the process. A single, long running serial task can take weeks to complete and barely produce any data exploration value. The serial experiment, shown in brown, is barely visible. Running a parameter sweep technique by executing many jobs in parallel from varying starting points provides some advantage over time. Interestingly, this human-involved process is comparable to our automated uniform sampling technique. We stress offline human



(a) Wall Clock Time (b) CPU Hours (c) Overhead (d) I/O Cost

Figure 3.8: Comparison of the costs incurred the five experiments: [S]erial, [P]arallel, [U]niform, [B]iased, [L]attice. Captures wall clock time (a) and unit costs in CPU Hours (b). Overhead, shown in red in (a) and (b) are expanded and broken by operation in (c). Subfigure (d) shows advantages gained through in situ analysis. It includes time for off-site data movement [Off], local file storage [Lstr], in situ analytics [InS] (without an in-memory cache) and in situ with an in-memory cache [cah]. I/O costs include both network and disk overhead.

analysis is *not* included on this graph which adds to the overall effort of the scientific process.

Given historical data, a supervised, biased sampling approach sees a marked improvement in overall results over time. Likewise, the lattice based approach also produced valuable scientific output. The key difference among these two techniques is the underlying approaches: biased sampling is a supervised technique and relies on a domain expert labeling schema; the lattice based approach is unsupervised and purely data centric relying only on the raw output.

Compared to simulation execution, overall costs for overhead to perform in situ analysis and control is negligible. Measured in CPU hours expended, which equates to financial cost per unit, we found on average 6.8% of computing was dedicated to the services and operations other than executing simulations for our adaptive approaches. In contrast, while the naive strategies have no internal overhead, we did include – and highlight – the data transfer burden to move large, raw output trajectories to remove locations for offline analysis. We also stress that additional data analytical costs are not included in these figures other than disk I/O. Such tasks include ETL (extract, transform, load), data reduction, and other human-interactive processing to complete the data exploration objective.

The actual costs incurred are shown in Figure 3.8. Given that a majority of processing

is consumed by the underlying scientific MD simulation, we show that adding a moderate amount of overhead to improve the data exploration is a value-added expense, as measured in CPU Hours. This is even more apparent when comparing the wall-clock times for shorter, controlled simulations to the execution of a long running task. Granted, long running simulations may potentially explore a deeper portion of the MD state space over time, adaptively restarting simulations increases the chances of finding rare events and, thus, can achieve the same exploration goal at a much quicker pace.

We further break down the overhead components to show where costs are dedicated in DINSAC in Figure 3.8 (c). For adaptive techniques, disk and network I/O costs to move data out of the HPC is significantly reduced as only relevant data is retained for downstream processing. For MD output, only protein coordinates are stored in the on-line catalog. Offline archival storage is always an option as well, but this task can occur as a background process. Much of the overhead is attributed to analysis. This includes local analysis co-located with simulation tasks on compute nodes as well as the global analysis macrothread tasks which drive system-wide control decisions for the ensemble. As expected, local analysis is consistent among all three techniques. For the global analysis operation, the lattice based approach incurred approximately two additional hours of unit-cost compared to the two supervised approaches.

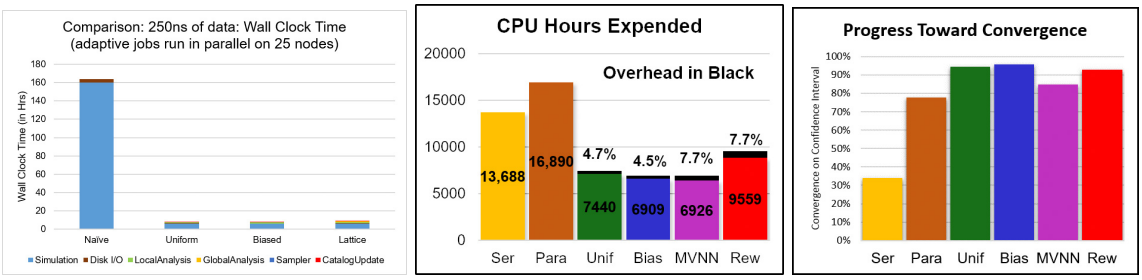
As described in Section 3.3, we incorporated flexibility to shut down the overlay and save on costs or persist data services and support time sensitive operations. The major penalty for either of these techniques is the amount of CPU time wasted by client processes (e.g. a controller worker task) to wait on the service. In contrast, worker delay occurs if the serverless catalog is not active in the network and a macrothread process must assume duties as an overlay node. We recommend that systems which employ serverless architectures should consider adding such a flexible capability in order to meet changing resource constraints and/or user demands.

As a final metric of comparison, we highlight the advantages gained through in situ execution by showing the time saved in I/O costs. Shown in Figure 3.8 (d), which compares I/O for

500GB of data, off-line analysis requires nearly 3.7 hours to move output. This transfer includes both disk I/O and the network requirement to copy data to a remote location. By coupling analytics inside the HPC, we eliminate the network traffic, reducing data transfer overhead to 41.4 minutes. We further demonstrate improved data access by running an in-memory cache service and reduce all data transfer overhead to 3.5 minutes.

3.6.2 Resource Utilization and Cost Benefit Analysis

In this section we examine the internal overhead costs to provide database management and decision control and show how the costs associated with integrating the framework are worth the value gained in overall scientific experimentation. Among the goals in coupling data analytics in situ with simulation execution is to ensure the overhead costs are minimal and provide value added to system productivity. Over several experiments, we found that total costs for running both an online catalog and caching service were less than 6.2% of total operating costs.



(a) Cost: Wall Clock Time (b) Cost: CPU Hours expended (c) Benefit: Convergence Progress

Figure 3.9: Resource costs of given experiments compared after generating 600ns of simulation data in left and center with the benefit gained shown on the right

Figure 3.9 (a) highlights the time saved in executing many, shorter and controlled simulation jobs. A single, long running serial task can take weeks to complete (in yellow); we found incorporating an adaptive sampling technique reduces this window an average of 14.8 hours. As we have mentioned, a single serial task could theoretically visit a much deeper state space along a single trajectory and meet one scientific objective to provide coverage over a protein's state space; however, we argue

that achieving this objective with a single trajectory in one Brownian walk is extremely unlikely; probabilistically the MD simulation will expend resources revisiting the same space, e.g. get stuck in a well. Adaptive sampling techniques can couple with generated data to preclude the system from exploring already visited (sub-)states and restart from input parameters most likely to reach unexplored regions.

We calculate total costs measured in CPU hours on a per core bases. Every worker and overlay task allocated 24 cores each, the maximum number available on a single node (hence costing 24 credits per hour); manager tasks only allocated one, but their processing overhead is minimal at less than one minute per job. Thus, manager overhead is not included in these results as they account for less than 0.1% of the total costs. As shown in red in Figure 3.9 (b), overhead costs associated with adaptive sampling are not a major setback. When employing an online catalog and a caching service, we found the total costs incurred an average of 6.2% of the total costs as measured in CPU hours. Taken into consideration averaged over across all experiments, dynamic allocation completes in 50.4% of the time as compared to naive simulation execution and, as shown in 3.9 (c), it provides more data exploration benefit with more converged results.

Overhead costs further sub-divide into two major categories. The first major cost incurred involves data management through overlay services, which includes the online data storage catalog and an online cache. As described in Section 3.3, we explored both an underlap and overlap method for managing serverless framework. The underlap technique allows a service to shutdown and save all in-memory storage to disk in order to reduce the costs of idling in the HPC. Overlapping the service ensures that a node is active somewhere in the cluster at any given time. The major penalty for either of these techniques is the amount of CPU time wasted by client processes (e.g. a controller worker task) to wait on the service to come online. Underlap delay occurs due to the load time when a process first detects an absence of the catalog (or cache) on the network and it must assume duties as the master overlay node; during loading times the service is unavailable. In contrast, overlap delays occur as as result of the handover between two nodes; during the handover period,

the protocol allows for read-only transactions. We found the handover period is approximately 4.8 times longer than the simple loading time. Recalling the proposed cost equations:

$$\frac{TLC_cN}{D} < Underlap < \frac{TLC_cN^2}{D} \ , \quad Overlap < \frac{THC_nN}{S} + C_nT$$

As a simple example of how to apply this equation to determine the best cost benefit for a service, we demonstrate using data garnered from experiments conducted. By fixing the number of jobs per control decision to 25 at a cost of 24 credits allocated per job, the simulation runtime to 30 minutes, the total time to 600 minutes (10 hours) and setting $H = 4.8L$ based on derived data, we can solve for the costs associated with overlap and underlap protocols:

$$\frac{(10hr)(15sec)(25)(24credits/hr)}{(30min)} < Underlap < \frac{(10hr)(15sec)(25)^2(24credits/hr)}{10hr}$$
$$Overlap = \frac{(10hr)(4.8)(15sec)(25)(24credits/nodehr)}{(4hr)} + (24credits/hr)(10hr)$$

Solving for these costs, implementing an underlap variant shows estimated costs for the service between 2400 and 60000 credits while the overlap costs are estimated to be no more than 1740 credits. Compared with the actual costs of the overlap protocol at 1087 credits, this shows how we can attribute a cost estimate to the protocol.

As mentioned in section 4, the caching service provided a 10 factor improvement for the reweight operator. This order of magnitude optimization is shown in figure 3.10 where we compare two adaptive sampling techniques. In these four experiments, we allocated 100 jobs per control decision and measured the time conducting the Back Projection and resampling. This task is a heavy data transfer intensive one requiring the sampler to retrieve a file from the underlying storage tier and perform a random access to select a specific coordinate position. The graph shows the

average time spent performing the back projection operation both with and without a cache for 50 control decisions clearly highlighting the benefits of integrating an in-memory serverless protocol with simulation and data analytics inside the HPC.

One additional advantage is the implicit advantage of capitalizing on local memory when coupling simulation and analysis. This demonstrates how the framework reduces the traditional data science task of ETL to a simple transform operation by eliminating the need to extract or load data. We leveraged native linux RamDisk capability to write simulation output files in DCD format directly to local memory on the computing node and subsequently reference the same memory location for the ensuing in situ process to classify, reduce, and cluster data. Although modest in comparison given that BPTI in implicit solvent is a fairly small system, this technique saved an additional read and write operation for each simulation executed. This accounts for 5 seconds of time saved on each traditional ETL operation which is modest compared to the 26.4 minute average to run a single simulation for 1 ns. In total, in situ analysis saved an average of 33 CPU hours for every macrosecond of simulation execution. For larger systems ran in explicit solvent, this savings could be very significant given that simulations can run in parallel, but data transfer cannot.

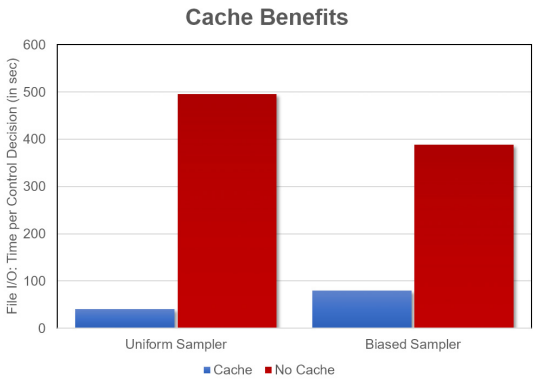


Figure 3.10: Time spent in data transfer to perform the back projection operation for two different sampling techniques. Red shows time incurred for disk I/O traffic without a cache; Blue depicts the time to back project the same amount of data using locally hosted, in-memory cache overlay service

Finally, we provide a graphical overview of the framework in action showing the lifecycle for a single experiment. The graphic shown in Figure 3.11 visualizes one iteration of an experiment employing the reweighting sampling technique. This experiment, which took approximately eight hours, shows every macrothread worker task which ran in the HPC cluster. This includes 700

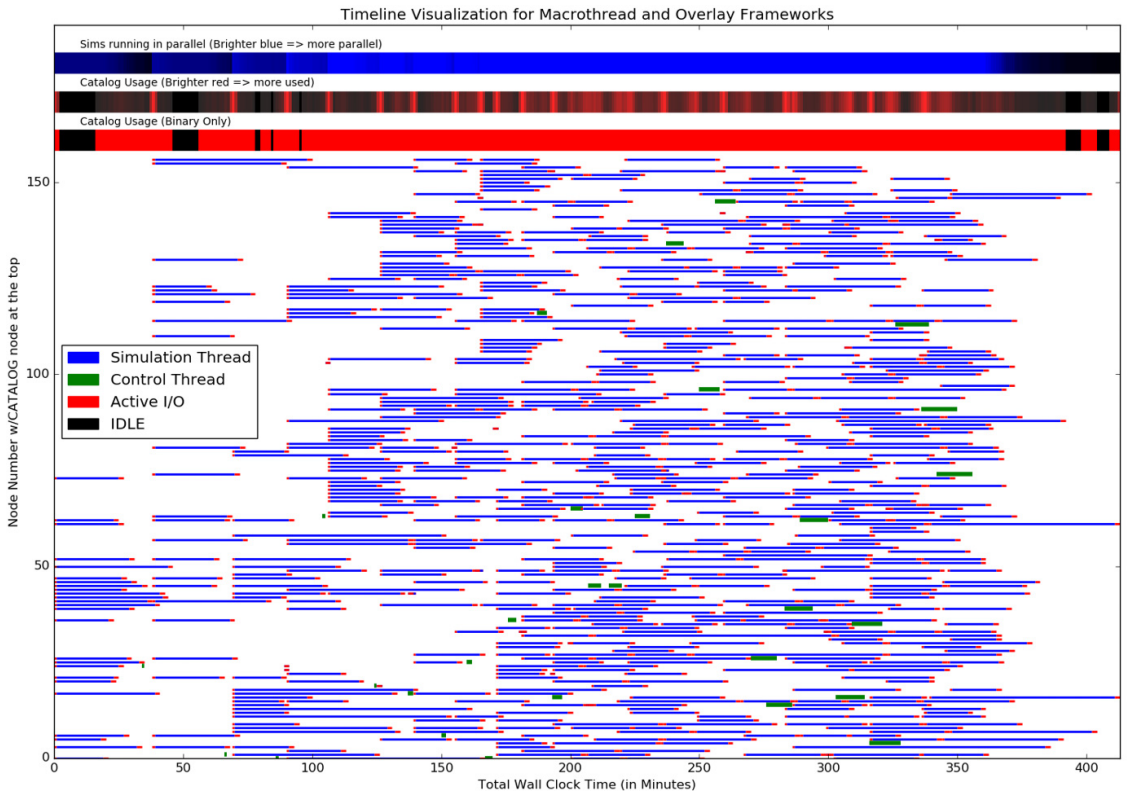


Figure 3.11: Graphical view of a simulations executed with overlay and macrothread frameworks

independent simulations (in blue) and 29 controller jobs (in green). Depicted in red are the points in time during which a worker task actively communicated with the catalog overlay. Note that the controllers have near continuous communication; however, we highlight them in green to distinguish them from other jobs. Conversely, simulation tasks only communicate twice during execution. Data I/O for these threads occurs when they first initiate to load the current state when send downstream output back to the catalog upon completion. The red bars aggregated all data below showing points in time when the service is more active and when it is idle. This visualization provides insight into the asynchronous execution in HPC. During the first hour of this execution, scheduling is fairly consistent with a series jobs completing every 30-40 minutes. The stochastic nature of the simulation results in varied execution lengths resulting in overlapping and asynchronous data flow. The controller framework, however, is able to smoothly manage this and dispatch simulations accordingly. Also

shown is the gradually increasing time to perform in situ data analysis for control decision; by the 300 minute mark, total data generated has exceeded 100GB. The controller tasks begin to overlap with multiple threads executing concurrently. Configured to growth elastically in a greedy manner, the system adapts to the availability of computing nodes in the cluster and accelerates the analysis process. A more conservative approach would defer control decision and save some computing costs while completing in a longer time span.

3.6.3 Elastic Resources Allocation

We compare cost benefits of different elasticity control plans by varying execution scope both at the single thread layer, individual simulation runtime, and at the macrothread control layer, the number of simulation jobs to schedule for each decision. We demonstrate how the system can be tuned to meet a user defined objective to either minimize running time or minimize costs. For this experiment, we implemented shorter, 250 picosecond simulations and varied the number of jobs to execute at various intervals from 5 to 200.

By elastically growing streaming resources (i.e. expanding the streaming faucet of data), we show how the system can quickly burst a number of jobs and improve convergence in short period of time. This is shown in Figure 3.12 where the green lines show high expenditure of resources in the first hour of execution. In contrast, shrinking the resource allocation can achieve a slightly better convergence goal at less cost, but at a much slower pace in real time as shown by the smaller step size in blue. Specifically, we found that we can achieve comparable scientific goals in as little as 20% of the execution time.

We also found that granular control of resource allocation can save as much as 21% of the total CPU hours expended. A more granular approach, while slower, ensures the system halts as soon as the convergence goal is achieved providing better parameter selection due to the increased number of control decisions. This last reason accounts for the variance in the final CPU costs as listed on figure 3.12: the 100-step size happened to complete prior to the 150ns termination trigger

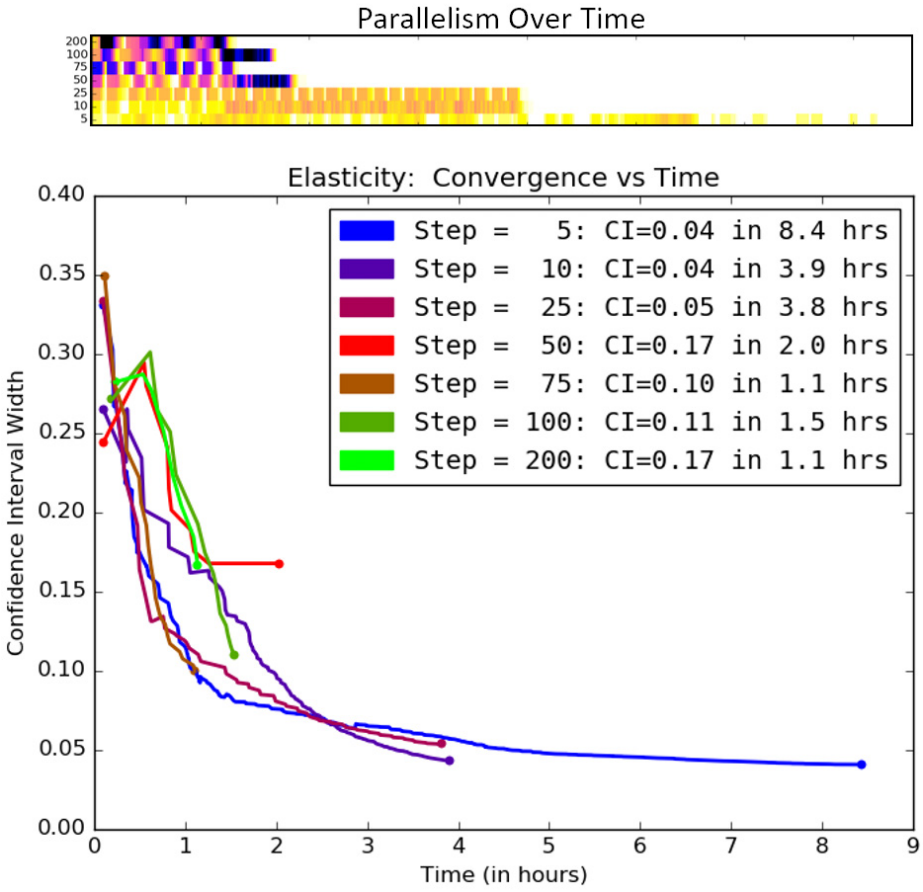


Figure 3.12: Comparison of 150ns of Simulation Experiments varied by the number of jobs launched on each decision cycle.

Elasticity Cost Comparison

Step Size	Final CI	Time (Hrs)	CPU Cost
5	0.040	8.43	1460
10	0.043	3.91	1351
25	0.054	3.81	1362
50	0.168	2.04	1462
75	0.099	1.10	1555
100	0.110	1.53	1711
200	0.167	1.38	1561

Figure 3.13: Cost shown in both real time (hrs) and in monetary costs (as CPU hours)

and hence, it ran extra jobs to reach termination.

One constant among the experiments run is the unit cost per computing hour. If we allow

unit cost to vary as is the case with spot instance in cloud infrastructure, a larger allocation of resources could achieve comparable results at comparable costs but at a much quicker pace. In addition, shown above the graph is the relative amount of parallelism observed for each experiment. Notably, the larger step sizes were never able to consistently achieve a full 100% level of parallelism. In fact, the 100 step size variant shows how it never gets to leverage available nodes until the 1.5 hour mark. Likewise, the 10-step implementation is able to achieve greater than 10 concurrent jobs due to overlapping controlling tasks: over time a controller puts new jobs on the queue while existing jobs are running due to backlog in HPC execution.

3.7 Related Work

Initially, we considered combining the efforts of SciDB with a scientific workflow manager. SciDB[75] leverages multi-dimensional arrays to not only organize data, but also to seamlessly integrate data into processing. Scientific Workflow Management Systems (SWfMS) which employs a business analysis approach for simulations. SWfMS provide a high-level, abstract mechanism to facilitate a scientist's ability to design, deploy, monitor, and analyze experimentation.[102,103] Modern systems can bring together operational process, such as running simulations or gathering external scientific observed data, with analytical engines and other computational intensive processes. They can operate in homogeneous environments or heterogeneous ones whereby the workflows can stitch resources from HPC, grid, cloud, and locally operated clusters. Often presented as a DAG, SWfMS connect pipelined processes to pass data from one tier to the next and provide a provenance database to track a datum lifecycle throughout the workflow. However, while SWfMS bridge analysis tasks and simulation engines, they perform so in an offline fashion[43] and target more high-level abstract views which falls short of integrating at the data management layer or providing in situ online analysis along side experimentation execution. Copernicus,[104] further integrates workflow processes by distributing execution across many platforms. A comparable ensemble manager which employs a

client-server design pattern, it lacks the data-driven control to steer a unifying exploration objective.

Our work parallels concepts introduced with Cumulon[105] which aims to capitalize on dynamic cost prices targeting Amazon EC2 cloud services. Huang, et al demonstrate how an active cost modeling process can optimize fluctuating prices (due to to supply/demand) in order to maximize resource utilization. They a implemented a structured fault tolerant system which can recover from unexpected and sudden lost resources (e.g. spot instances are instantly removed and work is lost) by combining reliable primary nodes with cheaper, transient nodes to accomplish a large-scale data analysis task. By employing a stochastic model to estimate costs at varying prices and numbers of transient nodes, the model predicts when spot instances would be reclaimed and work lost. Measuring the estimated synchronization time is critical since it could be extraneous computational work – at some point of diminishing returns, added transients nodes would be cost ineffective. Thus, the optimization strives to identify this ideal point.

Dataspaces[106] is a data management framework providing a virtual shared space abstraction layer to executing tasks enabling services, such as analysis, monitoring and visualization, to access live streaming data. It is an asynchronous, in-memory capability designed to integrate simulation processes with external, in situ processing jobs. Building on top of this framework, the authors in[107] implement both in situ analysis and in-transit analysis. They present resource optimization improvements through topological analysis, descriptive statistics, and visualization for molecular dynamic simulations. Presenting in situ analysis scheduling as a numerical optimization problem,[108] demonstrates a significant reduction in overall end-to-end time for both simulation and analysis. This effort focuses on *when* to implement in situ analysis by affecting job scheduling as compared to our work which focused on *how* to integrate analysis with execution. Clustering algorithms to support large scale distributed simulations levering in-memory data is presented in.[109] Similar to this effort which focuses specifically on clustering for feature-based objects tracking, our system couples analytical tasks directly in-memory and alongside executing simulation processes. Our efforts bring together many of these advances. Distinct from these, we focused on not only how to interject in

situ processing within the HPC, but also on leveraging the output to couple analysis and control to select simulation input in order to improve the larger, scientific exploration objective as a unified ensemble managing many, shorter simulation processes.

Other HPC data management efforts have also focused on integrating DBMS techniques inside supercomputing. One solution implements Spark on HDFS in HPC.[94] Other approaches, include IndexFS and ShardFS,,[110,111] which are middleware technologies designed to support file indexing for distributed file systems, such as Lustre. Unlike these solutions, our framework does not focus on the storage layer. Pertaining to data cubes, orthogonal efforts have emerged since the work was first introduced in.[55] Research has focused on materialization, construction, and optimizations with algorithms for bottom-up crawling (BUC)[112] and efficient processing.[113] We see our lattice-based approach as building on top of these concepts.

3.8 Open Directions and Takeaways

We foresee future work in adaptive control and in situ data management by integrating diverse computing platforms and locations into a heterogeneous and multi-user ensemble management. This effort coalesces resources from HPC, cloud environments, and locally managed hosts, enabling the system to capitalize on technological benefits of each environment. Adaptive, real time response presents opportunities to exploit cloud pricing fluctuations, such as Amazon's spot pricing, or to react to external pressures, such as dispatching tasks to alternate resources when a scheduling queue is backlogged. We discuss a multi-user ensemble manager in detail in Chapter 5.

Scientific and engineering fields rely on highly powered computational resources to conduct modeling in support of research and development efforts. Simulation engines, residing in supercomputing infrastructures, are highly optimized for performance, but lack an integration with follow on data analytics. Our research efforts on this framework aim to recreate the traditional, scientific process of conducting experimentation, observation, and analysis through an adaptively controlled

simulation ensemble manager. In doing so, we have shown how an online, data-driven in situ sampling approach to dynamically steer simulation parameters can improve science exploration efforts in HPC. This solution addresses the human-in-the-loop problem specified in Chapter 1. Specifically, we have demonstrated how a semi-autonomous system can assume some of the lower level tasks of tuning parameters, adjusting filters, and monitoring queries. This, in turn, frees the human to focus on higher level tasks in the data exploration process.

Throughout this chapter, we viewed simulations as independent, isolated tasks. Each task executed as a black box function taking as input, starting coordinates, and producing, as output a trajectory. We could easily switch the simulation ensemble management with a different field, more specifically, targeting machine learning. Instead of executing simulation jobs at each decision point, we schedule training tasks over models. The adaptive feedback loop analyzes the global progress over all models, such as calculating the global sum of the collective training convergence, and scheduling model training accordingly. However, if the models were not strictly isolated tasks, but rather overlapping, or semi-isolated, then we could integrate more intelligent controlling methods as part of a global optimization strategy.

We thus re-frame the original data exploration problem presented in Chapter 1 in a different context, addressing how both the data and the systems layers can integrate into the analysis layer. Combining the adaptive controlling abstract concepts presented in this chapter, we extend the exploratory scope from a single focused effort to a multi-model problem. In Chapter 2, we presented a lattice based approach for identifying rare events in a molecular dynamic simulation; however, we treated every simulation as a single input data stream to build one feature lattice in order to explore BPTI. What if, instead, we had constructed five separate lattices, one for each of the five states of BPTI?

This question alludes to two common challenges in machine learning: *over-generalization* and *over-fitting*. In the case of BPTI that we have just described, any machine learning training

task may over-generalize for any one of the individual states. For example, a model to train the task of predicting the likelihood of a transition to state 3 may not produce an accurate result for any one of the input states. The converse, training a model on a lattice constructed from only a subset of nodes of one state, may also produce an inaccurate model since it would result in over-fitting on the training data. We, thus, seek to strike a balance between the two challenges and concurrently train multiple models to provide a more effective solution.

This objective gives rise to resource demand constraints and other data challenges. If constructing and maintaining one lattice is compute intensive, then multiple lattices may be infeasible. In addition, given that rare events produce very minimal data, can we even gather enough input to generate a viable data structure for these states? We address these challenges through a semi-isolated, multi-model analyical approach in machine learning. Leveraging distributed systems abstraction primitives, we implement an adaptive control technique to steer data analysis toward a more effective and efficient goal. As a further extension, we can also integrate a multi-model approach with the data features lattice. This would enable adaptive control of a data exploration system integrated into machine learning; a future direction we mention in the final chapter.

Chapter 4

Distributed Machine Learning for Semi-Isolated Models

We present a system design pattern for machine learning applied to multiple models containing overlapping data domains. We quantify these as semi-isolated models whereby sub-components of each must remain isolated from all others, but models can share selected parameters and operators. As input, we organize data in a hierarchical manner where an individual domain represents a localized subset of a global population. We want to train each model separately, as if it were isolated, but leverage the overlapping components to provide a more accurate and better predictive model. We describe this design pattern and present it as a solution to support localized, or personalized, data analytic applications. In addition, we show how asynchronous primitives can dynamically affect distributed training, providing a better balance between resource efficiency and training effectiveness. We argue this is a necessary abstraction layer for machine learning systems by demonstrating how this capability can improve on existing solutions. As a running example we apply our methodology to personalized healthcare; we identify a single patient's data among all patients bedded in a given hospital as a sub-domain with an associated semi-isolated model designed

to predict the likelihood of a given hazard in that patient, such as detecting the onset of septic shock. Implemented on top of TensorFlow using both synchronous and asynchronous execution, we evaluate this framework on both the MIMIC II research dataset and on real world data. With a custom developed optimization operator, we show a 6-8X speed up for training time with prediction accuracy as high as 96% and demonstrate an adaptive optimization approach which improves on current state-of-art machine learning training.

4.1 Introduction

In Chapters 2 and 3, we demonstrated an approach to improve the exploration workflow though data organization and systems integration solutions. Data organization, in particular, allows the system to provide more autonomous oversight into lower level pattern matching: it self-organizes a feature lattice to cluster nodes which exhibit similar multi-variate data distributions. This, in turn, enables the human-in-the-loop to focus on more important tasks in the larger scoped problem, such as understanding a virus or developing a new drug.

Since data exploration is normally part of a larger effort, we now delve into the issues of applying the lattice within this larger context. Specifically, we seek to apply an associated analytic task to our data which drives an application domain requirement. For example, in studying a virus, a user has determined that a transition to particular state is critical in applying a specific neutralizing antibody. The user needs to know the likelihood of this transition occurring in order to assess the validity that the antibody can be further developed to fight a disease. The scientist could develop a single model using all observed conformation trajectory windows as training input data, however, such a technique fails to provide specific representation for any single virus sub-state. Even if the scientist trained the same machine learning task over each lattice cluster, any one cluster could still represent multiple data subsets leaving the resultant model to the same over-generalized consequence. In contrast, some clusters, which aptly represent a well defined sub-state, can over-fit

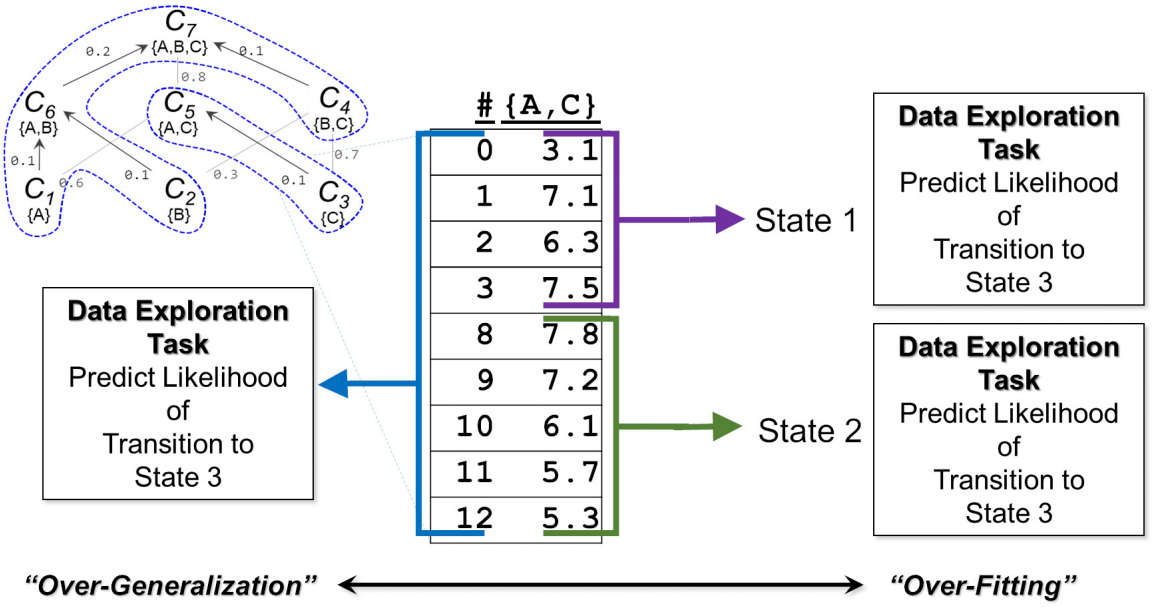


Figure 4.1: Over-generalization vs Over-fitting. Projecting the data points from a lattice node as defined in Chapter 2, we can create a single model to train a given task (shown on the left). However, this model suffers from *over-generalization* and does not accurately provide predictions for any one of the input data subsets. On the right, we highlight two subsets of data and show how we can train two models for each. Unfortunately, each of these models suffers from *over-fitting*.

the data inhibiting the model from providing an accurate prediction on the objective task. We, thus, identify a requirement to simultaneously prevent over-generalization and over-fitting during machine learning training.

We highlight this dichotomy in Figure 4.1 which depicts the feature lattice data organization output we described in Chapter 2. In this example, we show the itemset and data for the smaller set of nodes, labeled Cluster {A,C}. We follow up this clustering with a subsequent prediction task to determine the likelihood that any one molecular dynamic conformation would transition into State 3, which in our application domain example may be a critical point at which a scientist can affect the virus with an antibody. Extending this example, we also note that the data within this cluster is comprised of two different subsets characterized as State 1 and State 2, shown in purple and green, respectively. Applying a single model to all the data in this cluster (on the left of the figure) has the unfortunate drawback of *over-generalization*, whereby a resultant prediction is not highly accurate

for either of these two subsets of data. In contrast, we could train two separate models for each of the data subsets (on the right); however, this solution suffers from *over-fitting* whereby the trained model is unable to accurately predict on unseen generated data conformations.

While this dilemma has a relatively low impact on molecular dynamics, this problem can be a significant obstacle to requirements in other domains. Projecting this same dichotomy on a different field, such as healthcare, we can better articulate the challenges associated in precluding both over-generalization and over-fitting. In lieu of training a machine learning task to predict the likelihood of a transition to a specific state, a typical heathcare requirement would instead seek to train a model to predict the likelihood that a person may contract a particular disease. We replace the spatio-temporal dataset of molecular dynamic motions with the longitudinal data of a patient's medical history. We categorize a global input domain as all historical data for all patients in a given population group; a single person is considered a localized sub-domain. Because every person has a very unique family history, genetic make-up, and lifestyle habits, a model trained for a global population group cannot provide the necessary fidelity for a single patient. Such a model trained using the entire input dataset may over-generalize and under-represent the importance of a specific trait for an individual prediction.

As we noted with molecular dynamics and depicted in Figure 4.1, training a model for each sub-domain is a potential solution. However, it results in over-fitting the model especially since the input domain lacks sufficient data for any single person. For example, training a supervised model for a patient using only her medical history to predict heart disease cannot provide an accurate assessment if she has never been diagnosed with the condition (i.e no positive labeled data exists). In contrast, as we previously mentioned, we could create one single model applicable for the entire global population. Unfortunately, this second approach limits localized accuracy for any one input. A model trained on everyone's data to predict a given disease is less likely to accurately predict this disease in any individual. We, thus, identify a shortcoming in this field associated with personalized models, or in the case of medicine, personalized healthcare, which is the requirement to strike a

balance between over-generalization and over-fitting.

In scoping this effort, we restate this requirement as the need to provide a localized solution tailored for a subset of data within a larger population, to which we associate the term *personalization*. Specifically, we aim to train a model providing a personalized prediction fitted specifically for a population subset which, for example, can be one person. To solve this, we identify the differing requirements in training and provide an architectural design for this genre using a multi-model training methodolody. We overlap training among all models creating a common dependency; each parameter space divides into an isolated and a shared space. Our approach optimizes localized and isolated portions of the models to preclude over-generalization while integrating into a paired, global training process which optimizes the shared parameter space and precludes over-fitting.

Our approach contrasts to much of the research in machine learning optimization which focuses on a single model with a sole task to map an input domain to an outcome. Model training supporting multiple tasks is often viewed as a purely separate problem and addressed with traditional distributed or parallel systems. From a systems perspective, this is typically implemented through an abstraction layer treating each as an executing process: the system schedules and prioritizes model training, but does so without any internal knowledge of the data or execution. While this ensures task isolation, it precludes data and computational sharing among the different models. In contrast, model sharing and reuse can help improve machine learning outcomes, but if uncoordinated, it can become resource intensive and computationally infeasible.

We, thus, seek to merge an application and systems approach and improve on machine learning effectiveness while also maintaining resource efficiency. In our architectural framework, we train multiple models concurrently where each shares a portion of the execution graph. We quantify these as semi-isolated models whereby sub-components of each must remain isolated from all others. As input, we identify hierarchical sub-domains within a population sharing common data points. Throughout this paper, we use a running example of personalized healthcare. This is an extension of the work presented by Soleimani, et al, which focused on reliable uncertainty aware

event prediction,[114] where longitudinal patient data is not only sparse, but can have missing entries. We identify a hospital patient's medical history as a sub-domain within a global population of all patients in one or more hospitals. The machine learning task applied in the example is to predict the likelihood for a given hazard in a patient, such as the onset of septic shock.

Implementing this on a single computer follows an arduous sequential process which can be infeasible for a large population. Extending to a distributed environment, training can scale but only in a limited capacity. Due to the structured nature of the training, a synchronized solution ensuring data consistency of the global space among the distributed nodes provides a very effective technique, but at a cost of resource efficiency. The systems goal of this effort it to maximize all computational resources in a distributed machine learning setting to best improve overall model performance. The caveat to this goal is that maximization of system resources should provide value added productivity to the application objective.

To achieve this capability, a system demands synchronization primitives capable of promoting an objective to either support model accuracy or resource efficiency when needed. Synchronization at global computational steps ensures data consistency but at a cost of resource efficiency. Asynchrony, in contrast, can allow a system to progress a training object at a much quicker pace, but it can potentially degrade overall performance. Contemporary systems, most notably TensorFlow, provide some capabilities to statically define such primitives, but are limited.

We have thus developed a capability on top of TensorFlow to control the level of asynchrony in a dynamic and adaptive manner, providing a better balance between resource efficiency and training effectiveness. Such a dynamical technique can enable a system to enforce a consistent data view among all distributed nodes to promote improved accuracy or it can loosen consistency requirements, allowing nodes to maximize local resources promoting faster execution. We, thus, aim to improve machine learning optimization workload management for model training and testing and demonstrate this as applied to semi-isolated models in support of personalized healthcare.

4.1.1 Background

Within machine learning only a small subset of published efforts seek to exploit internal data management or computational capabilities to improve overall performance across multiple training tasks. The closest genre is transfer learning whereby the knowledge gained from one model is used to train a secondary one. Traditionally, the techniques include leveraging the parameters, hyperparameters, or model structure of an already trained model, known as the source, to a new untrained model, known as the target. Its purpose is to reduce the overall training time or to improve accuracy of the secondary model.[115]

Transfer learning divides into two basic categories: the first, inductive learning, implies that source and target tasks are different while using the same or a similar input. We note that if the two input domains are different, an implied relationship exists between them. Examples of inductive transfer learning include using the same input image set to label two different output spaces: a source task which labels animal images as cats and dogs with a corresponding target task using the same input images, but labeling them as sleeping or eating. In contrast, transductive transfer learning focuses on transferring knowledge from one model to another where the source and destination tasks are the same. This is a more direct association between multiple models with many applied techniques include importance sampling of the domain[116] and parameter weighting and reuse.[117] In many of these cases, the techniques apply to artificial neural networks using an image or language processing input domain. In addition, research in this field also includes multi-task learning, where different tasks are trained concurrently.[118,119] It distinguishes from transfer learning, where source model is trained before the target model.

4.1.2 Motivation

Our work applies to the space overlapping both transductive learning and multi-task learning, specifically, focusing on applying a computer systems framework to optimize concurrent training and improve accuracy for multiple models within a single operational environment. We identify two

motivating reasons to pursue this research effort. One derives from the need to improve training effectiveness for better accuracy of a single, or personalized model, and the other addresses efficiency to provide training results in a timely manner.

The first motivating reason to develop this approach stems from a lack of data in any one of the isolated target tasks within a multiple model environment. A problem arises whereby domains lack sufficient data on which to train resulting inaccurate or non-converged models. In some cases, a lack of labeled input data to create a viable training dataset precludes supervised machine learning techniques; in fact, labeling, itself, is often viewed as a separate problem. If we aggregate these smaller domains into a larger population, sufficient data exists to create a training dataset applicable to everyone. However, using only the global population data without a locally scoped, trained model results in a generic solution which may not accurately provide a prediction for any one domain

We, thus, seek to improve training *effectiveness* by striking a balance between the extremes of training only a single global model and training many isolated models. This, naturally, requires data sharing among the models which we include in the methodology below. Our application of a multiple model approach addresses this shortcoming: to create a personalized model for healthcare, for example, the system lacks an abundance of source data for a single person. While one could create such a model using an isolated personalized domain, the lack of data hinders model performance.

The second reason addresses the need to provide training in a resource efficient manner. This entails employing a static set of resources to either train more models in less time or train models which converge to a solution in less time (we could also present this as using a varying set of resources to train as accurately as possible within a fixed time). In the baseline case of multiple, independent models, a solution already exists whereby training is dispatched with a trivial parallel approach. Since each model and source dataset is isolated, training more models in a shorter time merely requires adding resources where training limitation is bounded strictly by cost alone. This is a simple, linear scaling of the resources to the number of trained models. However, when domains

and models overlap, isolated, parallel execution does not scale linearly. Outcomes from one model have a direct impact on the performance of other models, leading to a dependency among the tasks; adding more resources does not necessarily improve performance or reduce training time, and can potentially reduce model accuracy. The relationship, thus, requires a synchronized effort at either the data or algorithm layer or both. We study the synchronization impacts on training to identify a means to control synchronization, providing a capability to improve machine learning *efficiency* without sacrificing substantial accuracy.

4.1.3 Challenges

We identify the following shortcomings in multiple model machine learning optimization and identify opportunities to improve the field:

1. **Concurrent training of dependent models does not scale**. With model performance tied directly to the underlying infrastructure, such as GPU resources, a single model's predictive accuracy is improved with additional resources. This entails either more time on a static set of computing nodes or more computational power to achieve better results in a static time frame. Although many factors influence model training performance, such as the training dataset and hyperparameter selection, the amount of resources dedicated to the training task directly correlate with accuracy. For isolated models, this scaling is linear, as stated above, but with dependent models, this scaling is not clearly identified. The hurdle of understanding how scalability impacts performance is one that must be understood in order to increase training capability.

2. **No current solution exists to dynamically mediate model training**. Training several models concurrently necessitates a resource management solution in order to efficiently allocate and utilize resources (time, memory, GPU) while simultaneously optimizing aggregate model performance. We define aggregate model performance as the collective accuracy of all models;

in the personalized model, this is the prediction accuracy for everyone in the population. Multi-model machine learning would benefit from the ability to make traditional systems management decisions within the training process. Such examples include discarding or preempting non-productive training efforts and developing work scheduling plans for executing nodes. While such efforts have been applied to hyper-parameter optimization, they have not been directly applied to concurrent model training across different models.

3. **Data Management is segregated and isolated, precluding data sharing opportunities**. Current machine learning systems are designed to train a single model and/or support efforts of multiple, isolated models. Techniques, such as between graph replication, where a model and data are replicated across many worker nodes, support the efforts to train a single set of shared parameters. Personalized models, in contrast, require both isolated parameters unique to each node, and a shared global state among the models. Contemporary frameworks, to include TensorFlow, in its open source version, lack this capability to concurrently support shared and isolated data models within the same executing graph. They can support limited implementations, but are tailored to support a specific class of algorithms, most commonly found within convolution neural networks (CNN).

4.1.4 Applications

While the direct application of our methodology aims to improve health monitoring and disease detection, it is not isolated to this use case. We identify the following potential use cases where such a systematic approach to personalized model training management and optimization would directly apply.

Personalized Health Care. In general, personalized healthcare refers to dividing a population into various groups in order to separately diagnose and apply treatment to each. The motivating reason is that we acknowledge everyone is unique with different history, lifestyles, and genomes. A diagnosis or treatment applied to one person may be counter-effective on another. Taken to the

extreme, we can potentially define a personalized healthcare profile for everyone in a given population, which includes machine learning models to predict health risks, diseases, and make treatment recommendations.

As a running example, we apply our framework to the detection of sepsis. Formally, we define sepsis as a progressive inflammatory response due to internal infection resulting in the fatal condition of septic shock.[120] It is difficult to monitor and detect and any early warning to its onset can be life saving. In this case, we refer to the application of machine learning as a decision support effort helping hospital staff prevent the onset of shock. However, given that hospital lengths of stay are short and routine doctor visits are often sparse over a person's lifetime, sufficient data may not exist to create such unique models. Data collected from a bedded patient in a hospital is limited to a short span of time and lacks an abundance of data, let alone labeled data as input for a machine learning approach. Creating a personalized model tailored to the individual which can predict the onset of disease or the likelihood of a sudden fatal condition, such as septic shock, is challenging. Such an application can benefit from combining a global population dataset with an individual subset to create a personalized predictive model. Although we utilize this example throughout the paper, our methodology is extendable to other fields outside of medicine.

Internet Of Things As devices become more computationally capable, each is gaining the ability to integrate into a localized machine learning system. From kitchen appliances to televisions, home networks are slowly permeating throughout society; the ability to perform machine learning at home already exists. In addition, the list of sensors and data inject points is also increasing over time with improved video devices, wearable technology, and even personal digital assistants. Data analytics is already intersecting with the internet of things (IoT) to create an environment well suited to leverage the capabilities provided through incremental or on-demand machine learning and further improve everyday life. The potential applications include home security, energy efficiency, and even lawn care. Given this robust home environment, we identify the need to optimize the local computing resources within a personalized setting (e.g. home server, cloud appliance, or even mobile

device) to train a set of models which integrate data from a variety of sources. Specific examples may include: (1) developing a model to process home security video for suspicious activity and alert the server to increase recording time or frame rate, and (2) tracking home network activity to passively monitor and label children online behavior.

Time Sensitive Models. In some situations, the critical resource is time where a model is required as fast and as accurate as possible. Such cases may include emergency response or military operations. In these situations, features, tasks, and requirements may not be known in advance, precluding a system from developing models *a priori* which can be employed immediately for prediction or classification. Infectious disease response may seek to develop models which can quickly assess a human for contamination, severity, and contagiousness. A military unit may need to integrate newly discovered intelligence into a series of decision support systems applied to battlefield data in support of ongoing operations. In both scenarios, time is a vital resource where any optimization to improve performance across all training efforts would provide tremendous benefit.

4.2 Machine Learning

Machine learning is a genre within computer science that uses data to develop models to predict or discover information. Within machine learning, we focus on the systems employed to support supervised learning algorithms leveraging a labeled data set, consisting of an output for each input element, to learn a predictive function capable of classifying a newly unseen input. One common example is character recognition which accepts an image as input and produces an letter as output. Applied to personalized healthcare and septic shock, each model represents a function mapping a person's historical, demographic, and longitudinal data to make a prediction on the likelihood of a prescribed sepsis condition.

In comparison, regression algorithms find the "best fit" manifold for a multi-dimensional dataset and, in a more broad categorization, unsupervised learning garners information from an

unlabeled input domain. While the methodology and techniques contained herein can apply to regression and even unsupervised learning, they are not included in the scope of this paper. In this section, we discuss the system frameworks underpinning the tools employed in contemporary development environments. We highlight and focus on the TensorFlow platform, as it is among the most robust, capable, and widely used. Subsequently, we discuss training methodology and how it can vary in a distributed setting. Finally, we identify the specific distinctions which separate semi-isolated, multiple-model training from the traditional approach to machine learning.

4.2.1 Systems and Architecture

Machine learning has made incredible strides over the past decade due to significant advances in computational processing. Beginning with the large algorithmic leap presented by Geoffrey Hinton in 2006 where he and his colleagues presented a deep belief network,[121] researchers have moved quickly to improve performance and system efficiency to train large, multi-layered, deep neural networks using gradient descent process.

We divide the machine learning process into three phases. The first is to *build* a model defining a function to map an input X to an output Y. Models, $\mathcal{M}_i$, are algorithms traditionally represented as direct acyclic graphs, although in certain cases, cyclic graphs are acceptable. The second is to *train* the model which consists of executing the graph repeatedly to update and improve the model as an optimization problem minimizing an objective function, f. The last is to *test* the converged parameters whereby we use the trained model to make predictions on new data input. Most systems focus on the first two of these phases, but many are providing a service capability to more efficiently support testing and prediction.

While developing a single-node, multi-threaded implementation is feasible, such a solution is geared toward a single-focused task. Extending such capability or scaling the model to support a larger data domain necessitates a more broad architectural design and implementation. Systems such as Torch[122] and Theano[123] provide an interface and runtime engine to enable more users to

construct and develop complex machine learning models. A precursor to other machine learning engines, such as Google DistBelief framework[124] and Caffe,[125] these systems were revolutionary in the field as they presented a graph based engine with user-friendly programming extensions on top of scripting languages Lua and Python while leveraging compiled routines to train large models efficiently.

Distributed Machine Learning

Extending beyond a single system, distributed machine learning leverages the aggregate computational resources of many nodes to perform a single, unified machine learning objective. The basic design pattern for multi-host machine learning derives from the parallel version of the classic stochastic gradient descent algorithm.[126] This effort was a natural extension of multi-threaded, single node implementation. Similar to multi-threaded execution, distributed systems must ensure message coordination and parameter synchronization, a service provided by the executing machine learning engine. A relatively new territory in the field, distributed machine learning can further extend the limits of model complexity. However, the distributed approach to machine learning, naturally has its own advantages, disadvantages and considerations which we describe below.

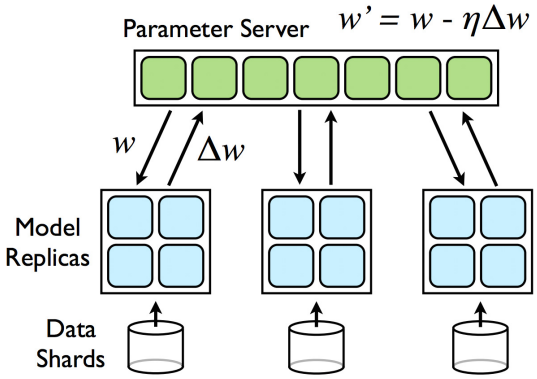


Figure 4.2: Distributed Machine Learning. The parameter server design employs one set of nodes to only manage the data input/output while a another set consists of workers executing a replicated copy of the same machine learning model.[124]

The critical difference in a distributed setting is that computation and data are no longer co-located. Each node maintains its own local view of the shared parameter values and applies its view at the beginning of each training step for forward feed graph computation. Based on a

user-defined training methodology, these views can either be fully consistent or inconsistent (note: we could enforce that parameters are *eventually consistent* or use a looser term and merely state less than fully consistent; however for simplicity we state that parameters are either fully consistent or not). While many machine learning systems provide a means to coordinate training using either of these two, TensorFlow provides an extended flexibility for users to define their protocols which can integrate these two extremes. We leverage this flexibility in designing the methodology for personalized model training.

In 2011, Agarwal and Duchi presented the parameter server model which they referred to as delayed stochastic optimization.[127] The model, depicted in Figure 4.4 applies gradient updates at the master every time a worker completes forward feed execution of the graph. Using a global timestamp clock, the system tracks gradient age. The authors showed how the error resulting from executing back propagation with older gradients in asynchronous execution scaled with the number of nodes.

Shortly after Google published its open source version for machine learning system, DistBelief, the precursor to TensorFlow. It implemented an asynchronous version of stochastic gradient descent called, Downpour SGD. using a parameter server model with a single graph replicated to each executing worker.[124] The DistBelief pattern segregates node functionality into a state and stateless execution. The stateless nodes, the workers, are responsible for the compute intensive task of executing the model graph. The stateful nodes, known as the parameter servers, perform simple computations, such as applying gradient updates, and focus system resources on data I/O since mutable state changes frequently during training. Systematically, this isolates the data write operations to few or a even single node, and allowing the workers to execute in a stateless fashion.

Project Adam enhanced the parameter server model for an improved convolution neural network design.[128] The Adam architecture distributed parts of the model to different nodes while retaining a common part which is replicated among all nodes. This allows data parameters local to the sub-components of the graph to remain isolated and reduce the synchronization to a smaller

portion of the model.

The Hogwild! system demonstrated and theoretically showed performance benefits of executing multi-threaded machine learning without any synchronized data consistency controls.[129] This lockfree approach enables single processing threads to update shared data asynchronously, including allowing a thread to overwrite updates from other concurrently executing threads. In stochastic gradient descent, this technique proves very effective since data access is generally sparse with each processing step only modifying a small relative portion of the global parameters. Through experimentation, the authors demonstrated near linear speed increased with extremely limited data inconsistency or error. Extended beyond single node system, the lockfree approach has been proven effective and efficient in machine learning application. Building on top of these successes, Petuum,[130] offers even more optimized lower level system integration, although it focuses on load balancing and reducing network latency.

TensorFlow

TensorFlow[131] is a machine learning system providing a programming interface and runtime engine to execute a wide variety of algorithms. Models are implemented as dataflow graphs encapsulating both state and computation: nodes represent operations, such as add, reshape, or reduce, and edges represent data as multidimensional arrays, or *tensors*, passed from one operator to another. Users define a set of operators with associated parameters and subsequently execute the graph in a session. Sessions manage the data flow between nodes by *feeding* input values into the graph as immutable objects executing any arbitrary sub-graph within the model.

In TensorFlow's delayed execution design, a preprocessing phase builds an executable model prior to training. This enables the engine to optimize for hardware performance and add control flow operations to the user-defined graph. This has the one disadvantage of forcing static graph construction prior to execution for a distribution system, precluding dynamic manipulation during training without completely rebuilding the graph. The training phase consists of executing the graph

and iteratively updating mutable state to optimize an objective function. Applied to traditional, neural network models, this consists of the forward feed and backward propagation. However, the underlying Tensorflow engine is designed to provide maximum flexibility, enabling users to develop any arbitrary models using control flow primitives, such as conditional operators and while loops to meet algorithm demands. It can execute multiple, concurrent, even overlapping subgraphs on the same device in a multi-threaded fashion, thus enabling lower level optimization tailored to specific hardware implementations.

With hardware capable of supporting multiple graphical processing units configured to provide general purpose programming, an empowered user can leverage TensorFlow on a single server to implement very complex models containing millions of parameters. However, model complexity ultimately reaches a computational boundary for a single node system and scalability can only be achieved through a distributed, multi-node solution.

As is the case with other machine learning systems mentioned, in TensorFlow worker nodes operate with a global parameter updated from a specific timestep, t. A staleness factor, τ_t is attributed to the gradient produced by this worker when it completes it local forward feed calculation on the loss function. Global calculations of parameters can account for this staleness factor by dropping older gradients.

The primary challenge for distributed execution is the management and consistency control of the large parameter sets for the training model among the executing nodes in the system. One extreme is to provide a standard locking mechanism on all data sources where each node attains a mutex lock on a shared memory location in order to make an inplace update on a parameter value. While this ensures consistency, it introduces significant overhead. In contrast, a system can completely remove all locks and allow the nodes to make updates to data values at will, as mentioned with Hogwild! which is extremely effective on sparsely updated data. However, for more dense data updates, systems require a means to synchronize among the working nodes which is the basis behind a the parameter server design found in the Project Adam and DistBelief systems. TensorFlow follows

this similar pattern, but provides more capability and functionality in data synchronization.

One of TensorFlow's greatest advantages is its capability to support very large models on many nodes. Like other distributed engines, TensorFlow provides data consistency controls at each step while maximizing parallel execution opportunity for computation. By assigning distinct sets of mutable state to specific stateful nodes, TensorFlow, like other parameter server based systems, can execute write operations on the shared global parameters while allowing stateless computations to safely read updates. Additional inter-node communication through data queues facilitates the timing and coordination between steps to inform workers when to read parameter updates and proceed to subsequent execution. A critical distinction in contrast to the traditional architecture, Tensorflow removed the computational limitations on parameter servers. Instead, the system organizes nodes into jobs, where any node in any job can execute part or all of the dataflow graph, providing an opportunity for developers to capitalize on algorithm requirements to meet unique demands, such as training isolated models for personalized health care.

4.2.2 Training

Machine learning is the process of developing and executing a *black box* function mapping an input domain to an output label. We define the input domain, $\mathcal{D}$, consisting of the input feature set, X, and the output, Y with an implied association of input to output, or $X \rightarrow Y$. The primary objective of training a machine learning model, $\mathcal{M}$, is to use given data to *learn* the function, $\mathcal{F}(\cdot) = P(Y|X)$. The model consists of an algorithm along with associated mutable - or *trainable* - parameters, w. Training the model requires optimizing an objective function, f, to minimize the difference between the model's performance and the actual result for a provided labeled dataset. At each step in the training phase, a computational engine executes one forward pass of the algorithm resulting in an observed output, Y', calculates the amount of loss in the system using the objective function, and updates model parameters using the gradient change with respect to the loss function:

$$w_t = \nabla_{(t-1)} w$$

This process is aptly known as *gradient descent*. By *descent*, optimization attempts to traverse *down* a cost effective path to minimize the objective function, f. At each step in the training process, the objective function is evaluated and gradients are calculated and applied to update parameters for subsequent training iterations. Training continues indefinitely until the process reaches a user-defined stoppage criteria, such as converging to local minima or exhaustion of limit-based resources (e.g. time).

Implementing machine learning in a distributed setting is not a novel concept. A fully synchronized methodology for classic algorithms, such as stochastic gradient descent, follows a very similar approach to a multi-threaded, single node solution: each executing processes selects its own random batch of input data and calculates a single gradient update. Once all processes have completed, a single, master process accumulates each gradient and applies an aggregated version to the trainable parameters in the model. Using the updated parameters, each process continues on its next step in the training cycle. This simple synchronization process ensures data consistency at each step while maximizing parallel execution opportunity for computation. With a single node performing write operations on the shared global parameters, all other process threads can safely read updates. Additional message processing can facilitate timing and coordination to ensure local parameter updates occur only after global parameters are updated.

While the fully synchronized method provides a means to vastly improve training effectiveness, the distributed execution introduces new overhead costs and challenges. Implementation requires additional layers of system component complexity to develop message passing protocols, networking, and data coordination. This has the side effect of adding more programming, processing, communication overhead, and supporting systems. Other networking, input and output, also contribute to overall delay, specifically in a constrained environment where nodes compete for shared

resources. A typical resource, such as data access, can result in reduced efficiency at the local level. That is, if two nodes need to share a common access path, either one will receive, on average half the available bandwidth. This results in a longer access time to read/write data, contributing to latency in model training. Another cause for system latency results from data imbalance among the working nodes. When distributed, data is sometimes not perfectly divided; this cause leads to the effect whereby some nodes must consume more resources to perform model training. This phenomena, *data skew*, results in a subset of slower progressing nodes, known as stragglers, which stagnate global optimization of the model.

Synchronization

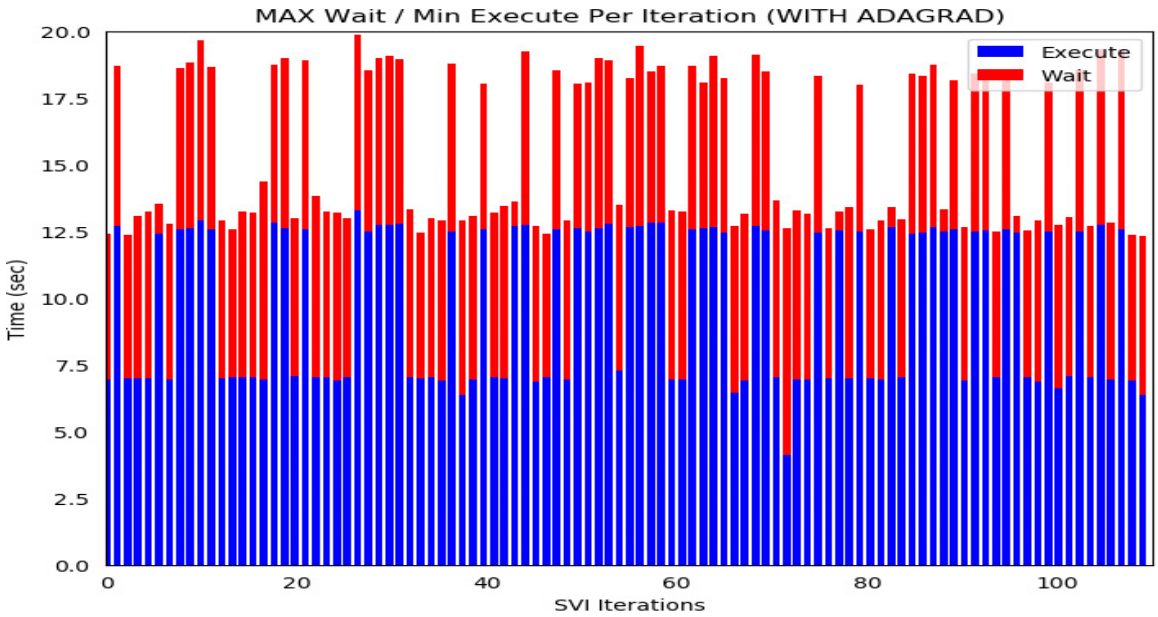


Figure 4.3: Profiling synchronized execution. This figure is a snapshop of training time for 120 iterations of the distributed LMC Algorithm. Each vertical bar represents a single global time step of training and highlights the "fastest worker" for the corresponding step. The blue bar indicates actual processing time to complete local work and the added red above indicates how long that worker waited to continue to the ensuing step.

Traditional distributed architectures, to include TensorFlow, determine the optimization technique *a priori* of whether to enforce a consistent data view at each step of training or to allow each

workers to execute local optimization with a potentially different value for globally shared state. This distinction, known as the contrast between a synchronized and asynchronous execution, has a direct impact on model training efficiency and its effectiveness of prediction accuracy. Synchronizing data at each step promotes a more accurately trained model, but the coordination adds latency which increases the time to reach convergence. Asynchronous training, in contrast, maximizes resource utilization, but may degrade model accuracy or prevent it from attaining a more optimal state.

In a synchronized setting, TensorFlow, like other distributed machine learning systems, enforces a consistent state for all workers' computation at each global step. Locally, each node reads the current global parameter values from the parameter server(s) and perform its forward feed step on its local share of the global data, calculating a single gradient update. Subsequently, each node waits for all others workers to complete this same step. Known as a barrier, this synchronization primitive control technique ensures data consistency but at a cost of resource efficiency since workers are blocked from performing any operations. Once all gradients are aggregated, the parameter server(s) apply an aggregated version to the trainable parameters in the model. Using the updated parameters, each worker continues on its next step in the training cycle. It should be noted that because gradient updates are communicative for synchronized training, the order in which the parameter server applies updates within the same global timestep can be arbitrary.

A fully synchronized algorithm which ensures total data consistency can only progress as fast as the slowest node. Because each executing process must wait until all other processes complete local execution prior to receiving updated parameters, slower nodes can hinder progress. This is graphically shown in Figure 4.3 depicting a 120-iteration sample of training. Each bar in this graph represents the operational processing for the fastest worker on a single global timestep in a training session. The height of the blue bars represent actual work performed and the red component is lost execution time due to idle processing. If we capitalize on this opportunity to eliminate the computing resources which wait in idle, we can improve overall system performance provided it can be accomplished with little to no loss in algorithm performance.

In an ideal setting, this fully synchronized methodology can provide the most accurate and efficient solution. In this scenario, all memory and computational processing are uniformly distributed with each parallel node supporting identical hardware configuration. If they each execute the same amount of computational operations, then global processing should follow a routine pattern whereby all threads start and finish local work at the same time, allowing the master to quickly update global state and initiate the next training step. Realistically, parallel processing rarely meets this ideal goal. Some processes finish local computation early and must wait on straggling efforts from other threads.

To eliminate the idle time in the distributed system, we can remove the barrier point and allow the nodes to execute training asynchronously. The key advantage is the increased computation time spent optimizing the model; however, the lack of a shared global view among the nodes represents a source of inconsistency in the system. With workers calculating an objective function using different initial parameters, global updates can actually lead to digressive results. In addition, the order in which parameters are applied also has an impact on the overall system state since each node may read different views of the data at different points in time.

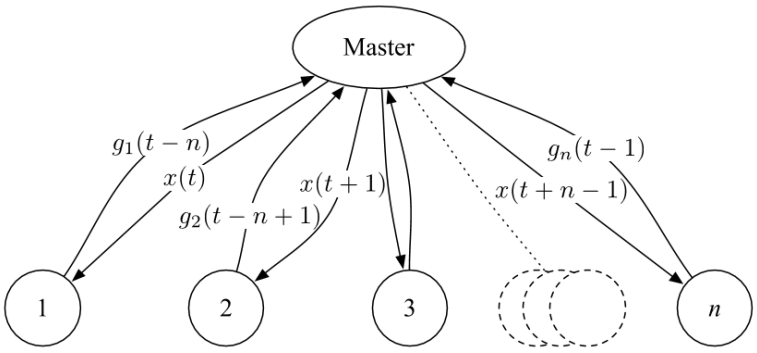


Figure 4.4: Distributed model of delayed stochastic optimization.[127] Worker nodes submit gradient updates g_i calculated from a previous timestamp, $(t-n)$ to the Master. The Master applies updates, increments the global timestep clock to and returns to the new parameter, $x(t)$ to that worker. Synchronized training would have all workers submit a gradient $g_i(t)$ and all receive the same new parameters, $X(t+1)$.

This design, shown in Figure 4.4, reveals how each gradient, g, can have a unique timestep from each worker, $g(t-n)$. Because distributed systems, are by nature, non-deterministic as a result of other external pressures, such as network latency, gradient updates may arrive at different times.

One key metric used in this setting is the staleness of a given update, which is measured in the number of global timesteps from the initial point a worker begins local computation to the time the master receives the resultant gradient update. The staleness factor can have a significant impact on overall performance. To counter this effect, we identify the aggregation of gradients as a tuning parameter which we call asynchrony affecting the amount of synchronization during model training.

4.2.3 Unique Aspects of Semi-Isolated Models

To support an overlapping training effort providing a localized model within a global context has some distinct characteristics. We term this as semi-isolated multi-model machine learning, since only a portion of the model parameters must be isolated. Described below, we present the unique challenges and requirements for this class of machine learning. Specifically, highlighting three primary differences pertaining to the input domain, the training requirements, and the testing/servicing phase.

Input Domain

The first key distinction is the organization of data into hierarchical input domains to support personalized applications. We define localized domains as subsets of the global one, $\mathcal{D}_i \in \mathcal{D}_\mathcal{G}$, given that all sub-domains share the same input feature space, X. In addition, all local sub-domains segregate into disjoint subsets such that:

$$\mathcal{D}_\mathcal{G} = \bigcup \lim_{i=1}^{I} \mathcal{D}_i, \text{ for each, } i \in I \text{ subset of data}$$

Each sub-domain has an associated machine learning model, $\mathcal{M}_i$, which is unique, or *personalized* for that input sub-domain. We retain the same algorithm and objective function for all models, but establish localized parameters for each, w_i. In addition, all models share a set of global parameters, w_G with a common set of operators. Applied to personalized health care data,

for example, we specify each patient, as a unique, localized data subset consisting of that person's historical, longitudinal data, such as heart rate or blood pressure over time along with her static demographic data (e.g. age, gender). This design provides a framework to develop hierarchical, taxonomy of sub-domains organizing data in a multi-layer tree structure of populations. In this extension, every inner node of the tree has a common set of parameters which are shared among all descendent nodes. Leaves of the tree represent the personalized models. However, for the scope of this research effort, we focus on the division of data into sub-domains applied to a two-layer global and local hierarchy.

We highlight the contrast between aggregating all data to train a single, unified model and of developing completely separate, isolated models. While complete aggregation is a possible implementation, it precludes the desire to develop personalized models providing a tailored, more accurate prediction function to the end user. Likewise, creating completely isolated models is not a feasible approach since each individual lacks sufficient training data to develop a unique model. Our approach, thus, balances between extreme globalization, or *over-generalization*, of data which defines a single, unified model, and complete segregation of models where every domain is trained separately and can result in *over-fitting*. This separation of localized data to support personalized models creates a distinct need for *isolation* in the machine learning implementation.

Training Requirements

The second key distinction pertains to the training demands for alternating updates to local and global parameters. In contrast to a typical machine learning implementation, which updates a single set of parameters, w, at each step, a personalized model must optimize one set of parameters, w_i, as a single step *within* the global optimization process. Thus, for each step to train the set of shared parameter, w_G, the engine must execute numerous training steps on a single model, $\mathcal{M}_i$.

For single machine implementations, this process is a sequentially executed algorithm. At each step, the machine learning system loads local parameters into memory, executes training

Algorithm 4 LMC Algorithm: Single Node Training

for t in MAX_ITERs **do**
  $\mathcal{D}_i \leftarrow$ select random sub-domain
  Optimize w_i (AdaGrad, L-BFGS)
  Calculate global gradient $\rightarrow \nabla_t$
  Apply global gradient: $w_{G_{(t+1)}} = w_{G_t} - \nabla_t$
for i in N **do**
  Optimize w_i (AdaGrad, L-BFGS)

to optimize the local parameters while keeping the global parameters fixed. Once converged, the algorithm computes and applies a global gradient update, freezing the local parameters, but allowing the global one to mutate. Using the new global parameters, the algorithm progresses to the next iteration using a different sub-domain of data, $\mathcal{D}_i$ and corresponding local parameters, w_i. Model selection can either proceed in a stochastic fashion, where sub-domains are randomly selected at each step, or a sequential one, where all models are trained in sequence.

We briefly note the opportunistic potential of including data-driven input model selection, which we described in Chapter 2, as an alternative to the batching or random selection associated with stochastic optimization. One of the drawbacks with basic sampling techniques is the lack of specificity or an ability to steer (or affect) the model training. Importance sampling aims to overcome this limitation by factoring in a target distribution into a source distribution when sampling. Often, this technique is necessary when the target distribution is either too complex or impossible to sample directly. Representing the importance aspect of the problem through a target distribution, or a importance sampling density $g_s(X)$, and calculating a weight with respect to the source distribution, $f_s(X)$ provides a re-focusing (or re-weighting) with a reduced variance resulting in a sampling that favors important samples.[32]

We note sampling as a potential key issue in this training methodology because each step, is itself, a complete training subset. Improved model selection can be a critical factor in improving overall time to convergence. Selecting the most opportune order to train models would certainly result in the best outcome and we note that, especially, the asynchronous nature of training results in varying rates of convergence, even when the order of data input is held constant. Unfortunately,

knowing this sequence *a priori* is not possible. Ergo, if we can apply an adaptively controlled technique which can make dynamic input decisions, we can drastically improve training efficiency and effectiveness.

Testing

The third key distinction is found in the testing phase. To serve models using new data, traditional, trained models only process a single forward feed pass of the model to make a single prediction. The model servicing component requires no mutable state and normally consumes fewer resources. This often results in smaller, condensed versions of the trained model enabling machine learning systems to permeate throughout society reaching more devices which less capable hardware. In contrast, to predict on personalized models, the system must execute local optimization of the sub-domains parameters, w_i. Global parameter, w_G, become immutable but the system allows local parameter to update, until at such point, sufficient local data within the sub-domain exists; we note that this is equivalent to a completely isolated and independent model.

The servicing component, must thus, tie directly into the training phase. This is especially the case with TensorFlow, and other major machine learning system, where we cannot leverage the optimized model prediction capability found within these servicing modules. For our purposes, we have identified an online prediction capability as a coupled requirement with machine learning training. This requirement is akin to traditional transfer learning since we are using the training from previous models to perform local optimization and prediction on a new dataset. The challenges arise from creating a labeled dataset in real time (or near real time) to support the training of the new sub-domain.

4.3 Methodolody

This section covers the technical implementation for the distributed algorithm implemented the Linear Models of Coregionalization (LMC) framework. It includes an detailed description of the

global optimization design patterns for both a synchronized and asynchronous setting along with the local optimization techniques. In addition, this section addresses how we tackle the challenges of *data skew* and our methods for providing an online prediction capability to service the model on new input sub-domains. We also present additional implementation details

4.3.1 Global Optimization

The global optimization is divided into two separate components: training at the local node for a single sub-domain, and the aggregation step to optimize the global parameters. As a comparison, the algorithm depicted in Algorithm 4, above, shows the basic steps to train both global and local parameters, w_G and w_i, respectively, for implementing the algorithm on a single machine where each step is performed sequentially.

Executing on a single node, the LMC algorithm can readily manage data flow between local and global optimization logic within a single TensorFlow graph. Updates to local parameters in step 3 which overwrite stored TensorFlow variables as each new batch of patients is sampled have no impact on the remainder of the executing code other than to support local and global optimization in steps 4 and 5. In contrast, such changes to variable parameters can have significant impact in a distributed setting if these TensorFlow variables are declared on a commonly shared, global graph.

Shown in Algorithm 5, we present the implementation pseudo-code for the distributed LMC Algorithm. This is the synchronized logic for both the master and worker nodes. The key changes are the added `SEND/RECV` operations for data coordination between master and worker. In TensorFlow, we implement this using a standard message queue as an operator in the model graph. The workers, otherwise, perform the same local optimization; the master node stores all workers' gradients upon receipt and updates global parameters when its has accumulated each worker's update. This version of the algorithm does not account for fault tolerance in execution. However, to account for lost worker nodes, the master would need to change the number of expected gradients from N to $(N-1)$, and then update this same value when (or if) the recovering node rejoins the

cluster and continues training. In addition, the system may benefit from a data redistribution to reallocate data assigned to the lost node.

Algorithm 5 Distributed LMC Algorithm (Synchronized)

Input: NumWorkers (N), InitParam (w_G), GlobalOptMethod, (f_m)

Master

Set $A, t = 0$
`SEND` $w_G, 0$ to each worker
while *true* **do**
  `RECV` ∇_i from worker, i
  accumulate: $\nabla_A = \nabla_A + \nabla_i$
  $A = A + 1$
  if $A == N$ **then**
    $\nabla_{G(t)} = \nabla_A / A$
    $w_{G(t+1)} = w_{G(t)} + f_m(\nabla_{G(t)})$
    $t = t + 1$
    $A = 0$
    $\nabla_A = \nabla_A * 0$
    if t == `MAX_ITERS` **then**
      $t = -1$
    `SEND` w_G, t to each worker

Worker

while True **do**
  `RECV` w_G, t from master
  if $t < 0$ **then**
    `BREAK`
  $\mathcal{D}_i \leftarrow$ select random sub-domain
  Optimize w_i (AdaGrad, L-BFGS)
  Calculate global gradient $\rightarrow \nabla_i$
  `SEND` ∇_i to master
for $\mathcal{D}_i$ in $\mathcal{D}$ **do**
  Optimize w_i (AdaGrad, L-BFGS)

Multi-Session, Multi-Graph execution

To implement the back-and-forth optimization between both global and local parameters in a distributed setting, we leveraged two separate graphs executing in alternating sessions in TensorFlow. Although we initially considered creating a single, uniform design using either in-graph or between graph models, we found neither of these were a feasible solution to meet the algorithm's demand. This shortfall in depicted in Figure 4.5. In-graph replication, where operators and data are allocated for each executing device, is not scalable as the number of workers increase. We performed a few simple checks on creating a single graph with this technique. Unfortunately, the current LMC graph requires approximately 4 GB of memory (not including data and intermediate requirements to support processing). While we could have used very large, memory intense nodes, we could not support such a very large graph nor could this scale in the number of models to a larger population.

Alternatively, we implemented a between graph solution where one graph is constructed

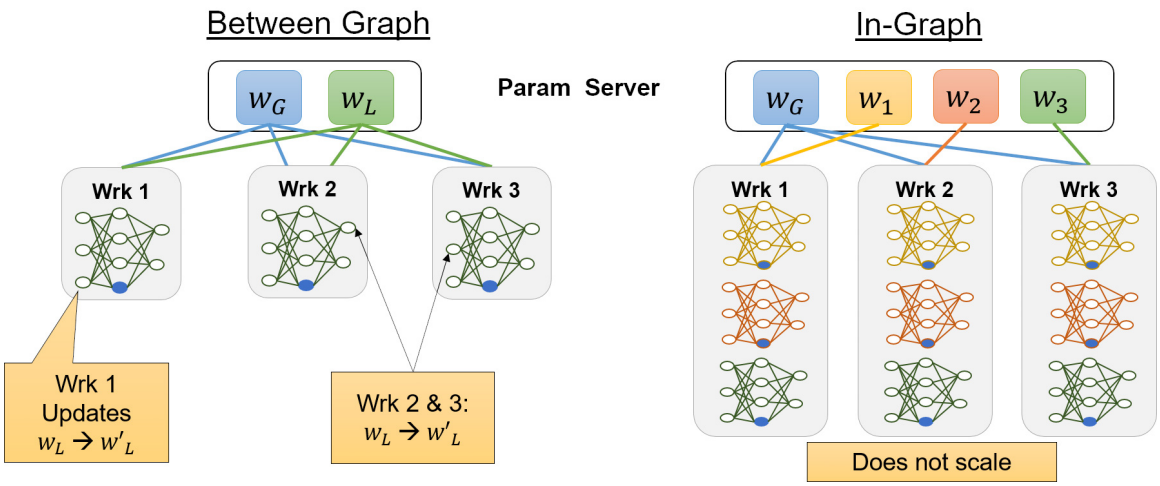


Figure 4.5: TensorFlow replication strategies. *Left*: Between graph replication enforces parameter consistency for local variable, w_L. *Right*: In-graph replication provides an isolated model for each worker, however, increased network latency and graph size make this option infeasible.

and replicated for each worker. When executing, we discovered that the local variables defined for each worker are not truly local: they are shared among all workers and exist on a single device in accordance with the parameter server design pattern. While each executing node retains a copy for local execution, the Tensorflow engine can periodically update these values. Typically, this has little impact on a traditional training step which executes one forward feed and one backward propagation as a single global training step. Furthermore, because we developed a custom operator to execute optimization over the personalized model, the local graph executes multiple times within a single session. We discovered that our local variables were synchronizing across all distributed nodes after approximately 40-50 timesteps whereby, every node's local view updated to reflect the values stored in the parameter server at that point in time.

Ultimately, we pursued the dual global/local graph pattern. The master executes solely within the global graph, managing the gradient coordination queues to `SEND` and `RECV`, as well as accumulating and processing the aggregated gradients with the user-specified global optimization method, f_m. Additional python control logic enables the master to process asynchronization primitives as defined below. The worker alternates between the two graphs: it communicates with the

master via queues and otherwise performs TensorFlow operations locally. The basic logic to execute the two graphs are shown in Listing 4.1.

```
# Define the cluster
isSupervisor = (FLAGS.job_name == 'ps')
tf_hostname  = lambda i: nodegroup + '-%d.'%i + svcname
ps_hosts     = [tf_hostname(0)]
worker_hosts = [tf_hostname(i+1) for i in range(num_workers)]
clusterDef={"ps": ps_hosts, "worker": worker_hosts}
cluster = tf.train.ClusterSpec.__init__(self, clusterDef)

# Define Local Session / Graph
serverLocal   = tf.train.Server.create_local_server()
with tf.device("/cpu:0"):
    sessionLocal = tf.Session()
    manylmc = manyLMC_joint_model.ManyLMC(train_data, id_list)
    manylmc.define_local_graph()
    local_vars, global_vars = manylmc.define_data_local()

# Define Global Session / Graph
serverGlobal  = tf.train.Server(cluster,
      job_name=FLAGS.job_name,
      task_index=FLAGS.task_index)
graphGlobal = tf.Graph()
with graphGlobal.as_default():
    with tf.device('/job:ps/task:0'):
        manylmc.define_global_graph(global_vars)
        ...
        global_init_op = tf.global_variables_initializer()
supervisor = tf.train.Supervisor(graph=graphGlobal,
    is_chief=isSupervisor,
    init_op=global_init_op)
sess_config = tf.ConfigProto(log_device_placement=(FLAGS.verbose>2))
sessionGlobal = supervisor.prepare_or_wait_for_session(
            serverGlobal.target,
            config=sess_config,
            start_standard_services=False)
manylmc.setSession(sessionLocal, sessionGlobal)
```

Listing 4.1: Global Graph for Gradient Processing

The first block defines the cluster specification automatically based on the prescribed number of workers, a environment variable passed in as a configurable parameter. Note that because we use deterministic hostnames, in our case through the Kubernetes container management statefulset, we can derive the fully qualified name and assign it accordingly. Otherwise, this would need to be passed in as well through an environment variable or accessible via a discovery or shared state service, such as Zookeeper or Dynamo.

The local session block creates the TensorFlow service on the node and instantiates the LMC joint model model, `manylmc`. This is the driver object defining the model over the local shard of training data. Note how we create the local graph first to define this object, which separates the local and global parameter list. The operators pertaining to the global session and graph are defined on the master, denoted with the `ps` label. Worker nodes are limited to interaction with the queue nodes only. When the supervisor object calls `prepare_or_wait_for_session`, the master only initializes all global variables and waits for all workers to join the session. We have added additional signaling and messaging (not shown) which ensures the nodes are all joined before any training begins to ensure any one node does not *race ahead* and bias training.

Synchronous Execution

Depicted in Figure 4.6, the global optimization step shows the alternating processing between local operations, in yellow, and global ones, in blue. A single iteration begins with the master sending a new set of global parameters, w_G, to each worker node input queue, $\{N_0, N_1, .., N_n\}$. Once received, each worker samples a window of local input which is a subset of one patient's longitudinal data or batch of windows with each a subset of different patients' data. We define a set of patient windows as overlapping with each containing the same initial start point, but different end points. Local optimization over a window minimizes the objective function with respect to the sample data and executes for up to 500 iterations using either via the custom operator described in Section 4.3.2 or with Python SciPy's minimize module. In either case, we found convergence is usually achieved

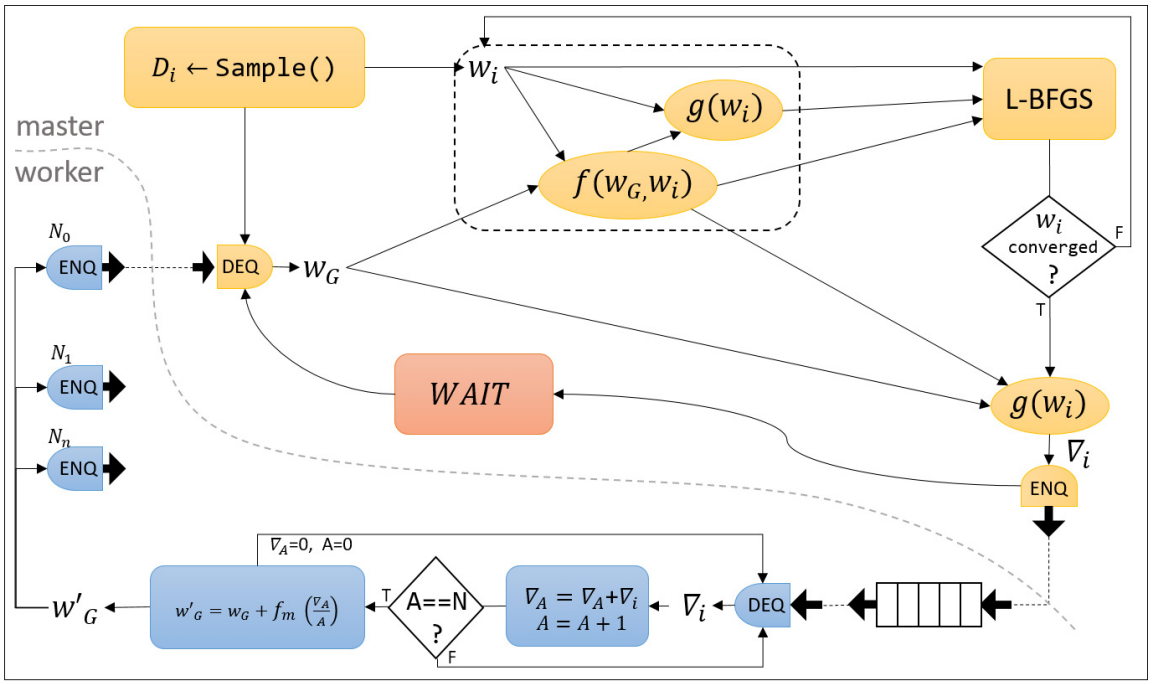


Figure 4.6: Implementation design for the distributed version of the Synchronized LMC Algorithm. Nodes in the yellow are executed on a locally isolated Tensorflow graph on each worker. Nodes in blue are part of the global graph and executed at the master.

within the first 50-100 steps. Once converged, the local worker calculates a global gradient, ∇_i and submits it to the master's queue for processing in the global session. To synchronize gradient calculations globally, each worker subsequently waits in idle until the master produces a new global parameter. Concurrently, the master accumulates all workers' gradients for a the given timestep, ∇_A; when all are received, it calculates a new global parameter, w'_G, and notifies all workers to continue execution. To calculate a new, w'_G, we provide a primitive enabling the system to execute a user-specified optimization strategy, f_m, of which we have implemented Adagrad, Adam, RMSprop and, simple gradient descent.

We provide two synchronous execution strategies. The first is the baseline, or full synchronization where each worker submits one gradient update to the master in each timestep. In this scenario, all worker gradient updates aggregate into a single, global parameter update. As a second extension of this, we also include a mini-batch capability where nodes train on a set of patients

within a single timestep. The aim of mini-batch execution is to reduce the overall idle time at each worker, but at the cost of potentially hindering overall global progress since the system calculates fewer global parameter updates for each local worker optimization step.

We show the basic graph operators used to construct the global portion of the this pattern in Listing 4.2. This highlights the major operators for gradient processing which are used for both synchronous and asynchronous execution.

```
wGlobal    = tf.Variable(wG_init, dtype=tf.float64)
gradAccum  = tf.Variable(tf.zeros_like(wG_init), dtype=tf.float64)
asynchrony = tf.placeholder(tf.int32, shape=[])
gradQueue  = tf.FIFOQueue(maxQSize,
            dtypes=[tf.int32, tf.int32, tf.float64],
            shapes=[[], [], wGlobal.get_shape()])
...
submitGrad = gradQueue.enqueue([myIndex, self.ts_i, self.grad_i])
storeGrad  = tf.assign_add(gradAccum, gradQueue.dequeue())
avgGrad    = tf.divide(gradAccum, tf.to_double(asynchrony))
update_wG  = tf.assign_add(wGlobal, learnRate * f_m(avgGrad))
applyGrad  = [tf.assign_add(self.tsGlobal, 1), update_wG]
```

Listing 4.2: Global Graph for Gradient Processing

The essential, core tensor objects used for gradient processing are defined in the first part of this listing. This includes the global parameters, w_G, ∇_A, A, and the queue to receive update locally processed gradients from the workers. The workers each submit gradients, ∇_i with the origin timestamp corresponding to the start of local optimization. The master's basic logic for processing is shown in final four lines and includes the optimization function, f_m, which could be AdaGrad, Adam, or another optimizer technique. We also note that these operations correspond to the global components of the graph design in Figures 4.6 and 4.7.

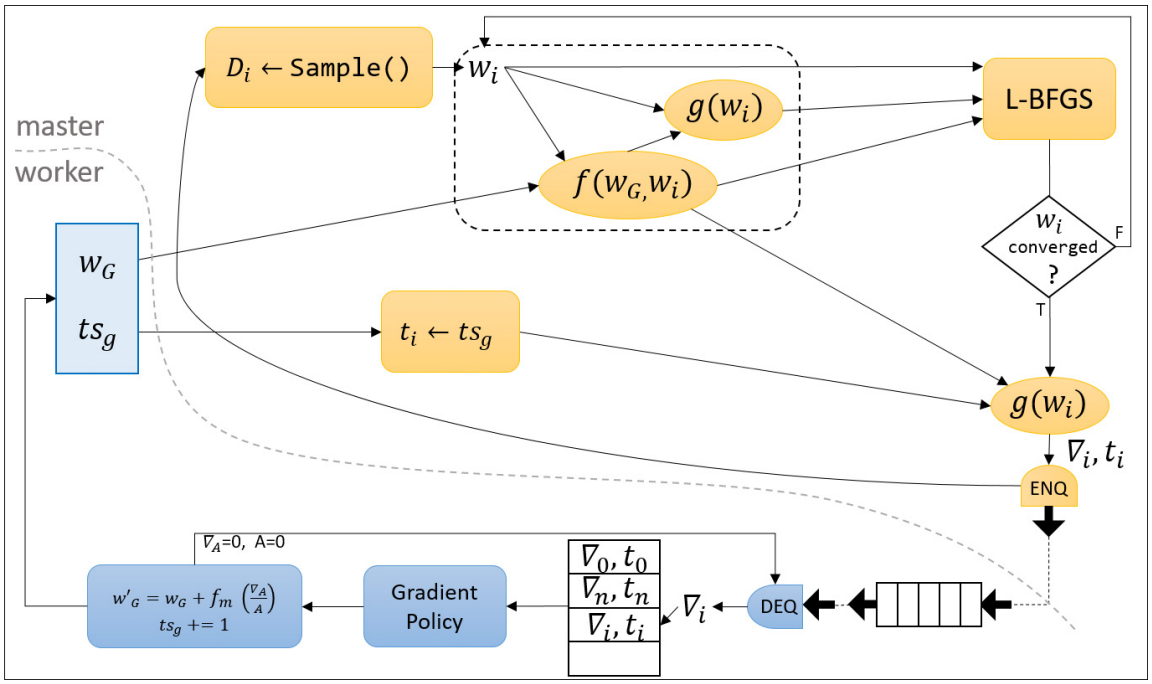


Figure 4.7: Asynchronous Training. Just as the synchronized version, local execution is depicted in yellow and global operation are in blue. The light blue box shows the shared global variables which are written by the master and read by the workers.

Asynchronous Execution

In the asynchronous version, shown in figure 4.7, workers perform the same basic processes as stated above. They each sample patient(s), execute local optimization and output a global gradient. The critical difference among the worker's logic is the lack of a wait step in the asynchronous variant. Instead of blocking on the next timestep parameter update, $w_{G_{(t+1)}}$ to begin local execution, workers, instead, read in the current global parameters, $w_{G(t)}$ immediately. Each worker retains the value of t as the source timestep in global gradient submission. This value can be used to calculate gradient delay, or staleness, τ, at the master supporting adaptive decisions regarding gradient pruning or parameter updating.

TensorFlow provides staleness factoring as part of its optimization techniques. However, its application for staleness only supports pruning strategies whereby gradients with a staleness above a predetermined threshold are discarded. In addition, gradient application is performed individually

on a first in, first out basis where gradients are applied in the order in which they are received at the master. We extend this range of synchronizing primitives by demonstrating asynchronous policies in distributed machine learning. Specifically, we show how the master can leverage measurable data, such as data skew and staleness, to influence prioritization for gradient calculations.

For global optimization and gradient application, we retain an asynchony setting, A, which serves as a tuning lever between more or less asynchrony. Lower values denote more asynchrony with more incremental updates to global parameters; this results in more diversity among the global views for the parameters, w_G, at all the workers. Higher values for A, in contrast, provide a less diversity among these views and we claim this is less asynchrony. Our results have shown that high asynchrony can lead to better overall performance; however, it also can have a higher variance.

Applying a more sophisticated approach, we show how the master can organize received gradients and execute additional preprocessing before updating parameters. Specifically, we show how the master can select and prune updates. We implemented this variant using a priority queue consisting of gradients sorted in two different manners: one is by the timestep from which they were originally calculated, the second is by the gradient similarity to global progress which we describe in more detail below. We have implemented the following asynchronous gradient application strategies:

1. *First-In, First-Out (FIFO).* FIFO is the most basic and commonly applied asynchronous technique. Requiring very little overhead, the master applies each gradient in the order in which it was received. By applying every gradient to the the global parameters without any aggregation, the system implements gradient optimization as a standard parameter server model.

2. *Least Stale First (LSF).* This strategy applies gradients in the most recent order based on global timestep when the worker initiated local optimization. Given that the staleness for a gradient, τ is the difference between the timestep associated with the calculated gradient and the actual global timestep at the master when received, we retain a sorted priority queue for each gradient received based on τ. To apply at each global step, the master pops the top-A

gradients based on the level of asynchrony. As an alternative, we also have optionally explored, *Most Stale First (MSF)*. The compliment to LSF, this strategy applies gradients in the order in which they were calculated based on the global timestep and staleness factor, τ.

3. *Cosine Similarity (COS)*. We apply an alternative strategy to selectively apply gradients based on how similar each is to the global progress. Using the cosine similarity function, we compute cosine distance between a given worker's gradient, w_i and the global progress:

$$1 - \frac{w_i \cdot (w_{G(t)} - w_{G_0})}{\|w_i\|_2 \|(w_{G(t)} - w_{G_0})\|_2}$$

We define global progress at a timestep, t, as the difference between the parameters at that step and an initial value for parameters. We include an initial bootstrapping period and set w_{G_0} to a parameters after this initialization period (for simplicity, we fix the initial parameters to the values after the first 10 updates and found this improved performance for cosine calculation). Using the cosine distance, we continually apply gradients which are most in-line with the global progress (see Figure 4.8); however, this technique can also apply the least similar gradients as well or alternate between. This is akin to an adaptive updating momentum factor, but instead of factoring in the momentum, we selectively apply gradients.

To prioritize in these alternate approaches, the master waits until a set number of gradients are received, selecting the top A-gradients to apply. As an additional tunable parameter, we can also determine how long the master waits until it makes a gradient decision; this is affected by controlling the minimum queue size. An abstract primitive technique, we can let the master defer a control decision in order to acquire more candidates from which to select for prioritization as part of the update calculation. Specifically, we state the master will aggregate the top A gradients from among a pool of $A + k$ gradients. A larger candidate pool can provide a better gradient step, but it can also preclude model progression by delaying updates to global parameters.

We note that both the LSF and the COS approaches represent an adaptive technique to

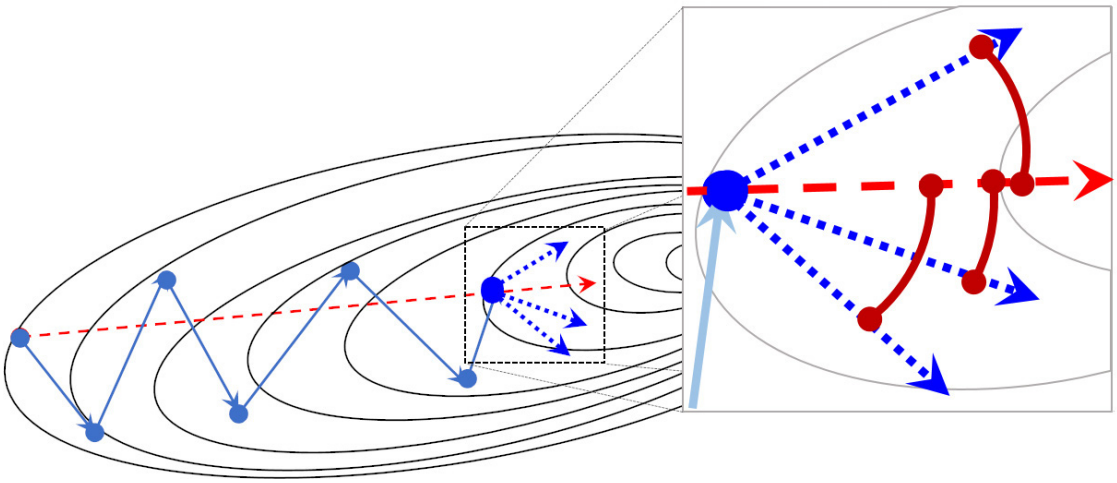


Figure 4.8: Cosine Similarity gradient selection policy. The zoomed in view shows a decision point for the master server which must prioritize gradients. Cosine similarity calculation of each gradient (in blue) to the global mean descent (in red) provides a comparison score to select gradients.

affect the global outcome. This adaptive strategy connects back to previous work on adaptive control data exploration from a systematic perspective.

Once calculated, the master updates the shared global parameters and increments it global timestep counter. By applying highest priority gradients and pruning those that are stale or lower priority, the master ensures that the global updates are applied with more current information. The drawback of this technique is the lost work: each discarded gradient corresponds to a computational resource which did not contribute to model progression.

In addition, we identify dynamic adaptation of synchronization parameters as a means to dynamically steer optimization toward a more efficient or a more effective path. This includes factoring each gradient by a weighted distribution, such as inverse data skew, to affect the global optimization. We implemented this using real time data skew, but found this had little impact on overall performance, given that we perform a load balanced sharding algorithm prior to training (see Section 4.3.5). In addition, we all employ momentum in an adaptive fashion to improve convergence time (Section 4.3.3).

4.3.2 Local Optimization

Solving for the local parameter, w_i, we employ a limited memory Broyden - Fletcher - Goldfarb - Shanno (L-BFGS) Algorithm. This solution iteratively identifies a new parameter, w' and calculates a objective loss, $f(w')$ and gradient $g(w')$, until a termination condition is met. While Tensorflow provides out-of-the box optimization operators, such as Gradient Descent and Adagrad, these performed very poorly on the semi-isolated model for LMC. We have thus employed both a SciPy optimization and a custom developed operator built using the TensorFlow OpKernal module.

The first local optimization strategy employs the SciPy minimize package within Python. This solution, derived from the GPFlow module[132] wraps a Python callable around a TensorFlow session call which passes into the SciPy minimize function with a set of solver options. This technique introduces significant overhead for each step of the optimization algorithm as the underlying parameters are casted between a TensorFlow "tensor," backed by the Eigen unsupported Tensor package, and the Python numpy, ndarray object. While both are statically allocated memory objects for dense operations, they not only require internal wrapping overhead when passed back and forth, but each of the two programming interfaces implement differing memory management protocols. Specifically, numpy statically allocates a fixed block of memory (which is not necessarily contiguous) for each ndarray upon creation. Numpy operations are designed to leverage this static allocation supporting lower level, vectorized operations. In contrast, TensorFlow may reallocate memory for objects during execution supporting other optimization techniques in a multithreaded environment. This is especially apparent in the distributed TensorFlow adaptation where variables reside on a separate systems.

To improve the efficiency, we identified the need to implement an optimization solution within a *single* TensorFlow session call. Execution control to perform local optimization is conducted via a co-routine programming design pattern. The two routines which execute concurrently are the Tensorflow graph, operating within a session, and an extension of the C++ implemented L-BFGS.[133] We implement the co-routine pattern using two lock-free threads which alternate control: one blocks

while the other executes. We originally designed the solution using a single mutual exclusive (mutex) lock whereby each thread obtained a lock prior to execution; however, we found this was not necessary and only added overhead.

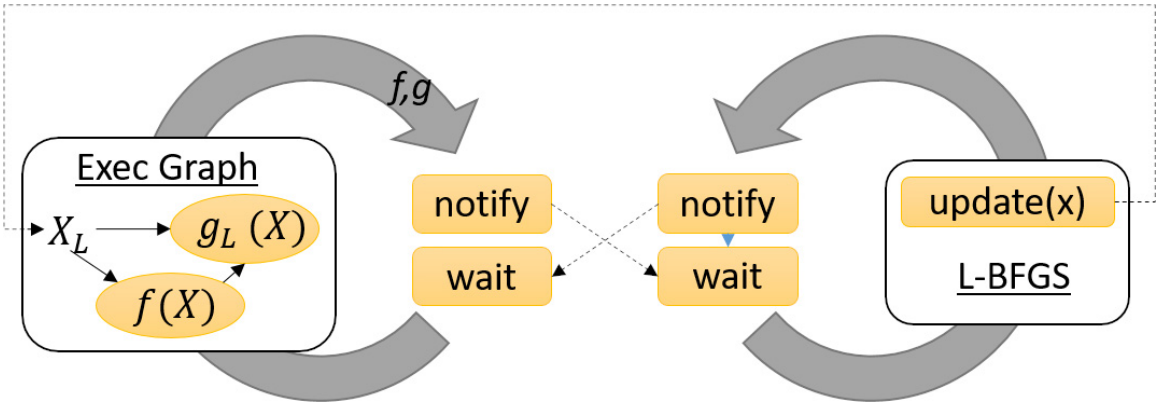


Figure 4.9: Co-routine design: On the left, the main executing thread operates within a TF while loop context calculating tensors for loss, f, and gradient, g. On the right, the L-BFGS solver updates input parameter, x, notifies the main thread and wait for updated tensors.

As shown in Figure 4.9, the two threads operate in a lockfree mutually exclusive execution pattern. On the left, the main TensorFlow thread which calls the custom solver operator, performs one pass forward of the graph calculating both the loss function, f, and the gradient, g, over the local parameters, w_i. Because we have defined the entire likelihood function and local gradient calculation within a TensorFlow `while_loop` context, the engine will re-run the graph using updated tensor values for the input variable, w_i. When the graph invokes the solver operator, it passes the updated, f, g values as tensors to the optimizer thread, shown on the right, and blocks execution. The solver thread continues its execution where it left off in the L-BFGS algorithm until it reaches a request for a new f and g using a new w'. Once requested, the solver thread notifies to main thread to rerun the graph and it blocks execution until the graph returns with new values for f and g.

We also provide a profiled view of graph operations taking place for one iteration of local optimization. Shown in Figure 4.10, we highlight, in red, the major components of execution for the LMC Algorithm. This includes overhead for flow control, the index and tensor processing operator,

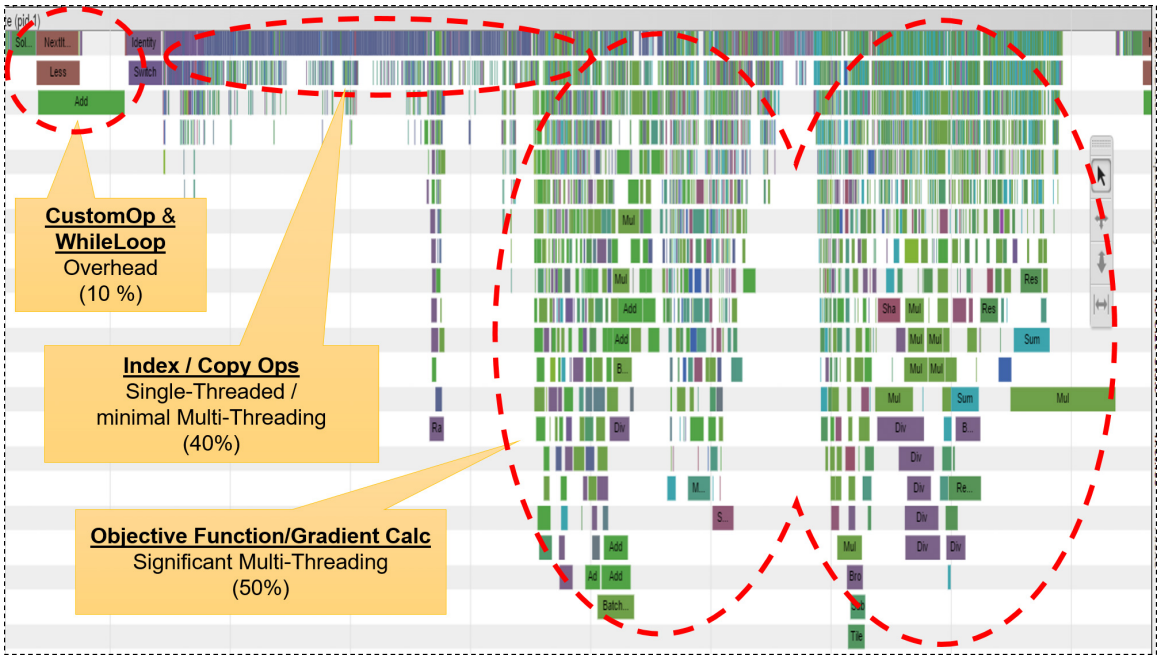


Figure 4.10: TensorFlow profile of the custom operator depicting one loop iteration over a 50 millisecond time period (X-Axis). Each row corresponds to a thread of execution. Green/Purple nodes are individual operators in the TensorFlow graph. *Top left:* Loop control overhead for the Custom operator and while context. *Top:* Many, shorter reshape, slice, and indexing operators which have very limited multi-threading support. *Center and Right:* Operators performing the objective function and the gradient calculation. Note the barrier in between enforcing serial executing of these two sub-graphs.

as well as the objective function, $f()$ and the gradient calculation $g()$. This profiling reveals how the custom operator consumes only a smaller portion of total execution time (depicted as the X-axis) while the GPFlow operators to perform index and reshaping on tensors consume a significantly larger portion. Because of the limited mutli-threading support for these operations, optimization is limited. This contrasts to the heavily optimized TensorFlow operators to calculate the objective function and gradient on the right half of the image.

For additional implementation details on the customized operator, see Appendix A.

4.3.3 Adaptive Strategies

Integrated within the model design, we include data-driven, dynamic capabilities in the machine learning process with the goal of overcoming systematic deficiencies and improving algorithmic performance. Specifically, we have implemented several techniques to steer training outcomes toward (1) a better objective state, such as producing a more accurate machine learning model, or (2) a converged system state using less resources, such as less time, or, in some cases, both of these two objectives. We describe these adaptive techniques below.

Adaptive Gradient Norm

The first is an adaptive check on individual gradient magnitudes. We observed a distinct difference in data distributions between real world evidence (RWE), such as real time data input from hospital metrics and staff workers, as compared to fabricated benchmarking data, such as the Mimic II dataset.[134] Aside from the many known benefits of using RWE, most notably the direct application to real world demands,[135] RWE captures additional factors among a population, such as outlier cases or human data collection errors which are hard to replicate in simulated data.

To account for such outliers, we apply an adaptive normalization technique to all gradients. For each gradient received at the master, ∇_i, we calculate the L^2-norm, $||\nabla_i|| = \sqrt{\sum {x_j}^2}$, where ∇_i is a vector of size j. We retain a sliding window, W_n^g over all gradients consisting of the previous n gradient norms. Performing simple statical calculations, we clip the magnitude of all gradients to be less than two standard deviations from the average over this window. Thus, for large gradients where $||\nabla_i|| > 2\sigma\overline{W_n^g}$, we scale them using:

$$\nabla_i' = \frac{2\sigma\overline{W_n^g}(\nabla_i)}{||\nabla_i||}$$

Note that we set a max tolerable magnitude of 2-standard deviations from the window mean; however, this could also be an adjustable parameter.

Adaptive Momentum

The learning rate, α is the hyperparameter applied to the gradient update at each training step. It is the adjustment to the stepsize corresponding to the gradient norm. Often a critically tuned value, the learning rate can have significant impact on model accuracy and convergence rate: if set too high the model may converge quickly, but never attain accurate results, if set to low the model may eventually very accurate, but at a cost of exceptionally high resources.

In contrast to setting a static parameter, one can define a heuristic to adjust the learning rate as the algorithm progresses. Known an annealing functions, these simple techniques decay the learning rate over time. Common examples include $\alpha' = \alpha_0/(1 + kt)$, $\alpha' = \alpha_0 e^{-kt}$ where t is the timestep and k is a tunable hyperparameter. We include an annealing function presented in[136] which uses two hyperparameters to define a delay, τ, and a forgetting rate, κ:

$$\alpha_t = (t + \tau)^{-\kappa}$$

More popular strategies for adaptive gradients optimization include AdaGrad,[137] which divides a learning rate by the square root of the sum the L^2-norm of all received gradients and Adam which maintains an exponential moving average of both the gradients and the squared gradients, but adds a decay rate.[138] We employ the AdaGrad technique in both local optimization, prior to executing L-BFGS algorithm, and provide it, along with Adam, as an option for global optimization.

For adaptive techniques, more robust methods should maintain a memory of the gradient direction for previous timesteps. One simple technique is to employ a momentum term when updating parameters. This term retains previous timestep gradient calculations and gets added to the current optimization step:

$$w_{(t+1)} = \alpha \nabla_t w_t - \mu_m \nabla_{(t-1)}$$

A relatively new concept, accounting for implicit momentum within distributed execution

captures the impact of asynchronous systems in machines learning.[139] As more nodes are adding into distributed training, the momentum of learning changes. Thus, when applying gradients asynchronously, one should always account for the implicit momentum in the system. The authors proposed an implicit value for momentum as

$$\mu_m = 1 - \frac{1}{N}$$

where N is the number of nodes in the system. We extend this as a more adaptive approach to

$$\mu_m = 1 - \frac{A}{N}$$

where A is the number of applied gradients, or asynchrony, for a single timestep.

4.3.4 Online Prediction

The training component described in the Section 4.3.1, details the global optimization to train the common components, w_G of the prediction model among all sub-domains. Referring to the tree structure taxonomy of input domains, this equates to training the parameters associated with the root node of the tree. Additional optimization is necessary to train each sub-domain and develop the semi-isolated models. Section 4.3.2, which describe the parameter optimization, is implemented for both training and testing. Once the global optimization phase is converged, or it exceeds a pre-determined number of training iterations, the system conducts localized and isolated testing and evaluation on a training subset of the input data. Decoupled from the global training optimization, testing can occur either in sequence after training or at a later time and it can be distributed in a strict parallel fashion. That is, since the global parameters, w_G are fixed in this stage, each model can train in complete isolation, enabling a linear scaling of resources: to train more models quickly one needs to only add more systems. As a solution, we have developed a separate executable component to conduct online prediction of live data. This capability is designed

to support medical practitioners with real time predictions of the likelihood of sepsis occurring within a future, user-defined time horizon.

However, maintaining prediction capabilities in real time can be a resource efficiency problem whereby data generation exceeds a systems capability to continually train and predict outcomes. To overcome this, the system can apply dynamic selection techniques to prioritize model selection. For example, models whereby predictive outcomes are high should be prioritized over lower scoring models. As applied to personalized healthcare in hospitals: those patients with a higher likelihood of experiencing a sepsis condition should have their personal model updated at a faster rate as compared to routine patients.

4.3.5 Data Sharding

In order to distribute data among the worker nodes, we have implemented a pre-processing data sharding algorithm resulting in a 10x improvement on data allocation. The purpose of this component is to diminish the impact of data skew. which we described in Section 4.2.2, on training in the distributed environment. Specifically, this process creates an set of S shards for the N nodes where $S >> N$. To facilitate allocation of nodes, we assign values to S and N which are powers of 2; specifically, S is 128 and N is between 2 and 64; however, this technique can support any value for N, provided $N < S$. A primary advantage of this methodology to easily distribute shards of data using the modulo function where each node is given the shards corresponding to its index in the cluster of workers. For example, in a 32-node system with 128 shards, worker #1 processes patients in shards 1, 33, 65, an 97.

The algorithm to create the shards provides a load balancing capability which helps to decrease overall training time. It begins by collecting a count of the number of data elements for each patient in the input dataset and filtering out patients which do not meet user-provided minimal selection filters (e.g. hospital length of stay less than six hours). Subsequently, we split the data into positive and negative patients based on outcome labels and perform stratified sampling to identify

the training set, $\{T^+, T^-\}$ and the test set $\{E^+, E^-\}$ for both positive and negative patients. Using the count of the number of data items for each patients, we sort the sets from largest to smallest and conduct a greedy load balancing allocation by assigning each patients to shards whereby

$$s_n = \{t_i | i \ mod \ S == n, \forall t_i \in T^+\} + \{t_j | j \ mod \ S == n, \forall t_j \in T^-\}$$

The impact on using a load balanced sharding methodology produces data allocated more evenly across each node in the system. We show the impact of data distribution on system efficiency in Table 4.1. This captures the variance of execution time among the nodes within a single distributed training session. As a baseline, we show how a random, uniform sampling strategy can result in significant data skew with some nodes continuing execution as long after others have finished. The far right column shows the results after implementing our sharding algorithm. With data more evenly balanced among all the workers, all nodes complete training and testing within a few minutes of each other.

# Nodes	**Random Distribution (Baseline)**	**Load-Balanced Sharding**
4	7.72	0.67
8	13.26	2.95
16	17.67	1.70
32	21.62	3.03

Table 4.1: *Comparison of data skew.* This table shows the difference in variance for training time among the nodes in a given cluster. Each row represents a cluster configuration comparing the baseline distribution technique against our load-balanced sharding technique.

As one final finding on data allocation, we stress that the initial allocation is only an estimate of expected training and testing time. We assumed, initially, that patients with more data would take longer to train and that by evenly dispersing patients' data among the nodes by size, we could achieve more balanced systematic testing, in terms of time. Our hypothesis was upheld and validated for this data domain as demonstrated by the results in Table 4.1; however, other factors, such as data quality, can impact training and testing time. For other domains or datasets, a more

dynamic, adaptive technique may be required to reallocate and rebalance data among nodes during testing and training. The current distributed LMC framework can easily be adapted to include a more dynamic load balancing methodology whereby the master node collects training time metrics for each patient and, as needed, reallocated patients to different workers (e.g. from the *slower* nodes to the *faster* ones) using a partition table or through messaging queues.

4.3.6 Additional Implementation Details

This section covers some additional details on the supporting logic and framework to support the training and testing effort.

Preprocessing. The driver program, contains the entrypoint (the main function) for the LMC training systems. Prior to execution, it processes input parameters, reads source data, creates the shared global cluster, and builds the TF graphs. To facilitate a shared state among all modules on a given node, we leverage the TensorFlow `app.flags` settings. This provides a singleton based key value store to manage state. The system reads in environment variables and any command line arguments as input parameters to the program. Some additional settings are also inferred in the logic, such as automatically detecting cloud and containerization environments (AWS/Kubernetes). For input, the system is designed to support data read from both file-based and Postgres sources. An interface package provides an abstraction layer in between the LMC framework and the data source. It is designed to process all extraction, transformation and loading operations prior to executing any Tensorflow based logic.

Input and Output. The system is designed to support both file base input and output as well as database integration. For file based support, we use comma separated value files (.csv) and numpy binary objects (.npy). Usfuel for testing and debugging the file base support allows developers to create local sandbox versions of the distributed program on their local machine. For database integration, we utilize the PostgreSQL database management system. The model is set up to support two dataset sources: the first is the Mimic II dataset,[134] a research developed data

model, and hospital generated data using the Clarity data model, with data update in real time. We have implemented an abstraction layer over the data input to facilitate expansion of data sources to include more than these two as it can be easily extended for to support other formats such as HDFS or key-value stores.

Distributed Cluster. TensorFlow implementations for a distributed cluster follow an architecture consisting of one or more parameter servers supporting any number of "worker" nodes. Although typical deployment utilize parameters servers to manage all data structures operating within an event handler loop, we implement our design using a more traditional master-worker pattern. The master performs synchronization and communication tasks and is capable of adaptive adjustments to system wide parameters, such as the learning rate or alternating between synchronous and asynchronous learning steps. Specifically, within TensorFlow, the master and the workers are implemented as two separate jobs within the cluster.

Cloud Deployment. To facilitate distributed execution, we leveraged the Kubernetes container management system inside Amazon Web Services (AWS). Individually, each node executes as a pod on a single elastic cloud compute (EC2) instance. Coordinated through a Kuberbernetes statefulset object which provides deterministic hostnames, each pod starts a docker image with a set of input parameters, executes a startup script and implements the training algorithm. Using a postgres backend database, workers read in patients data and output metrics, parameters, and other data. Upon completion, the master node collects all workers' local optimzation metric data and calculate aggregate output scores.

Fault Tolerance. Fault tolerance is currently provided in the cloud deployed version through AWS and a Kubernetes statefulset. If a running pod fails, the Kubernetes manager will instantly restart the pod on an available node. If the entire node fails, the AWS cloud provider will start a new node and once the instance is online, the Kubernetes manager will assign the lost pod to that existing node. Internally, we have included design capability supporting fault tolerance within the LMC framework. If a working node fails during training, it will restart and recover lost work. Since

gradient updates are submitted to the master at each interval, global progress is never lost when a worker node fails. Master node failure can be catastrophic, but we have we have designed the system to more readily support failure and provide a brief overview for a more robust solution below.

For local parameter values, the system can currently checkpoint parameters, w_i, to an associated database. When a node recovers, it would restore all values from the previous checkpoint and begin executing local training where it had left off, losing only the updates from the previous checkpoint. There is a resource trade off for fault tolerance: saving intermediate data requires computational resources and storage; thus, these requirement must be balanced against the desire to maximize progress on the machine learning objective. For the master node, the system stores all global updates, w_G at each interval. Since all nodes execute the same program, the system can be configured to enable a worker node to serve as a back up master node, continuing global optimization from the point of failure. This would require additional engineering work and a decision on whether to off-load the back up worker's computation to other nodes or dual-purpose this node as both a worker and a master.

4.4 Experiments

We executed several experiments testing optimization techniques, scalability, cost, and synchronization strategies.

Metrics

- **Time**. To quantify the efficiency for a given setting, we utilize wall-clock time for execution. Unless otherwise specified, time is measured in minutes and is independently collected on each node. It includes the time from program initialization to experiment completion.
- **Accuracy**. We quantify effectiveness for a given technique, strategy, or setting using sensitivity and specificity performance measures. Accuracy is measured by executing optimized

models with a testing dataset to make a boolean prediction for a hazard. Predictions are subsequently compared against the actual results to quantify a true positive rate (TPR) and false positive rate (FPR) as well as true negative and false negative rates. We utilize the following two scoring metrics:

1. Receiver Operating Characteristic Area Under Curve (ROC AUC). This measures the sensitivity and specificity for predictions made by a model against a test set of data. It is factors the true positive rate of all predictions against the false positive rate. A ROC AUC of 1.0 is a perfect prediction score.
2. Positive Predictive Value Area Under Curve (PPV AUC). This measures the recall precision among all positive predictions. Is it defined as the number of true positive outcomes divided by the total number of predictions for which a model returned positive. These score are given the nomenclature, `recall precision`.

Independent vs Aggregate: A prediction is made on each window for each patient. We calculate two sets of scores: one treats each window as a unique prediction score and the other groups each all windows for a patient into a single prediction. These are each labeled `independent` and `aggregate` in the scoring nomenclature.

Vanilla vs Robust. We calculate TPR and FPR on each window to make one set of prediction scores, which we label `vanilla` as a baseline comparable metric for time event prediction. In addition, we include an alternative scoring metric to address the challenges of accurately predicting over windows with limited data input. For these windows, we defer the prediction by not labeling it as positive or negative, although the window is included among the total number of prediction when calculating TPR and FPR. This scoring metrics is labeled with the `population robust` nomenclature.

Given the two aggregation techniques and two scoring standards for both ROC AUC and PPV AUC, we calculate eight total accuracy scores. We highlight Aggregate Population Robust and

Aggregate Population Robust Recall Precision as the two we assess most accurately reflect the domain and application for personalized healthcare. We refer to these as ROC AUC and PPV-TPR for the remainder of this section and these are the scores we show in the summary tables below. We also list the eight metrics explained above:

- *Aggregate Population Robust (ROC AUC)
- *Aggregate Population Robust Recall Precision (PPV-TPR)
- Aggregate Vanilla
- Aggregate Vanilla Recall Precision
- Independent Robust
- Independent Robust Recall Precision
- Independent Vanilla
- Independent Vanilla Recall Precision

Experiment Set Up

All results demonstrated are collected from experiments executing on the Amazon Web Services (AWS) Elastic Cloud Computing (EC2). To facilitate implementation, we leveraged the Kubernetes container management system (v1.5.4) to deploy a set of workers with a single master as part of a Tensowflow (v1.0.1) distributed cluster. A single individual training and testing implementation requires N+1 nodes where N represents the number of worker hosts along with one master host acting as the parameter server. We establish the following experiment hyperparameters for which we separately vary within each experiment:

1. Local Optimization Operator. This is a choice in the local optimization routine to either utilize the Python SciPy minimize operator or the custom developed one which we described in Section 4.3.2.

2. Global Synchronization Policy. We vary the global parameter updating strategy policy by applying the techniques defined in Section 4.3.3. This includes fully synchronous global updates and the techniques defined for asynchronous execution.

3. Amount of Asynchrony. This defines the number of gradients aggregated at each step of global optimization.

4. Number of workers in the given experiment. We vary the number of workers to demonstrate scalability.

5. System Configuration. To highlight cost and value, we vary the number of cores per node.

6. Number of Training Iterations. As indicated in each experiment, we specify the number of training steps executed at each worker. This step refers logic executed in reference to Algorithm 5 from Section 4.3.1.

Settings Nomenclature

We use a short nomenclature to distinguish these variables which denotes the synchronization policy and amount of asynchrony. Where appropriate, we also include other distinguishing annotations. We provide the following examples with explanation to better explain this nomenclature:

`SYNCH` - Synchronous Policy

`FIFO2` - Asynchronous Policy using First-In, First-Out, with asynchrony of 2 gradients per timestep

`COS10` - Cosine Similarity policy with 10 gradients applied at each timestep.

We further describe the system set up, the values at which non-varying hyperparameters are fixed, and any specific nomenclature below with each experiment.

Synchronous Training

Solver	ROC AUC	PPV-TPR	Time
SciPy	0.96	0.68	6:34:36.0
CustOp	0.95	0.71	56:58.8

Asynchronous Training

Solver	ROC AUC	PPV-TPR	Time
SciPy	0.93	0.63	3:07:47.4
CustOp	0.96	0.73	24:36.6

Table 4.2: `SciPy` vs `CustomOp`. Comparison of local optimization operator. *Top*: Compared using synchronous training method. *Bottom*: Comparison of the average of 6-8 iterations of asynchronous training for each operator.

4.4.1 Exp 1: Local Optimization

Question: Does an optimization operator executing within a single TensorFlow graph session improve overall training efficiency without sacrificing model accuracy?

Set Up: This experiment only varies the choice of operator employed for local optimization between the SciPy and CustomOp operator. We test these two using a 32-worker cluster using both synchronous and asynchronous execution policies. For the asynch policy in this section, we aggregate two gradients per global time step using the FIFO method; for this section we refer this as "asynchronous" as we focus on the comparison between the SciPy and the CustomOp, but note that we could have used any of the asynchronous policies. We fix the system configuration using 16 cores per node and train each using 2400 iterations per worker. For source data, we utilize a one-year sample of 73,624 bedded patients from real world evidence (RWE). We filter out short stay patients, or those with hospital length of stay (LOS) less than six hours, reducing the dataset to 29,471 patients. Of these patients, 1,542 have been identified as positive for severe sepsis. From this, we randomly select 60% of the patients as the training source, preserving 40% for testing.

Table 4.2 shows the results comparing the SciPy operator an the CustomOp. Clearly, the CustomOp completes training in much less time with accuracy scores as good as the SciPy using either method of global optimization. We stress that each of these training sessions execute the

same number of global optimization steps and the only difference between the two synchronous versions and the two asynchronous ones are the local operator employed. The customized operator implementation eliminates memory allocation and session management overhead and drastically reduces optimization training time.

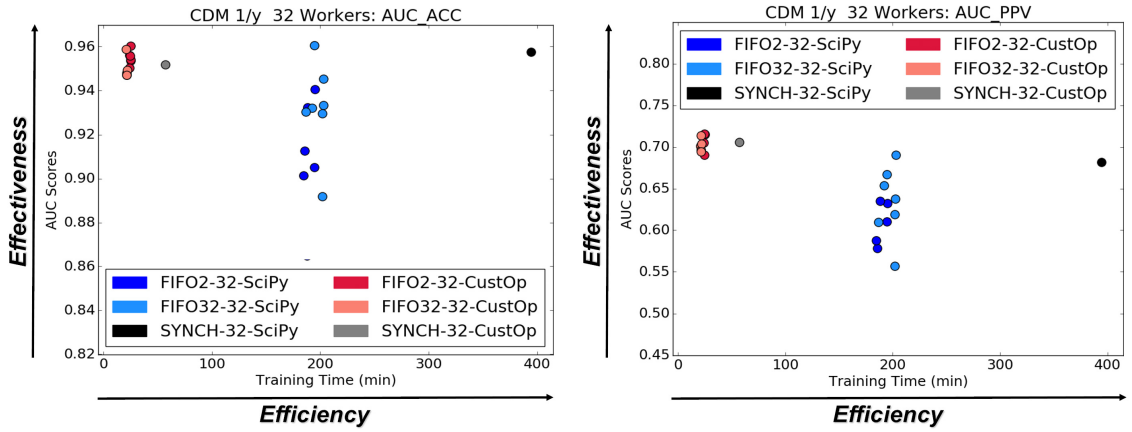


Figure 4.11: Scatter plot comparing Convergence of AUC scores as measured over the duration of training in global timesteps. *Left*: ROC AUC metric. *Right*: PPV-TPR metric.

We show a supplemental set of results comparing CustomOp with SciPy. Figure 4.11, highlights the gains in efficiency with from the CustomOp as opposed to the SciPy variant. We show significant improvement through a reduction in training time by a factor of nearly 8X when comparing the fully consistent, synchronized versions, `SYNCH-32`, for SciyPy (in black) and CustomOp (in gray). In addition, we provide result for asynchronous execution, which will be explored below, highlight the more effective and efficient cluster of red points in the upper left portion of the graph as opposed to the SciPy results in blue along the center of these graphs.

Conclusion: The major takeaway is the impact on convergence for using a single-session optimization operator: CustomOp reduces synchronized training to 1/6 of the time as compared to the SciPy operator and for asynchronous, we see this ratio reduce to 1/8. Accuracy scores are also slightly higher for the customized operator as compared to the SciPy implementation for asynchronous execution, although in the synchronous variant, SciPy performs slightly better. By

keeping control flow within a single run session, we have shown a means to reduce TensorFlow overhead attributed to alternating between Eigen data objects used in TensorFlow and numpy ndarrays in python and recommend this as a design consideration.

4.4.2 Exp 2: Scalability

Question: How does adding working to a training cluster affect overall training time and performance when global parameters are synchronized at each training step?

Set Up: We now use the CustomOp local optimizer to focus on the impact on training when we scale the number of workers in a distributed machine learning cluster. For these experiments, we used the MIMIC II dataset, selecting a sample of 3150 patients which is randomly divided in to a 75%/25% training/testing split. Each patient's training data is further subdivided into at most 10 windows. System configuration is standardized with a common instance platform consisting of 16 cores and 30 GB of RAM per node and vary the number of workers, as indicated, in the cluster. Each experiment executed 1000 global training steps using the synchronous optimization strategy. We measured the time each node spent in a certain phase of execution defined as:

`Initialize`: Includes time to load data, build the local TensorFlow graph, and join the cluster as part of the distributed global session.

`Train`: Time spent executing the computations of the local graph.

`Wait`: Idle time an executing worker is waiting to receive global updated parameters from the master.

`Test`: Portion of execution after global optimization is complete when a workers tests and predicts on its local shard of testing data.

The first key point to highlight is the scaling of wait time which increases as we add worker nodes. However, we found that idle time levels off after 16 workers to about 10.5 minutes - about

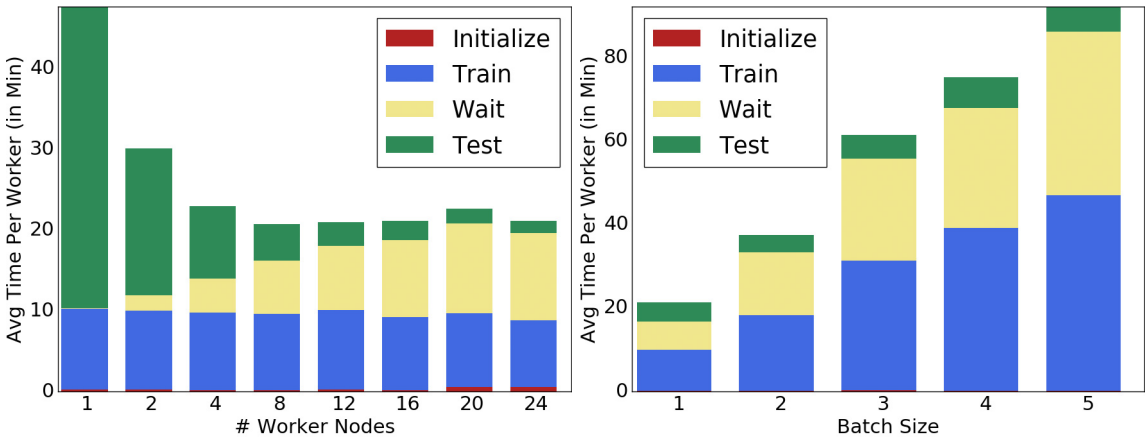
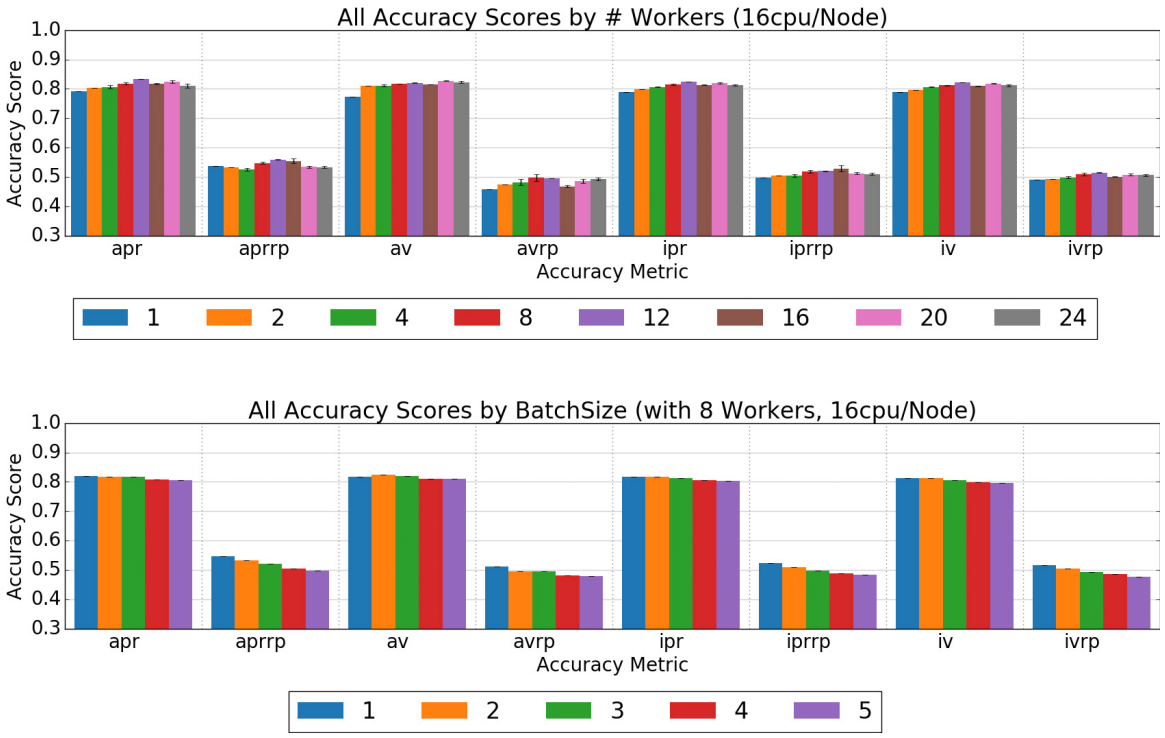


Figure 4.12: Comparison of Execution time for the major training phases. *Left*: Varying the number of workers. *Right*: Varying the batch size of data for each worker.

half of the total execution time - for each training session. As expected, smaller configurations have much less idle time which is a direct factor of the *slowest* worker. Considering that the data is non-homogenous, a single training step can vary depending on the amount and nature of the selected input data. This results in workers' local models converging at significantly different rates within the same global timestep. As data distributes across more nodes with increased workers, the system has a higher probability of training one of these "slower" patients in each training step. Because the synchronized methodology dictates that each step is dependant on the slowest node, if any node selects a "slow" patient, then all workers will wait in idle until that model completes training locally, which, experimentally, we observed stops increasing at around 16 nodes.

The second key point is that accuracy generally improves as we add workers up to the 12-worker configuration. This is shown in Figure 4.13 comparing the accuracy scores for the different variations in this experiment. We highlight that the 12 worker performance scores all resulted in a 2 standard-deviation improvement over all configuration of 8-workers or less. For all metrics, except aggregate pop robust recall precision (aprrp), accuracy improves as we add workers up to 12 Beyond that, we observer fluctuations in the results which could be side effect of a converged system. Since we are executing the same number of iterations in each session at each worker, training may converge



apr	Agg Pop Robust	ipr	Indep Pop Robust
aprrp	Agg Pop Robust Rec Prec	iprrp	Indep Pop Robust Rec Prec
av	Agg Vanilla	iv	Indep Vanilla
avrp	Agg Vanilla Rec Prec	ivrp	Indep Vanilla Rec Prec

Figure 4.13: Comparison of AUC metrics with acronyms provided. For definitions of each, refer to Section 4.4. *Top*: Comparison of scalability by varying the number of workers in the distributed cluster. Note that we observed the best performance with 12 workers which showed at least 2 standard deviation improvement over the baseline 1 worker configuration. *Bottom*: Using a fixed cluster size of 8 workers, the compares adjusting the batchsize applied at each worker in each timestep.

faster with 12 workers. With 16 or more, training had already converged when we stopped execution and we observe the oscillating scores. The one exception, aprrp, has higher recall with fewer workers due to a lower true positive rate (TPR) among the predictions.

The graph also show how increasing workers reduces both the initialization and local learning/prediction phases of the algorithm. With increased data distribution, each worker requires less local computation. The single node execution spends almost all of its time in these two phases whereas the same gross computation is distributed across several nodes as the system scales up. We also note that the total training time is not exactly consistent across the different configurations.

Much of the relatively small variances in time is attributable to external system pressures. This includes network traffic from other executing jobs in the cloud based cluster environment.

In another set of experiments, we increased the batchsize to train multiple patient data within a single training cycle. For these five configuration, we fixed the total number of workers at 8 with 16 cores per worker and varied the amount of data the workers on which each workers trained among the different experiments. We found that batchsize resulted in a linear scaling of both execution time and waiting time. Furthermore, we discovered that performance actually decreased as we increased the batchsize. Thus, our findings have shown that with the current algorithm, varying batchsize is not a promising effort; however, with a much larger scaled problem leveraging batchsize could be a possible means to improve efficiency as long as it does not diminish performance.

Conclusion: We identify two critical findings pertaining to the impact of scaling working on both the wait time, which affect overall training time, and on model accuracy. Given that each worker performed the same amount of computational execution locally, We found that idle time increases as we add workers, but levels off after 16. We also found training to converge when we have 12 workers or more. Thus, for this dataset and training, we identify that the best configuration is 16 workers for efficiency and 12 workers for effectiveness, resulting in an optimal configuration range of 12-16 workers. The main takeaway is that by increasing scalability, one can improve efficiency and effectiveness up to a certain point, after which adding additional workers does not improve performance.

4.4.3 Exp 3: Synchronous vs Asynchronous

Question: Can an asynchronous global methodology for training semi-isolated models demonstrate comparable accuracy results at a fraction of the time as compared to synchronous execution?

Set Up: Now that we have demonstrated the impact of scaling workers in a distributed

cluster, we turn our attention to compare synchronous against asynchronous execution. Based on our findings from the previous experiment, we will fix the total number of workers in the cluster at 12 and compare the two strategies. Each experiment trains for an average of 2500 iterations per worker for a total of $12x2500 = 30,000$ global gradients. Because of the non-deterministic nature of asynchronous execution, some workers contribute more work than others, however, by fixing the aggregate number of work, we can properly compare synchronous against asynchronous execution.

We utilize the same 1-year dataset described in experiment #1 above, but use a subsample of the testing set consisting of 10% of the total source population (as opposed to 40%) while retaining the same ratio of positive to negative patients. We found the subsampling has no impact on our experiment and allows us to more efficiently focus on the comparison of synchronous and asynchronous execution. As the global optimization method, we employ AdaGrad with a fixed learning rate of 0.025 using the CustomOp for local optimization.

For this experiment, we compare synchronized gradient updates against the baseline First-In, First-Out (FIFO) asycnhronous technique where gradients are applied in the order at which they are received at the master. We also vary the amount of asynchrony, A, in each experiment at intervals of 1, 2, 4, 6, and 12 to explore the impact of aggregating gradients at different rates.

Method	**ROC AUC**	**PPV-TPR**	**Training Time**
SYNCH	0.913	0.586	53.19
FIFO1	0.912	0.600	26.93
FIFO2	0.909	0.598	26.91
FIFO6	0.909	0.601	27.14
FIFO12	0.908	0.622	27.05

Table 4.3: Synchronized vs Asynchronous results. Note that time includes *only* the training phase and is measures in minutes.

We show the summarized results in Table 4.3 with the accuracy graphs in Figure 4.14. In addition, we captured the accuracy over time for these experiments in Figure 4.15. For efficiency, we clearly see that asynchronous execution is must faster than synchronous. In fact, all of the 12-worker asynchronous experiments in this section completed at about the 27 minute mark. As highlighted in

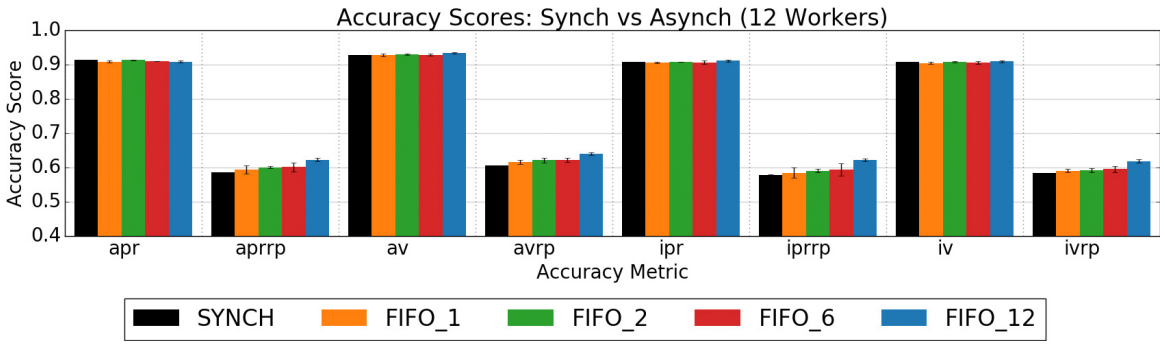


Figure 4.14: Comparison of AUC metrics for synchronized execution (in black) and asynchronous using a FIFO technique. We vary gradient aggregation among 1, 2, 6 and 12 per global timestep.

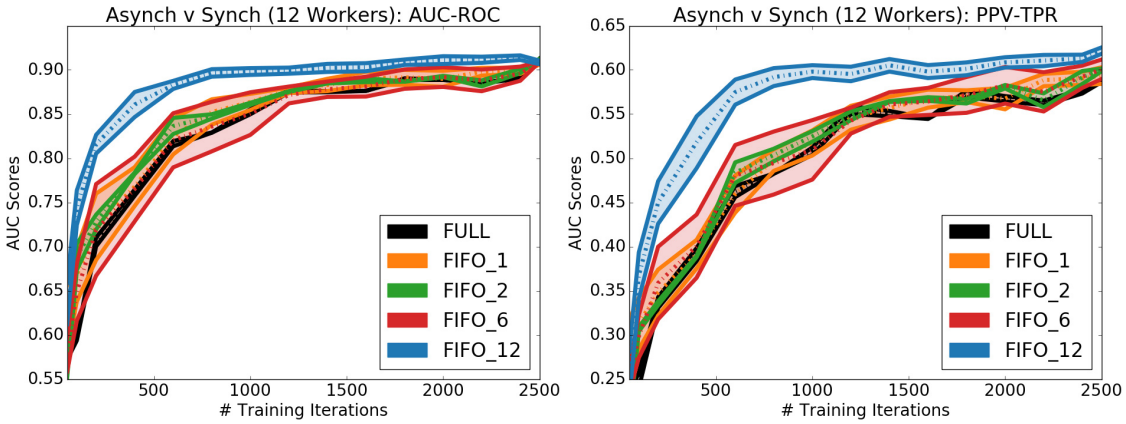


Figure 4.15: Convergence of AUC scores as measured over the duration of training in global timesteps. *Left*: ROC AUC metric. *Right*: PPV-TPR metric.

the previous experiment, the difference is reflected in the idle time workers spend waiting on updated global parameters at each step. By eliminating idle time in the distributed cluster, we reduce the training time by one half.

The second aspect to explore is how asynchrony impacts training effectiveness. Overall ROC AUC results are fairly consistent among all variants (`apr, av, ipr` and `iv`). Synchronous has the highest score; however, is not a statically significant difference between that method and the others. However, for the PPV-TPR scores (`aprrp, avrp, iprrp` and `ivrp`), we observe improvements by as much as 2 standard deviations with especially higher precision recall with asynchronous execution as compared to synchronous. In controlling asynchrony, we find FIFO-12 to be the most

effective and efficient: it provides more consistency among the workers at each step, but capitalizes on continuous execution by precluding idle time due to its asynchronous method.

Highlighting the accuracy score over the duration of training, observe that FIFO-12 actually achieves better performance in less time, reaching nearly 0.90 AUC-ROC within 1000 iterations (see Figure 4.15), which equates to approximately 10 minutes. While we also observe slightly less difference among the other variants.

Sync vs Async (32 Worker Configuration)

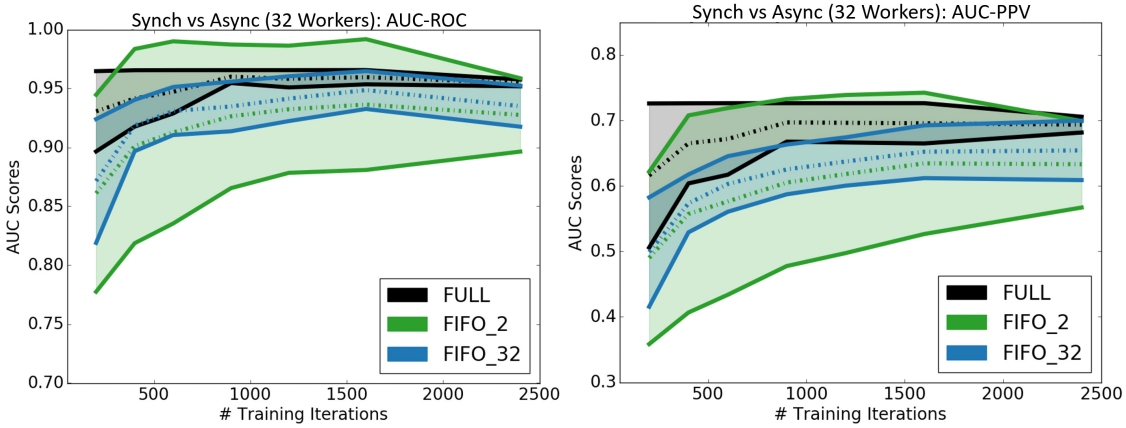


Figure 4.16: Convergence of AUC scores as measured over the duration of training in global timesteps for 32 worker configurations. *Left*: ROC AUC metric. *Right*: PPV-TPR metric.

As reinforcement of the results from the 12 worker configuration, we conducted similar experiments using a 32-worker set up. We compare the fully synchronized coordination policy with two asynchronous variations: `FIFO_2` and `FIFO_32` where we set the asynchrony value to 2 and 32 respectively. As shown in the ribbon plot in Figure 4.16, we found that providing more consistency (`FULL`, in black)) produces the best overall accurate results. Furthermore, a stronger consistent policy of aggregating more gradients per timestep (`FIFO_32`, in blue) also perform slightly better than aggregating fewer gradients. By progressing the optimization with more updates, we also observed higher variance in the outcome as depicted with the much wider ribbon band of convergence (`FIFO_2`, in green).

Sync vs Async (1-Month Dataset)

We provide additional evidence of the impacts identified with asynchronous execution using a 1-month dataset from real world evidence of hospitalized patients. For these experiments, we use the SciPy operator and a system configuration with 16 cores per node. We fix the total number of local updates applied at 3200 and vary the number of nodes as well as the number of global updates executed as each node. We compare synchronous training using 8, 16, 24, and 32 workers against the basic FIFO asynchronous policy using 8 and 32 workers. In addition, we provide two variations of asynchrony for each configuration. The one month dataset contains 6088 bedded patients, split 60% training, 40% testing, with each patient's training data further subdivided into at most 10 windows.

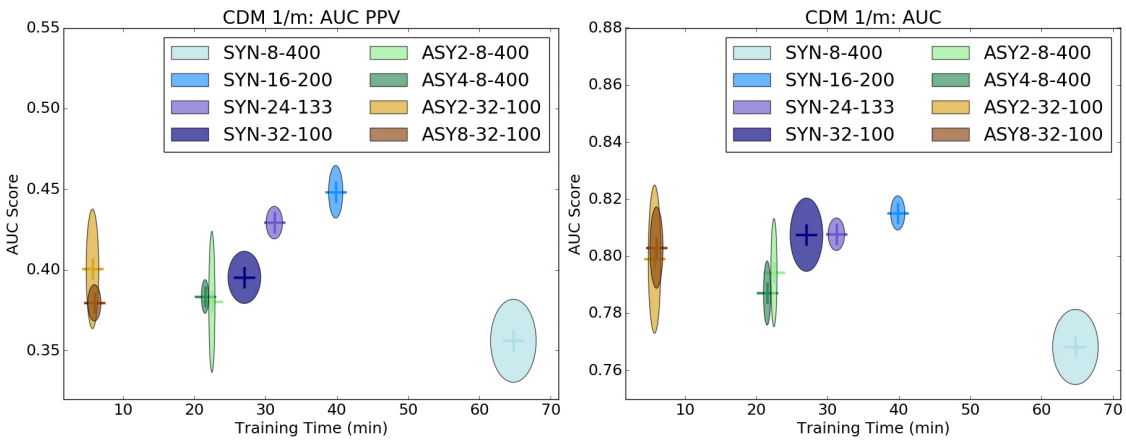


Figure 4.17: Model performance for AUC scores on CDM 1-Month dataset using accuracy score (left) and recall precision (right). Each ellipse encompasses a min/max of data points for the given training session with the center point denoting the average.

Conclusion: These result also clearly show the reduction in overall training time between asynchronous execution which eliminates all idle time which we highlighted in the previous two sections. We see very little, if any, drop off in training effectiveness and, thus, argue that training asynchronously is the preferable option for distributed machine learning over semi-isolated models. We also note that adjusting the amount of asynchrony, as we have done by controlling the number of gradients aggregated at each global timestep, can serve as a tuning hyperparameter for model

training.

Here we showed how asynchronous can perform better than synchronous execution and that we can affect training outcomes by adjusting the level of asynchony. Although this technique applies a simple FIFO methodology, combining asynchrony with adaptive techniques can be more effective. We now explore more adaptive approaches and demonstrate how they can improve results above and beyond the effectiveness we have already seen.

4.4.4 Exp 4: Asynchronous Control

Question: Can we apply an adaptive technique to an asynchronous execution policy and improve on model training performance?

Set Up: We use the same set up as in the previous experiment with 12 workers executing the same 2500 iterations and use the same training and testing dataset. In this case, we are only varying asynchronous global optimization methods. Specifically, we compare the FIFO strategy against the least stale first (LSF) and the cosine similarity (COS) as well as showing the impact of adaptive momentum which we described above.

For these results, we focus solely on effectiveness of training with the AUC scores. We begin with the impact of the global optimizater applied at the master. We show how our adaptive implicit momentum technique compares against Adam and AdaGrad using both low and high value set for the asynchrony tuning we described in the previous section. Specifically, we compare FIFO2 as the lower setting and FIFO12 as the higher setting.

We observe that adaptive implicit momentum improves on the AdaGrad optimizer we had employed in previous experiments. This technique is comparable to the Adam optimizer, even out-performing with a larger asychrony value.

We complete the comparison of asynchrony by showing how an adaptive approach to gradient application can improve the global training objective. We make one small change in this

Asynch Method	Optimizer	ROC AUC	PPV-TPR
FIFO2	ADAGRAD	0.909	0.598
FIFO2	ADAGRAD w/Momemtum	0.914	0.595
FIFO2	ADAM	0.917	0.620
FIFO12	ADAGRAD	0.903	0.622
FIFO12	ADAGRAD w/Momemtum	0.915	0.632
FIFO12	ADAM	0.915	0.616

Table 4.4: Asynchronous results comparing optimizer and adaptive momentum techniques using both a lower and higher setting for asynchrony.

experiment set up by using FIFO10 instead of FIFO12 as the higher setting. This ensures we keep asunchrony less than the total number of workers to enable the adaptive strategies to apply (i.e. if $A = N$, then our forumla for adaptive momentum, $1 - A/N$, is canceled out). In both cases, we set $k = 2$ and let the master wait until it receives $A + 2$ total gradients before making a decision on gradient application.

Using FIFO as the baseline, we compare it against both the least stale first (LSF) and the cosine similarity (COS) strategies. For these experiments, we fix the global optimizer by using ADAM, considered among the best optimizer choices, in order to demonstrate improvement above the current state-of-the-art in machine learning optimization.

Asynch Method	ROC AUC	PPV-TPR
FIFO2	0.909	0.598
LSF2	0.927	0.623
COS2	0.930	0.658
FIFO10	0.908	0.622
LSF10	0.920	0.617
COS10	0.923	0.636

Table 4.5: Asynchronous methods comparing gradient policies. Compared against First-In, First-Out (FIFO), the results show the added benefit of using adaptive strategy for steering global training toward a common machine learning objective. Both the Least Stale First (LSF) method and the Cosine Similarity (COS) method return better outcomes with the COS2 showing the best of all techniques.

The results in Table 4.5 and Figure 4.18, show how the adaptive techniques of LSF and COS can further improve on the results obtained previously. Because the distributed nature of the system implies a non-deterministic ordering of gradients, the global parameters may be updated differently in

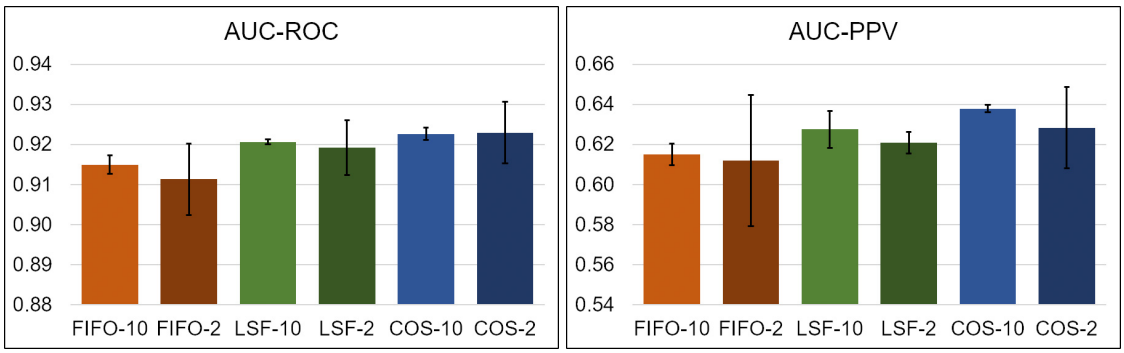


Figure 4.18: Performance scores for the gradient aggregation policy techniques of FIFO, LSF, and COS. *Left*: ROC AUC metric. *Right*: PPV-TPR metric.

different executions. We can, however, control this ordering and apply a more productive capability. Using a adaptively controlled methodology, we can steer the training objective toward a better goal, or, alternatively, toward the same goal but at a quicker pace. By letting a master organize and apply gradients in a selective and intelligent fashion, the system, as whole, can converge to a better result. We also note that we did not further explore internal hyperparameter options, such as changing the master's delay before making a decision. We leave additional fine tuning for these adaptive strategies as an open direction for future work.

Conclusion: Focusing on the effectiveness of training, we observed that adaptive approaches to global optimization improve on traditional machine learning techniques. Specifically, we find that a Cosine Similarity technique, whereby gradients are applied in the order in which they are most similar to global progress, has proven to be the top performing strategy of all with which we have experimented. This demonstrates how an asynchronous primitive, in this case controlling an adaptive gradient application, can dynamically affect distributed training, providing a better balance between resource efficiency and training effectiveness. This highlights a future direction in distributed machine learning systems which should look to incorporate such capabilities into existing architectures.

4.4.5 Exp 5: Cost

Question: We conclude our experiments by re-visiting the effectiveness vs efficiency paradigm by asking, which configuration is preferable to find the most cost benefit solution for machine learning training: fewer nodes with more cores per nodes or many *cheaper* nodes each with a limited number of cores?

Set Up: These experiments utilize the same MIMIC II dataset and the fully synchronized global optimization strategy as described in experiment #1. However, in this case, we fix the aggregate number of cores (as best as possible) among the different configurations listed, but allow the number of workers and the number of cores per worker to vary for each of the five configurations we tested. We note that we fix this at 128 cores, except for 4x36 which has 144 cores. This is due to Amazon instance availability which does not provide a 32-core system at the comparable processor model. The nomenclature used reads `#workers` X `#coresPerWorker`.

We note that, in general, the ratio of cost to cores is about the same between the variants: for example, a 16-core EC2 instance costs approximately $0.80 per hour and a 2-core instance costs approximately $0.10 per hour. Granted the prices vary slightly based on specific instance type and region, however, in general the costs are comparable per hour. Thus, overall efficiency to train a model can be measured by the total training time per unit. We also do not include any added benefit for leveraging spot price instances. We did, however, utilize spot pricing to help reduce overall experiment costs, but are including the listed prices for on-demand EC2 instances. While spot prices can help drive down costs, they are applied evenly across all systems, and, thus had no impact on the results.

The results in Table 4.6 show that for synchronized training, a configuration of 8 workers with 16 cores was the best balanced mix. As we increase the number of workers, we increase the amount of idle time among all the executing nodes since each must wait on the slowest worker to

# Workers	CPUs	Hourly Rate	Time (hrs)	ROC AUC	Cost
4	36	$1.675	0.53	0.808	$3.73
8	16	$0.838	0.36	0.818	$2.60
16	8	$0.419	0.60	0.828	$4.68
32	4	$0.209	1.04	0.807	$7.71
64	2	$0.105	2.38	0.788	$17.22

Table 4.6: Tabular comparison of cloud costs for system configurations. Total cost factors in the number of nodes, cost per unit, and the total time required for machine learning. In general, the 16 worker, 8-core configuration is the most valuable at comparable rate of $2.60. Costs based on Amazon EC2 On-Demand prices as of July 3, 2017.[140]

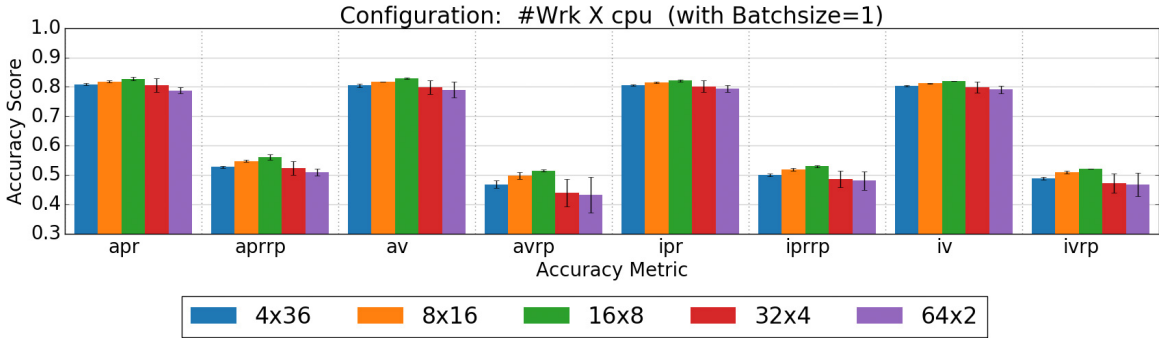


Figure 4.19: Comparison of accuracy metrics using various configuration of workers X cores per worker. For metric acronyms see Figure 4.13

progress the algorithm. This is shown in configuration 64x2 which used the least expensive EC2 instance type (m4.large), but took three times as long as the more robust nodes. On the other extreme, the 4x36 configuration had much less idle time, but each node also had must more data to process. We also highlight that the most effective configuration were the 16x8 workers as shown in Figure 4.19. Reinforcing the findings from our second experiment, we see that training performs optimally when we have 16 workers.

Conclusion: The key takeaway is that neither extreme between (a) many *cheaper* computing nodes and (b) fewer more *powerful* nodes is preferable; a balance between the two is more ideal. For the given dataset, we found that the 8x16 configuration produced the most cost effective measure; however, the 16-worker configuration returned the highest accuracy. The optimal cost benefit lies somewhere in between these two extremes and that the exact fit depends on the dataset to be

trained as well as any other hyperparameters. We recommend that users determine the best course of action when choosing a cluster configuration based on priorities. To maximize cost savings and achieve the best possible outcome, do not go too low in cluster configuration: a relatively smaller cluster with better processing power per node is the best choice. To maximize model performance, use sightly more nodes and, if necessary to remain within a fixed budget, use slightly fewer cores per nodes.

4.5 Open Directions and Takeaways

We foresee future work in this effort on three directions. The first is an natural extension of the techniques we have developed. This includes additional hyperparameter adjustments and fine tuning to gain a better understanding of how these adaptive approaches affect the global/local training pattern. We believe we have only begun to explore this space and additionally, more intelligence adaptive approaches are yet to be uncovered. We also describe two other orthogonal efforts:

Hierarchical Personalized Models. In describing the semi-isolated model, we initially stated that this research effort focused on only a global and a local layer. Every individual model shares the same set of global parameters and each had a unique and isolated portion of the model. However, we could develop a taxonomy of sub-domains where each subset in a multi-tiered hierarchy shares a common set of parameters. Training would need to be more involved and require an adjustment to the algorithm as well as scheduling and dispatching of sub-domains in a distributed cluster.

Sampling for Control. As an alternative direction, we could integrate a mediation control loop to better sample sub-domains for training. The current implementation randomly selects input at each step using a naive uniform distribution. However, we could add an additional component to organize and adapt the system and selectively choose models at each step. This direction implies an importance sampling methodolody whereby we could incorporate a lattice-based approach to

organized sub-populations.

We have implemented a distributed version of the linear models of coregionalization algorithm. Our framework provides a two-graph methodology which combines a globally shared model with a locally trained optimization graph implemented at each worker node. A master node updates global parameters in either a synchronous or asynchronous fashion. In conclusion, we feel this effort has enabled training and prediction for personalized medical machine learning models in a more time efficient and practical manner and that it can integrate into real world hospital use.

Chapter 5

Takeaways and Open Directions

We began this thesis with an overview of the current computing landscape. In its description, we identified the modern challenges of interjecting evolving capabilities into structured data management architectures. The specific problem we identified pertained to human-in-the-loop management of data generation and distributed analysis tasks. This human directed oversight inhibits efficient online, iterative data exploration leading to delays in knowledge discovery and decision making. This dilemma has put a strain on systems to maintain pace with data exploration requirements which can result in severe consequences in certain fields and settings, to include military operations or personalized healthcare. Our goal all along was to address this problem through architectural support to reduce the human requirement and shift the burden to a more semi-autonomous capability. We focused on the data and systems components of the exploratory process to enable a human to focus on higher level tasks. In Chapters 2, we showed how a lattice based data organization approach precludes human oversight in pattern matching allowing the system to steer input parameter selection for exploratory tasks. We implemented this concept in Chapter 3 through a serverless framework integrated into scientific simulations in high performance computing. In Chapter 4, we demonstrated adaptive asynchronous primitives in distributed machine learning providing a capability to enable more autonomous hyperparameter tuning.

Throughout, we retained a running theme of improving system efficiency and effectiveness by coordinating efforts of individual tasks with an adaptive control methodology. We addressed the shortcoming of a data exploration workflow hindered by a human-in-the-loop; we overcame this problem by coordinating asynchronous execution through an abstraction mechanism to reduce the human direct involvement in lower lever tasks, such as parameter selection, data filtering and query monitoring. We applied these techniques to two direct use cases of scientific computer simulations in high performance computing and machine learning for personalized healthcare. In both, we showed how to coordinate the efforts of many, shorter independent tasks to collectively reach a common data exploration objective.

As part of the molecular dynamics effort, we identified how the human-in-the-loop was a hindrance to efficiently executing the data exploration task. Traditional scientific simulation workflows entail a bio-physicist to carefully select a set of input parameters for a given experiment. The jobs execute in high performance computing node over the course of several days or even weeks resulting in very large trajectory output data files, typically measured in multiple terabytes. Specifically, we found that a single long running serial simulation for even a small bio-molecular system can take a week to execute and output a file approximately 250 gigabytes in size. As part of the human-in-the-loop process, the scientist manually copies the raw data file(s) out of the HPC cluster to a local resource, which, itself can take an extraordinary amount of time. Given that average internet speed is approximately 7.2 Mbps,[141] transferring this raw trajectory output could take approximately 79 hours. Only the preparatory step, this is followed with further processing and examination on local resources which are typically less capable than the supercomputing nodes. The subsequent task of filtering raw data and pattern matching consumes additional days whereby the scientist uses interactive tools, such as a jupyter notebook to understand shortfalls in the data collection in order to identify a new set of input parameters. This all results in a very prolonged turn-around time to execute a follow up set of experiments and drive the iterative data exploration objective.

Our methodology to integrate an architectural framework was specifically aimed to preclude the human scientist from performing these manual, time-intensive tasks. We demonstrated a capability which completely eliminates the need to copy and filter raw data. We instead injected a more semi-autonomous capability to push analytics closer to data generation and keep the raw data filtering requirement within the supercomputing infrastructure. We also highlight that this framework specifically allows the human-in-the-loop to focus her efforts on conducting off-line analysis over the reduced and filtered data. The scientist can instead become forward-thinking by applying the information gained to synthesize new knowledge and integrate the outcomes into higher level queries focused on solving domain specific problems.

The second use case focused on the integration of dependent tasks within machine learning applied to personalized healthcare. In this case, we identified the parameter tuning efforts as very human intensive ones. While some autonomous hyper-parameter optimization technique exists, they do not address the optimization for multiple, dependent model training. Such applications, found within localized training for personalized healthcare, require additional coordination layers to finely tune consistency requirements at the lower level. This includes a mechanism to determine when models are diverging and need to coordinate for consistency and the ability to apply a more adaptively steering gradient selection policy, such as the least stale first or using a cosine distance function we introduced, on top of an optimization technique. Similar to the lattice which assumed many of the lower level pattern matching tasks, the asynchronous primitives outlined within Chapter 4, demonstrated how we can enable the system assume a more semi-autonomous role in data exploration. The human-in-the-loop, can instead focus on the outcomes of the model, such as addressing the actions to take in response to a positive prediction of the onset of sepsis in a given patient.

Although we targeted applications pertaining to supercomputing and machine learning, the framework and methodologies presented herein are applicable to other domains. Our efforts provide a proof of concept as a means to improve resource utilization for a variety of operating environments, including high performance computing clusters, austere settings, and military battlefield systems. By

demonstrating a significant reduction in resource utilization, we showed how to improve productivity in data exploration. The mediation control loop and sampling methodologies outlined in Chapters 2 and 3 serve as a roadmap: future efforts should seek to drive input sampling via data organization techniques, such as the data lattice. Applying these adaptive controls and synchronization primitives, individual tasks become more productive, as we demonstrated through increased rare event detection. Their collective efforts aggregate to provide greater contribution to a common objective. We have, thus, shown how to focus human efforts and improve data exploration.

We conclude with three open directions for future work in adaptive control for data exploration. For the first, we revisit the application we mentioned in Chapters 3 where the serverless framework could extend to a multi-user system. We frame the research issues pertaining to users, scheduling and fairness policies. Our second direction is an orthogonal one within machine learning. When we initially addressed the analysis tier of data exploration we viewed the underlying tasks as *black box* functions, commonly associated with machine learning. However, what if we sought to open the black box and understand what happens inside as a means to garner information and further improve data driven control of our process? We examine the background behind this concept known as explainability. Finally, we describe how we could apply the feature lattice concept presented in chapter 2 with the multi-model approach in chapter 4 as a bridge to enhance machine learning training and servicing.

5.1 Multi-User Ensemble Management

A long range goal of the molecular dynamics database project is to provide a basic framework to support large scale, multi-user analytical requests for a consolidated research effort. Generalizing beyond scientific simulations, we feel opportunities exist to capitalize on the synergy of combining many user demands into one unifying system. Such an integrated, adaptive system would need to tackle additional challenges, such as scheduling and fairness. We present the follow-

ing design as a foundation for this conceptual system which can extend the work we have already demonstrated.

5.1.1 User Interaction

A larger deployed system which supports multiple users must determine that amount of interaction either required or allowed for its user base. We identify the following potential categories of human interaction:

1. *None:* Closed loop, internal feedback system without any interactive users. Simulation-generated data serves as the input stream to the analysis tier. Analysis task results are fed into the system control which drive decisions on subsequent tasks to launch based on an initially established system-wide objective function.

2. *Read-Only:* Users can ask simple, read-only queries based on generated data. Typical user queries involved asking select/project-like queries on either raw data, local-diffused data, or global data. Should consider integrating a QoS with user queries so users can receive either a guarantee of results or a confidence index on accuracy (or "as-of-time") for the data. System would need to prioritize user requests with on-going system control tasks which are meeting an objective function.

3. *Interactive:* Users can ask both read and write queries. In addition to the factors of read queries, this type of interaction allows the user to impact running processes which could entail anything from stopping analysis to updating the objective function directly.

4. *Crowd Sourced:* A long-term goal of the system. In this case, a crowd of scientific experts can influence the system control decision based on their pattern of queries. There is no direct user-interaction with the system control, but the objective function will change over time to ideally become optimal.

5.1.2 Solutions for Job Scheduling

In a multi-user cluster, the system must make resource decisions among a myriad of competing demands. Specifically, the system must decide how to divide a finite number of computing hours, given multiple users who want answers to queries. While this resource decision is commonly applied throughout systems operations, processes are treated as isolated entities. The critical difference in our system is the interdependence of the tasks which integrate into a common objective. We, thus, ensure an abstraction layer between the users and the actual tasks which can assume a many-to-many relations: one task can support multiple users and likewise one user's query may require multiple tasks to complete.

In our framework, we view part of the system's responsibility as receiving queries transforming them into potential jobs via a sampling strategy. We recognize that an additional declarative interface and query processor would be required. Given this, we foresee the outcome of this step as a probability distribution over a set of potential jobs. Using the distribution, the system can then drive decisions to (1) decide among the users who receive resource prioritization and then (2) which specific job to schedule. Given a finite number of computer hours, N, we identify two potential models to implement this layer:

Allocate and Sample

In this two step process, the system first executes a fair-policy scheduling algorithm to distribute a portion of the computer hours to each user.

	Advantages	Disadvantages
EQ/WD	Simple	Long running users (not near convergence) may never finish
EQ/SL	Simple, Deterministic	Some users may never finish; cannot launch for continuous values
MM/WD	Fair allocation among users	Need to convert CI -¿ discreet # of computing hrs
MM/MM	Fair among uses and jobs	Complex: 2 layers of resource computation
MM/SL	Fair & Deterministic	Some jobs may never run; cannot launch for continuous values

- EQ: **Equal Allocation of resources**: Each user receives $\frac{N}{|U|}$ hours in either a parallel (divide-and-conquer) or a round robin fashion.

- MM: **Max-min algorithm**. Implementation to maximize the minimum amount of resources needed among all users. Define max per user as the largest confidence interval among all sampling variables. CI's would need to be represented in a common unit of measure with users specifying a desired CI and error, (e.g. as states, select high-level data variable X to be discovered (determined, solved, identified) at $CI = 95\%$ within an error of $\Theta = 3\%$). Thus, each user's max is a function applied to that user's confidence interval. This can be represented as follows:

- WD: **Weighted Distribution** Probability distribution over a continuous interval value (with a μ, ρ) or a set of discreet values (s.t. sum is 1)

- SL: **Sort and Launch** Given set of discreet values, sort in descending order and submit each in succession until all resources are consumed. May loop if needed or launch more than one task per value.

```
Fair Scheduler Algorithm:
Given:
  μ_i  →  Calculated variable estimate for variable, V_i
  C_lo, C_hi  →  Confidence Interval
  Θ_i  →  User Defined Error toleration
  R_t  →  Available compute resources (e.g. hrs, nodes) allocated at time, t
  R_D  →  Resources desired to use at time, t (based on external usage)
  R_I  →  Idle Resources
  NOTE:  R_t ≤ R_D

Let:
  Θ*_i = (CI_hi − CI_lo) / μ_i,  ∀i ∈ V_i
```

$\overline{\Theta_t^*} = \frac{\sum \Theta_{i,t}^*}{|V_i|}$, NOTE: $\overline{\Theta^*}$ will converge to $O(n^{\frac{-1}{2}})$
$R_t = R_{t-1} + \left(\frac{1}{\sqrt{t}}k - \overline{\Theta_t^*} \right) * R_{STEP}$, k is a constant
$R_D = R_t + MIN(R_I, R_t)$
$\vec{D} = \{R_D * (\Theta_i^* - \Theta_i)\}, \forall i \leftarrow$ User Demand Vector
(***may want to normalize $\vec{D}$ if $\sum \vec{D} < 1$)

Solve:
Apply Max-Min: $f(\vec{D}, R_t)$

Aggregate and Sample

As an alternative, the system can aggregate all user values into one large global pool. This would change the scheduler's fair scheduling policy from one of a fair-to-user to one of fair-to-jobs, where all user jobs are competing for resources (vice users). Implementation would be a two step process: (1) aggregate all users into one global set of values or probability distribution function (which we note is akin to running the reweight operator which we described in Chapter 2) and then, (2) sample among global set of all values. This may not be advantageous to the previous model, but it does allow for users jobs to be distributed in accordance with a global probability. The only scenario where this could be beneficial is if it could identify overlapping user demands and bias sampling accordingly. Ergo: this method tries to run jobs which may benefit multiple users, promoting synergy in the system.

5.1.3 Allocation and Deallocation

One final part of this research effort which we explored is the manner in which resources are fairly allocation. We also examined the inverse, deallocation, which coincides with the same considerations of long-running queries in database management systems. We provide background material on those two topics since they directly apply to this concept system:

Fairness

For single threaded tasks, the decision is moot: give all resources to the task until completion; however, in a multi-user or resource competitive environment, the scheduler needs to balance user goals to complete a given task/query with resource utilization. Known as a fairness policy, the system aim to ensure each user receives a *fair share* of the resources while maximizes resource utilization. Several techniques have been developed to strike a balance between optimization and

fair use to ensure all users make progress toward a goal and preventing *starvation* - the condition where one or more users is denied any resources. Several naive approaches are baseline comparisons for any resource management policy:

1. *Equal Allocation.* Each user receives the exact same amount of resources. If a cluster has N nodes with K users, then each users receives exactly $\frac{N}{K}$ share of the resources exclusively. A comparable policy, round robin, evenly distributes resources in time (vice space) by iteratively scheduling all N nodes to each user. While this policy is *fair*, it does not optimize resource utilization.

2. *First Come First Serve (FCFS).* Users compete in an aggressive (greedy) fashion for available resources where uses consume everything all available resources they request. While this may optimize resource utilization, it may prevent some users from making progress (e.g. one user can request everything and leave nothing for everyone else).

3. *Shorted Job First (SJF) and Shortest Time-To-Completion First (STFC).* Similar in nature, these two policies identify the ideal *next* user to schedule on a queue. In SJF, the system uses the user's estimated resource requirements while in STFC, the system prioritizes based on which user is expected to complete her execution the soonest.[142] In either case, users with long running jobs or queries which are far from completion may never make progress.

4. *Proportional Scheduler.* As an extension to a basic equal allocation fair scheduler, a proportional scheduler factors in a weight per user for each allocation. Thus, users receive a fair share proportional to their associated weights. Weight can be tied to priority, completion time, or requested resources. If proportions are not fairly distributed (e.g. one user has a significantly large priority or request), starvation for lower-weighted users can still occur.

5. *Max-Min Allocation.* With max-min allocation, the objective of a scheduler is to maximize the minimum required allocation for each user. Allocation is conducted in an iterative fashion whereby each allocation choice is to fairly distribute either the minimum requested resource for each user or a fair amount (whichever is smaller). Formally, if U = vector of user requests (for each user where $u_i > 0$, and N = # of available nodes (or resources), then the scheduler allocates MIN($\frac{N}{|U|}$, MIN(u_i)) each round until all resources are allocated. The main limitation of max-min policies is the dependence on accurate resource requests.

6. *Dominant resource fairness (DRF).* Implemented in the Mesos framework, DRF is a max-min strategy which overcomes the limitation of accurate user resource requests while balancing multiple uses across multiple resources.[143] Ghodsi, et al define four common properties a fair scheduler should consider:

   (a) Sharing Incentive: each user is better off sharing the cluster

   (b) Strategy Proofness: users cannot benefit by lying about resource demands

   (c) Envy-freeness: uses should not prefer another user's allocation (fairness)

   (d) Pareto efficiency: zero-sum of resources (gain to one user is a reduction for another)

   DRF identifies a dominant resource per user and then subsequently selects the user with the smallest dominate share and executes. An an extension the authors implements a weighted DRF which incorporates a weight in the the calculation of each user's dominant share.

7. *Lottery Scheduler.* As an implementation of a weighted proportional scheduler, a lottery scheduler[144] treats the weights as a probability distribution. Every time resources must be allocated, the system distributes individual allocations randomly using the user's weights (demand, priority, request) as the pdf. While this ensures probabilistic fairness given sufficient resources and ensures 100% optimization usage, it does not provide deterministic fairness (e.g. users could be starved of resources if they have a low probability to receive resources). Lottery scheduling can be applied to the local distribution with a individual users' tasks and, thus, provide a global probabilistic resource scheduling solution where each user has competing tasks.

8. *Stride Scheduler.* The deterministic version of a lottery scheduler, a stride scheduler selects tasks using a logical time-stamping system. Each unit of time is a *quantum* and users are given *tickets* to use quantums where the number of tickets is directly proportional to the user's weight (priority, demand, etc..). A user's *pass* is her next earliest available time slot when she can receive resources. The stride algorithm iteratively selects the user with the lowest pass value, allocates resources, and then increments the user's pass by her stride value. The *stride* is inversely proportional to the user's weight; thus, users with a high weight get allocated more frequently. How the inverse function is applied to the calculate the weights determines the relative allocation among the users. Using a ticket analogy provides for extensions such as a global ticketing system which can allocate quantums to individual user tasks, user ticket

inflation/deflation to change priorities without changing other allocation policies, and ticket transfers between users. One potential pitfall is that a super-user could demand a very large portion of the quantum; to overcome this, the authors developed a hierarchical stride schedule which groups users into a tree structure: users are consolidated in a binary tree with leaves representing users and internal nodes containing the minimal pass of all subordinate users, and aggregate stride and ticket values (for all subordinate branches). Tickets are issued recursively down the tree and each node in the allocation path updates its pass accordingly.

Managing Long Running Queries

The determination of how to handle long-running queries is an execution decision. It is a reaction to event in which the system must consider if and hot to interfere with the on-going process acting on behalf of a query. As mentioned above, the system can do anything from nothing to kill. By introducing an external source to the system (a.k.a. user interaction), we must explore how to handle long running queries which may have multiple and possibly conflicting objectives.

The primary concern with long running queries is what do with them if/when they become contentious and/or need to be stopped and/or need to be rescheduled. Queries could be stopped due to a variety of reasons, to include system overload, priority of other tasks, external system maintenance, change of objective function, and others. For simplicity and to bound the issue, it is probably best to explore this decision in HPC for scientific purposes as a factor of query priority with respect to both time and space as opposed to resource contention, fault tolerance, or other reason. For example, a user submits a query and the system must decide if it needs to free up disk resources for intermediate results (space) and if it needs to schedule it immediately or if it can queue it at a more ideal time in conjunction with system convergence tasks whereby it could opportunistically leverage cached intermediate results and minimize overall CPU time to respond to the user. This would have to be intertwined with QoS expectations provided to the user, resource budget constraints within the computing environment, garbage collection policies, I/O to archived data, and the ratio or prioritization of tasks between system convergence and user queries. There are various ways to react when the system must decide on interrupting a process:

1. *Nothing:* Doing nothing means letting the process finish to completion. The system could warn the user (or admin). Doing nothing could impact other processes' decisions (e.g. archiving, garbage collection).

2. *Kill (& Restart)*: A process is terminated and no state is saved. Any data or work produced by the process is considered lost and would need to restarted from its initial state. Because no results/data are expected in return, the system should attempt to kill a process at the first opportunity in order to minimize lost resources. The system can decide at any point whether to restart that query, provided its initial state is not dependant or does not factor into any other tasks (e.g. garbage collection, resource utilization).

3. *Stop (& Resume)*: A process is halted in place. Depending upon the nature of the process, it could return intermediate results to the user. When stopping, the system must decide (or prioritize) whether it saves the process's state so that it can either resume operations or be referenced for a future use. When stopping the system has control over when to best stop the process (e.g. checkpoint immediately or continue until the next potential checkpoint operation) or whether it should stop in place and revert to the previous checkpointed location; with the latter, the system would have to replay lost work in order to return to the process's state.

4. Rescheduling: System allows a current process to finish, but changes schedule of follow-on tasks. For example, the system could have a simulation task running with an immediate analysis task to follow; it could allow the simulation task to run and reschedule the analysis for a later time.

Other factors to consider with long running queries:

- Proactively determine optimal checkpointing moments during execution which is described in[145] as a means of query suspend and resume. This could be natural analysis breaks. For example, after every window it analyzed, the system follow up with a quick aggregation calculation and outputs the result to disk. This allows the system to have checkpoints already established and makes stopping and resuming the analysis task much more efficient.

- Keep a progress tracking metric to drive stopping vs killing decisions in the future. The metric should include statistics on how much work is completed, how much useful data is generates/available, how much

5.2 Explainability

Explainability (a.k.a. "Semantic Provenance") of dynamic systems defines the the efficacy linking feature sets (spatial, cause) with the lineage of events (temporal, effects). If a function links input to an output, then explainability defines the linkage from output to input; we, define this a *back-projection* function. If we could understand where an output derived, then we could select the perfect input for our exploratory system, such as telescopic coordinates to view a future cosmological event or simulation starting points to observe a specific action.

An orthogonal effort explored as part of this research endeavor is to incorporate any knowledge garnered from within a machine learning model into the adaptive process. This concept, still in its infancy and not well structured in literature is deemed explainability. If we could understand how the latent features directly tie to a labeled outcome, we could utilize the inner model parameters more effectively in steering our training. As a future consideration, such understanding could even lead to a dynamic model system whereby we actually change the structure of localized models based on the relationship between output and either input or latent features. We start with a background on the literature in provenance, describing the forward trace of input to output and then describe the inverse concept of explainability.

5.2.1 Provenance in Data Management

Traditional DBMS literature defines provenance as the lineage of data which traces raw data from its input (e.g. the source table) to its output (e.g. a tuple in a resultant query). Provenance adds value to data by explaining how it was obtained as well as providing a means of correctness and attribution. Uses include view updates, expressiveness, and annotation propagation. In contemporary data warehouses and distributed or online analytics, understanding the source of data can play a critical role in trust giving more value to the output and clarifying the correctness of a result.[146] From a systems perspective, retaining this lineage in an online fashion, a database can more efficiently update and maintain a view.

One of the earliest efforts which formally defines data provenance was in[147] where Cui et al clarify that a lineage of an output tuple is the collection of all input data which contributes to that tuple. For a simple, select-join-project (SJP) query, this is merely the input tuple from the source tables matching the selection and projection criteria in the associated relations. More complicated

lineage is defined as well for aggregations, inner queries, and others.

Collectively, describing the source data set is known as the *why-provenance*[148] component and is often referred to as the witness, or the subset of necessary input data items to ensure a given tuple exists in an output view. Here, provenance refers to explaining *why* a tuple, t, is formed, specifically linking a set of source to a destination. Formally, we state the lineage of a tuple is defined as the collection of input data that contribute to the output tuple, t, and/or the data that helped to produce t. This notion is based on the witnesses to a query. A witness is a subset of the database which ensures that a given tuple is in the output. Since the total number of witnesses can be exponentially high, a witness basis is often uses which encompasses only a compact representation of the of the set of all witnesses. In addition, we can also view, in a more detailed manner, the *how-provenance* of data lineage. This describes not only the set of source tuples, but also the manner in which they contributed. Finally, we can also define the *where-provenance* which pertains to the location of the source data contributing to the output tuple. Specifically, it refers to the source tables in a database from which the input rows contributed.

When applying provenance, we often consider the overhead associated with the process. Because it is added computation associated with a result and not the actual result, we must decide when and how to perform this step, known as materialization. In a lazy approach, computation is performed only when needed. The main drawback in lazy processing is that the system must ensure it retains enough information to execute the computation when necessary. Specifically with provenance, this may require additional techniques to infer information from the source data set if that information is not available after query execution (e.g. data is transformed and no longer in its original state). Conversely, in a greedy fashion, materialization is computed immediately. With provonance, this mean that the lineage information is produces when the query is executed. This allows the provenance of an output element to be derived from the results of the query as it is returned to the user. This may be a necessary technique when the provenance information may not be available after query processing. As noted, however, greedy processing incurs additional overhead.

5.2.2 Explainability in Data Analysis

While the notion of lineage and provenance provides a means to interpret an output tuple in a relational data model, this concept does not generalize from data management to data analysis.

Specifically, we view this through the perspective of a machine learning systems used for analytical processing. These systems solve complex, often non-linear problems using highly connected deep neural networks. Combining a myriad of input features they produce a means to predict output for a given input. One way to describe the explainability is to define what happens "inside the box." In a deep neural net, for example, such explainability could be viewed as identifying the hidden, latent nodes within the model which most contribute to an outcome. These learned features provide a direct lineage from source nodes to output classification or prediction. However, the nature and complexity of the underlying model makes such a task untenable for human consumption.

Defined in terms of input, one way to view explainability is purely by describing an output item by using the input feature set. Such a technique is more akin to the why-provenance of data management described above. Input features provide a witness basis for an predicted output. In cases such as decision support (e.g. medical recommendations), this may provide a sufficient explanation for result: A triage model recommends a patient to be admitted into a hospital because she has several contributing symptoms along with a history of pneumonia. Such simplified, seemingly linear explanations, may not always be as apparent.

The concept of applying explainability to an analysis process is akin to answering the question, what happens inside the black box? If we view a machine learning algorithm as a "black box" function with input features and an output result, the goal of an explanation is to help humans better understand the underlying functionality. When applied to a deep neural networks, for example, this entails providing a meaning or value to the hidden layers and the latent features. Lipton describes three different ways to view transparency.[149] The first is the human layer whereby a model is transparent if a human can comprehend the entirety of it at once – in a less boolean sense, a model's simulatability is described in terms of the amount of the underlying model that a human can comprehend. The second is the data layer, which is the decomposability of the model. This implies an explanation in terms of the feature space and relies on a set of interpretable features as a baseline, excluding complex or derived features which, themselves, are not explainable. The final layer is the algorithmic one, whereby an explanation provides insight into the nature of how an algorithm reaches a given outcome. This latter element of transparency provides a context under which we can prove a given algorithm converges or produces a given result. Unfortunately, due to the complex nature of machine learning models, one can never "prove" that they are bound to converge on a specific result.

We could view explainability as an extension of trust. The application of trust in technology was fully explored by Muir[150] who studied the adoption of an automatic system as compared to a manual one. She found that trust is the principle value which directly influences a human's decision on whether to adopt an automated process. The notion of trust is predicated on uncertainly and vulnerability in so much as one must rely on trust if she lacks complete confidence in a result or possesses some degree of risk in the consequences of accepting a result. Stating in the contrast: an individual has no need to trust anyone or anything if he is completely certain of the validity of information/process and has no risk to its consequences.[151] Applied to the modern digital age, the very complex nature of contemporary digital system precludes complete human certainty. In various studies, information which reduces this level of uncertainty, has been shown to increase trust and, hence, acceptance of the predicted knowledge or adoption of the automated system.

Trust is the focus for systems such as LIME, where the authors[152] take a more human-centric approach. They see explainability as a presentation of evidence (e.g. textual or visual artifact) to provide a qualitative description which relates a subset of an input domain, such as a group of pixels or a set of words, to the predicted result. This definition is centered around the underlying motivation of the concept of trust: a doctor is more apt to trust a predicted result if he/she has a better understanding of why the model reached is predicted conclusion. This attests to an aspect of human nature whereby the more information we garner about a problem, the more likely we are to use it.

An alternative viewpoint of explainability is to define it within the context of data exploration, which is our application in this effort. Such context creates a means to understand why underlying correlations exist leading to previously undiscovered knowledge. Often in our data exploration process, we define correlations but not necessarily causality: two input points, for example, which cluster together are said to "correlate", but do we really know why? A K-means algorithm merely provides a label for each individual data point, it does not provide any reason or mechanism to discern why two points exist within the same cluster. We can, however, infer such correlations based on the data items and describe their associations in terms of the feature space.

Finally, we can also view explainability in a spatio-temporal sense. Unlike independent and identically distributed datasets (i.i.d), spatio-temporal data is a series of dependant data where the value of one timestep is dependant on the previous step. Explainability can be redefined as understanding the temporal lineage of the data. At a higher-dimension, this temporal association

can be completely described in terms of the input features. This allows us to draws connections between higher and lower dimensions of datasets.

5.3 Multi-Model Clustering

We now examine the integration of the data exploration contributions presented herein as multi-model clustering with a lattice based approach for adaptive control. In Chapter 2, we presented the feature lattice as a means to organize and cluster data to drive a sampling process. It purpose was to facilitate the sampling process and provide a increasingly updated data structure which can continuously improve the candidate pool of input parameters which provided the most probable outcomes to improve the data exploration objective. We can apply this same methodology to multi-model machine learning, and specifically to semi-isolate models for both training and testing. We first discuss the related context, meta-learning and the describe how the lattice can improve on this process.

5.3.1 Meta-Learning

Meta-learning, loosely termed the process of "learning how to learn" is described as a sub-system within machine learning which adapts with experience gained by exploiting metaknowledge that is collected with from prior learning on one or more domains.[153] Inherent in the metalearning process is bias – or the influence of choosing a hypothesis to steer a training objective. It is distinct from traditional learning in that traditional approaches discover or "learn" a model as applied to a specific domain in order to to solve a single focused task. Metalearning, in contrast, derives experience across multiple domains and applications in order to discover insight or knowledge about the learning process. It can be viewed as a supporting effort to help users improve underlying machine learning tasks and it ties directly into the adaptive control framework and semi-isolated model approaches we presented.

The common application of metalearning is algorithm recommendation. A general search and matching problem, algorithm recommendation seeks to identify the best possible class of models given a descriptive objective task. It helps users narrow the search scope and recommend the single best class of algorithm to solve a problem, such as determining a specific clustering approach between K-mean, self-organizing maps, or a linkage algorithm.[154] Once a class of algorithms is determined,

metalearning can also apply to the model design phase. A typical use case is to identify the best existing model from which to warm-start a new task by initializing parameter values. Extending these concepts to the more general application of *model discovery*, metalearning can facilitate the combination of base models to identifying a more appropriate "super model" for the user. We can specifically apply a feature lattice to organize input models based on a subset of the feature domain. Using the lattice or hypercube approach, we can identify groups of models which cluster together and leverage this knowledge to warm-start new model training based on the parameters from all other models in the cluster.

Metalearning can also help account with resource management decisions. Akin to a cost model used within query optimization, metalearning is the process of retaining various statistics and features on the learning process. Such data can be recalled to estimate training time or resource requirements. Direct applications of training cost estimates can steer decisions on hyperparameter optimization, model training prioritization, or cost-benefit of continued training. We can employ metalearning by simply gathering and analyzing model statistics, such as parameter values or training accuracy, or by applying a secondary function to calculate a derived score or metric. One example may entail performing a k-NN algorithm on data items with common features to identify degree of redundancy (similar input and output), correlation (different input, same output), or inconsistency (same input, different output). An alternative view for meta-learning is landmarking, which refers to information garnered from the performance of either simplified models over a common dataset or the performance of various models using simplified datasets.[155] The key concept is to quickly compare and assess model or data selection as well as gauge performance expectations in order to make more detailed training, resourcing, and hyperparameter decisions. We can apply our data exploration solution in a meta-learning construct through a multi-model clustering approach with an adaptive controlling methodology in order to further improve on both model training and testing.

5.3.2 Implementation Concept

We organize models based on the input feature set. This is where a domain expert can help drive the process, providing feature engineering oversight. For example, with personalized healthcare, we can organize patients by static boolean features (e.g. family history of a disease). Using multiple features, we can categorize sub-domains in a concept features lattice as outlined in Chapter 2 and identify low dimensional latent attributes, akin to the various states of a molecular dynamic system.

We can subsequently roll-up or drill down domains as part of the exploration process. Clustering and organizing models can subsequently drive the training and the testing phases.

Training

During the training phase, the goal, as we have mentioned previously, is to maximize model accuracy while minimizing resource utilization. The solution employs various adaptive techniques to the LMC algorithm which, at its core, resembles stochastic gradient decent (see Algorithm 4). In the dual global-local graphs outline in Chapter 4, we randomly selected models for training at each iteration. However, could we, at each global training step, decide on a more optimal model or category of models from which to sample instead of a uniform random selection?

Clustered models offer a structure by which we can apply stratified sampling or any of the other sampling techniques highlighted in Chapter 2. We note that importance sampling has been used in machine learning[156] although we feel this is an area worthy of additional research efforts. We believe we can further enhance importance sampling in gradient descent approaches by leveraging a feature lattice to organize and cluster input sub-domains as part of a multi-model system.

Our sampling strategy would appropriately fit into this context so long as we do not introduce bias into the training and overfit our model. Integrating meta-learning concepts mentioned above, we can also incorporate statistical methods we have demonstrated. Employing the techniques we introduced to track convergence using a bootstrapping approach, we could allocate and prioritize resources to train sub-domains which are less converged over those that are more converged in order to improve aggregate training efficiency in the multi-model system.

Testing

Although we briefly covered the testing and servicing requirements of a multi-model system in Chapter 4, we believe adaptive techniques, data organization, and sampling approaches are paramount to scalability. In particular, the healthcare example we followed applied to patients in a single hospital and although we have extended this capability to support multiple hospitals, the system only trained on prediction of the onset of septic shock. If we entertained multiple tasks, (e.g. cardiac arrest, stroke), then this problem becomes even more complex.

To solve this, we could use the feature lattice to organize the testing dataset. After training one model for each node in the lattice, we use the organized structured to identify and warm-start

new models. We could even cluster the lattice using a roll-up operation to further reduce the problem complexity. As each personalized model is selected, we can leverage its peer models from the same node (or cluster) to warm-start new parameters. In addition, the parameters can be further maintained as an aggregation over all models within the same node: i.e. as we train more models, we create more robust average parameters. Adaptively, we can select nodes and/or sub-domains to test in one of two ways: (1) at the coarse grain level to provide better clusters, the system samples sub-domains contained within nodes whose average parameter values are least converged, and (2) at the fine grain level, we can use a nearest neighbor selection and train sub-domains who have the fewest neighbors within an epsilon bubble or who have the largest epsilon bubble of k-nearest neighbors. This last approach gives us a mechanism to identify the models which are not only untrained, but also most unlike any other model in system and, hence requiring the most training with the inverse assumption that models which are close in proximity (i.e. nearest neighbors) are most likely to be similar and, thus, the target model would require less training time. To summarize, these approaches help drive resource decisions with a cost-based decision, which, in turn, provide the tool to conduct adaptive control with sampling over multiple models in machine learning.

Bibliography

[1] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10, 2010.

[2] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed Data-parallel Programs from Sequential Building Blocks," in *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, ser. EuroSys '07. New York, NY, USA: ACM, 2007, pp. 59–72. [Online]. Available: http://doi.acm.org/10.1145/1272996.1273005

[3] J. W. Tukey, *Exploratory data analysis*. Reading, Mass.: Pearson, 1977.

[4] J. Rowley, "The wisdom hierarchy: representations of the DIKW hierarchy," *Journal of information science*, vol. 33, no. 2, pp. 163–180, 2007.

[5] D. J. Power, "What is the "true story" about using data mining to identify a relation between sales of beer and diapers?" 2002. [Online]. Available: http://www.dssresources.com/newsletters/66.php

[6] D. G. Pascual, *Artificial Intelligence Tools: Decision Support Systems in Condition Monitoring and Diagnosis*. CRC Press, 2015.

[7] M. Spick, *The ace factor air combat & the role of situational awareness*. Annapolis, Md Naval Institute Press, 1988. [Online]. Available: http://openlibrary.org/books/OL2065123M

[8] J. Lundberg, "Situation awareness systems, states and processes: a holistic framework," *Theoretical Issues in Ergonomics Science*, vol. 16, no. 5, pp. 447–473, 2015.

[9] C. von Clausewitz, M. Howard, and P. Paret, *On War*, ser. Princeton paperbacks. Princeton University Press, 1989. [Online]. Available: https://books.google.com/books?id=DRoL0NmsrlwC

[10] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized Streams: Fault-tolerant Streaming Computation at Scale," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13. New York, NY, USA: ACM, 2013, pp. 423–438. [Online]. Available: http://doi.acm.org/10.1145/2517349.2522737

[11] N. I. O. Standards and Technology, *NIST Special Publication 800-94 Guide to Intrusion Detection and Prevention Systems (IDPS)*. Gaithersburg, MD: U.S. Department of Energy, 2007.

[12] "Joint Publication 2-01.3: Joint Intelligence Preparation of the Operational Environment," 2009.

[13] S. Gilbert and N. Lynch, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services," *SIGACT News*, vol. 33, no. 2, pp. 51–59, jun 2002. [Online]. Available: http://doi.acm.org/10.1145/564585.564601

[14] W. Jean-Jacques E.. Slotine and Li, *Applied nonlinear control*. Prentice Hall Englewood Cliffs, 1991, vol. 199.

[15] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[16] F. Darema, "Dynamic data driven applications systems: A new paradigm for application simulations and measurements," in *Computational Science-ICCS 2004*. Springer, 2004, pp. 662–669.

[17] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, and Others, "C-store: a column-oriented DBMS," in *Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment, 2005, pp. 553–564.

[18] J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, and P. J. Haas, "Interactive data analysis: The control project," *Computer*, vol. 32, no. 8, pp. 51–59, 1999.

[19] M. N. Garofalakis and P. B. Gibbon, "Approximate Query Processing: Taming the TeraBytes," in *Proceedings of the 27th International Conference on Very Large Data Bases*, ser. VLDB '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 725—-. [Online]. Available: http://dl.acm.org/citation.cfm?id=645927.672356

[20] I. Lazaridis and S. Mehrotra, "Progressive approximate aggregate queries with a multi-resolution tree structure," in *ACM SIGMOD Record*, vol. 30, no. 2. ACM, 2001, pp. 401–412.

[21] M. L. Kersten, S. Idreos, S. Manegold, E. Liarou, and Others, "The researcher's guide to the data deluge: Querying a scientific database in just a few seconds," *PVLDB Challenges and Visions*, vol. 3, 2011.

[22] K. Dimitriadou, O. Papaemmanouil, and Y. Diao, "Explore-by-example: An Automatic Query Steering Framework for Interactive Data Exploration," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. New York, NY, USA: ACM, 2014, pp. 517–528. [Online]. Available: http://doi.acm.org/10.1145/2588555.2610523

[23] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[24] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "HaLoop: efficient iterative data processing on large clusters," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 285–296, 2010.

[25] S. Idreos, O. Papaemmanouil, and S. Chaudhuri, "Overview of data exploration techniques," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 277–281.

[26] M. Bayes and M. Price, "An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfrs," *Philosophical Transactions (1683-1775)*, pp. 370–418, 1763.

[27] H. Robbins, "Some aspects of the sequential design of experiments," in *Herbert Robbins Selected Papers*. Springer, 1985, pp. 169–177.

[28] J. C. Gittins, "Bandit processes and dynamic allocation indices," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 148–177, 1979.

[29] W. K. HASTINGS, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970. [Online]. Available: http://biomet.oxfordjournals.org/content/57/1/97.abstract

[30] G. M. Torrie and J. P. Valleau, "Nonphysical sampling distributions in Monte Carlo free-energy estimation: Umbrella sampling," *Journal of Computational Physics*, vol. 23, no. 2, pp. 187–199, 1977.

[31] C. R. Palmer and C. Faloutsos, "Density Biased Sampling: An Improved Method for Data Mining and Clustering," *SIGMOD Rec.*, vol. 29, no. 2, pp. 82–92, may 2000. [Online]. Available: http://doi.acm.org/10.1145/335191.335384

[32] R. Y. Rubinstein and D. P. Kroese, *Simulation and the Monte Carlo Method*, ser. Wiley Series in Probability and Statistics. Wiley, 2008. [Online]. Available: https://books.google.com/books?id=1-ffZVmazvwC

[33] A. Deshpande, Z. Ives, and V. Raman, "Adaptive Query Processing," *Found. Trends databases*, vol. 1, no. 1, pp. 1–140, jan 2007. [Online]. Available: http://dx.doi.org/10.1561/1900000001

[34] J. M. Hellerstein, P. J. Haas, and H. J. Wang, "Online Aggregation," in *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '97. New York, NY, USA: ACM, 1997, pp. 171–182. [Online]. Available: http://doi.acm.org/10.1145/253260.253291

[35] D. Feldman, M. Schmidt, and C. Sohler, "Turning Big Data into Tiny Data: Constant-size Coresets for K-means, PCA and Projective Clustering," in *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '13. SIAM, 2013, pp. 1434–1453. [Online]. Available: http://dl.acm.org/citation.cfm?id=2627817.2627920

[36] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica, "BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data," in *Proceedings of the 8th ACM European Conference on Computer Systems*, ser. EuroSys '13. New York, NY, USA: ACM, 2013, pp. 29–42. [Online]. Available: http://doi.acm.org/10.1145/2465351.2465355

[37] B. Efron, "Bootstrap Methods: Another Look at the Jackknife," *The Annals of Statistics*, vol. 7, no. 1, pp. 1–26, 1979. [Online]. Available: http://www.jstor.org/stable/2958830

[38] A. Pol and C. Jermaine, "Relational Confidence Bounds Are Easy with the Bootstrap," in *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '05. New York, NY, USA: ACM, 2005, pp. 587–598. [Online]. Available: http://doi.acm.org/10.1145/1066157.1066224

[39] D. Frenkel and B. Smit, *Understanding Molecular Simulation*, 2nd ed. Orlando, FL, USA: Academic Press, Inc., 2001.

[40] W. Wriggers, K. A. Stafford, Y. Shan, S. Piana, P. Maragakis, K. Lindorff-Larsen, P. J. Miller, J. Gullingsrud, C. A. Rendleman, M. P. Eastwood, and Others, "Automated event detection

and activity monitoring in long molecular dynamics simulations," *Journal of chemical theory and computation*, vol. 5, no. 10, pp. 2595–2605, 2009.

[41] V. Yeguas and R. Casado, "Big Data Issues in Computational Chemistry," in *Proceedings of the 2014 International Conference on Future Internet of Things and Cloud*, ser. FICLOUD '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 389–392. [Online]. Available: http://dx.doi.org/10.1109/FiCloud.2014.69

[42] F. Chirigati, V. Silva, E. Ogasawara, J. Dias, F. Porto, P. Valduriez, and M. Mattoso, "Evaluating Parameter Sweep Workflows in High Performance Computing," in *SWEET'12: 1st International Workshop on Scalable Workflow Enactment Engines and Technologies*. Scottsdale, Arizona, United States: ACM, may 2012, p. 10. [Online]. Available: http://hal-lirmm.ccsd.cnrs.fr/lirmm-00749968

[43] M. Mattoso, J. Dias, K. A. C. S. Ocaña, E. Ogasawara, F. Costa, F. Horta, V. Silva, and D. de Oliveira, "Dynamic Steering of HPC Scientific Workflows," *Future Gener. Comput. Syst.*, vol. 46, no. C, pp. 100–113, may 2015. [Online]. Available: http://dx.doi.org/10.1016/j.future.2014.11.017

[44] D. E. Shaw, P. Maragakis, K. Lindorff-Larsen, S. Piana, R. O. Dror, M. P. Eastwood, J. A. Bank, J. M. Jumper, J. K. Salmon, Y. Shan, and Others, "Atomic-level characterization of the structural dynamics of proteins," *Science*, vol. 330, no. 6002, pp. 341–346, 2010.

[45] R. Salomon-Ferrer, A. W. Gotz, D. Poole, S. Le Grand, and R. C. Walker, "Routine microsecond molecular dynamics simulations with AMBER on GPUs. 2. Explicit solvent particle mesh Ewald," *Journal of chemical theory and computation*, vol. 9, no. 9, pp. 3878–3888, 2013.

[46] S. Lee and S. Choi, "Landmark MDS Ensemble," *Pattern Recogn.*, vol. 42, no. 9, pp. 2045–2053, sep 2009. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2008.11.039

[47] D. Xin, J. Han, X. Li, and B. W. Wah, "Star-cubing: Computing Iceberg Cubes by Top-down

and Bottom-up Integration," in *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, ser. VLDB '03. VLDB Endowment, 2003, pp. 476–487. [Online]. Available: http://dl.acm.org/citation.cfm?id=1315451.1315493

[48] Y. Sismanis, A. Deligiannakis, N. Roussopoulos, and Y. Kotidis, "Dwarf: Shrinking the petacube," in *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. ACM, 2002, pp. 464–475.

[49] A. Rabkin, M. Arye, S. Sen, V. S. Pai, and M. J. Freedman, "Aggregation and Degradation in JetStream: Streaming Analytics in the Wide Area," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 275–288. [Online]. Available: http://dl.acm.org/citation.cfm?id=2616448.2616474

[50] S. Lee, J. Kim, Y.-S. Moon, and W. Lee, "Efficient Distributed Parallel Top-down Computation of ROLAP Data Cube Using Mapreduce," in *Proceedings of the 14th International Conference on Data Warehousing and Knowledge Discovery*, ser. DaWaK'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 168–179. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32584-7{_}14

[51] T. Milo and E. Altshuler, "An Efficient MapReduce Cube Algorithm for Varied DataDistributions," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16. New York, NY, USA: ACM, 2016, pp. 1151–1165. [Online]. Available: http://doi.acm.org/10.1145/2882903.2882922

[52] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration." *VISAPP (1)*, vol. 2, no. 331-340, p. 2, 2009.

[53] E. Bernhardsson, "ANNOY: Approximate nearest neighbors in C++/Python optimized for

memory usage and loading/saving to disk, 2013," *URL https://github. com/spotify/annoy*, 2013.

[54] B. Ganter and R. Godin, *Formal concept analysis.* Springer Berlin/Heidelberg., 2005.

[55] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh, "Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals," *Data Min. Knowl. Discov.*, vol. 1, no. 1, pp. 29–53, jan 1997. [Online]. Available: http://dx.doi.org/10.1023/A:1009726021843

[56] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," in *Proceedings of the 20th International Conference on Very Large Data Bases*, ser. VLDB '94. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 487–499. [Online]. Available: http://dl.acm.org/citation.cfm?id=645920.672836

[57] B. Goethals and M. J. Zaki, "Advances in frequent itemset mining implementations: report on FIMI'03," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 109–117, 2004.

[58] R. J. Bayardo Jr., "Efficiently Mining Long Patterns from Databases," *SIGMOD Rec.*, vol. 27, no. 2, pp. 85–93, jun 1998. [Online]. Available: http://doi.acm.org/10.1145/276305.276313

[59] C. Okasaki, *Purely functional data structures.* Cambridge University Press, 1999.

[60] Y. Rubner, C. Tomasi, and L. J. Guibas, "A Metric for Distributions with Applications to Image Databases," in *Proceedings of the Sixth International Conference on Computer Vision*, ser. ICCV '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 59—-. [Online]. Available: http://dl.acm.org/citation.cfm?id=938978.939133

[61] A. Gupta, I. S. Mumick, and V. S. Subrahmanian, "Maintaining Views Incrementally," *SIGMOD Rec.*, vol. 22, no. 2, pp. 157–166, jun 1993. [Online]. Available: http://doi.acm.org/10.1145/170036.170066

[62] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom, "View maintenance in a warehousing environment," *ACM SIGMOD Record*, vol. 24, no. 2, pp. 316–327, 1995.

[63] "Maryland Advanced Research Computing Center," \url{https://www.marcc.jhu.edu}.

[64] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, and K. Schulten, "Scalable molecular dynamics with NAMD," *Journal of computational chemistry*, vol. 26, no. 16, pp. 1781–1802, 2005.

[65] R. T. McGibbon, K. A. Beauchamp, M. P. Harrigan, C. Klein, J. M. Swails, C. X. Hernández, C. R. Schwantes, L.-P. Wang, T. J. Lane, and V. S. Pande, "MDTraj: A Modern Open Library for the Analysis of Molecular Dynamics Trajectories," *Biophysical Journal*, vol. 109, no. 8, pp. 1528–1532, 2015.

[66] S. Nutanong, N. Carey, Y. Ahmad, A. S. Szalay, and T. B. Woolf, "Adaptive Exploration for Large-scale Protein Analysis in the Molecular Dynamics Database," in *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, ser. SSDBM. New York, NY, USA: ACM, 2013, pp. 45:1—-45:4. [Online]. Available: http://doi.acm.org/10.1145/2484838.2484872

[67] P. A. Boncz, M. Zukowski, and N. Nes, "MonetDB/X100: Hyper-Pipelining Query Execution." in *CIDR*, vol. 5, 2005, pp. 225–237.

[68] H. Plattner, "A common database approach for OLTP and OLAP using an in-memory column database," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 2009, pp. 1–2.

[69] D. Abadi, P. Boncz, S. Harizopoulos, S. Idreos, S. Madden, and Others, "The design and implementation of modern column-oriented database systems," *Foundations and Trends{®} in Databases*, vol. 5, no. 3, pp. 197–280, 2013.

[70] S. Chandrasekaran and M. Franklin, "Remembrance of streams past: overload-sensitive management of archived streams," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment, 2004, pp. 348–359.

[71] I. Alagiannis, R. Borovica, M. Branco, S. Idreos, and A. Ailamaki, "NoDB: Efficient Query Execution on Raw Data Files," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12. New York, NY, USA: ACM, 2012, pp. 241–252. [Online]. Available: http://doi.acm.org/10.1145/2213836.2213864

[72] A. Lakshman and P. Malik, "Cassandra: A Decentralized Structured Storage System," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, apr 2010. [Online]. Available: http://doi.acm.org/10.1145/1773912.1773922

[73] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," *ACM SIGOPS operating systems review*, vol. 41, no. 6, pp. 205–220, 2007.

[74] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[75] M. Stonebraker, P. Brown, D. Zhang, and J. Becla, "SciDB: A database management system for applications with complex analytics," *Computing in Science & Engineering*, vol. 15, no. 3, pp. 54–62, 2013.

[76] B. Mozafari, P. Sarkar, M. Franklin, M. Jordan, and S. Madden, "Scaling Up Crowd-sourcing to Very Large Datasets: A Case for Active Learning," *Proc. VLDB Endow.*, vol. 8, no. 2, pp. 125–136, oct 2014. [Online]. Available: http://dx.doi.org/10.14778/2735471.2735474

[77] D. J. Abadi, Y. Ahmad, M. Balazinska, M. Cherniack, J.-h. Hwang, W. Lindner, A. S. Maskey,

E. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The design of the borealis stream processing engine," in *In CIDR*, 2005, pp. 277–289.

[78] Y. Ahmad, O. Kennedy, C. Koch, and M. Nikolic, "DBToaster: Higher-order Delta Processing for Dynamic, Frequently Fresh Views," *Proc. VLDB Endow.*, vol. 5, no. 10, pp. 968–979, jun 2012. [Online]. Available: http://dx.doi.org/10.14778/2336664.2336670

[79] M. Nikolic, M. ElSeidy, and C. Koch, "LINVIEW: Incremental View Maintenance for Complex Analytical Queries," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. New York, NY, USA: ACM, 2014, pp. 253–264. [Online]. Available: http://doi.acm.org/10.1145/2588555.2610519

[80] P. C. Shyamshankar, Z. Palmer, and Y. Ahmad, "K3: Language Design for Building Multi-Platform, Domain-Specific Runtimes."

[81] C. Xu, D. Tao, and C. Xu, "A survey on multi-view learning," *arXiv preprint arXiv:1304.5634*, 2013.

[82] A. Shon, K. Grochow, A. Hertzmann, and R. P. Rao, "Learning shared latent structure for image synthesis and robotic imitation," in *Advances in Neural Information Processing Systems*, 2005, pp. 1233–1240.

[83] J. Kreps, N. Narkhede, J. Rao, and Others, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, 2011, pp. 1–7.

[84] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.

[85] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *ACM SIGOPS operating systems review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[86] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin, "HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 922–933, 2009.

[87] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, and Others, "Spanner: Google's globally distributed database," *ACM Transactions on Computer Systems (TOCS)*, vol. 31, no. 3, p. 8, 2013.

[88] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: Wait-free Coordination for Internet-scale Systems." in *USENIX annual technical conference*, vol. 8. Boston, MA, USA, 2010, p. 9.

[89] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center." in *NSDI*, vol. 11, no. 2011, 2011, p. 22.

[90] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, and Others, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.

[91] S. Plimpton, "Fast Parallel Algorithms for Short-range Molecular Dynamics," *J. Comput. Phys.*, vol. 117, no. 1, pp. 1–19, mar 1995. [Online]. Available: http://dx.doi.org/10.1006/jcph.1995.1039

[92] E. Paci, R. W. Pastor, C. B. Post, J. Z. Pu, M. Schaefer, B. Tidor, R. M. Venable, H. L. Woodcock, X. Wu, W. Yang, D. M. York, and M. Karplus, "CHARMM: The Biomolecular Simulation Program."

[93] S. Haldar, *SQLite Database System Design and Implementation*. Sibsankar Haldar, 2015.

[94] N. Chaimov, A. Malony, S. Canon, C. Iancu, K. Z. Ibrahim, and J. Srinivasan, "Scaling

Spark on HPC Systems," in *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2016, pp. 97–110.

[95] F. Zhang, S. Lasluisa, T. Jin, I. Rodero, H. Bui, and M. Parashar, "In-situ feature-based objects tracking for large-scale scientific simulations," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*. IEEE, 2012, pp. 736–740.

[96] J. R. Boyd, "Destruction and creation," US Army Comand and General Staff College, Tech. Rep., 1987.

[97] S. Krompass, H. Kuno, J. L. Wiener, K. Wilkinson, U. Dayal, and A. Kemper, "Managing Long-running Queries," in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, ser. EDBT '09. New York, NY, USA: ACM, 2009, pp. 132–143. [Online]. Available: http://doi.acm.org/10.1145/1516360.1516377

[98] C. Binnig, A. Crotty, A. Galakatos, T. Kraska, and E. Zamanian, "The End of Slow Networks: It's Time for a Redesign," *Proc. VLDB Endow.*, vol. 9, no. 7, pp. 528–539, mar 2016. [Online]. Available: http://dl.acm.org/citation.cfm?id=2904483.2904485

[99] J. M. Hellerstein, M. Stonebraker, and J. Hamilton, "Architecture of a Database System," *Found. Trends databases*, vol. 1, no. 2, pp. 141–259, feb 2007. [Online]. Available: http://dx.doi.org/10.1561/1900000002

[100] L. Lamport and Others, "Paxos made simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.

[101] S. Kumar, C. Huang, G. Zheng, E. Bohm, A. Bhatele, J. C. Phillips, H. Yu, and L. V. Kalé, "Scalable Molecular Dynamics with NAMD on the IBM Blue Gene/L System," *IBM J. Res. Dev.*, vol. 52, no. 1/2, pp. 177–188, jan 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1375990.1376006

[102] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "A Survey of Data-Intensive Scientific Workflow Management," *J. Grid Comput.*, vol. 13, no. 4, pp. 457–493, dec 2015. [Online]. Available: http://dx.doi.org/10.1007/s10723-015-9329-8

[103] M. Mattoso, K. Ocaña, F. Horta, J. Dias, E. Ogasawara, V. Silva, D. de Oliveira, F. Costa, and I. Araújo, "User-steering of HPC Workflows: State-of-the-art and Future Directions," in *Proceedings of the 2Nd ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, ser. SWEET '13. New York, NY, USA: ACM, 2013, pp. 4:1—-4:6. [Online]. Available: http://doi.acm.org/10.1145/2499896.2499900

[104] S. Pronk, P. Larsson, I. Pouya, G. R. Bowman, I. S. Haque, K. Beauchamp, B. Hess, V. S. Pande, P. M. Kasson, and E. Lindahl, "Copernicus: A New Paradigm for Parallel Adaptive Molecular Dynamics," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 60:1—-60:10. [Online]. Available: http://doi.acm.org/10.1145/2063384.2063465

[105] B. Huang, N. W. D. Jarrett, S. Babu, S. Mukherjee, and J. Yang, "CÜMÜLÖN: Matrix-based Data Analytics in the Cloud with Spot Instances," *Proc. VLDB Endow.*, vol. 9, no. 3, pp. 156–167, nov 2015. [Online]. Available: http://dx.doi.org/10.14778/2850583.2850590

[106] C. Docan, M. Parashar, and S. Klasky, "DataSpaces: An Interaction and Coordination Framework for Coupled Simulation Workflows," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 25–36. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851481

[107] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, and Others, "Combining in-situ and in-transit processing to enable extreme-scale scientific analysis," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*. IEEE, 2012, pp. 1–9.

[108] P. Malakar, V. Vishwanath, T. Munson, C. Knight, M. Hereld, S. Leyffer, and M. E. Papka, "Optimal Scheduling of In-situ Analysis for Large-scale Scientific Simulations," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: ACM, 2015, pp. 52:1—-52:11. [Online]. Available: http://doi.acm.org/10.1145/2807591.2807656

[109] S. Lasluisa, F. Zhang, T. Jin, I. Rodero, H. Bui, and M. Parashar, "In-situ Feature-based Objects Tracking for Data-intensive Scientific and Enterprise Analytics Workflows," *Cluster Computing*, vol. 18, no. 1, pp. 29–40, mar 2015. [Online]. Available: http://dx.doi.org/10.1007/s10586-014-0396-6

[110] K. Ren, Q. Zheng, S. Patil, and G. Gibson, "IndexFS: scaling file system metadata performance with stateless caching and bulk insertion," in *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2014, pp. 237–248.

[111] L. Xiao, K. Ren, Q. Zheng, and G. A. Gibson, "ShardFS vs. IndexFS: Replication vs. Caching Strategies for Distributed Metadata Management in Cloud Storage Systems," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, ser. SoCC '15. New York, NY, USA: ACM, 2015, pp. 236–249. [Online]. Available: http://doi.acm.org/10.1145/2806777.2806844

[112] K. Beyer and R. Ramakrishnan, "Bottom-up Computation of Sparse and Iceberg CUBE," *SIGMOD Rec.*, vol. 28, no. 2, pp. 359–370, jun 1999. [Online]. Available: http://doi.acm.org/10.1145/304181.304214

[113] V. Harinarayan, A. Rajaraman, and J. D. Ullman, "Implementing Data Cubes Efficiently," in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '96. New York, NY, USA: ACM, 1996, pp. 205–216. [Online]. Available: http://doi.acm.org/10.1145/233269.233333

[114] S. S. H. Soleimani, J. Hensman, "Scalable Joint Models for Reliable Uncertainty-Aware Event Prediction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[115] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[116] W. Dai, G.-R. Xue, Q. Yang, and Y. Yu, "Transferring naive bayes classifiers for text classification," in *AAAI*, 2007, pp. 540–545.

[117] J. Gao, W. Fan, J. Jiang, and J. Han, "Knowledge transfer via multiple model local structure mapping," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 283–291.

[118] R. Caruana, "Multitask learning," in *Learning to learn*. Springer, 1998, pp. 95–133.

[119] T. Evgeniou and M. Pontil, "Regularized multi–task learning," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 109–117.

[120] M. M. Levy, M. P. Fink, J. C. Marshall, E. Abraham, D. Angus, D. Cook, J. Cohen, S. M. Opal, J.-L. Vincent, G. Ramsay, and Others, "2001 sccm/esicm/accp/ats/sis international sepsis definitions conference," *Intensive care medicine*, vol. 29, no. 4, pp. 530–538, 2003.

[121] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[122] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A Matlab-like Environment for Machine Learning," in *BigLearn, NIPS Workshop*, 2011.

[123] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: A CPU and GPU math compiler in Python," in *Proc. 9th Python in Science Conf.*, 2010, pp. 1–7.

[124] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, and Others, "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[125] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[126] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Advances in neural information processing systems*, 2010, pp. 2595–2603.

[127] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," in *Advances in Neural Information Processing Systems*, 2011, pp. 873–881.

[128] T. M. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project Adam: Building an Efficient and Scalable Deep Learning Training System." in *OSDI*, vol. 14, 2014, pp. 571–582.

[129] F. Niu, S. J. Wright, B. Recht, and C. Re, "HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent," in *Advances in neural information processing systems.*, 2011, pp. 693–701.

[130] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu, "Petuum: A New Platform for Distributed Machine Learning on Big Data," *arXiv preprint arXiv:1312.7651*, 2013.

[131] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, and Others, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[132] A. G. d. G. Matthews, M. van der Wilk, T. Nickson, K. Fujii, A. Boukouvalas, P. León-Villagrá, Z. Ghahramani, and J. Hensman, "{GP}flow: A {G}aussian process library using {T}ensor{F}low," *arXiv preprint 1610.08733*, oct 2016.

[133] P. Wieschollek, "No Title." [Online]. Available: https://github.com/PatWie/CppNumericalSolvers

[134] M. Saeed, C. Lieu, G. Raber, and R. G. Mark, "MIMIC II: a massive temporal ICU patient database to support research in intelligent patient monitoring," in *Computers in Cardiology, 2002.* IEEE, 2002, pp. 641–644.

[135] S. Jadhav, ""Using Real World Data to Enhance Clinical Trials"," *Informatics News*, 2017.

[136] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 1303–1347, 2013.

[137] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[138] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[139] I. Mitliagkas, C. Zhang, S. Hadjis, and C. Re, "Asynchrony begets momentum, with an application to deep learning," *arXiv preprint arXiv:1605.09774*, 2016.

[140] "Pricing for Amazon EMR and Amazon EC2," 2017. [Online]. Available: https://aws.amazon.com/emr/pricing/

[141] "Akamai's State of the Internet," 2017. [Online]. Available: https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q1-2017-state-of-the-internet-connectivity-executive-summary.pdf

[142] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, 0th ed. Arpaci-Dusseau Books, may 2015.

[143] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 323–336. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972490

[144] C. A. Waldspurger and W. E. Weihl, "Lottery Scheduling: Flexible Proportional-share Resource Management," in *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI '94. Berkeley, CA, USA: USENIX Association, 1994. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267638.1267639

[145] B. Chandramouli, C. N. Bond, S. Babu, and J. Yang, "Query Suspend and Resume," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '07. New York, NY, USA: ACM, 2007, pp. 557–568. [Online]. Available: http://doi.acm.org/10.1145/1247480.1247542

[146] J. Cheney, L. Chiticariu, and W.-C. Tan, "Provenance in Databases: Why, How, and Where," *Found. Trends databases*, vol. 1, no. 4, pp. 379–474, apr 2009. [Online]. Available: http://dx.doi.org/10.1561/1900000006

[147] Y. Cui, J. Widom, and J. L. Wiener, "Tracing the Lineage of View Data in a Warehousing Environment," *ACM Trans. Database Syst.*, vol. 25, no. 2, pp. 179–227, jun 2000. [Online]. Available: http://doi.acm.org/10.1145/357775.357777

[148] P. Buneman, S. Khanna, and T. Wang-Chiew, "Why and where: A characterization of data provenance," in *International conference on database theory*. Springer, 2001, pp. 316–330.

[149] Z. C. Lipton, D. C. Kale, C. Elkan, R. Wetzell, S. Vikram, J. McAuley, R. C. Wetzell, Z. Ji, B. Narayaswamy, C.-I. Wang, and Others, "The Mythos of Model Interpretability," *IEEE Spectrum*, 2016.

[150] B. M. Muir, "Trust in automation: Part I. Theoretical issues in the study of trust and human intervention in automated systems," *Ergonomics*, vol. 37, no. 11, pp. 1905–1922, 1994.

[151] K. Kelton, K. R. Fleischmann, and W. A. Wallace, "Trust in digital information," *Journal of the American Society for Information Science and Technology*, vol. 59, no. 3, pp. 363–374, 2008.

[152] M. T. Ribeiro, S. Singh, and C. Guestrin, "" Why Should I Trust You?": Explaining the Predictions of Any Classifier," *arXiv preprint arXiv:1602.04938*, 2016.

[153] C. Lemke, M. Budka, and B. Gabrys, "Metalearning: a survey of trends and technologies," *Artificial Intelligence Review*, vol. 44, no. 1, pp. 117–130, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10462-013-9406-y

[154] D. G. Ferrari and L. N. de Castro, "Clustering algorithm recommendation: a meta-learning approach," in *International Conference on Swarm, Evolutionary, and Memetic Computing*. Springer, 2012, pp. 143–150.

[155] R. Vilalta, C. G. Giraud-Carrier, P. Brazdil, and C. Soares, "Using Meta-Learning to Support Data Mining." *IJCSA*, vol. 1, no. 1, pp. 31–45, 2004.

[156] D. Needell, R. Ward, and N. Srebro, "Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm," in *Advances in Neural Information Processing Systems*, 2014, pp. 1017–1025.

Appendix A

Distributed LMC: CustomOp Implementation

Under the hood, TensorFlow uses the C++ unsupported/Eigen library. Highly efficient and optimized, the Eigen package is widely used in many applications. In order to improve optimization efficiency, we sought a solution to minimize the TensorFlow session overhead and opted for a custom operation developed in C++. This technique employed a extension of the OpKernel class compiled in C++ and imported into the TensorFlow application as a shared objected. The UML for the operator is provided in Figure A.1. The steps to implement such a solution are provided below:

1. (C++) *Define the input and output tensors.* The first step is to provide datatypes for the tensor objects. If the operator needs to infer tensor sizes, one must also provide a shape function. Tensorflow provides a set of datatypes based on numpy (e.g. float64, int32). Optionally one can define an alias to a list of these types and allow the graph to pass multiple data types; however, output must be specifically declared. To formally define the operator for use in Tensorflow, invoke the macro `REGISTER_OP` by providing the appropriate input and output information:

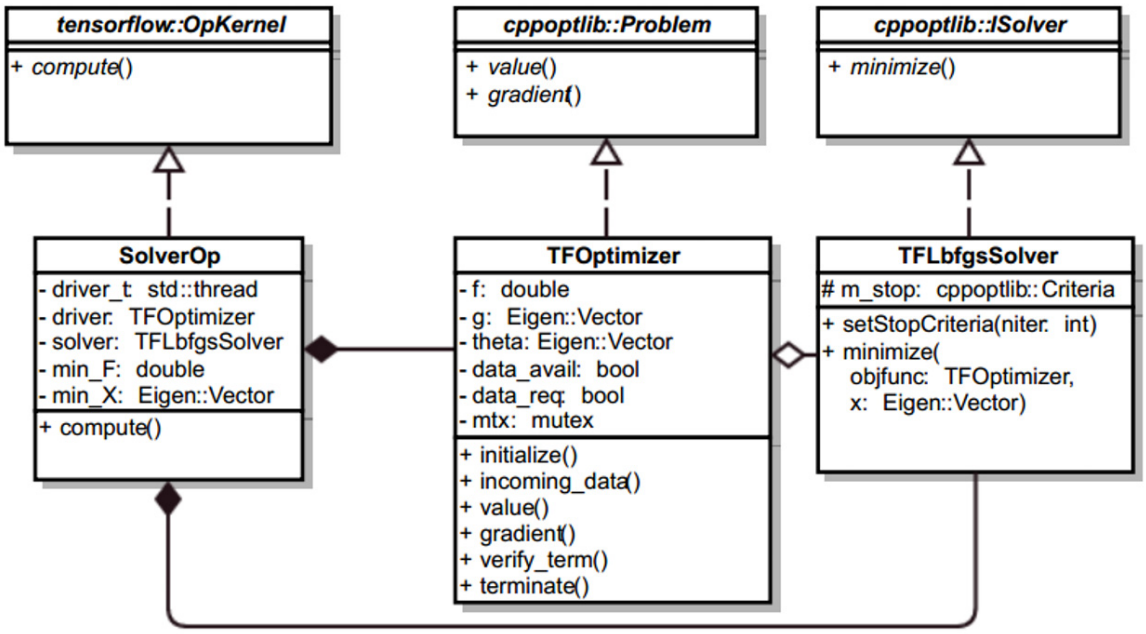


Figure A.1: UML diagram implementing a custom L−BFGS solver operator within TensorFlow.

```
REGISTER_OP("SolverOp")
  .Input("f: float64")
  .Input("g: float64")
  .Input("x_in: float64")
  .Input("t: int32")
  .Output("status: int32")
  .Output("x_out: float64");
```

2. (C++) *Inherit the `OpKernel` class.* The base class used to for all custom operations in Tensorflow is `OpKernel`. C++ implementation need to extent this class and override the `compute` function which is the callable invoked in he Tensorflow graph within a session. The function call contains one parameter, `OpKernelContext` which provides reference objects for all the input and output tensors.

```
class SolverOp : public OpKernel {
```

```
public:
  void Compute(OpKernelContext* context) override {
    //Implementation goes here
  }
};
```

3. (C++) *Register operation call with a device.* This provides a named operator to the Tensorflow graph along with allowable devices. GPU implementation are possible and can further improve the algorithm efficiency. This step is implemented via a macro call, `REGISTER_KERNEL_BUILDER`

```
REGISTER_KERNEL_BUILDER(Name("SolverOp")
  .Device(DEVICE_CPU), SolverOp);
```

4. (Shell) *Compile and Build.* Compiling using g++ is useful for debugging. The compiler needs to know the location of the Tensorflow library which are defined the environment variable, `TF_INC` below. Also be sure to use the same version which will be employed in the application. The following is a sample compilation command line:

```
g++ -std=c++11 -Wno-write-strings -fno-inline -g
  -o solver_op.so -fPIC -I $TF_ALL_INC -I $TF_INC
  -lpthread -shared solver_op.cc -O2
```

   It may be necessary to use the bazel toolkit in order to fully to integrate the custom operator into the Tensorflow application. This will require building Tensorflow from source (note: TF will only build in its entirety once). Further information on using bazel can be found at https://bazel.build/versions/master/docs/install.html.

5. (Python) Load and integrate. In the Tensorflow application, the custom operator is exposed by invoking the `tf.load_op_library` command. This will provide a module reference to the

shared object file and the custom operator class is called by invoking the registered name. Note that Tensorflow's naming convention require camelcase definition in the C++ which will result in underscore notation in Python (this is a quirk).

```
custom_op_library = tf.load_op_library(custom_op_lib)
status = custom_op_library.solver_op(f, g, x, token)
```

As shown in step 1 above, our implementation of this operator contains four input and one output tensor. They include the f and g tensors which are the loss and gradients, a reference to the x_i parameter, and a token. The use of a reference object provides the calling operator access to the source tensor's memory handle. In this case, we pass in the memory address for the x_i parameter which represents the batch of patients' local variable tensor in the graph. This allows us to optimize data update for the iterative L-BFGS algorithm: instead of passing a new local parameter, x_i' back to the TF graph, the custom op updates the parameter in place. We note that x_i passed into the from the TF graph is only set initially; however, each time the TF engine invokes this operator, the memory address may have changes – hence it is necessary to always pass in the most recent memory address for this parameter.

The final input parameter, a token, is used as a means to communicate from TF graph to the custom op. This token is used to set the max number of iteration the L-BFGS algorithm should run which allows us to configure this a as tunable parameter in the system. This same token can also be used to pass in a termination flags, if needed, to provide a preemption capability. For two way communication, the output value, status, represents a signal from the C++ operator to the TF graph in python using the following convention:

```
1 → Inner loop execution (linesearch)
Positive N → Outer loop completed (N=# iterations)
Negative N → Optimization complete (N=# iterations)
```

The co-routine concept describe in 4.3.2 is implemented with three object classes (see UML,

Figure A.1). The custom solver operator (as described above) inherits the TensorFlow `OpKernel` class and defines the `compute` function which is invoked at each call in the TF graph. This class runs in the main execution thread of the TF program and maintains state in between session calls; we leverage this stateful design and define driver and solver objects to implement the numerical Eigen optimization. The L-BFGS algorithm executes its own thread via the a `TFLbsfgSolver` minimize function call, using the provides problem object's `value()` and `gradient()` method calls to request a new f and g, respectively.

We extend the base Problem class in the numerical solver with a `TFOptimizer` class to mediate data flow between the two co-routines. This object serves as a driver class containing blocking routines in both threads with and shared data objects. We have retained a single mutex lock to facilitate proper a termination sequence, ensuring the optimizing thread properly joins with the main thread upon completion.

Appendix B

Distributed LMC Technical Documentation

B.1 Getting Started

This README walks through the detailed instructions to install, use, develop, and deploy the the distributed LMC machine learning framework for training and predicting health events (e.g. severe sepsis) in patients. If you would like to bypass this and simply run (and go) with a pre-packaged version, we have created a docker image, `damsl/tf:dev` (TAG TBD) which includes all software, dependencies and a sample data file. To get going:

1. Pull the Docker image and start a container:

   ```
   docker pull damsl/tf:dev
   docker run -it --entrypoint=/bin/bash damsl/tf:dev
   ```

2. Then, execute the local run script:

   ```
   ./golocal.sh
   ```

This will start one master service and one worker node on the local host (inside the container) on ports 2220 and 2221. By default, the worker's output will display in stdout and the master's output is redirected to /tmp/lmc.ps. These locations and all other program parameters can be modified by changing the start `golocal.sh` script.

B.2 Install and Set up

Prerequisite Requirements include:

- Python v2.7 (python v3 not yet supported)
- Tensorflow v1.0.1 (see note below)
- gcc 4.8.2+

Additional python packages needed to run LMC are listed in the file, requirements.txt.

Note: If you plan on extending/developing the customized L-BFGS operator (in C++) you will need to install and build Tensorflow from source using the Google Bazel build system. For more information: https://www.tensorflow.org/install/install_sources.

Input Sources

1. File based input (MIMIC). Mimic file based input is provided by setting the patient_data command line flag to the loation of the csv-formatted patients data file.

2. CDM Data (file). To use a locally stored, numpy-formatted CDM data source, set the flag to 'LOCAL'

3. DB Integration. Setting the patient_data flag to 'SQL' will povide a connection to the AWS-located Postgress service. When running locally (outside of AWS), be sure to set the database to 'test.' When running on AWS, set the database to 'dw' which has a pre-configured connection setting included. In addtion, you will need to set the environment variable DB_PASS to

the database password. Optionally, you can also set the connection for name, host, and user via command line options (password will need to be provided).

B.3 Execution

The LMC Framework is invoked via the python script, `joint_model.py`. While some options/features are currently directly integrated into the code (e.g. hard-coded feature list options), many are passed in via command line options. The Tensorflow engine provides a global flags object, `tf.app.flags` which stores global configuration. To see the complete list of options, type `python joint_model.py --help` or see the `set_flag()` function call near the top of the joint_model.py file. Be sure to set the LMCHOME environment variable to the location of the source directory (which is, by default the current working dir).

Local Execution Within a cluster of host machines, the framework operates with one parameter server (a.k.a the master) and 1 or more worker hosts (note: a host refers to a single server). Each host runs the exact same python code – the only difference is the assigned job (which is either `ps` or `worker`) and the task index. Both of these are set as command line options, `--job_name` and `--task_index` passed into the program. In addition, set the flags `--ps_hosts` and `--worker_hosts` with the corresponding list of machines names and ports. **NOTE: worker index numbers MUST match the corresonding list passed into worker_hosts** (e.g. if node_b is the first worker, it must have task_index=0). The following example best illustrates a simple, 3-node configuration:

In this example, the Master will run on node_a. Two workers will run on hosts, node_b and node_c. All will use port 2222 on their local machine. The following command should be executed on each machine:

On node_a run:

```
python joint_model.py \
        --ps_hosts=node_a:2222 \
        --worker_hosts=node_b:2222,node_c:2222 \
```

```
        --job_name=ps \
        --task_index=0
```

On node_b run:

```
python joint_model.py
        --ps_hosts=node_a:2222 \
        --worker_hosts=node_b:2222,node_c:2222 \
        --job_name=worker \
        --task_index=0
```

On node_c run:

```
python joint_model.py \
        --ps_hosts=node_a:2222 \
        --worker_hosts=node_b:2222,node_c:2222 \
        --job_name=worker \
        --task_index=1
```

For larger deployments and for better orchestration, it is recommended to put this command inside a bash run script and set environment variables (see the run scripts for more detailed examples).

AWS/Kubernetes Integration The distributed version is designed to operate on the opsdx AWS cluster. Cluster configuration/orchestration is automated via a Kubernetes statefulset integrated with a service to manage deterministic hostnames. Run-time environment variables are defind within the pod-spec template definition (defined in the execution yaml file) and passed into the docker container run script, gok8s.sh. These parameters subsequently pass into the lmc framework via command line options (see below on how to add a new command line option). For kubernetes, The LMC framework is designed to automically detect a k8s environment and construct the cluster (the ps_host and worker_host list) based on the number of workers. To execute the framework on AWS:

1. *(optional) Create a Launch Configuration.* To select a specific instance type, you need copy and create a new launch config (you cannot modify an existing launch config in AWS). To set initial docker docker images or add other startup commands, modify the user data script in

the advanced detailed panel. For spot pricing, select the spot price check box and enter a price in the launch.

2. *Start the Nodes.* In the AWS web interface, select an ASG and up-scale the ASG to the desired number of nodes (e.g. 3 nodes for a 2-worker + 1-master configuration). Be sure to select the desired launch config. For spot pricing, check the current prices and, if needed, switch availabilty zones (you may need to configure a new subnet for this).
3. *Set runtime configuration.* Modify the yaml file with appropriate runtime parameters: it is important to set the number of replicas key and NUM_WORKERS environment variable, where the number of replicas is always 1 more than the number of workers. In addition, be sure the opsdx_nodegroup value corresponds to the ASG in which you want to run (i.e. if you selected the tf1 ASG, be sure to set the nodegroup value to 'tf1' at the bottom of the yaml file).
4. *Apply.* Launch the statefulset using the following command and ensure that both the namespace(s) are created and you have set up a secrets file connecting to the backend DB: `kubectl apply -n tf1 -f tflmc.yaml -f dbpass.yaml`
5. *Delete.* When the job completes, you need to manualy delete the statfulset: `kubectl delete -n tf1 tflmc1`
6. (optional). If you are down, you can down-scale the ASG to 0 nodes.

B.4 Implementation Details

PSQL Database The tools/db.py file is designed as a standalone module – You can import it as a local session on OPSDX and use it in an interactive (or jupyter) Python environment. To connect to it, you need to set the DB_PASS environment variable which is set upon launch in the gok8s.sh script, but needs to be set in any local session. The test database is accessible from outside AWS, but has limited capabilities. To access the dw (or dev/prod) databases, you need to

connect from inside the AWS local subnet. Some notes on schema, setup and I/O:

- Output metrics (timing and AUC scores) are written in key-value form to the table tflmc_metrics. For this table. All nodes ID-s of 999 corresond to the master node.
- Global parameters (and local params when implemented) are stored as numpy arrays.
- There is a catch-all key-value table whch stores JSON formatted data. Every experiment's runtime config is stored in this table under the key, 'flags'

Logging and Output The program uses two directories to output data files. The WORKDIR ennvironment variable defines the location of a tempoarary working directory – it should be local to the host machine. This includes params files, index files and the node's log file. An output directory, defined via the OUTDIR environment variables, serves as the location for cluster-consolidated files. It is currenly only used for local cluster deployments (non-SQL) to collect metric, log and other data from workers nodes for processing at the master; however, it can serve as a consolidation point for any other large data files. The output directory should be accesible to all nodes (e.g. an NFS share).

For logging, we utilize the standard Python logging module to print routine an debugging information to both console and to file. A custom timelogger objects is defined and implemented (see the tools folder) which provides elapased timestamp messages relative to program start.

Program Parameters Tensorflow provides a globally accessible key-value module, which can be accessed anywhere in the codebase where tensorflow is imported. The module, `tf.app.flags`, is used to process input parameters and for program configuration settings. All options are defined in the set_flags() method which is defined near the top of the joint_model.py file. To add a new parameter:

1. Add the appropriate data definition call in the set_flags() method and be sure to use the correct data type (e.g: `tf.app.flags.DEFINE_integer('new_param', 0, 'My New param')`.

2. For k8s: Update the yaml file to include the environment variable which must match the variable in the gok8s.sh run script. (e.g. add `-name: NEW_PARAM_ENV_VAR` with a value field set to your desired value)

The Custom LBFGS Operator Written in C++, the custom op extends the PatWie C++ numerical solver (`https://github.com/PatWie/CppNumericalSolvers`). It uses the Eigen Matrix and Eigen Tensor libraries which are header-only libraries (`https://eigen.tuxfamily.org/dox/`).

NOTE: Although similar, eigen Tensors are different from Eigen Matrices. Tensors are part of unsupported Eigen library. For a good online reference use this link: `https://bitbucket.org/eigen/eigen/src/677c9f1577810e869f4f09881cabc3e503a810c1/unsupported/Eigen/CXX11/src/Tensor/README.md` Some details: Under the hood, Tensorflow stores data as memory-mapped Eigen Tensors. When accessing the tensor, you actually create a map to the data object and then access the underling data handle.

Vita

Benjamin A. Ring, Lieutenant Colonel, U.S. Army was commissioned as an Armor Officer in 1996 with a Bachelor in Computer Science from the U.S. Military Academy in West Point, NY as an honor graduate. After completing the Armor Officer Basic course, he was assigned to the 1st Infantry Division in Schweinfurt, Germany with 1-77 Armor Battalion and served tours in both Bosnia and Kosovo. In 2000, upon completion of the Armor Captain's Career Course as the distinguished honor graduate, he was assigned to 1-8 Cavalry, 1st U.S. Cavalry Division, Ft. Hood, Texas. From 2002-03, he commanded the headquarters company and in 2003-04, he commanded A/1-8 CAV, to include combat deployment to Iraq. In 2006, he earned his Masters in Computer Science from Boston University and taught at West Point, 2006-09, receiving his Assistant Professor promotion in 2008. From 2010-11, he served as Senior Systems Manager for Task Force 101, Regional Coalition Forces East in Afghanistan, and from 2011-14, he was the Academic Systems Manager for the U.S. Army Command and General Staff College, Ft. Leavenworth, KS. A member of Upsilon Pi Epsilon and Phi Kappa Phi Honor Societies, he has several publications in both technical and leadership journals. His military decorations include the Bronze Star, the Defense and Army Meritorious Service medals, the Combat Action Badge, and the Parachute and Air Assault Badges. Beginning in September, 2017, Lt. Colonel Ring will be assigned as the Chief Operations Officer, U.S. Army Cyber Protection Brigade, Ft. Gordon, GA.