University of Nevada, Reno

Big Data Application and System Co-optimization in Cloud and HPC Environment

A dissertation submitted in partial fulfillment
of the requirements for the degree of Doctor of
Philosophy in Computer Science and Engineering

by

Xinying Wang

Dr. Feng Yan, Dissertation Advisor
Dr. Dongfang Zhao, Dissertation Co-Advisor

May 2022

The Graduate School

We recommend that the dissertation
prepared under our supervision by

Xinying Wang

entitled

Big Data Application and System Co-optimization in Cloud and HPC Environment

be accepted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

Feng Yan, Ph.D.
Advisor

Dongfang Zhao, Ph.D.
Co-Advisor

Lei Yang, Ph.D.
Committee Member

Haoting Shen, Ph.D.
Committee Member

Lipeng Wan, Ph.D.
Committee Member

Lei Cao, Ph.D.
Graduate School Representative

David W. Zeh, Ph. D., Dean
Graduate School

May 2022

Abstract

The emergence of big data requires powerful computational resources and memory subsystems that can be scaled efficiently to accommodate its demands. Cloud is a new well-established computing paradigm that can offer customized computing and memory resources to meet the scalable demands of big data applications. In addition, the flexible pay-as-you-go pricing model offers opportunities for using large scale of resources with low cost and no infrastructure maintenance burdens. High performance computing (HPC) on the other hand also has powerful infrastructure that has potential to support big data applications. In this dissertation, we explore the application and system co-optimization opportunities to support big data in both cloud and HPC environments. Specifically, we explore the unique features of both application and system to seek overlooked optimization opportunities or tackle challenges that are difficult to be addressed by only looking at the application or system individually. Based on the characteristics of the workloads and their underlying systems to derive the optimized deployment and runtime schemes, we divide the workflow into four categories: 1) memory intensive applications; 2) compute intensive applications; 3) both memory and compute intensive applications; 4) I/O intensive applications.

When deploying memory intensive big data applications to the public clouds, one important yet challenging problem is selecting a specific instance type whose memory capacity is large enough to prevent out-of-memory errors while the cost is minimized without violating performance requirements. In this dissertation, we propose two techniques for efficient deployment of big data applications with dynamic and intensive memory footprint in the cloud. The first approach builds a performance-cost model that can accurately predict how, and by how much, virtual memory size would slow down the application and consequently, impact the overall monetary cost. The second approach employs a lightweight memory usage prediction methodology based on dynamic meta-models adjusted by the application's own traits. The key idea is to eliminate the periodical checkpointing and migrate the application only when the predicted memory usage exceeds the physical allocation.

When applying compute intensive applications to the clouds, it is critical to make the applications scalable so that it can benefit from the massive cloud resources. In this dissertation, we first use the Kirchhoff law, which is one of the most widely used physical laws in many engineering principles, as an example workload for our study. The key challenge of applying the Kirchhoff law to real-world applications at scale lies in the high, if not prohibitive, computational cost to solve a large number of nonlinear equations. In this dissertation, we propose a high-performance deep-learning-based approach for Kirch-

hoff analysis, namely HDK. HDK employs two techniques to improve the performance: (i) early pruning of unqualified input candidates which simplify the equation and select a meaningful input data range; (ii) parallelization of forward labelling which execute steps of the problem in parallel.

When it comes to both memory and compute intensive applications in clouds, we use blockchain system as a benchmark. Existing blockchain frameworks exhibit a technical barrier for many users to modify or test out new research ideas in blockchains. To make it worse, many advantages of blockchain systems can be demonstrated only at large scales, which are not always available to researchers. In this dissertation, we develop an accurate and efficient emulating system to replay the execution of large-scale blockchain systems on tens of thousands of nodes in the cloud.

For I/O intensive applications, we observe one important yet often neglected side effect of lossy scientific data compression. Lossy compression techniques have demonstrated promising results in significantly reducing the scientific data size while guaranteeing the compression error bounds, but the compressed data size is often highly skewed and thus impact the performance of parallel I/O. Therefore, we believe it is critical to pay more attention to the unbalanced parallel I/O caused by lossy scientific data compression.

Acknowledgments

I would like to express my deepest gratitude to my advisors Dr. Feng Yan and Dr. Dongfang Zhao for the excellent guidance, caring, patience, and engagement throughout the course of my study at University of Nevada, Reno. Thank you for all your intellectual and emotional support throughout this dissertation work especially during hard times. I am very fortunate to have you as my advisors. I am also particularly grateful to Dr. Lipeng Wan, the mentor during my intership. He has been a great friend and guided me during my internship. In addition, I would like to express my sincere gratitude to my committee members, Lei Yang, Haoting Shen and Lei Cao for their valuable intellectual inputs in improving the dissertation work. I would also like to thank all the members of the Intelligent Data and Systems Lab (IDS Lab) and HPDIC Lab with whom I had a chance to work in close collaboration. Their diverse backgrounds, inspiring suggestions and in-depth discussions have formed an intellectual environment for interdisciplinary research.

In addition, I would like to thank my parents for their endless love and support, without whom I cannot have come this far.

Finally, I am thankful to all the institutions for the support during my Doctorates studies. The work presented in this dissertation is part supported by Amazon Web Services (AWS) Research Grant, a Microsoft Azure Research Award, and National Science Foundation (NSF) awards CCF-1756013 and IIS-1838024 .

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Overview

Computational science is the usage of hardware, algorithms and other scientific knowledge to simulate the physical world on powerful computing. Big data applications are usually composed of many small tasks and there can be heterogeneous computing demands among these tasks causing load imbalance. Existing methods for achieving load balance are usually application specific and require significant amount of manual efforts to make changes to the application code. Elasticity and pay-as-you-go features in cloud computing have potential to address the load imbalance issue through a general resource scheduling and migration approach without changing application code. So based on the different characteristics of applications and the features of systems, we aim at developing efficient scaling methodologies to derive the optimized deployment and runtime schemes. However, there are many challenges due to the unique features and requirements of applications such as memory-intensive applications which usually consume lots of memory that can be quite expensive if not carefully planned. There are aslo challenges in achieving optimal deployment, such as predict growth rates of applications, storage capacity usage, and I/O bandwidth. To overcome all these issues, efforts are needed to revolve around all the constituent problems, and the spectrum of research topics relevant to scalability. This dissertation studies efficient scaling in Clouds and HPC environment with application and system in synergy.

1.2 Dissertation Contributions

This section briefly highlights the contributions of this dissertation.

1.2.1 Toward Cost-effective Memory Scaling in Clouds: Symbiosis of Virtual and Physical Memory

When deploying memory-intensive applications to public clouds, one important yet challenging question to answer is how to select a specific instance type whose memory capacity is large enough to prevent out-of-memory errors while the cost is minimized without violating performance requirements. The state-of-the-practice solution is trial and error, causing both performance overhead and additional monetary cost. This paper investigates two memory scaling mechanisms in public clouds: physical memory (good performance and high cost) and virtual memory (degraded performance and no additional cost). In order to analyze the trade-off between performance and cost of the two scaling options, a performance-cost model is developed that is driven by a lightweight analytic prediction approach through a compact representation of the memory footprint. In addition, for those scenarios when the footprint is unavailable, a meta-model-based prediction method is proposed using just-in-time migration mechanisms. The proposed techniques have been extensively evaluated with various benchmarks and real-world applications on Amazon Web Services: the performance-cost model is highly accurate and the proposed just-in-time migration approach reduces the monetary cost by up to 66%.

1.2.2 HDK: Toward High-Performance Deep-Learning-Based Kirchhoff Analysis

The Kirchhoff law is one of the most widely used physical laws in many engineering principles, e.g., biomedical engineering, electrical engineering, and computer engineering. One challenge of applying the Kirchhoff law to real-world applications at scale lies in the high, if not prohibitive, computational cost to solve a large number of nonlinear equations. Despite recent advances in leveraging a convolutional neural network (CNN) to estimate the solutions of Kirchhoff equations, the low performance is still significantly hindering the broad adoption of CNN-based approaches. This part proposes a high-performance deep-learning-based approach for Kirchhoff analysis, namely HDK. HDK employs two techniques to improve the performance: (i) early pruning of unqualified input candidates and (ii) parallelization of forward labelling. To retain high accuracy, HDK also applies various optimizations to the data such as randomized augmentation and dimension reduction. Collectively, the aforementioned techniques improve the analysis speed by 8× with accuracy.

1.2.3 Toward Accurate and Efficient Emulation of Public Blockchains in the Cloud

Blockchain is an enabler of many emerging decentralized applications in areas of cryptocurrency, Internet of Things, smart healthcare, among many others. Although various open-source blockchain frameworks are available in the form of virtual machine images or docker images on public clouds, the infrastructure of mainstream blockchains nonetheless exhibits a technical barrier for many users to modify or test out new research ideas in

blockchains. To make it worse, many advantages of blockchain systems can be demonstrated only at large scales, e.g., thousands of nodes, which are not always available to researchers. This paper presents an accurate and efficient emulating system to replay the execution of large-scale blockchain systems on tens of thousands of nodes. In contrast to existing work that simulates blockchains with artificial timestamp injection, the proposed system is designed to be executing real proof-of-work workload along with peer-to-peer network communications and hash-based immutability. In addition, the proposed system employs a preprocessing approach to avoid the per-node computation overhead at runtime and thus achieves practical scales. We have evaluated the system for emulating up to 20,000 nodes on Amazon Web Services (AWS), showing both high accuracy and high efficiency with millions of transactions.

1.2.4 Unbalanced Parallel I/O: An Often-Neglected Side Effect of Lossy Scientific Data Compression

Lossy compression techniques have demonstrated promising results in significantly reducing the scientific data size while guaranteeing the compression error bounds. However, one important yet often neglected side effect of lossy scientific data compression is its impact on the performance of parallel I/O. Our key observation is that the compressed data size is often highly skewed across processes in lossy scientific compression. To understand this behavior, we conduct extensive experiments where we apply three lossy compressors MGARD, ZFP, and SZ, which are specifically designed and optimized for scientific data, to three real-world scientific applications Gray-Scott simulation, WarpX, and XGC. Our analysis result demonstrates that the size of the compressed data is always skewed even if the original data is evenly decomposed among processes. Such skewness widely exists in

different scientific applications using different compressors as long as the information density of the data varies across processes. We then systematically study how this side effect of lossy scientific data compression impacts the performance of parallel I/O. We observe that the skewness in the sizes of the compressed data often leads to I/O imbalance, which can significantly reduce the efficiency of I/O bandwidth utilization if not properly handled. In addition, writing data concurrently to a single shared file through MPI-IO library is more sensitive to the unbalanced I/O loads. Therefore, we believe our research community should pay more attention to the unbalanced parallel I/O caused by lossy scientific data compression.

1.3 Conclusion and Discussion

In this dissertation, we have studied the optimization of performance and scalibity in the cloud. Chapter 3 studies memory scaling in the cloud with hybrid method. We developed a performance-cost model that is driven by a lightweight analytic prediction approach through a compact representation of the memory footprint, as well as a meta-model based prediction method using just-in-time migration mechanisms. Chapter 4 analyzes the Kirchhoff-Based computationally-prohibitive problem. We proposed a high performance deep learning method named HDK to conduct Kirchhoff law which can efficiently and accurately detect abnormal values in electric area. Chapter 5 presents an accurate and efficient emulating system to replay the execution of large-scale blockchain systems on tens of thousands of nodes. Chapter 6 introduces the I/O imbalance caused by lossy compression and the serious impact.

In chapter 3, we focus on developing a practical system tool for preventing OOM error

while balancing the performance and cost trade-off for big data applications in the public cloud rather than making theoretical contributions. The proposed memory scaling methodology is based on two scenarios: 1) when some information of memory footprints is known a priori; 2) when the memory footprints is unknown in advance. Specifically, when the memory footprint of an application is unknown a priori, the application can start with the just-in-time application migration as it does not impose any assumption on memory footprint. After running a while, if a repeating pattern is detected in history memory footprints, it can switch to the cost-ware exploitation of virtual memory approach for better cost-effectiveness. The three main contributions is as following: *(i) We develop analytic models to predict applications' performance and cost when various portions of virtual memory are used in public clouds.* As a result, users will know how, and by how much, virtual memory will affect the overall performance and cost of their applications before actually executing them. *(ii) We propose multiple cost-effective approaches for preventing OOM error in public clouds.* Instead of periodically checkpointing memory states, the proposed elevation-migration approaches incur only one batch of memory accesss when encountering memory depletion. *(iii) We extensively evaluate the proposed techniques using benchmarks and real-world applications.* Experimental results demonstrate high effectiveness of both techniques on AWS.

In chapter 4, we propose a new deep-learning-based approach to efficiently conduct Kirchhoff analyses that are widely used in various engineering fields. The new approach employs multiple techniques including early pruning of unqualified input data, parallelization of forward labelling, data augmentation through random errors and dimension reduction, all of which collectively enable an efficient and accurate mechanism for large-scale Kirchhoff analyses. We implement the proposed approach with the latest machine learning framework and the parallel computing library, and evaluate it with real-world engineering applications showing promising results.

In chapter 5, we propose a very first blockchain *emulator* with both high accuracy and high scalability. In contrast to Bitcoin-Simulator [39], BlockLite comprises a specific module to execute real PoW workload and thus making it an *emulator* rather than a simulator,supports fine-grained transaction management, and scales out to 20,000 nodes thanks to its efficient network communications built upon distributed queues along with PoW preprocessing that incurs negligible runtime overhead. Different than VIBES [86], BlockLite is fully decentralized with no single point of failure or performance bottleneck. It should be noted that even on a single node the decentralization philosophy of blockchains still holds for a blockchain *emulator* because each user-level thread is now considered as an individual node.

In chapter 6, we first characterize the skewness of three real world compressed scientific data by studying three widely used data compressors. Then, we analyze the potential impacts of the skewed compressed data on parallel I/O performance. Finally, we compare the write time and write throughput under different use scenarios.

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter provides some necessary general background knowledge and detailed related work.

2.1 Memory Management in Cloud

Memory management in cloud computing recently draws plenty of research interests in the community. One of the hot topics is predicting memory usage to prevent application crashes or detect abnormality, and a common method is using time series. In particular, Spinner et al. [85] proposed to predict the memory usage of applications on virtual machine in a proactive manner using time series analysis (with a training period usually in terms of days). Santosh Aditham et al. [69] applies LSTM recurrent neural networks for prediction the memory usage as a part to prevent data attacks. Also, Lingxue Zhu et al. [87] were motivated by recent Long Short Term Memory networks and successfully apply it to large-scale time series anomaly detection at Uber. In contrast to aforementioned related work, this dissertation for the first time combines checkpointing with time series.

To reduce cost, a rich literature focuses on the optimization of resource allocation and cost-effectiveness for a cluster of cloud instances. In [54], authors proposed to reconfigure the constituent instances serving the same workload with lower cost. More recently, contracts-based resource sharing model [98] was proposed to save the cost. Some

works [50,96] focused on minimizing the cost of spot instances; Furthermore, some works [58] showed that it is even possible to achieve both the high reliability of on-demand instances and the low cost of spot instances on AWS using the proposed scheduling techniques. To the best of our knowledge, this dissertation is the first work focusing on cost reduction for memory-intensive applications using virtual memory technology.

Besides, much work has been focused on the modeling and scheduling part in resource management. In [99], an automatic resource allocation model for scaling virtual machine was developed; a similar work [44] on automatic leasing virtual machines was published in the same year. In particular, in [100] authors proposed an autonomic and elastic resource scheduling framework was proposed; a scheduling algorithm based on Lyapunov optimization was proposed in [84]. It should be noted that those techniques for better resource allocation are also applicable to other metrics in addition to cost, such as energy consumption that is one of the most important research topics, such as [13]. While the primary goal of this dissertation is to investigate new approaches to reduce the monetary cost for memory-intensive applications, the reduced cost also implies reduced energy consumption as a co-product.

In more sophisticated scenarios such as both Infrastructure-as-a-Service (IaaS) and Software-as-a-Service (SaaS) being offered, a two-stage optimization model was developed in [93]; for hybrid clouds (i.e., applications deployed to both an on-premises cluster and a public cloud), various techniques [36, 56] were proposed to minimize the overall cost; for cloud federation (i.e., inter-cloud), various techniques [12,32,57] were developed to automate the resource selection and configuration. Although in this dissertation we assume the underlying cloud infrastructure comes from a single cloud vendor (i.e., AWS), there is nothing technical to prevent users from applying the proposed approach to multiple, heterogeneous clouds.

Many other domains (e.g., high-performance computing (HPC) [21,46,65], databases [91, 92], networking [95,102], big data systems [51,103]) are switching from conventional cluster computing to cloud computing and minimizing the cost is also actively researched. Of note, *working set* has been actively studied in HPC (e.g., [23,68]), which refers to the overall size of the program along with the initial and intermediate data to be held in (virtual) memory. The techniques proposed by this dissertation, although mainly targeted on and evaluated on public clouds, have the potential to be extended to apply to other domains as well.

2.2 Optimizations of Kirchhoff-Based Analyses

The Kirchhoff law is one of the most widely used physical laws in many engineering principles, e.g., biomedical engineering, electrical engineering, and computer engineering. There are two main challenges in solving a Kirchhoff-based system of nonlinear equations: (i) the root of unknowns is not unique because the unknowns appear at the denominators of the equations; (ii) the computation for finding the root takes a long, usually prohibitive, time. Several studies in the literature proposed effective methods to optimize Kirchhoff-based analyses. In [41], the MRFSPICE algorithm—a combination of the Metropolis and Besag's ICM (Iterated Conditional Modes) algorithms—was proposed for optimizing highly nonlinear and non-continuous analog circuits using the Kirchoff law. In addition, based on the Kirchhoff Law about an arbitrary sinusoidal steady-state circuit, a principle of dynamic optimization method was adopted by Lin et al. in [63] to compute complicated alternating-current circuit network. In [42], a simple transformation was proposed to efficiently obtain the solutions of the autonomous Kirchhoff equation or system using the known solutions of the corresponding local equation or system. Similarly, an iterative method was proposed

in [29] for solving a nonlinear biharmonic equation following the Kirchhoff law. In [24], an automated Satisfiability Modulo Theories (SMT)-based method was proposed for the formal analysis of Switching Multi-Domain Linear Kirchhoff Networks (SMDLKN).

2.3 Blockchain and Blockchain Simulator

Blockchain is an enabler of many emerging decentralized applications in areas of cryptocurrency, Internet of Things, smart healthcare, among many others. Several researches on blockchain are being under focus among the distributed computing community apart from the main stream blockchain systems like Bitcoin [15], Ethereum [33], and Hyperledger [48]. In order to mitigate the bottleneck with the storage a blockchain framework named Jupiter [43] is designed for mobile devices. Similarly, to alleviate storage bloating problem another framework [28] is proposed based on Network Coded Distributed Storage. To enable customization and enhancement in arbitrary scenarios Inkchain [49] is designed that is built with the flavor of permissioned blockchain based on Hyperledger.

Reliability and security in order to maintain data integrity is being considered another major concern for the distributed ledger technology. BigchainDB [14] is known to have all the features from database (i.e., indexing, querying structured data) and the blockchain properties (i.e., decentralization and immutability) while providing better fault tolerance. To improve the security at the Transport layer, Certchain [22] is proposed. Smart contract technology is leveraged in [45] to make sure the validity of data based on decentralized privacy preserving search scheme. Similarly, 2LBC [8] is designed to manage the data integrity in distributed systems based on leader rotation approach.

Distributed data provenance [26, 27] has been another research attraction among the file system and database communities. Most recently, a consensus protocol called proof-of-reproducibility (PoR) [5] is crafted to manage distributed in-memory ledger for HPC systems in order to support scientific data provenance. For storage-level provenance several I/O optimization techniques [26, 27] for file systems are proposed. There is also an emerging trend for conventional workloads, such as high-performance computing and networking, to move to the cloud platforms [59, 60, 104, 106]. Various solutions [10, 71] are designed for the improvement of provenance in database transactions.

Though, a recent work namely SimBlock [9] focuses to develop a blockchain network simulator that supports changing node behavior on run time in order to investigate behavior of nodes; to the best of our knowledge, this dissertation presents a blockchain framework for the first time that supports emulation with very large scales of nodes in terms of user level threads along with the plug-in facility of custom components (i.e., ad-hoc consensus protocols) for domain specific applications.

2.3.1 Neural Networks for Engineering Applications

Neural networks have been found useful for a wide range of real-world engineering applications. For instance, in automobile applications, neural networks were used to provide an accurate estimation of the remaining energy in high-capacity electric vehicle batteries. Neural networks enabled rapid adaptation to battery nonlinearities as well as changes in drivers and driving conditions that are difficult to model or characterize [89]. In the oilfield services and equipment industry, Coveney et al. [25] utilized neural networks to predict cement compositions, particle size distributions, and thickening-time curves from the dif-

fuse reflectance infrared Fourier transform spectrum of neat cement powders. This, in turn, allowed the estimation of cement quality and detected batch-to-batch variability in cement reliably. In medical engineering, Lim et al. [62] employed neural networks to efficiently and inexpensively screen large populations for diabetic retinopathy, which may lead to blindness if left untreated. The efficiency and accuracy of the neural networks introduced appropriate transformations that exploit general knowledge of the target classes. Similarly, Nigam et al. [70] proposed a method for the automated detection of epileptic seizures from electroencephalograms (EEG) signals using neural networks. The authors asserted that in comparison to the manual approach, the application of neural networks significantly reduced the time required to analyze lengthy recordings.

Compared to the related work discussed above, this work falls in the category of leveraging neural networks (i.e., deep learning) to estimate the computationally-prohibitive solutions to a complex system of nonlinear equations in the context of electrode arrays. Although the idea of employing neural networks in electrode array is not new, this work represents the very first study on efficiently collecting, as opposed to *simulating*, the training data at practical scales. In addition, this work applies various optimization techniques such as parallelization, augmentation, and dimension reduction, to further improve the analysis performance without compromising the accuracy. This work, thus, lays out the road to widely adopting neural networks in real-world engineering applications at scale.

2.4 Unbalanced I/O in Lossy Compression

As the advancement of computational power has greatly out paced capacity and bandwidth of I/O systems over the last decade, storing the whole scientific data has become infeasi-

ble as it will be prohibitory expensive. To reduce the cost of I/O and speed up scientific computations, using compression is a promising direction. Namely, scientific data are first reduced using lossless or lossy compressor before transferring through the I/O systems. Lossless compressors [35, 38, 47, 76, 107] offer the capability of compressing data and preserving bit-wise identical information content in decompressed data. As scientific data become increasingly large with the advancement of scientific simulations and experiments, relative low compression ratios obtained though lossless compression can no longer satisfy both the time and resource constrains in modern scientific computing. As not every bit of the scientific data necessarily contributes to the useful information in data, lossy compression is known as a more favorable approach for greatly reducing the cost of I/O. Especially, to make sure important information contents are not lost during compression, several lossy compressors for scientific data have been proposed with guaranteed error control. For example, SZ [61] lossy compressor is built based on using multiple prediction methods. ZFP [64] lossy compressor is built based block transformation. MGARD [1–4] lossy compressor is built based on multilevel decomposition.

Based on compression techniques, many works has been focusing on applying compression to reduce the cost of I/O. For example, [97] proposes to use lossless compressors such as LZO and BZip2 to reduce the amount of data transferred over the network. They build I/O Forwarding Scalability Layer in the communication libraries so that the compression and decompression are transparent to users' applications. [108] proposes to use both lossless and lossy compressors to reduce the data movement cost between scientific simulation code and in-situ analytics. Based on their evaluation, they propose an adaptive compression service for the in-situ analytics middle-ware. [37] focuses on applying transparent compression between the computing and the storage systems. They build on-line decision system that can predict whether to compress at runtime, allowing guaranteed QoS for the I/O systems. [80] proposes to adaptively applying compression at highly-compressible regions and

perform direct I/O on less-compressible regions to optimize the overall I/O performance.

As lossy compression are continuing to evolve for achieving higher compression ratios, it is anticipated that there will be higher disparity in terms of compressed sizes across regions in scientific data. This could lead to even more unblanaced I/O workload, which results inefficient I/O operations. However, very few existing studies have identified such issue or proposed solutions to improve the I/O performance.

Part I :

Dissertation Contributions

CHAPTER 3

TOWARD COST-EFFECTIVE MEMORY SCALING IN CLOUDS: SYMBIOSIS OF VIRTUAL AND PHYSICAL MEMORY

3.1 Introduction

Data intensive applications usually consume lots of memory, which can be quite expensive if not carefully planned. More importantly, big data applications generate a highly dynamic amount of intermediate data that needs to be persisted in memory for performance. The OOM error in big data applications is often deemed as one of the most frustrating errors as it usually implies that the developers would need to spend a lot of time in further splitting the already complex application with finer granularity, reconfiguring the underlying big data system, redeploying the big data application, and recomputing many time-consuming results. Despite all such efforts, the application is not guaranteed to work without OOM errors—possibly making all the time and resource investment worthless. Part of the challenge comes from the unpredictability of memory footprint when the input data size changes. An intuitive solution would be to estimate the memory footprint based on different input sizes, which, unfortunately, is also challenging because in the real world the relationship between the two could be nonlinear in one of our more recent works [51]. The number of streamlines represents the input size for an Magnetic Resonance Imaging (MRI) application detailed in [66].

One conventional way to scale memory is using operating system's virtual memory (e.g., swap in Unix-like systems). Compared to scaling up physical memory (e.g., selecting a

more powerful instance[1]), virtual memory usually yields degraded performance (thus potentially longer running time and higher cost). Therefore, an effective way to determine the performance and cost impact when using different amount of virtual memory is highly desirable for users to make decisions. To quantitatively study the trade-off between performance and cost caused by virtual memory, the first goal of this work aims at building a performance-cost model that can accurately estimate the performance and cost using virtual memory.

This chapter aims to answer the following research questions: *how could we prevent data-intensive applications from experiencing OOM errors while retaining high resource utilization and low monetary cost.* To this end, we propose two techniques, one building on virtual memory of the operating system (OS) and the other inspired by statistical prediction models aiming to eliminate the overhead of the conventional checkpoint-based mechanism. To summarize, this dissertation focuses on developing a practical system tool for preventing OOM error while balancing the performance and cost trade-off for big data applications in the public cloud rather than making theoretical contributions. Specifically, we make the following three main contributions.

(i) We develop analytic models to predict applications' performance and cost when various portions of virtual memory are used in public clouds. As a result, users will know how, and by how much, virtual memory will affect the overall performance and cost of their applications before actually executing them.

(ii) We propose multiple cost-effective approaches for preventing OOM error in pub-

[1]For memory-intensive applications, memory is the bottleneck, so a more powerful instance with better other resources (e.g., CPU, networking) would not necessarily improve the runtime performance. In the applications studied in this dissertation, we observed less than 100% utilization in resources other than memory capacity, such as CPU cycles and memory access bandwidth.

lic clouds. Instead of periodically checkpointing memory states, the proposed elevation-migration approaches incur only one batch of memory accesss when encountering memory depletion.

(iii) We extensively evaluate the proposed techniques using benchmarks and real-world applications. Experimental results demonstrate high effectiveness of both techniques on AWS.

3.2 Cost-aware Exploitation of Virtual Memory

3.2.1 Overview

For applications whose memory footprint exceeds the available physical memory[2], it is possible to extend the memory usage to the swap space (i.e., virtual memory) with expected performance slowdown due to the data swap between the physical memory and the disk. In theory, the entire hard disk drive can be used as virtual memory; in Unix-like systems, this can be easily configured before the applications starts. Consequently, in theory, an application would unlikely run into out-of-memory (OOM) errors as long as we simply extended the virtual memory to the entire disk assuming the application's data can be accommodated by the disk size; but this might not be always practical because the performance overhead could be prohibitive in the real world.

As a concrete example, we show that an MRI application's performance on two different

[2]By "physical memory", we do not exclude the memory allocation in a virtual machine; it is used in this context only to differentiate the "virtual memory" or "swap space" used in Unix-like systems.

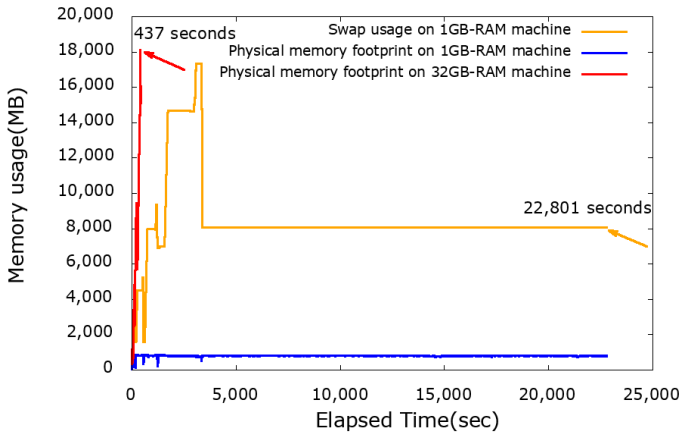


Figure 3.1: Huge Overhead Introduced by Improper Use of Virtual Memory

(and extreme-case) setups of virtual memory in Figure 3.1. The application incurs a peak memory usage of about 18 GB and completes in less than 500 seconds (red line) when the machine is equipped with enough memory (i.e., swap portion P, defined by the occupied swap size divided by occupied swap size plus occupied RAM size, is 0%). However, when we specify a combination of 1 GB physical memory and a 18 GB swap space on the machine (i.e., swap portion $\frac{18}{19} > 94\%$), the same application's total runtime exceeds 20,000 seconds. The red line is the real-time memory footprint when the application runs on a 32GB-RAM machine; the orange and blue lines record the swap and physical memory usages, respectively (on the 1GB-RAM 18GB-swap machine). However, the overall cost on the 1GB-RAM 18GB-swap machine might be lower than running the application on a 32GB-RAM machine.

The key question is: *if the users are willing to trade some time off to complete their jobs on the current instances (with enough swap space without OOM errors*), what would the monetary benefit and time overhead look like, quantitatively? That is, users would know better about the distribution of cost-time correlations in the parameter space between the two extreme-cases presented in Figure 3.1.

3.2.2 Assumptions

We assume the swap space would be able to accommodate the application's memory usage at all times. This assumption is easy to satisfy in the real world, as the capacity of hard disk drives is usually orders of magnitudes larger than physical memory. In addition, adding more hard disk drives is usually one of the most economic upgrades if more swap space is needed.

We also assume the swap portion is known in advance (we address the swap portion unknown scenario in Section 3.3). In many areas such high-performance computing and Mapreduce-like workloads, swap access-patterns including memory footprint are well studied (e.g., approaches including profiling an sample run on a subset of input data). Therefore, as long as the hardware specification of the machines is determined, it is easy to calculate the swap portion based on the physical memory capacity and the application's swap access patterns.

The last assumption is that the pricing of different instance types is fixed. Indeed, there are cases where instance prices are versatile (e.g., AWS's spot instances), but we do not consider those scenarios in our models because they are highly dependent on the non-technical contexts such business models that are beyond the scope of this dissertation.

3.2.3 Methodology

We aim to develop models that characterize the interconnection between applications' performance and instances' swap portion (i.e., "swapness") under various instance types with

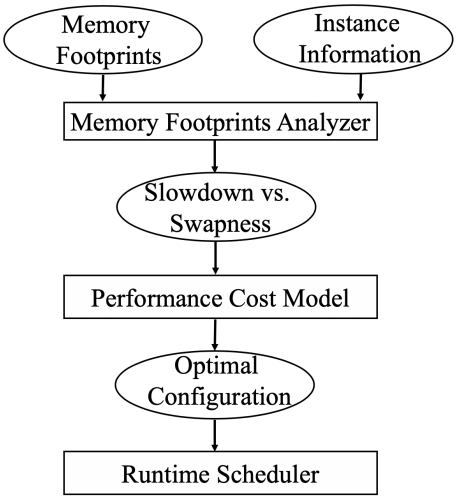


Figure 3.2: Overview of Cost-Aware Exploitation of Virtual Memory Approach

different memory capacities. The overview of just-in-time application migration approach is shown in Figure3.2. The approach needs both application's memory footprints and the information of the instance as inputs, and outputs Slowdown vs. Swapness. The models are crafted for specific memory sizes for two reasons: First, they will achieve higher accuracy than a single global model applied to all memory capacities; Second, most cloud vendors offer a limited number of instance types regarding memory capacities.

To study the correlation between performance slowdown and swap portion in normal running conditions, we start with measuring the swap-introduced slowdown in one of the most widely used matrix operations: matrix initialization. We chose this application as the baseline benchmark because of its representative, i.e., uniform, access to the (virtual) memory. Specifically, a 17,000 $\times$ 17,000 two-dimensional matrix is initialized with random integer values, which incurs about 18 GB peak memory usage. The test bed is an AWS `t2.medium` instance with 4 GB memory. The benchmark application is decomposed into various phases according to their memory usages, while the end-to-end execution times of each phases are recorded with the corresponding swap portion. For data-intensive applications, recording all the information on swap portions at a fine granularity is both space- and time-consuming. To address that, we apply cumulative density functions (CDF) to com-

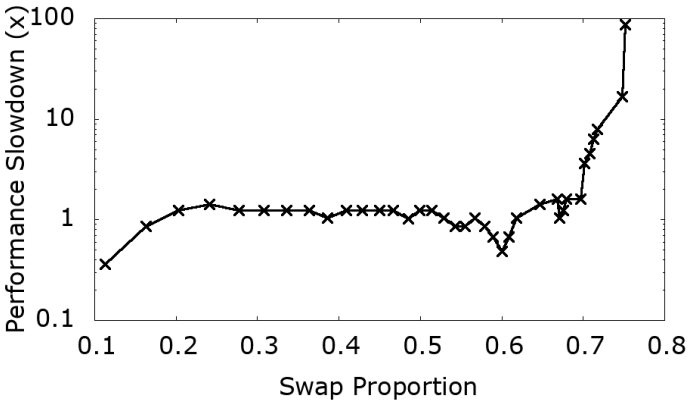


Figure 3.3: Slowdown vs. Swapness with 4GB-RAM Physical Memory

press the numbers and sizes of those records rather than storing all data points in their raw format.

As shown in Figure 3.3, the benchmark exhibits a strong exponential curve between the slowdown and the swapness. Although this is the trend exhibited with 4GB physical memory, similar trends are indeed observed with other memory capacities (we will discuss more when evaluating the system in Section 5.4). In theory, a higher proportion of swap space should imply a super-linear increase in swap access cost if the replacement policy is least-recently-used (LRU) [105]. As a result, the skeleton of the model we chose to fit is exponential in the following form:

$$S = \alpha \cdot e^{\beta \cdot x} + \theta, \tag{3.1}$$

where S is the performance slowdown, e is Euler's number, x is the swapness, and (α, β, θ) are coefficients to be determined during the model fitting.

The following illustrates how we fit the model for performance slowdown and swap portion. We segment the benchmark's execution into pieces with different swap portions at

runtime. For each swap proportion (i.e., swapness), we maintain a bucket to store the number of pieces falling into the bucket to save space (i.e., a cumulative density function, CDF). We applied binary searches on each of the coefficients in the model and determine them when the overall error is minimal. The resultant model is:

$$f(x) = (3.09E - 13) \cdot e^{44.21x} + 0.35 \tag{3.2}$$

Because of the fluctuations exhibited by Figure 3.3, we also provide confidence intervals (maximum and minimum) as follows:

$$f_{max}(x) = (1.16E - 10) \cdot e^{36.34x} + 13.33 \tag{3.3}$$

$$f_{min}(x) = (4.48E - 11) \cdot e^{34.89x} + 0.075 \tag{3.4}$$

The f_{max} models a subset of the benchmark data points landing on the top-left edge, while the f_{min} models the subset of the benchmark data points landing on the bottom-right edge. We will be using the above models to predict more real-world applications and report its accuracy and more importantly, the correlation between the overall monetary cost and the performance, in Section 5.4.

3.3 Just-in-Time Application Migration

3.3.1 Overview

There are various reasons why users decide not to use swap space for out-of-memory errors. For instance, the applications might be highly memory access-intensive and using swap, even by a very small portion, would slow the application down by orders of magnitude. As another example, the application's memory footprint and memory access patterns are not well studied, then the techniques proposed in Section 3.2 are not applicable.

The question now becomes: *Without introducing swap, how could we reduce the overall monetary cost in face of memory depletion?* Obviously, the risk of encountering memory errors would become the lowest if users started with the most powerful (and, expensive) instance to run the applications; doing so implies, however, the highest chance of underutilization of the memory resources and consequently incurs unnecessary monetary cost. One heuristic approach would be starting with the cheapest instances and then migrate the application to a larger instance when memory is depleted, which means additional performance overhead from periodical checkpointing and relaunching virtual machines.

Recall the significant overhead of checkpointing as shown in Figure **??**. Ideally, the migration should occur when only absolutely necessary as it may introduce performance degradation such as starting and warm up the memory of the new instance, transferring the data from old instance to new instance, i.e., at the point just before the memory errors out, what we called *just-in-time application migration* in the following discussion. The terminology is inspired by the well-known compiler technique "just-in-time compi-

lation," meaning that the source code is only compiled at runtime rather than prior to the execution—one of the unique features in functional programming languages like Lisp. It is worth to point out that here we focus on predicting "when" to do migration instead of "how" to do migration, so any live migration is compliment arty to our approach proposed in this section.

3.3.2 Assumptions

We assume the application is migrated between different instance types without physical data movements. This is not true in conventional clusters but very common in cloud vendors, for example in AWS the current instance can be shut down with saved status (i.e., snapshot) and then get restarted with larger memory capacity allocated along with other possible upgrades. Microsoft Azure and IBM BlueMix provide similar functionalities. Note that, in conventional cluster computing, a failed node's data are first checkpointed from memory to disk and then transferred to a healthy node, usually incurring a higher overhead.

Another assumption is that swap is completely excluded. That is, the swap portion in the remainder of this section is 0%. Indeed, there is nothing preventing us to combine the models in Section 3.2 and what we will discuss in this section, meaning that swapness can be between 1% and 100% in the real world. In practice, both approaches work with SWAP. It is also worth to note that our experimental results show that even with SWAP on, OMM can happen if the memory demand is much larger than the RAM capacity. However, this section will focus on only the techniques of just-in-time application migration, which is isolated from others so we know how, and by how much, just-in-time application migration

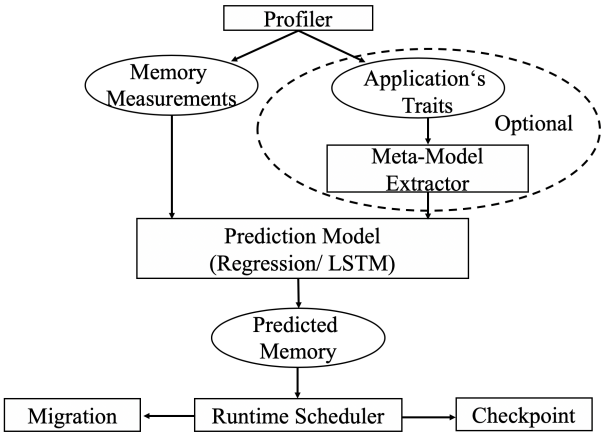


Figure 3.4: Overview of Just-In-Time Application Migration Approach

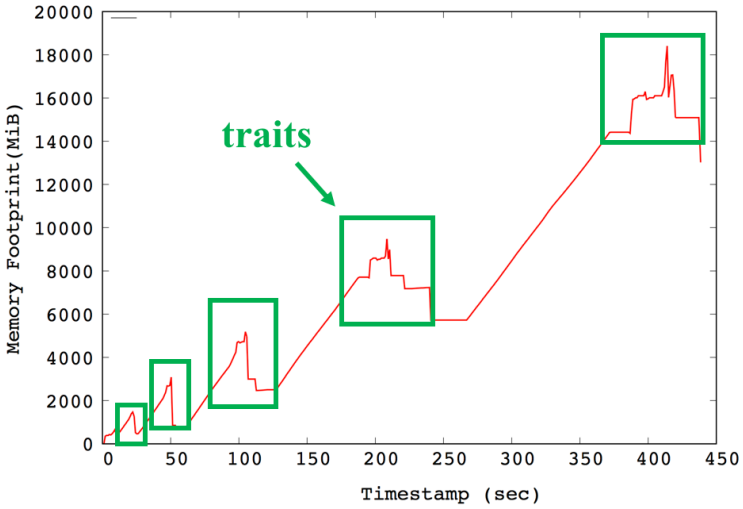


Figure 3.5: Memory Footprint of MRI

can facilitate the cost reduction.

3.3.3 Methodology

Since we plan not to apply periodical checkpointing, the key challenge is how to accurately predict when the memory will error out. Our approach considers both statistical models and the hint extracted from the application. The overview of the approach is shown in Figure 3.4. Specifically, our approach takes multiple fitting models (e.g., polynomial, exponential)

and looks back different numbers of data points (i.e., history depending on certain decay functions). Conventional models simply look back a certain number of data points, apply regression to achieve the least aggregate errors, and calculate a future data point; In contrast, our approach considers those fitting models as inputs and takes the application's own traits extracted from profiling a small portion of execution as another input, both of which constitute a higher-level of model that we call *Meta-Model* (MM). That is, the meta-model we propose is not only fit by the existing data but also correlated to the application's sample runs. Figure 3.5 shows the memory footprint of MRI over time when running on a 32G memory instance. There are 5 similar shaped repeated patterns highlighted with boxes and we call the repeated pattern as trait. As the the application running, the trait becomes bigger in terms of memory footprint and time span. Therefore, by modeling the size of the traits, we can predict future traits' memory footprint and time span. Assume we use f_1 and f_2 representing the scale of application's traits, f_3 representing the position of the traits, we can derive a quadratic model as following:

$$MM(x) = f_1 \cdot x^2 + f_2 \cdot x + f_3 \tag{3.5}$$

where the vector $\vec{F} = [f_1, f_2, f_3]$ represents the relation between the coefficient and the adjusted impact to the application. In the simplest form, $\vec{F}$ can be a linear transformation according to the swap access patterns we can observe from profiling a subset of sampled data points. In other words, our model extends the conventional fitting approaches by generalizing those constant coefficients into additional function that characterizes the application's own information.

After calculating the prediction values of multiple meta-models, we compute the prob-

ability of memory crash. The checkpoint and migration is triggered when 90% of RAM capacity is reached to offset the prediction error. If the probability falls below a threshold, the application will be paused and ready to be migrated to another instance (with larger memory capacity). Also, we adjust the aggressiveness of prediction algorithm based on the immediate history RAM usage and thus can avoid the accumulation of the errors. Specifically, if the predicted memory value is smaller than the measured value, we will make our prediction algorithm more aggressive. An alternative approach is to run a vote from all the participating meta-models and the majority wins (either continue with the current instance or migrate to a larger one).

If the system decides (or, predicts) that a memory crash is to occur, the application will be migrated to the least expensive instance type that has larger memory capacity than the current running instance. The reasoning is that in most public cloud vendors, the memory capacity roughly follows an exponential pattern (i.e., 2 GB, 4 GB, 8 GB, and so forth). Our protocol is conservative and hopes that doubling the memory capacity could satisfy the application's memory requirement. A more aggressive protocol is possible, for example instead of increasing 2× memory size we can multiple 3, 4, or even larger factors. In practice, doing so would imply missing some intermediate instance types. This dissertation will only discuss the scenarios where all the instances are strictly ordered by the memory capacity. Finding out the optimal factor of multiplying memory capacity is an interesting question and might be addressed in our future work.

3.4 Evaluation

3.4.1 Experimental Setup

All experiments are carried out on AWS [7]. The instances for various memory capacities and prices are listed in Table 4.1. We implement our models using Python and Shell scripts. The source code is available at the project website: https://www.cse.unr.edu/hpdic/proj/cme.

Table 3.1
AWS INSTANCES USED FOR EVALUATION

Instance Name	Memory Capacity (GB)	Price (US$ per Hour)
t2.medium	4	0.0464
t2.large	8	0.0928
t2.xlarge	16	0.1856
t2.2xlarge	32	0.3712

We evaluated the proposed techniques with three real-world applications. The first application is an MRI image processing scientific application from [66]. The second application is a data mining application on taxi's location data from Didi Inc described at [30]. The third application is a common numerical application on adding dense matrices extracted from the popular Python library Numpy [75].

3.4.2 Cost-aware Exploitation of Virtual Memory

The goal of this section is two-fold: a) demonstrating the accuracy of the proposed performance model with various real-world applications; b) reporting quantitative monetary benefit (and the compromise made on performance) when different portions of swap space are taken into account. All the experiments were carried out with 4GB-RAM instances on

AWS (i.e., `t2.medium`).

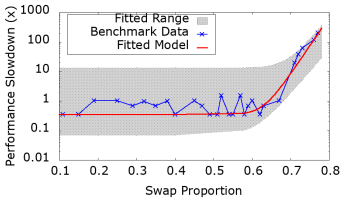


Figure 3.6: Swap model for MRI [66]

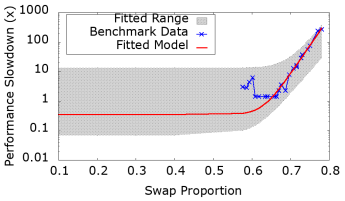


Figure 3.7: Swap model for Didi [30]

Figure 3.8: Swap model for AddMatrix [75]

Figure 3.6 shows the slowdown of the MRI application when swap portion is between 0.1 and 0.8. We do see some fluctuations in the real slowdown, which is expected as arbitrary (e.g., not LRU) memory-access patterns occur in this application. However, the trend still follows an exponential curve in the big picture. More importantly, all data points fall into the ranges (i.e., the gray shadow) of the model, indicating a high accuracy of the proposed model. To quantify that, we apply the Pearson correlation coefficient (PCC) defined as:

$$\text{PCC} = \frac{\Sigma_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\Sigma_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\Sigma_{i=1}^{n}(y_i - \overline{y})^2}} \tag{3.6}$$

where n indicates the total number of data points, x_i and y_i indicate the real data and modeled data, and $\overline{x}$ and $\overline{y}$ indicate the mean of each data series. The correlation of the real data points and our model is extremely strong, as the PCC turns out to be 0.994.

Similarly, Figure 3.7 shows the slowdown of the data mining application and the swap portion is between 0.55 and 0.8. The portion range is different from the first application because this one is more memory-hungry (or, more memory-intensive); that is why the swap space starts to kick in directly from the 0.55—more than half of the memory is "virtual". Because of the memory-hungry nature of this application, we observe even more fluctuations than the MRI application. But again, most of the real data points fall into the

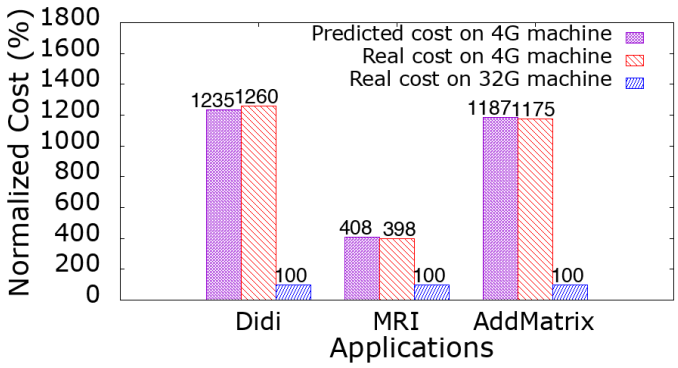


Figure 3.9: Execution time normalized to the 32G-RAM machine

prediction intervals (i.e., the gray shadow) and the PCC of this application is also extremely high: 0.988.

Lastly, Figure 3.8 shows the slowdown of the matrix adding application and the swap portion is between 0.1 and 0.8. This application exhibits a similar pattern as MRI; the PCC coefficient is also extremely high: 0.991.

Figure 3.10 compares the cost predicted by the proposed model and the real cost on the 4GB-RAM instance, both of which normalized to the cost on the 32GB-RAM instance that provides enough physical memory for both applications (the peak of real memory footprint is about 18 GB). Since in all cases the application is not migrated between instances, the cost is the same in terms of both execution time and monetary cost. We can see that the cost predicted by our model is highly accurate: The error rates are: Didi 4%, MRI 1%, and AddMatrix 3%. The figure also shows that using swap does not guarantee reduced cost: the cost of the Didi application using swap is increased by more than 158%, for MRI the cost is reduced by 50%, and the cost for AddMatrix is increased by 47%. The root cause of these results is that the Didi application is highly memory-intensive, meaning that introducing swap on a memory-restrained instance will likely incur much more running time outweighing the benefit of the cheaper per-hour price. The findings, in fact, prove the effectiveness of our model: our model can tell, in a very high accuracy, that whether

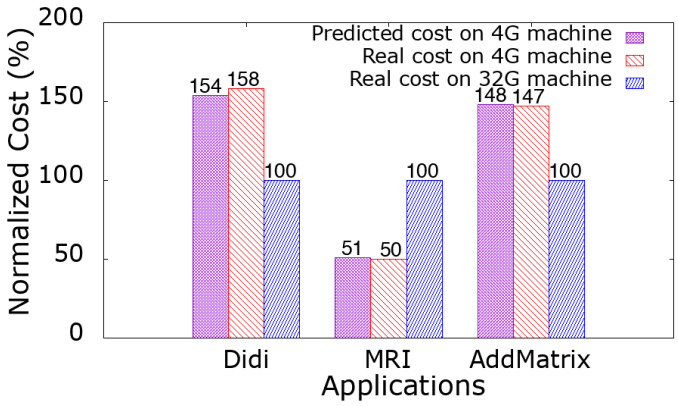


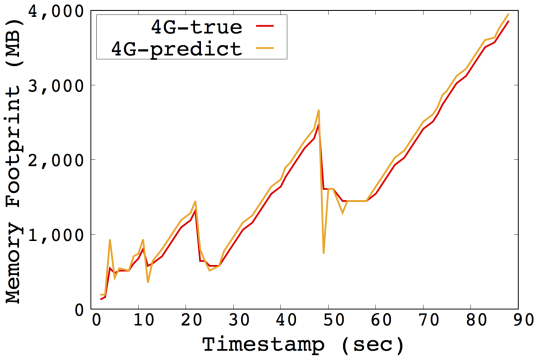
Figure 3.10: Monetary cost normalized to the 32G-RAM machine

applying swap on a memory-constrained and inexpensive instance would result in higher or lower total cost.

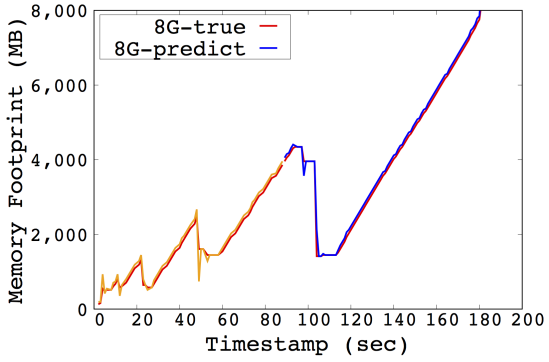
We also report the execution time for all three applications in Figure 3.9. We observe an order of magnitude higher time cost for both Didi and AddMatrix, which is partially attributed to their I/O intensiveness between swap and physical memory and therefore cause the overall monetary cost exceeding the baseline case. What is more interesting, however, is the MRI application. The MRI results make a strong case for trading off performance for cost: by compromising 4X running time the overall cost can be reduced to half. This is exactly the point of the first contribution of this dissertation—allowing users to make compromise between time and cost.

3.4.3 Just-in-Time Application Migration

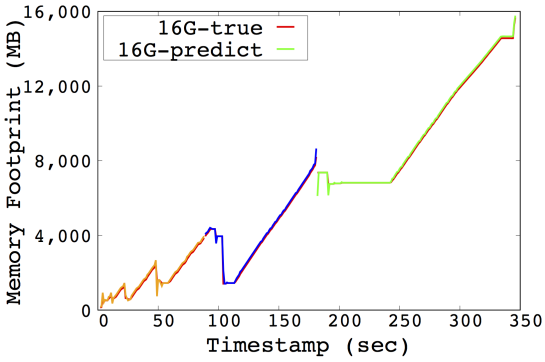
This section answers the following questions using real-world applications: a) illustrating the real-time performance for applications continuously deployed to a series of cloud instances in an increasing order of memory capacities (application migrates to another instance when the current one's memory is to be depleted); b) reporting the quantitative



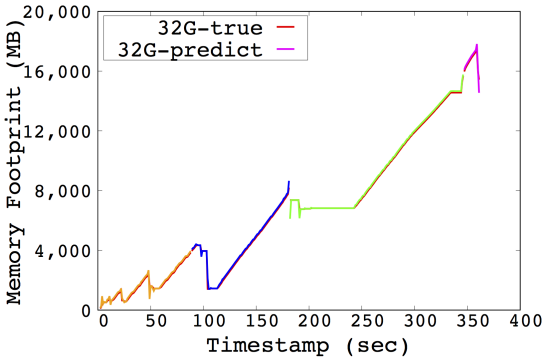
(a) Stage 1 (4GB-RAM)



(b) Stage 2 (8GB-RAM)



(c) Stage 3 (16GB-RAM)



(d) Stage 4 (32GB-RAM)

Figure 3.11: Memory elevation after migrating applications

monetary benefit when applying the proposed techniques of predicting memory footprint and elevating memory capacity.

Figure 3.11 shows the MRI application's memory footprint over its entire course using our prediction-elevation approach. The lifespan of the application comprises four stages on four instance types: 4GB-, 8GB-, 16GB-, and 32GB-RAM instances, as shown in sub-figures. For each stage, we plot the true values and the predicted values using the approach we discussed in Section 3.3. As we can see, our predicted values are highly accurate for most of time, except that at around 50 seconds the predicted value is noticeable lower than the true value. The reason for that is because the application just completed a memory-

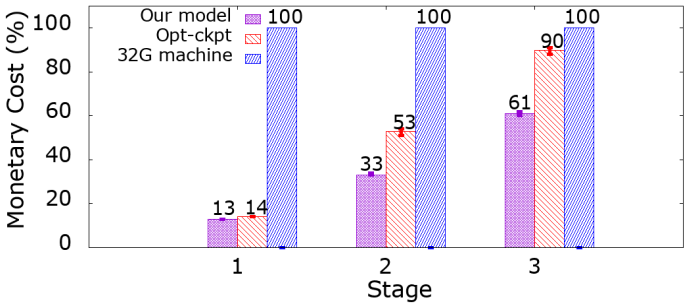


Figure 3.12: Cost comparison (normalized to the 32GB-RAM machine)

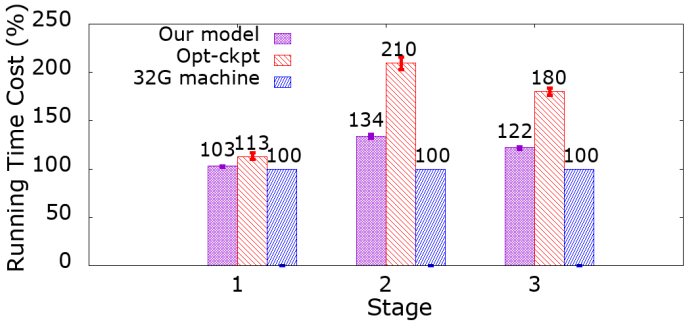


Figure 3.13: Time comparison (normalized to the 32GB-RAM machine)

intensive iteration and the system is busy with memory recycling (i.e., garbage collection). We will further fine-tune our model by considering such corner scenarios in our future work.

Figure 3.12 illustrates the overall costs at various stages normalized to the baseline cost (we did not show the 32GB-RAM comparison since they all incur the same cost). We can see that for all types of instances, our proposed approach incurs the least cost—significantly cheaper than both the checkpoint approach and the baseline with all physical memory allocated. Specifically, our approach costs only 12% of the baseline approach at 4GB-RAM instances, and 33% and 60% on 8GB-RAM and 16GB-RAM instances, respectively. The optimal-checkpointing approach is also cheaper than the baseline but more expensive than our proposed approach: 14% (vs. 13%), 53% (vs. 33%), and 90% (vs. 61%) on the above instances. Overall, our approach takes only 35% of the baseline cost and 66% of the optimal-checkpointing approach.

Indeed, the saved cost does require more running time, and Figure 3.13 reports the time overhead on each type of instances normalized to the baseline running time. Again, we did not report the 32GB-RAM case since the numbers would look exactly the same. We can see that for each instance type, the proposed approach does not incur as significant overhead as the optimal-checkpoint does. For the first three stages, the proposed approach takes 3%, 34%, and 22% more time than the baseline while the optimal-checkpointing approach takes much more: 13%, 110%, and 80%. Taking all together, the time overhead of the proposed prediction-elevation approach is only 22% but saves 65% cost comparing to the baseline.

3.5 Summary

This dissertation presents two techniques to help deploy big data applications with dynamic and intensive memory footprint on cloud-based big data systems with low monetary cost. The first approach assumes the users are well aware of the application's swap access patterns such as uniform access, and the proposed performance-cost model can accurately predict how, and by how much, virtual memory size would slow down the application and consequently, impact the overall monetary cost.The second approach removes the assumption of *a priori* memory access patterns by proposing a lightweight memory usage prediction methodology. The key idea is to eliminate the periodical checkpointing and migrate the application only when the predicted memory usage exceeds the physical allocation of the big data systems based on dynamic meta-models adjusted by the application's own traits. Taking both techniques together, this work covers a wide spectrum of big data applications regarding the trade-off between performance and cost using both virtual and physical memory scaling approaches.

CHAPTER 4

HDK: TOWARD HIGH-PERFORMANCE DEEP-LEARNING-BASED KIRCHHOFF ANALYSIS

4.1 Introduction

In various disciplines such as medical engineering and healthcare devices, scientists can only measure the end-to-end electrical resistance values that are derived from the complex, nonlinear transformation of individual resistances [73]. Nonetheless, the objective of many domain-specific applications is to find intrinsic resistance values, and the orthodox approach is to solve the system of large numbers of nonlinear equations formed according to the Kirchhoff law [53].

Solving a Kirchhoff-based system of nonlinear equations poses two technical challenges: (i) the root of unknowns is not unique because the unknowns appear at the denominators of the equations; (ii) the computation for finding the root takes a long, sometimes prohibitive, time. For instance, Niu et al. [72] reported that forming the Kirchhoff equations would take hundreds of days even for a small-scale 40×40 electrode array. Although [72] presented an algorithm to reduce the number of equations from exponential to polynomial, how to efficiently solve them remains an open challenge to both the biomedical engineering and parallel computing communities.

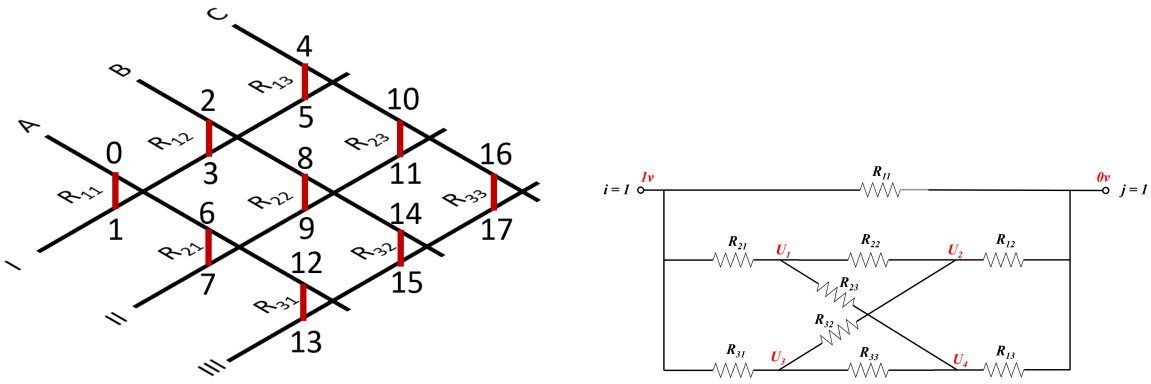
Inspired by the recent advances in deep learning (DL), researchers started to seek non-analytic paradigms to *estimate* the solution, e.g., training a convolutional neural network

(CNN) to accurately predict the unknown resistor distribution in an electrode array [88]. This approach demonstrated to be an effective means to "learn" the nonlinear function between inputs and outputs, as the error rate is reported as low as 0.49%.

However, before the CNN-based approach can be practically adopted, one critical issue concerning performance must be addressed. Specifically, the CNN-based approach [88] simply *simulates* the input and output data for the neural network, assuming the training set is known a priori, which is not the case in the real world. In fact, it is one of the most challenging problems to efficiently obtain the training data set for an electrode array due to the computational complexity of Kirchhoff nonlinear equations. As shown by Niu et al. [72], a commodity workstation took less than one minute to generate all the nonlinear equations for a 20×20 electrode array, but then the time increased to more than four hours for a 100×100 electrode array—prohibitively slow in the practice. This is partly why in [88] authors *simulated* tens of thousands of training data and tested it on a small-scale 16×15 electrode array[1]. The slow performance has significantly hindered CNN-based Kirchhoff analysis from being broadly adopted in real-world applications.

To overcome the above challenges, this dissertation proposes a high-performance DL-based method for Kirchhoff analysis, namely HDK. HDK employs two specific techniques to improve the performance of DL-based Kirchhoff analysis: (i) early pruning of unqualified candidate inputs and (ii) parallelization of forward labelling. To retain the high accuracy of the DL models, HDK also introduces random errors to augment the training set and at the same time applies dimension reduction to the original data. Collectively, the aforementioned techniques reduce the analytical time from hours to 15 minutes.

[1] In literature, an electrode array can be organized in a ring or a grid topology. Of note, in [88], the array is organized as a 16-electrode ring, which is equivalent to a 16×15 two-dimensional array referred to by [72].



(a) Electrode array with $n = 3$ (b) Converted vertex-based topology (for one path)

Figure 4.1: A 3×3 electrode array, where each of the 18 vertices follows the Kirchhoff law.

4.2 Problem Formulation

Formally, let vector $\vec{Z}$ represent the set of measured values, the goal is to find the vector $\vec{R}$ that satisfies $\vec{Z} = K(\vec{R})$, where $K(\cdot)$ indicates the Kirchhoff law. The Kirchhoff law states that *the aggregation of the incoming current flows must be equal to the aggregation of the outgoing ones*. The law must be applied to each joint on an electronic device, therefore usually comprising a system of equations, which is why the constraints are represented by vectors.

To make matters more concrete, we exemplify $\vec{Z} = K(\vec{R})$ in a 3×3 electrode array, as shown in Figure 4.1a. One set of axes (A, B, and C) are interconnected to another set (I, II, and III) on nine resistances: $R_{11}, \cdots, R_{33}$. In scientific experiments, we could only measure the resistant values from the endpoints, indicated by Z_{ij}'s. For instance, $Z_{B,II}$ is the measured resistance value between axis B and axis II. An equivalent, vertex-oriented topology can be achieved; Figure 4.1b shows the converted topology for a specific path between the axis A (i.e., $i = 1$) and axis I (i.e., $j = 1$). The actual voltage will not matter as the measurement is all on the resistances; for the sake of simplicity, we assume the source voltage is 1 volt, and the end (ground) voltage is 0 volt, as indicated in Figure 4.1b. Let

R_{ij} indicate the resistance on the i-th x-axis and the j-th y-axis. Without loss of generality, the end-to-end measured resistance on the first x-axis and the first y-axis must satisfy the following system of equations as per the Kirchhoff law:

$$\begin{cases} \frac{1-U_1}{R_{21}} = \frac{U_1-U_2}{R_{22}} + \frac{U_1-U_4}{R_{23}} \\ \frac{U_2}{R_{12}} = \frac{U_1-U_2}{R_{22}} + \frac{U_3-U_2}{R_{32}} \\ \frac{1-U_1}{R_{31}} = \frac{U_3-U_2}{R_{32}} + \frac{U_3-U_4}{R_{33}} \\ \frac{U_4}{R_{13}} = \frac{U_1-U_4}{R_{23}} + \frac{U_3-U_4}{R_{33}} \\ \frac{1}{Z_{11}} = \frac{1}{R_{11}} + \frac{U_2}{R_{12}} + \frac{U_4}{R_{13}} \end{cases} \tag{4.1}$$

where U_k ($k \in \{1, 2, 3, 4\}$) indicates one of intermediate voltages at joints, as illustrated in Figure 4.1b. The first four equations represent the four Kirchhoff constraints, and the last equation calculates the composite resistance between $i = 1$ and $j = 1$.

In practice, only Z's can be measured, and all U's and R's are unknowns. Although our goal is to find R values, voltage values U's have to be found as well. Note that Equation 4.1 speaks of the relation between a single pair of $i = 1$ and $y = 1$ only. The entire network would have nine systems of nonlinear equations in the form of Equation 4.1, making the total number of constraints equals 45. Also, note that in each equation we have four unknown U's, and there are nine unknown R's shared by all nine equations. Therefore, there are a total of 45 unknowns as well. That is, we have the same number of unknowns and constraints—a well-defined system of equations to solve. In a more general sense, there are $n^2(2n - 1)$ constraints and $n^2(2n - 1)$ unknowns for a given $n \times n$ electrode array.

However, the problem is proven ill-posed, meaning that the unknown variables cannot be uniquely and stably reconstructed from measurements alone [11]. If we look closer at

the unknowns in Equation 4.1, the majority comes from the voltages, i.e., $n^2(2n-2)$ of U's, and only a small fraction of them are R's: n^2; unfortunately, such a small fraction of R's (i.e., $\frac{1}{2n-1}$) are the root cause of the technical challenge because they all appear in the denominators, which makes the entire system of equations *nonlinear*. The problem then becomes ill-posed because:

- **Non-uniqueness of Root.** According to the rank of an augmented matrix, we could determine whether a system of linear equations has no root, a unique root, or infinite numbers of roots. For a system of nonlinear equations like Equation 4.1, we know there must be at least one root because they follow the physical Kirchhoff law. Literature [82] shows that there exist ways to find all possible roots; and yet, in this application, there should be only one real root, and we have no computational means to verify whether the found root is the real one.
- **Computational Complexity.** Unlike a system of linear equations, most algorithms for solving nonlinear equations are problem-dependent and a lot more complicated. Literature [72] shows that solving a system of nonlinear equations derived from a 20×20 electrode array takes more than five hours. In addition, the time-scale trend follows an exponential pattern: the solving procedure takes about 10 seconds for a 10×10 electrode array. Note that, in practice, the scale is much larger than 20; our device prototype is 64×64. The computation time increases 2,000 times from $n = 10$ to $n = 20$, and will not be acceptable at $n = 64$.

4.3 Methodology

4.3.1 Early Pruning: Upper-Bound of Input Parameters

In theory, the input data Rs can be arbitrary, although the values are usually within a specific range depending on the applications. That is, the parameter space of Rs is infinite: any positive real numbers are eligible. How to select a meaningful input data range remains an open problem. What we need here is to prune off the impossible candidates, in the context of electrode arrays, before selecting the training data set Rs. We thus must explore the intrinsic properties embedded in this specific engineering application. In the remainder of this section, we will pursue an analytical model for estimating the input Rs' range derived from the measured Z values.

To start with, we introduce a set of parameters, which constructs a parameterized model for the topology shown in Figure 4.1b. Instead of having a 3×3 array, we assume the dimension n is sufficiently large for practical usages, e.g., $n = 64$ in our prototype device. It follows that there are $(n-1)$ horizontal paths between a starting and ending endpoints. Each of these horizontal paths comprises three resistors with two distinct voltages in between, namely U^k_{left} and U^k_{right}, where k ($1 \leq k \leq n-1$) indicates the path index, $left$ indicates the left voltage and $right$ indicates the right voltage (assuming the current flows from left to right). It should be clear that the three resistors on those $(n-1)$ have nothing to do with the dimension of the sample 3×3 electrode array: all these paths would have three resistors anyways regardless of the dimension n [72]. In addition to the horizontal paths, there are $(n-2)$ "cross" paths starting from each U^k_{left}.

If we look closely at the topology from endpoint i to endpoint j in Figure 4.1b, the current at U_1 is diluted into two currents toward R_{22} and R_{23}. On the other hand, the current at U_2 is aggregated from currents through R_{22} and R_{32}. Therefore, we observe a distributive and symmetric divide-and-conquer workflow in this engineering application. If we assume, for the sake of analysis simplicity, the resistors follow a uniform distribution in a general $n \times n$ electrode array, then the current must satisfy the following

$$I_{ik} = (n-1) \cdot I_{kk} = I_{kj} \tag{4.2}$$

where we denote the start point as i, end point as j, and I_{ik} as the current between i and U^k_{left}. According to Ohm law, it follows that

$$U^k_{right} = (n-1) \cdot (U^k_{left} - U^k_{right}) \tag{4.3}$$

Consequently, if n is sufficiently large in practice, we then have $U^k_{left} \approx U^k_{right}$. An important implication is then the resistors between both intermediate voltage points are computationally negligible. Formally, we have

$$\lim_{n \to \infty} R_{kk} = 0 \tag{4.4}$$

for a current topology from i to j where $k \neq i$ and $k \neq j$. It should be clear that this conclusion is for the computation between a specific pair of endpoints; it does not imply the physical resistor is cut down to zero at large scales.

Equation 4.4 lays out a cornerstone for further analysis of the targeting parameter space. With Equation 4.4, we could safely eliminate all the Rs in the intermediate positions, i.e., those that are not adjacent to either endpoint. Back to the example in Figure 4.1b, it means

that we can remove R_{22}, R_{23}, R_{32}, and R_{33} when we compute the current from $i = 1$ to $j = 1$. It follows that the simplified topology now has only n horizontal paths: the top one comprises a single R_{ij}, and each of the remaining $(n-1)$ horizontal paths comprises two resistors R_{ik} and R_{kj}. It follows that, again, if we assume a uniform distribution of R, then according to the Kirchhoff law between endpoints i and j:

$$\frac{1}{Z} = \frac{1}{R} + (n-1) \cdot \frac{1}{2R} \tag{4.5}$$

or,

$$R = \frac{n+1}{2} \cdot Z \tag{4.6}$$

Therefore, we can pick the training data Rs in the range of $\left(0, \frac{(n+1)\cdot Z}{2}\right]$, where Z is the measured value and n is the array size.

4.3.2 Parallel Forward Labelling

We propose to label the data set in the forward direction, which is parallelized with multiple processes. Conventionally, if we have a general relationship $K(\cdot)$ between $\vec{R}$ and $\vec{Z}$ s.t. $\vec{R} = K(\vec{Z})$ where $\vec{Z}$ is the set of measured values and $\vec{R}$ is the set of unknown individual resistances, then we must have obtained the $\vec{R}$ calculated from $K(\cdot)$ along with the $\vec{Z}$. As discussed above, $K(\cdot)$ is prohibitively expensive to calculate. Therefore, we ask: can we obtain both $\vec{R}$ and $\vec{Z}$ through the inverse function of $K(\cdot)$, namely $K'(\cdot)$, such that:

$$\forall r \in \vec{R} : z = K'(r) \;\; \texttt{iff} \;\; r = K(z) \tag{4.7}$$

and we hope that the inverse function $K'(\cdot)$ is significantly easier to solve than $K(\cdot)$. This is known as a forward problem.

If we reexamine Equation 4.1, the inverse function $\vec{Z} = K'(\vec{R})$ consists of only one unknown in the denominator: Z_{11}. In fact, we could solve all the unknown U's in the first four equations and calculate Z_{11} trivially. In other words, the inverse function $K'(\cdot)$ can be built by solving n^2 systems, each of which comprises $(2n-2)$ linear equations and $(2n-2)$ unknowns. This is the simplest form for solving systems of equations: all unknown coefficients are linear, and the number of unknowns is equal to the number of equations. Therefore, a much simpler (inverse) function becomes available for generating the training set.

With a linear number of unknowns and the new topology, our next optimization is to parallelize the axis-oriented Kirchhoff equations. There are n^2 possible end-to-end measured resistance values from an $n \times n$ electrode array. Even if we have reduced the number of equations from n^2 to $2n-2$, there are still a total of $2n^2(n-1)$ equations from the entire array—we hope that parallelization could significantly reduce the time for processing these many equations.

The key challenge is how to isolate the shared z-axes between different paths across pairwise endpoints, such that these paths can be calculated in parallel. Although each z-axis is touched by a single x- and y-axis, respectively, when those unknowns are boiled down to the Kirchhoff equations, all of the unknowns are interconnected and cannot be trivially parallelized.

Our approach to parallelizing the Kirchhoff equations is to leverage an essential property of circuits: as long as the voltage and underlying individual resistance are kept unchanged,

the measured end-to-end resistance and circuits value are also unchanged. That is to say, regardless of how many times we measure the network, the results should be the same; it does not matter whether we measure all the n^2 end-to-end resistance values at once or n^2 times—basically one single pair of (i, j) each time, $1 \leq i, j \leq n$. Therefore, we could parallelize the $2n^2(n-1)$ equations along with the combinations of x- and y-axis. Indeed, this parallelization is still somewhat coarse-grained: we can parallelize the n^2 factor and each thread still needs to process $(2n-2)$ equations. In practice, however, n^2 could be a fairly large number and implies a high degree of parallelism. Take our device prototype for example again where $n = 64$, the proposed approach can be parallelized by up to 4,096 threads. Although linear scalability is unlikely in practice due to various overhead, n^2 threads imply strong parallelism; we will report quantitative results in the evaluation section when we apply the message passing interface (MPI) to the equations.

4.3.3 Augmentation through Random Errors

In real-world engineering applications, measurement errors are the norm to exist. To take this factor into account, we employ a procedure to randomized the $\vec{Z}$'s when we train the model. Another benefit of doing so is the fast augmentation of the training set. As discussed before, the training set is parallelly generated from the inverse calculation based on the Kirchhoff law. Nevertheless, it still takes tens of minutes to find a single mapping between a 64×64 $\vec{R}$ and $\vec{Z}$, as we will see in the evaluation section.

One tricky question for taking this approach is how much randomization should be applied. In this context, the degree of randomization is two-fold: (i) for a single element, what is the upper and lower bound of the randomized input and output; and (ii) how many

new inputs and outputs should be generated from the randomization. In engineering applications, 5% errors are usually acceptable; we control all the randomized data within 1% of the exact values from Equation 4.1. From the application's perspective, there is no hard requirement over the number of new inputs and outputs. However, the number of extended mappings between $\vec{R}$ and $\vec{Z}$ might significantly influence the model accuracy, as we will see in the evaluation.

4.3.4 Dimension Reduction

One common challenge in data analysis of real-world engineering applications lies in the high dimension. For instance, the electrode arrays from our wet lab are 64×64, totaling in 4,096 dimensions. Instead of directly training over these 4,096 features, we apply principal component analysis (PCA) to the measurements and hope to improve the accuracy of the model.

The critical question in the proposed approach is how many dimensions we want to reduce the original data. In theory, the dimensions should be collectively tuned by both PCA and DL models. That is, the PCA should remain much information (usually 95+%) with a small subset of dimensions and at the same time, such a small set of dimensions result in a few neurons in the DL model leading to high accuracy in predicting the intrinsic resistance values R's. We will experimentally show that the optimal choice of the reduced dimensions will be application-dependent, and usually resides close to the point where less than 1% – 5% information is lost from the PCA.

4.4 Evaluation

4.4.1 System Implementation

We have implemented the proposed method, mainly with Python and MPI, and released the source code hosted at Github: `https://github.com/hpdic/HDK_On_Electrode_Arrays`.

4.4.2 Experimental Setup

4.4.2.1 Test Bed

Our experiments are primarily carried out on the Amazon Web Services (AWS). Table 4.1 lists the instance types for the evaluation. For the parallel forward labelling experiment, we use both of the *t2.2xlarge* and the *c5.18xlarge* instances. For model training, we use only the *t2.2xlarge* instance. All instances are installed with Ubuntu 18.04, Anaconda 2019.3, Python 3.7, NumPy 1.15.4, SciPy 0.17.0, mpi4py v2.0.0, and mpich2 v1.4.1.

Table 4.1
AWS INSTANCES USED FOR EVALUATION.

Instance	CPU	Memory
t2.2xlarge	Intel Xeon E5-2686	32 GB
c5.18xlarge	Intel Xeon Platinum 8124M	144 GB

4.4.2.2 Data Sets and Baseline Systems

We evaluate the proposed approach using two data sets. Both data sets are collected from our wet lab with measured Zs using 64×64 electrode arrays. The core data set comprises 100 data points, which are augmented into 1,400 data points through up-to-1% random errors.

According to Equation 4.6, the input Rs should be in the range of $\frac{64+1}{2} = 32.5\times$ of the measured Zs. We name the first data set as *cell medium*, representing the estimated original range of data in a biological experiment. Specifically, the *cell medium* data set has its R's about 15×–25× of the measured values. The second data set is called *wound surface*, representing an application where the electrode array is applied to a patient's wound skin in a clinic. The typical R's values are about 25×–35× of the measured Zs.

For both data sets, we hold out 20% of the data for prediction, and the remaining 80% for training. We use 10-fold cross-validation and report the variance in the error bar if it is noticeable.

We compare the proposed work with two baseline systems recently published: (i) the conventional Kirchhoff analysis on electrode arrays [72] and (ii) a CNN-based Kirhohoff analysis on electrode rings [88]. The metrics include scalability, accuracy, and performance.

4.4.2.3 DL Models

The models are trained through Keras [52], a high-level API built upon TensorFlow [90]. Models in Keras are defined as a sequence of layers. We create a sequential model and add layers one at a time until the network reaches high accuracy. The number of inputs are initially set to $input\_dim = 512$ for both models. In both models, we use a fully-connected network structure with two hidden layers, and the number of neurons is 2,048. Table 4.2 illustrates the network architecture of the DL models.

Table 4.2
DL NETWORK ARCHITECTURE FOR 64×64 ARRAYS.

Layer	Input	Dense-1	Dense-2	Output
Neuron	100–1,400	2,048	2,048	100–1,400

During the training, we initialize the network weights with a custom initializer (*He initialization*) as we use the rectifier *relu* activation function on the first two layers for a reduced likelihood of vanishing gradient. We apply no activation function to the output layer because we aim to train a regression model instead of a classification model. When compiling, we use the *mean_squared_error* loss function and the gradient descent algorithm *adam* that appears to be highly efficient for our data.

4.4.3 Speedup from Parallel Forward Labelling

This section reports the performance of the parallelized generation of training data set following the Kirchhoff law. Specifically, we report that to solve a system of equations derived from a 64×64 array, it takes almost two hours (6,787 seconds) to solve the system of 126 equations using a serial implementation, as shown in Figure 4.2. The majority of computa-

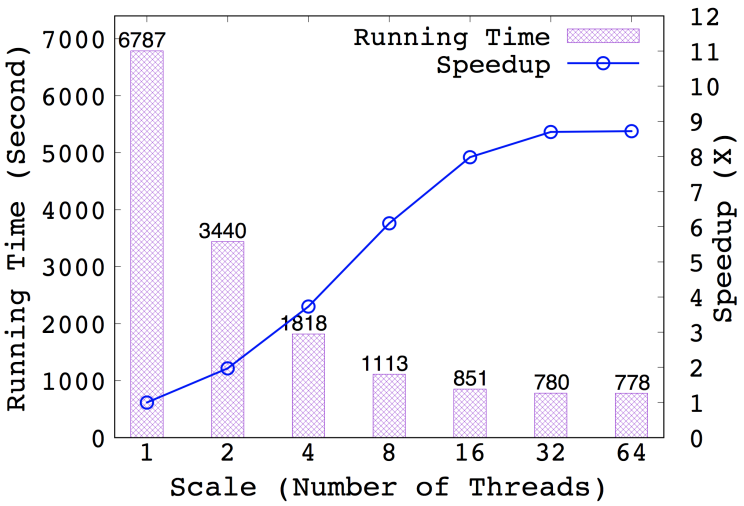


Figure 4.2: Time for labelling data from $\vec{R}$ to $\vec{Z}$.

tion time comes from the generation of the equations. To this end, we explore the potential parallelism at a fine-granularity. Specifically, we break down the equations into smaller batches, each of which is assigned to a rank in the message passing interface (MPI) [67]. This parallelism is only possible because the converted, vertex-oriented equations can be formed independently.

Figure 4.2 also reports the performance when various levels of parallelism are applied. Of note, we observe significant speedup on 2, 4, 8, and 16 threads—exhibiting almost linear scalability. However, the improvement becomes marginal when more threads join the computation, and the speedup stops at 8.7× with 64 threads. This can be best explained by the fact that the underlying I/O overhead is maximally amortized; that is, the I/O bandwidth is almost saturated by 16 concurrent threads. Therefore, we believe scaling out the application into multiple physical nodes would further improve the performance by having multiple I/O devices; we will leave this as our future work that focuses on the scalability of DL-based Kirchhoff analysis.

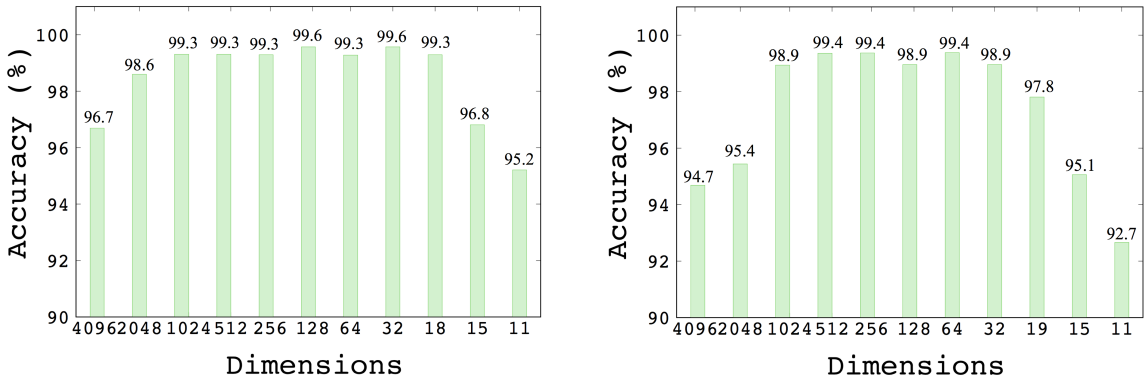
It should be noted that this conclusion should not be generalized as the experiment con-

sists of 126 equations, and the underlying hardware is a commodity solid-state disk (SSD) drive. And yet, regardless of specific applications and hardware, the resource is expected to be saturated at some point. A general model would be useful and is an interesting research question, which is beyond the scope of this dissertation.

4.4.4 Sensitivity and Overhead of Dimension Reduction

Figure 4.3 reports the accuracy when we reduce the original 4,096-dimensional data into various levels. The total number of data points is 1,200 for all cases; as we discuss before, 1,200 is sufficient to train accurate models (see Figure 4.5). The left-most columns indicate the accuracy of the models with no dimension reduction over the original data: 96.7% and 94.7% for the cell medium and wound surface data, respectively. After reducing the dimension to 1,024, both models are significantly improved to be around 99% accuracy. The high accuracy then does not change much until a very low dimensionality is reached: for the cell medium data set, the accuracy drops to 96.8% on 15 dimensions; for the wound surface data set, the accuracy drops to 97.8% on 19 dimensions. This can be best explained by the over-reduction where too much information is lost after PCA. Both experiments exhibit a convex curve over the accuracy, which is not accidental: neither high dimensionality nor overly-reduced dimensionality would be the optimal choice. We believe this is an interesting research question to find out the optimal dimensionality for ANN models and will address this in our future work.

We report the reduced dimensions with the corresponding information retained from the PCA in Table 4.3. The minimums of input and output neurons (i.e., dimensionality) for both data sets are 8 and 10, respectively, when 45% information is lost; in order to maintain



(a) Cell Medium (b) Wound Surface

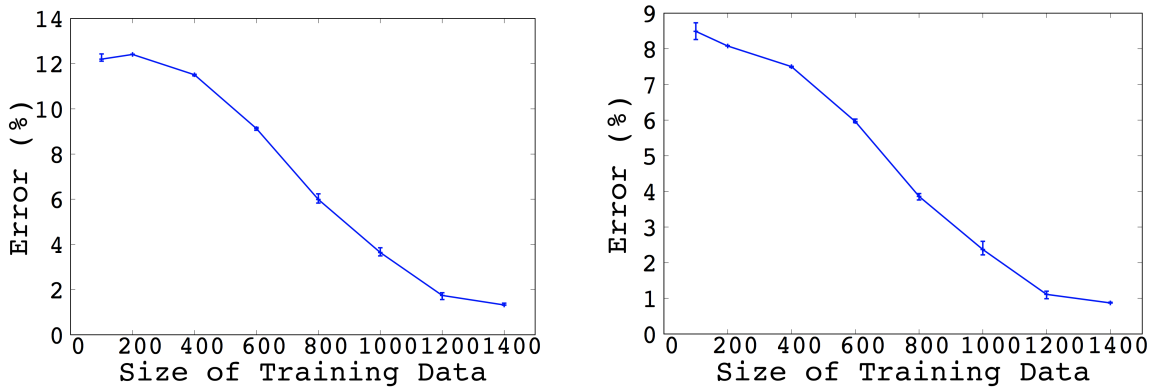
Figure 4.3: Accuracy with various dimension reductions.

99% of the information, both input and output in two data sets require 19 neurons. This result is aligned well to our findings on accuracy as reported in Figure 4.6: in cell medium, the accuracy reaches 99.3% with only 18 dimensions; in the wound surface, although the accuracy reaches a bit lower than Cell Medium (97.8%), a slightly higher dimensionality (i.e., 32) leads to an accuracy of 98.9%, and the 64-dimension becomes highly accurate, 99.4%.

Table 4.3
RETAINED INFORMATION AT REDUCED DIMENSIONS.

Information (%)	55	65	75	85	95	99
Cell Input Dim. (#)	8	10	12	14	17	19
Cell Output Dim. (#)	10	12	14	16	18	19
Wound Input Dim. (#)	8	10	12	15	18	19
Wound Output Dim. (#)	10	12	14	16	19	19

The overhead to reduce the dimensions is 135 seconds for all cases. The reason why it takes noticeable time is that mainstream libraries (e.g., scikit-learn [81]) do not support PCA at specific low dimensions, which is the case for electrode arrays. To that end, we implemented a PCA module from scratch specifically for this application. It should be clear



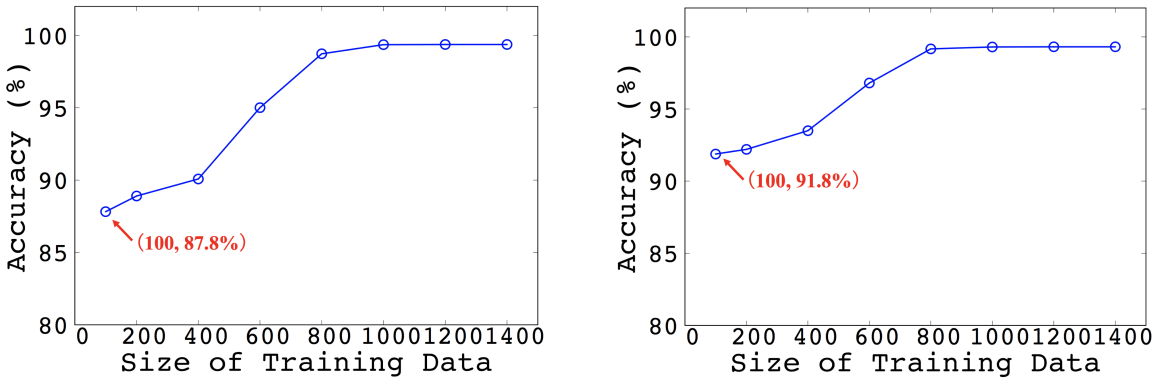
(a) Cell Medium (b) Wound Surface

Figure 4.4: Error rate of 10-fold cross-validation.

that the overhead at this level, i.e., 100s seconds, is negligible compared to the overall training time for the application. The performance of our implementation is highly dependent on the dimensions of the input data. When we apply the PCA implementation to smaller matrices, $2,048 \times 2,048$ matrices take 19 seconds, and $1,024 \times 1,024$ matrices take only four seconds.

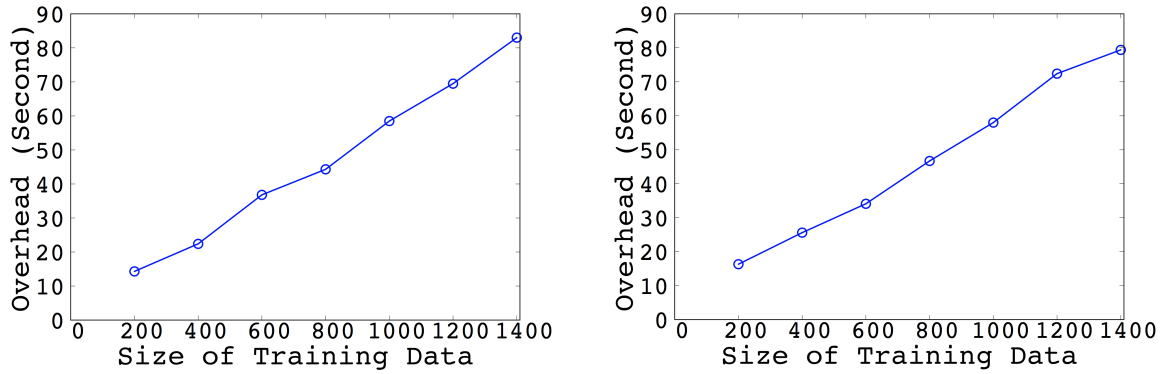
Figure 4.4 shows the error rates of 10-fold cross-validation at different augmentation scales. The variances are also plotted as error bars at each point. When the data set is augmented into 1,400 points, the error rate reaches around 1% with unnoticeable variances.

Figure 4.5 reports the prediction accuracy when various degrees of augmentation is applied to both data sets (with dimensionality = 512), respectively. Both original data sets comprise of only 100 data points, and we apply the randomness to augment the data sets into 200, 400, 600, 800, 1,000, 1,200, and 1,400 data points. With only 100 data points, we can only achieve 87.8% and 91.8% accuracy, respectively, which are unacceptable in real-world engineering applications. However, the accuracy gets drastically improved and passes 99% when 800 data points are available. Over-fitting issues start to appear when more than 800 data points are involved. How to find the optimal number of augmented



(a) Cell Medium (b) Wound Surface

Figure 4.5: Accuracy over various sizes of training data.



(a) Cell Medium (b) Wound Surface

Figure 4.6: Performance overhead of augmentation.

data points is beyond the scope of this dissertation; in practice, because the randomness can be quickly achieved, a trial-and-error approach is feasible.

Figure 4.6 reports the overhead to augment two data sets through random errors. The linear correlation between the overhead and the size of the randomized data points is expected: the computational and I/O time should be proportional to the augmentation of the input data. We also observe that the overhead is in terms of tens of seconds. It should be noted that this overhead is counted as part of the training phase; Given that the core training data might take hours to collect (i.e., 24 hours for about 100 points), this overhead is negligible in the entire training phase.

4.4.5 Put Everything Together: End-to-End Comparison

We compared the proposed HDK method with two baseline systems recently published: (i) the polynomial Kirchhoff analysis (PKA) method on electrode arrays [72] and (ii) a CNN-based Kirchhoff analysis (CKA) on electrode rings [88]. We re-implemented the baseline systems and deployed them to the same testbed when comparing them with the proposed HDK method. Table 4.4 illustrates their accuracy and performance on the same testbed along with the largest scales initially reported by the respective work.

Table 4.4
COMPARISON BETWEEN STATE-OF-THE-ART METHODS.

Method	PKA (2018)	CKA (2019)	This Work
Scale	20×20	16×15	64×64
Accuracy	100%	92.5%	99.6%
Time	5+ hrs	2.1 hrs	15 mins

In terms of accuracy, the CNN model (CKA) described in [88] yielded an accuracy of 92.5%, while HDK achieved accuracy higher than 99%. In the electrode array applications, 1% or lower error rate is acceptable due to the existence of measurement errors (which we leverage to augment the training data, actually). PKA always calculates the exact root of the Kirchhoff equations, thus exhibiting 100% accuracy.

The overall end-to-end execution time of the proposed HDK method takes about 15 minutes for processing 64×64, or 4,096-dimensional, electrode arrays. The majority portion comes from preparing the training data. CKA [88] was a proof-of-concept over simulated 16-dimensional data and did not report the real execution time. Since CKA was not open

source and did not have a phase for collecting real training data, we re-implemented a neural network with a forward model and deployed it to our testbed. Overall, CKA took about 2.1 hours to collect the data and train the CNN model. PKA [72] reported its analytics time as 5.6 hours over a 20×20, or 400-dimensional, array; its training time over a 64×64 array took about seven minutes. We re-implemented PKA with parallelized data training: we were able to reduce the training time from seven minutes to three minutes, and yet the calculating (i.e., Kirchhoff equations) takes a similar time, i.e., more than five hours, on only a 20×20 sub-array and does not stop for larger scales in a reasonable time (a couple of days). In summary, the proposed HDK is orders of magnitude faster than PKA with negligible accuracy loss and delivers more than 8× (2.1 hours / 15 minutes) performance improvement over CKA without compromising the accuracy.

4.5 Summary

This dissertation proposes a new deep-learning-based approach to efficiently conduct Kirchhoff analyses that are widely used in various engineering fields. The new approach employs multiple techniques including early pruning of unqualified input data, parallelization of forward labelling, data augmentation through random errors and dimension reduction, all of which collectively enable an efficient and accurate mechanism for large-scale Kirchhoff analyses. We implement the proposed approach with the latest machine learning framework and the parallel computing library, and evaluate it with real-world engineering applications showing promising results: the accuracy is as high as 99.6% with up to 8× performance improvement over the state-of-the-art.

CHAPTER 5

TOWARD ACCURATE AND EFFICIENT EMULATION OF PUBLIC BLOCKCHAINS IN THE CLOUD

5.1 Introduction

Blockchain, a decentralized and immutable database, has drawn a lot of research interests in various communities, such as security [17,55], database [6,31], network [34], distributed systems [101], and high-performance computing [5]. Although many existing blockchain frameworks [33, 48] are open-source and offer docker images accessible in major cloud vendors (e.g., Google Cloud, Amazon Web Services (AWS), and Microsoft Azure), there are yet more challenges for blockchains to be widely adopted, such as (i) the lack of resources to carry out large-scale experiments and (ii) much, if not prohibitive, engineering effort to modify sophisticated production (despite open-source) systems to timely test out new ideas. To this end, multiple blockchain simulators were recently developed, two of the most popular ones being Bitcoin-Simulator and VIBES.

Bitcoin-Simulator [39] follows the same architecture and protocol of Bitcoin [15], the foremost application in cryptocurrency built upon blockchains. Users of Bitcoin-Simulator can specify various protocol and network parameters, such as the number of nodes and network bandwidth. The main goal of Bitcoin-Simulator is to study the trade-off between performance and security. Because of its design goal, Bitcoin-Simulator simulates the execution of a blockchain network at the block level rather than the transaction level. Bitcoin-Simulator does not provide a fine-grained control over the application, limiting its applica-

bility for broader adoption. In addition, Bitcoin-Simulator simply inserts a series of static time stamps to simulate the proof-of-work (PoW) consensus protocol, which does not precisely characterize the behavior of real-world blockchain systems: for instance, Bitcoin dynamically adjusts the PoW difficulty and the nodes (as known as miners) usually complete the tasks in stochastic time intervals. Last but not least, Bitcoin-Simulator's network is built upon NS3 [74], a discrete-event network simulator, which limits the scalability on up to 6,000 nodes. Bitcoin network currently consists of more than 10,000 nodes [16], implying that Bitcoin-Simulator cannot simulate the entire network of Bitcoin as of the writing of this dissertation.

VIBES [86] extends Bitcoin-Simulator with the following improvements. First, VIBES supports a web-based interface for users to visually track the growth of the network. Second, VIBES improves the scalability of Bitcoin-Simulator by employing a fast-forwarding algorithm, which essentially designates a coordinator to control the events according to existing nodes' best guess on the block creation time. Such a centralized coordinator might be acceptable for a single-node simulator at small- or medium-scale, and yet could be a performance bottleneck for extreme-scale applications. Similar to Bitcoin-Simulator, VIBES takes the same approach of inserting time stamps to hypothetically carry out the PoW workload. Both Bitcoin-Simulator and VIBES are coarse estimators of real-world blockchain executions due to the lack of real PoW implementations or a decentralized architecture.

This dissertation presents **BlockLite**, the very first blockchain *emulator* with both high accuracy and high scalability. In contrast to Bitcoin-Simulator, BlockLite comprises a specific module to execute real PoW workload[1], supports fine-grained transaction management, and scales out to 20,000 nodes thanks to its efficient network communications built upon distributed queues along with PoW preprocessing that incurs negligible runtime

[1]Thus making it an *emulator* rather than a simulator

overhead. Different than VIBES, BlockLite is fully decentralized with no single point of failure or performance bottleneck. It should be noted that even on a single node the decentralization philosophy of blockchains still holds for a blockchain *emulator* because each user-level thread is now considered as an individual node.

5.2 System Design

The objective of BlockLite is to provide blockchain researchers and practitioners an easy-to-use and lightweight emulator to develop new components and evaluate new ideas, such as ad-hoc consensus protocols customized for domain-specific applications. To achieve that objective, BlockLite is designed to be deployed to a single node, with loosely-coupled components for flexible customization. In its infrastructure, BlockLite has implemented all the building blocks for a basic blockchain system. This section details how these common facilities are designed, the challenges we encounter, and the approaches we take to build up the emulator.

The high-level architecture of BlockLite is illustrated in Figure 5.1. While the interface, i.e., BlockLite API, will be detailed in §5.3, the infrastructure can be broken down into three categories: storage, computation, and networking. In the context of blockchains, they are usually referred to as *distributed ledgers*, *consensus protocols* (e.g., PoW), and *network communications*.

Distributed Ledgers. The transaction data of a specific blockchain are replicated, either fully or partially, in distinct files each of which is associated to a hypothetical node. The data, also called ledgers, could have been partially duplicated if the following two con-

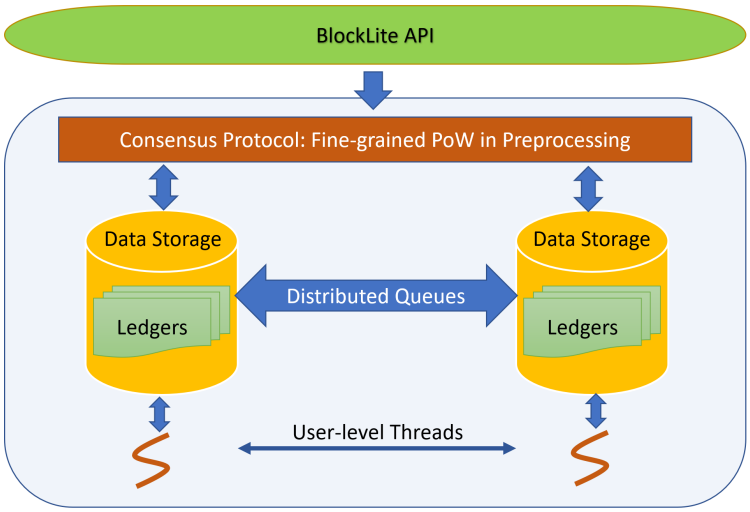


Figure 5.1: BlockLite Architecture.

ditions are satisfied: (i) more than 50% of nodes have agreed on that the new block (of transactions) is valid and (ii) the current node (other than those nodes who have voted) does not process any request regarding the new block. Regardless of specific consensus protocols, a blockchain requires only 50% votes supporting the new block's validity.

Consensus Protocols. A basic proof-of-work protocol is implemented from scratch in BlockLite. In contrast to other simulators where the difficulty is *simulated* by time delay and timestamps, BlockLite, as an *emulator*, conducts the real PoW workload by solving the puzzle. The puzzles we define in BlockLite are similar to the Nakamoto protocol in Bitcoin [15] in the sense of comparing blocks' hash values against predefined thresholds. Nonetheless, BlockLite exhibits an additional feature that preprocesses PoW allowing for fine tuning of puzzle difficulty, which is detailed in §5.2.1.

Network Communications. It is one of the most challenging components to emulate the networking in BlockLite that is designed to be working on a single node. Fortunately, BlockLite is designed for emulating public blockchains that are based on PoW, which is

compute-intensive rather than network-intensive[2]. Therefore, the real network impact for PoW-based blockchains lies in the network infrastructure's latency rather than bandwidth. BlockLite applies a statistical estimation of time delays for transmitting the messages between nodes, each of which is emulated by a user-level thread whose requests are buffered in a distributed queue. We will discuss the distributed queue in more detail in §5.2.2.

5.2.1 PoW Preprocessing for Fine-grained Calibration across Heterogeneous Systems

One cornerstone of Nakamoto consensus protocol, or any PoW variants, is the puzzle-based winner selection:[3] the hash value of the (block of) transactions is compared against the predefined "small" number. Because BlockLite is designed to be running on an arbitrary node that can be heterogeneous case by case, we must provide an efficient yet flexible mechanism to ensure the compatibility across heterogeneous machines. To this end, we design BlockLite puzzles as follows. A puzzle's difficulty is expressed by two sub-fields, L and M, in the form of $L.M$ (assuming SHA256 [83] is used as the hash function): L indicates the required number of leading zeros in the 64-hex (i.e., 256-bit) hash value; M indicates the minimal number of zeros in the middle of the hash value.

Figure 5.2 illustrates how $L.M$ is constructed. The first part L is semantically equivalent to the Nakamoto protocol: checking whether a hash value is smaller than a predefined threshold is essentially the same to counting the number of leading zeros in the binary or hex form of the hash value. L is a coarse-level adjustment of difficulty because the same L

[2]Private blockchains are indeed network-intensive due to the quadratic number of messages.
[3]As known as "leader" in the context of distributed systems

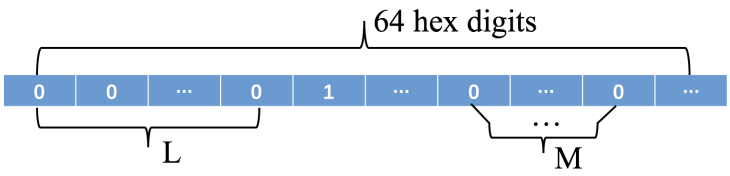


Figure 5.2: Two-phase Puzzle for Efficient Preprocessing of PoW in BlockLite.

might imply a wide spectrum of computation time, and this is exactly why Bitcoin dynamically adjusts the difficulty every 2016 blocks [15]. To address that, BlockLite introduces the M part to allow the system to check whether there are M zeros in the middle of the hash value satisfying the following conditions: (i) Any leading zeros in L are not considered; (ii) Tailing zeros, by definition, are counted towards M; and (iii) Zeros need not be continuous.

The benefit of the additional M-zero checking is that we can adjust the puzzle difficulty under the same meta-difficulty, i.e., same L but different M's. In addition, M is positively correlated to the puzzle difficulty: a larger M implies more computation time. To see this, we can think of a larger M representing a super-set of the sets of less zeros with smaller M's. As a consequence, a smaller M has a higher chance to meet the requirement—the difficulty is lowered.

While the flexibility is significantly improved, one limitation of this $L.M$ two-phase puzzle is that the two arbitrary difficulty numbers do not follow partial orders in terms of computation time. That is, if $T(\cdot)$ indicates the computation time of a specific difficulty, it is possible that

$$T(L_1.M_1) > T(L_2.M_2) \text{ and } L_1 < L_2, \tag{5.1}$$

if M_1 is significantly larger than M_2. The root cause of this counter intuition is that L and M are, essentially, incomparable. For instance, if L is much smaller than M, then finding out M zeros, despite from random positions, is still much harder than locating a few leading zeros. As an extreme case, if we have two setups as $L_1.M_1 = 1.63$ and $L_2.M_2 = 2.1$,

obviously the former case is much harder where we will seek for a hash value with all 64 zeros, as opposed to finding a hash value with two leading zeros and another zero from any of the remaining 62 hex digits.

5.2.2 Optimization for Extreme-scale Networking through Distributed Queues

In contrast to existing blockchain simulators, BlockLite does not simply insert timestamps for the the completion of PoW; instead, it solves the real puzzle to accurately *emulate* a real blockchain system. The downside of this approach is the cost and overhead for large-scale systems. For instance, Bitcoin has about 10,000 mining nodes as of January 2019 [16]. A single machine, despite its multi- or many-cores, is not able to efficiently emulate tens of thousands of nodes each of which works on a compute-intensive puzzle such as finding out a qualified hash value.

BlockLite overcomes the scalability challenge by delegating one node (thread) to solve the puzzle in a preprocessing stage and when the real application runs at a specific difficulty, the assigned nodes (or, threads) simply replicate the behavior of the delegation node. In doing so, BlockLite achieves the best of both worlds: real execution of PoW and low overhead (i.e., high scalability). Since the calibration is carried out in a preprocessing state, no runtime overhead is introduced.

The second technique taken by BlockLite to achieve high scalability is the usage of queue-based network communication. Specifically, we implement a priority queue who manages all the events in the order of their creation time. That is, the head of the queue

always points to the earliest event, followed by later events each of which is requested by a specific node. Therefore, the queued events implicitly determine the the orders of nodes completing their tasks (e.g., submitting transactions, solving puzzles, appending blocks), which significantly reduces the network traffic.

5.3 Implementation and Interface

BlockLite is implemented in Java with about 2,000 lines of code. We have been maintaining the source code at `https://github.com/hpdic/blocklite`.

Because users' machines are equipped with different resources, the very first step to deploy BlockLite is to calibrate the parameters in accordance to the system's specification. For instance, a throughput of 10 transactions per second might require 7.x difficulty on a high-end server with 32 cores, and the same throughput might require 4.x difficulty on a mainstream laptop with four cores. The calibration, also called PoW preprocessing, is to allow BlockLite to adjust the difficulty by considering factors input by users (e.g., expected throughput, consensus protocols) as well as system specification (e.g., number of cores, memory size).

When BlockLite runs for the first time, it generates a `difficulty-time` map between difficulty levels and the execution time. This map is implemented as a HashMap and is accessible to all the nodes. Whenever a node is waken up according to the consensus protocol, the node will consult with the difficulty-time map and replay the behavior with controlled randomness.

Algorithm 1 Calibration

```
1:  function runMillsOfDiffcults
2:      mainDifficult ← 1
3:      for i ∈ {0 ··· MAX_DIFF} do
4:          subDiffcult ← 0
5:          for j ∈ {0 ··· MIN_DIFF} do
6:              Start Timer
7:              powProof ← newProofWork(i, j);
8:              mineBlock();
9:              End Timer
10:         end for
11:     end for
12: end function
```

Specifically, the emulator starts by asking the user to specify the values of two parameters: MAX_DIFF and MAX_SUBDIFF. The emulator then goes on to repeatedly mine the blocks in a nested loop as shown in Algorithm 1. The complexity of the algorithm is $O(n^2)$ by observing the two levels of loops, one for the main difficulty and the other for the sub difficulty.

Algorithm 2 Mine Block

```
1: function mineBlock
2:     nonce ← 0
3:     n ← mainDifficult + subDiffcultf
4:     target ← A string of ⌊mainDifficult⌋ 0's
5:     while !blockID.startwith(target) or countOfZero(blockID) <n do
6:         nonce ++;
7:         blockID ← calculateHash(lastBlockID + timeStamp + nonce + ...);
8:     end while
9: end function
```

Algorithm 2 illustrates the mechanism of BlockLite to mine a specific block. The main difficulty `mainDiff` corresponds to L in Figure 5.2, `subDiff` indicates the auxiliary difficulty corresponding to M, and `miner()` is the implementation method of mining. The overall complexity is therefore $O(n)$, where n indicates the number of attempts before we find out the qualified nonce number.

BlockLite provides an easy-to-use interface for users to plug in application-specific components. Listing 5.3 illustrates a simplified code snippet of the interface with two core

methods. Users can implement both methods to inject customized consensus protocols. For instance, `generateProof` solves the puzzle; in PoW, this means to check many nonce numbers until the hash value of the combined data satisfies the condition.

Therefore, when the node wants to create a new Block, the program will call `mineBlock()` in block.java to solve the puzzle, i.e., select the appropriate difficulty to control the mining time according to the difficulty calibration map.

```
1 public interface Provable {
2     public boolean verifyProof(Block);
3     public String generateProof(Block);
4 }
```

Listing 5.1: BlockLite Plug-in Interface.

Both aforementioned methods take as input a `Block` object, whose fields are explained in Table 5.1. The class comprises all the necessary information for the system to manipulate the blocks and more important, the transactions—the dominant data format in blockchain-based applications. A more detailed declaration of the class can be found in the source code.

Table 5.1
BLOCK MEMBER VARIABLE

blockID	ID of the Block
creationTime	Creation Time of Current Block
creatorID	Creation ID (Node ID) of Current Block
parentBlock	Parent Block of Current Block
depth	Depth of Current Block
previousHash	Hash Value of Parent Block
childList	Child List of Current Block
numChild	Numbers of Current Block Children
txnList	Numbers of Current Block Transactions
proof	Consensus protocol (Nakamoto by default)

5.4 Evaluation

5.4.1 Experimental Setup

We perform extensive experimental evaluation of BlockLite on AWS. We pick four different instance types, the processors of which represent a wide spectrum of CPUs from both Intel and AMD. Table 5.2 lists the instance types along with their processor specification.

Table 5.2
DIFFERENT AWS INSTANCES USED FOR EVALUATING BLOCKLITE

t2.large	2 Intel(R) Xeon(R) CPU E5-2686 v4 @2.30GHz
t2.2xlarge	8 Intel(R) Xeon(R) CPU E5-2686 v4 @2.30GHz
c5.18xlarge	72 Intel(R) Xeon(R) Platinum 8124M CPU @3.00GHz
m5a.12xlarge	48 AMD EPYC 7571

The data we use for our evaluation are transactions in the same format of Bitcoin trades. Specifically, more than two million transactions are fed into the emulator for real-scale applications. We repeat all experiments for five times and report the average; we do not report the variance if it is unnoticeable.

5.4.2 Overhead

We report the overhead to find the map between the difficulty and the block-creation time in Figure 5.3. Specifically, we build the maps of four VMIs for the following three example intervals: 10 minutes, 20 minutes, and 30 minutes. It should be noted that, however, these overheads are incurred only during the preprocessing stage and would not be applied to the real-time execution.

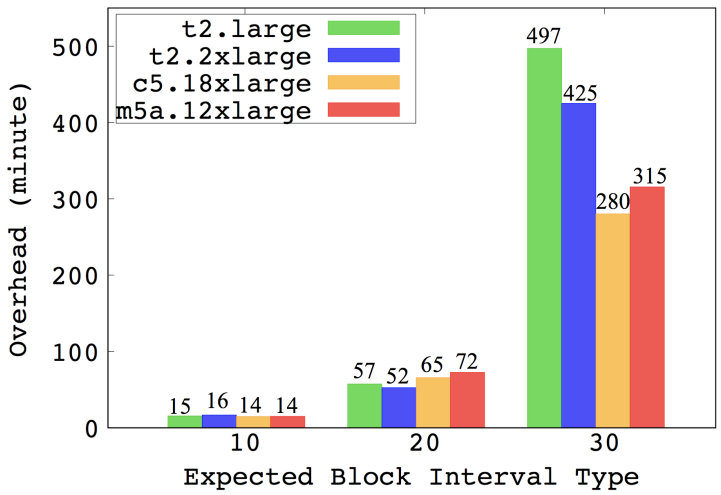


Figure 5.3: Overhead of Various Instances.

We can observe that the overhead increases at an exponential rate with respect to the intervals; take the `t2.large` instance for example, the overhead increases by 3.7× from 10 to 20 minutes and then increases by 8.7× from 20 to 30 minutes. Another observation is that these VMIs do not exhibit much difference in overhead at shorter intervals like 10 or 20 minutes; and yet, for longer interval like 30 minutes, the smaller instances (i.e., `t2.large` and `t2.2xlarge`) indeed incur much longer overhead (than `c5.18xlarge` and `m5a.12xlarge`). This phenomenon can be best explained by the fact that smaller VMIs are equipped with slower CPUs that need to solve the same puzzle in a longer time.

5.4.3 Accuracy

Figure 5.4 shows the calibration map of the *m5a.12xlarge* instance. For practical time intervals, e.g., ten minutes or more, the emulated processing times are close to the expected time intervals. We also plot the preprocessing difficulties in the figure; the result suggests that most difficulty values range in between six and eight, covering mining time from 1 to 30 minutes.

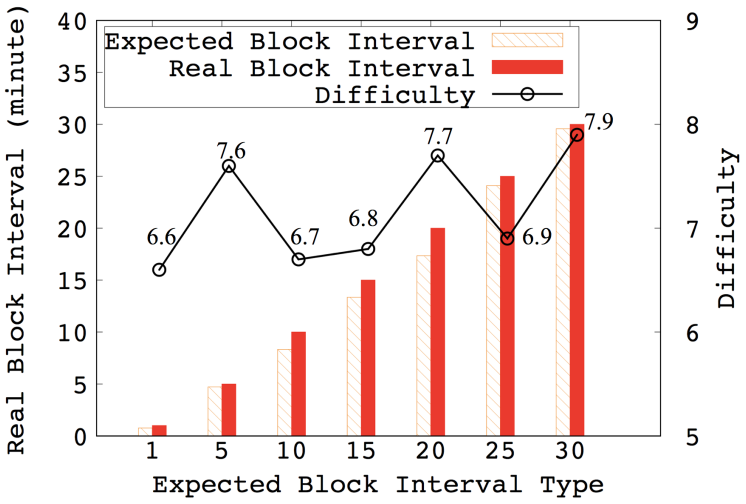


Figure 5.4: Difficulty v.s. Block Creation Time

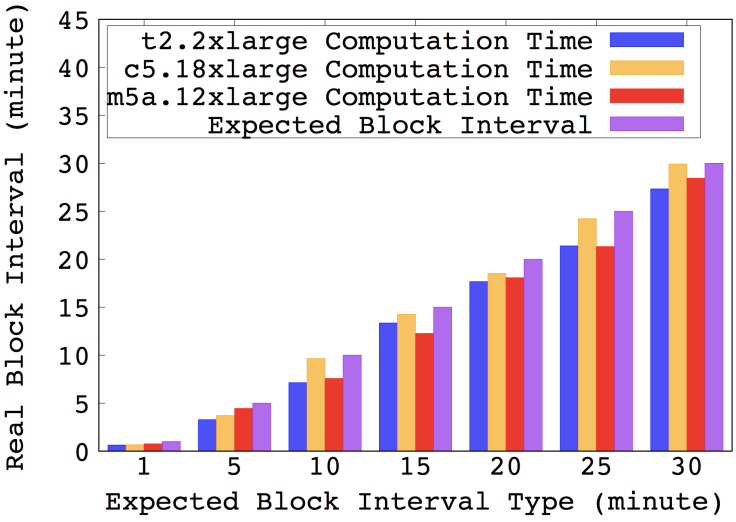


Figure 5.5: Block Creation Rate with Various Instances Type.

We then report the accuracy on various instance types in Figure 5.5. The for sake of clarity, we do not plot the corresponding difficulty in the figure. As we can see from the figure, not all instance types exhibit the same accuracy; in our tested VMIs, the `c5.18xlarge` seems to achieve the highest accuracy except for the trivial case of 1- and 5-minute intervals.

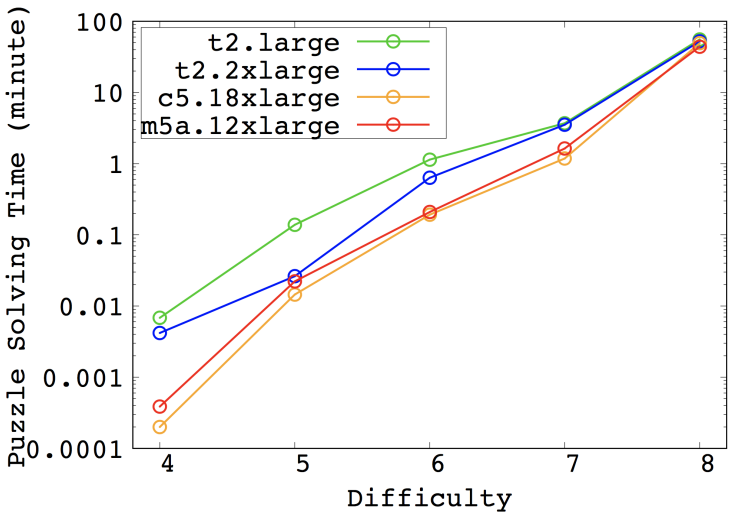
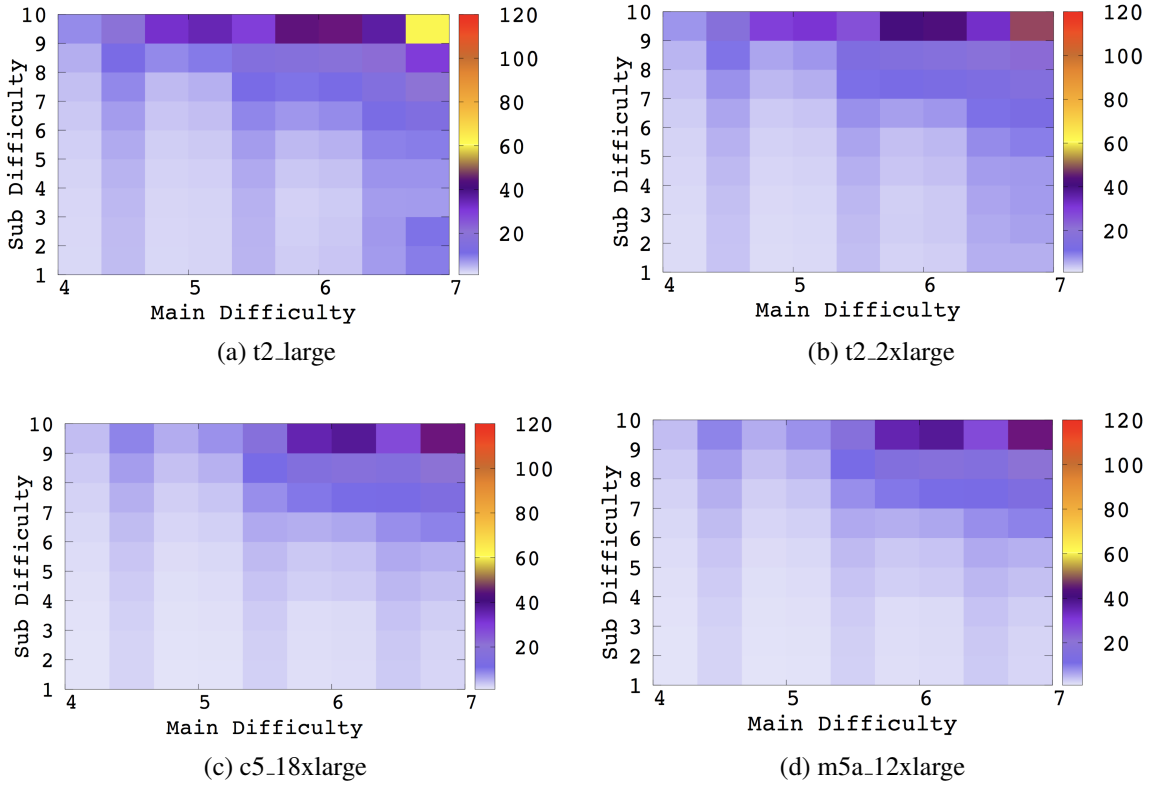


Figure 5.6: Difficulty and Puzzle Solving Time

5.4.4 Sensitivity

This section evaluates the sensitivity of BlockLite when deployed to various VMIs. Figure 5.6 shows the puzzle-solving time on four different instances with respect to practical difficulties from four to eight. For clarity, we only compare the `main difficulty` between VMIs in the figure (we will report `sub difficulties` later on). The figure clearly shows that smaller instances (`t2.large`, `t2.2xlarge`) take more time than the larger instances for all difficulties, which, again, can be explained by the different CPU performance on these instances.

In Figure 5.7, we report the puzzle-solving time of both main and sub difficulties. Indeed, the highest computation time appears on the top-right corner of the map in all cases, as that corner represents both the highest main difficulty and the highest sub difficulty; similarly the lowest value appears at the bottom-left corner. Nonetheless, we do observe different gradients from these four heat-maps. For instance, the largest instance (`m5a.12xlarge`) seems to have the most low-value cells (i.e., shorter puzzle-solving time).



(a) t2_large

(b) t2_2xlarge

(c) c5_18xlarge

(d) m5a_12xlarge

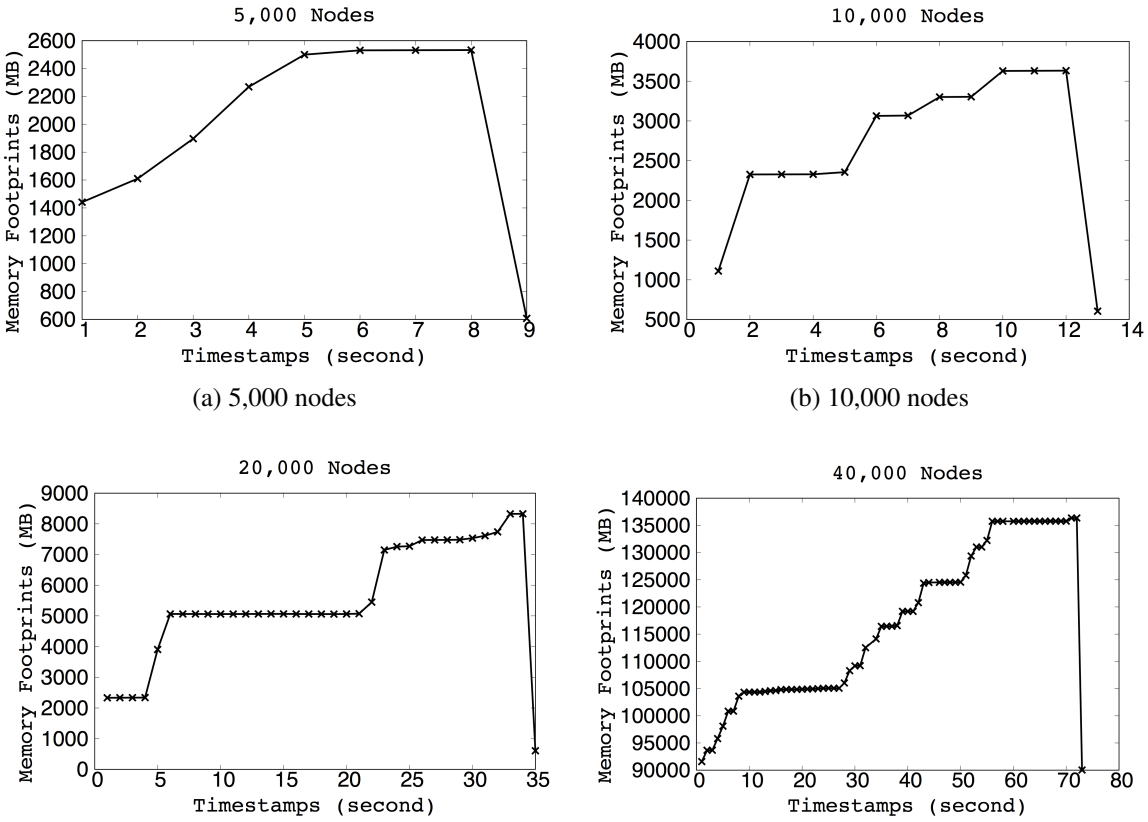
Figure 5.7: Puzzle-solving time of both main and sub difficulties.

5.4.5 Monetary Cost

5.4.6 Scalability

We tested BlockLite's scalability by emulating up to 20,000 nodes on the instance with 48 AMD EPYC cores and 192 GB memory (`m5a.12xlarge`, see Table 5.2). The workload is comprised of more than one million transaction data.

Figure 5.8 shows BlockLite's real-time executions, along with memory footprint, on 5,000, 10,000, 20,000, and 40,000 nodes, respectively. The puzzle difficulty is set to one for the sake of fast demonstrations. We can observe that even at the the real scale of Bitcoin—



(a) 5,000 nodes

(b) 10,000 nodes

(c) 20,000 nodes

(d) 40,000 nodes

Figure 5.8: Scalability and Memory Footprint of BlockLite.

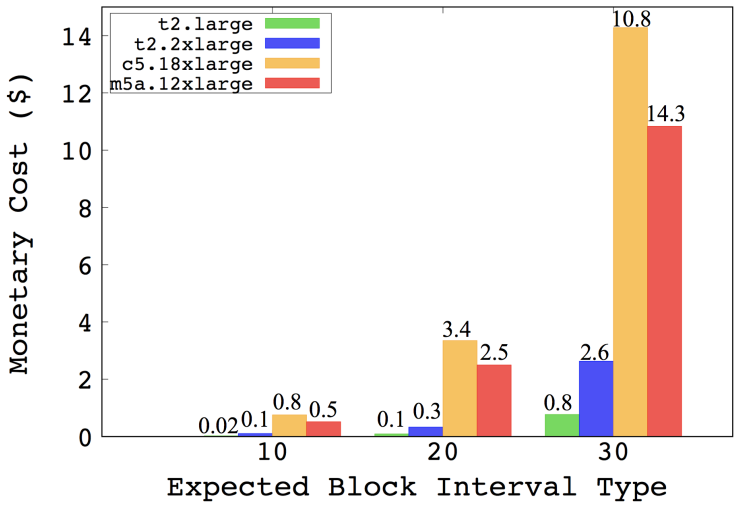


Figure 5.9: Monetary Cost of Various Instances.

10,000 nodes, BlockLite can finish the emulation in 13 seconds with reasonable memory footprint of less than 4 GB.

This section evaluates the cost incurred at the cloud computing vendors when various CPU or instance types are selected. As we can see in Figure 5.9, small instance `t2.large` incur the lowest cost in all intervals while the `c5.18xlarge` instance is the most costly one. This can be explained by the high unit price of the `c5.18xlarge` instance: although the execution time is comparable with the other large instance (`m5a.12xlarge`), the overall charge is higher because of the different unit price.

5.5 Discussion

There are a few optimizations that could have been applied to current BlockLite design, which are not supported at the writing of this dissertation. Although there are many more open questions to be answered, we list two of most interesting optimizations here in the following for the sake of limited space.

One possible optimization to the current BlockLite implementation is to leverage the underlying multi- or many-cores in modern processors. It should be noted that in many computation-based consensus protocols, the workload is embarrassingly parallel. Therefore, we should be able to parallelize the proof-of-work (POW) protocols, which will be studied in our future work.

Another possible extension to this work is to implement an inter-node communication such that the emulator can be distributed over a cluster of nodes. This would further improve the adaptability of BlockLite if a single node is not capable of conducting the intensive computation expected by some large difficulties.

5.6 Summary

This dissertation presents BlockLite, the very first system who can emulate large-scale public blockchains on up to 20,000 nodes. BlockLite achieves such a high scalability through an offline calibration of PoW execution and distributed queues. The emulation also achieves high accuracy through a two-phase adjustment over the underlying consensus protocol. In terms of usability, BlockLite provides an easy-to-use interface to plug in application-specific components such as ad-hoc consensus protocols.

CHAPTER 6

UNBALANCED PARALLEL I/O: AN OFTEN-NEGLECTED SIDE EFFECT OF LOSSY SCIENTIFIC DATA COMPRESSION

6.1 Introduction

As several exascale supercomputers are anticipated to be operational in the next a few years, scientific applications running on those machines are projected to generate massive amount of data at enormous velocity. For example, nowadays the X-point Gyrokinetic Code (XGC) [20] developed by the Princeton Plasma Physics Laboratory can easily produce more than 1PB of data per day when running on the OLCF's Summit supercomputer. As a conservative estimate, the data rate will increase to 10PB per day if running on an exascale supercomputer. Although data storage technologies have also improved tremendously over the years, absorbing data generated at such high rates is almost an impossible mission for most of the data storage systems built under rational budget. To address this critical and challenging issue, multiple lossy compression techniques that are specifically designed and optimized for the data generated by scientific applications have been proposed in recent years. Promising results from existing studies [1, 2, 61, 64] demonstrate that these lossy compression techniques can significantly reduce the size of scientific data while guaranteeing that the compression error is within certain bound. Since the data size can be effectively reduced by lossy compression, it is natural for scientists to expect the overhead caused by writing or reading their data can also be reduced accordingly. For example, if the size of the data is reduced by 100X, it is reasonable for scientists to believe that their data can be written out about 100X faster. This is true when their codes are run

at small scale and only small amount of data is written out using a few processes. However, for data-intensive scientific applications running with massive parallelism on high-performance computing systems, the I/O performance does not simply depend on the size of the data. It also depends on how efficient the concurrent I/O bandwidth can be utilized. Ideally, the maximal concurrent I/O throughput is achieved if all the processes write out or read in the same amount of data. This is why when scientists configure their simulations, they tend to divide the global simulation space into regions of equal size and assign one region to each process to ensure that the I/O loads from all processes are balanced. If the I/O loads among processes are unbalanced, meaning that some of the processes need to write out or read in much more data than others, the overall I/O throughput would decrease as the I/O time is determined by the slowest process that finishes the I/O. Unfortunately, when we apply lossy compression to the data on each process, this I/O imbalance issue often occurs. Specifically, as the data assigned to each process after domain decomposition are often heterogeneous in nature, the information density of these data portions usually varies.

Therefore, applying lossy compression to such heterogeneous domain decomposed data would inevitably result in significantly different compressed data sizes across the parallel processes and cause the imbalance of the parallel I/O loads.

To verify the aforementioned conjecture and characterize the imbalance in compressed data size, we apply three widely used lossy compressors MGARD [1–4], ZFP [64], and SZ [61] to the datasets generated by three real world scientific applications: Gray-Scott simulation, WarpX [94] and XGC [19, 20]. We aim to answer the following questions.

- If the original data is evenly decomposed and assigned to each parallel process, would the compressed data still be imbalanced among the processes?

- Would imbalance exist in compressed data regardless of which compression method is used?
- How imbalanced can the compressed data sizes be?
- How would the imbalance of the compressed data sizes affect the overall I/O performance?

Our analysis shows that the compressed data is always skewed even if the original data is evenly decomposed and such skewness exists in all the compressed data from different compressors. The difference in compressed data can be more than 10X and Weibull distributions can be used to fit the sizes of the compressed data on each process. During the write out or read in process of scientific applications, the overall write or read performance is usually bounded by the slowest process. Therefore, the skewness in compressed data can potentially degrade the parallel I/O performance if not handled carefully. To demonstrate such negative impacts caused by unbalanced parallel I/O due to skewed compression data, we conduct experiments on OLCF's Andes cluster [77]. In our experiments, we launch tests to simulate the highly skewed data among processes caused by lossy compression and measure the write performance when different write patterns are used. From the results, we observe that the I/O imbalance caused by lossy compression can significantly reduce the efficiency of I/O bandwidth utilization if the processes that own high information density data blocks happen to run on the same compute node. Moreover, witing data concurrently to a single shared file through MPI-IO library is more sensitive to the unbalanced I/O loads.

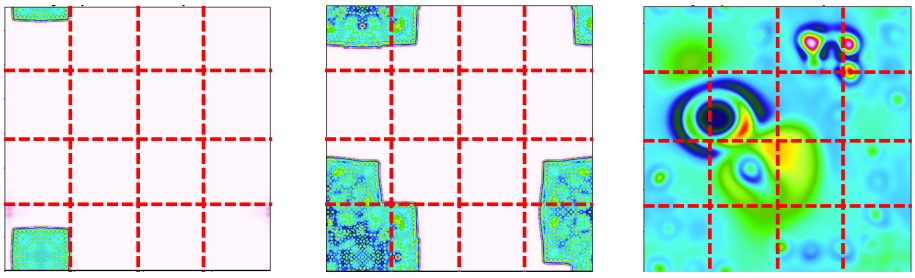


Figure 6.1: Different simulation steps of domain decomposition in parallel scientific applications.

6.2 Analysis of Lossy Compressed Data Size

Scientific codes are often run with a large number of processes in parallel on high-performance computing systems to achieve satisfactory speedups. Particularly, during the execution of these codes, each process is often assigned a single or multiple portions of the data to operate on, which is known as domain decomposition or "data parallelism". For instance, when scientists launch an MPI-based particle-in-cell simulation code, each MPI process is assigned to simulate the movements of particles in certain areas of the entire simulation region. Since the movements of particles are driven by complex physics models, the information density of the data that each process operates on is usually quite different. In some areas, the large distribution of the particles leads to high information density, while in others the data is sparse due to the lack of physics phenomena. This type of *heterogeneity* in domain-decomposed data, is commonly observed in many other parallel scientific applications [19].

Figure 6.1 shows an example of the domain decomposition in a parallel Gray-Scott simulation [78]. Each of these three sub-figures illustrates the concentration of a chemical species at different simulation steps. The simulation is run by 16 MPI processes and each of them operates $\frac{1}{16}$ of the entire simulation region. In each sub-figure, we use red dashed lines to differentiate the areas each process owns. The first sub-figure shows that the data on most of the processes is sparse. In the second sub-figure, the information density of data

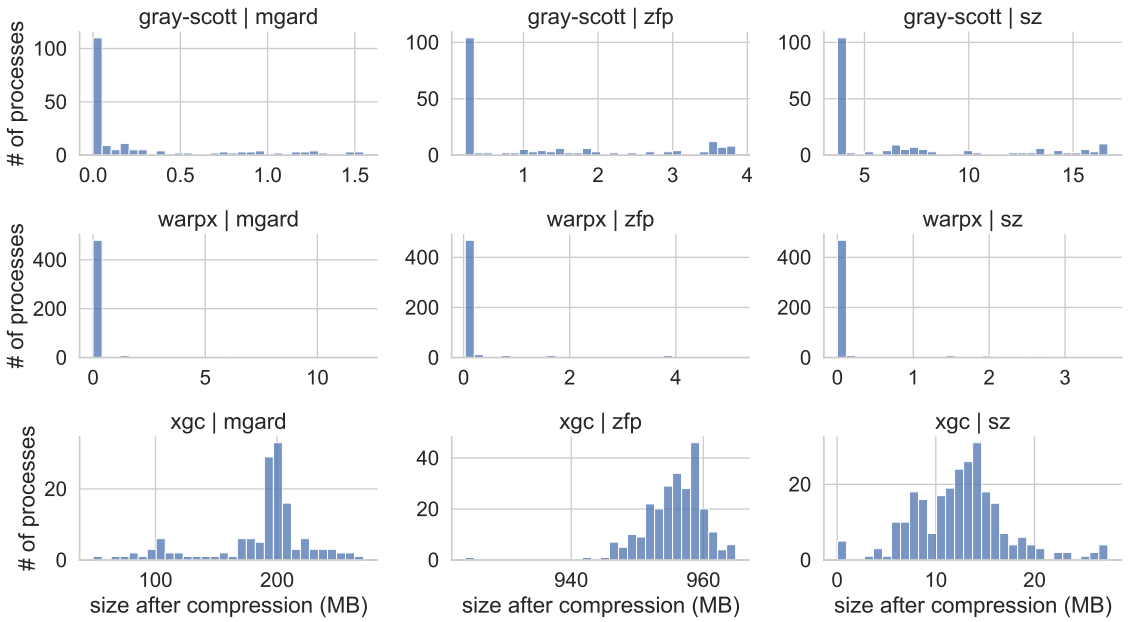


Figure 6.2: Distribution of data sizes among processes after applying different lossy compressors to three scientific datasets.

increases and demonstrates more differences across processes. In the third sub-figure, the data on each process has rich information content, but their characteristics are quite dissimilar. According to the information theory [79], the size of the compressed data is strongly correlated with the information content which is defined by the entropy in the original data. If we apply lossy compression to the data on each process, our conjecture is that the sizes of the data after compression would be significantly different from each other in many real world scenarios since the data on each process has quite different information density.

To verify our conjecture, we conduct characterization studies by applying different lossy compressors to datasets generated by three real world scientific applications. The three lossy compressors we select for our study are MGARD [1–4], ZFP [64], and SZ [61]. These lossy compressors are specifically designed and optimized for compressing scientific data and their effectiveness are verified by existing works [18]. The datasets we use in our study are generated by three parallel scientific codes. Gray-Scott is a 3D 7-point stencil code to simulate Gray-Scott reaction diffusion model. It's a simple system simulating the

time evolution of the spatial distributions of two interacting chemical concentrations. And parameters for the simulation such as the diffusion coefficient can be easily adjusted by the user. WarpX [94] is an advanced multi-platform electromagnetic Particle-In-Cell code. It supports many features including Perfectly-Matched Layers (PML), mesh refinement, and the boosted-frame technique. In addition, WarpX includes load balancing capabilities to achieve better performance. XGC [19, 20] is a gyrokinetic particle-in-cell code, which specializes in the simulation of the edge region of magnetically confined thermonuclear fusion plasma. The simulation domain can include the magnetic separatrix, magnetic axis, and the biased material wall. All of these datasets are written out through ADIOS2 [40], a library that manages the parallel I/O and stores the data in a self-describing format. The metadata of this self-describing format contains rich information about how the dataset is generated, such as which area each data block belongs to in the global array, which MPI process produces which data blocks, etc. By leveraging such information, we can launch a job to read in the exact data blocks produced by each MPI process, and compress them using different lossy compressors. Then we study how different the sizes of the compressed data can be among all the processes. Since different compressors might adopt different error metrics, when we choose the error bounds for each compressor, we try our best to make the peak signal-to-noise ratio (PSNR) of the entire compressed data produced by each compressors fall into a similar range. One thing we would like to emphasize here is that our results cannot be used to indicate which lossy compressor is better in terms of compression ratio since the PSNR of the compressed data from each compressor are not exactly the same. In this study, we only focus on how skew the distribution of the data sizes among processes can be after lossy compression.

For the dataset generated by the Gray-Scott simulation, the entire data is evenly decomposed and assigned to 192 processes. The size of the original data each process owns is 18MB. As shown in Figure 6.2, after the lossy compression, the sizes of the data on most

processes are reduced to less than 0.5MB. However, no matter which lossy compressor is used, the compressed data on a few processes are significantly larger due to their high information density. For example, when SZ is used, the compressed data owned by 5% of the processes are on average more than 10X larger than other processes. As a result, the distribution of data sizes among processes after lossy compression is highly skewed.

When we use the dataset generated by the WarpX simulation for our experiments, the sizes of the compressed data on almost all of the processes are similarly small since the entire dataset is very sparse. However, there are still several processes that have much larger data after compression. The compressed data on these processes can be hundreds of times larger than the minimal data size among all the processes. When we apply different lossy compressors to the data generated by the XGC simulation, the sizes of the compressed data on each process are also dispersed in a wide range. From all these results, we can see that for parallel scientific applications, even the original data is evenly decomposed and assigned to each process, the sizes of the compressed data on each process can be significantly different from each other due to the nonuniform distribution of the information density in the datasets.

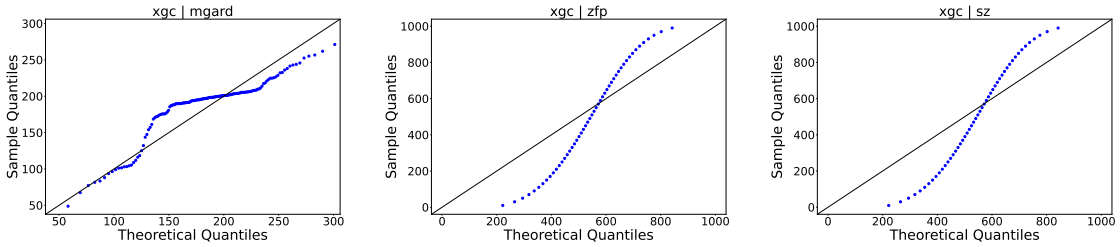


Figure 6.3: Q-Q plot of using Weibull distribution to fit the XGC data with different lossy compressors.

6.3 Evaluation

When parallel scientific codes write out or read in their data, the overall write or read performance usually depends on the slowest process. This is because a synchronization is often needed among processes before each computation step and no process can start the computation of the next step until the slowest process finishes its I/O. Therefore, scientists tend to decompose the data into chunks of the same size and evenly assign them to among processes, so that all the processes can finish the data writing or reading at about the same time to diminish the "straggler" effect. However, based on the observations we obtain in Section 6.2, the amount of data each process writes out or reads in can be significantly different after lossy compression even if the original decomposed data is evenly distributed across processes, see Figure 6.2. Therefore, applying lossy compression can potentially leads to imbalanced I/O loads across processes and thus cause "straggler" effect. In this section, we study how the imbalance of I/O loads caused by lossy compression affects the overall I/O performance of parallel scientific applications.

6.3.1 Experimental Setup

We conduct all our I/O tests on OLCF's Andes cluster. Andes is a 704-compute node commodity-type Linux cluster. Each of these compute nodes has 32 AMD EPYC 7302 cores and 256GB memory. Andes mounts the same center-wide GPFS file system as Summit, which offers more than 3GB/s per node I/O bandwidth. Since after decomposition the existing datasets we have are relatively small compared to the memory size on each compute node, we notice that the caching effect becomes a dominant factor when measuring the write performance. The measured performance numbers do not reflect the actual I/O

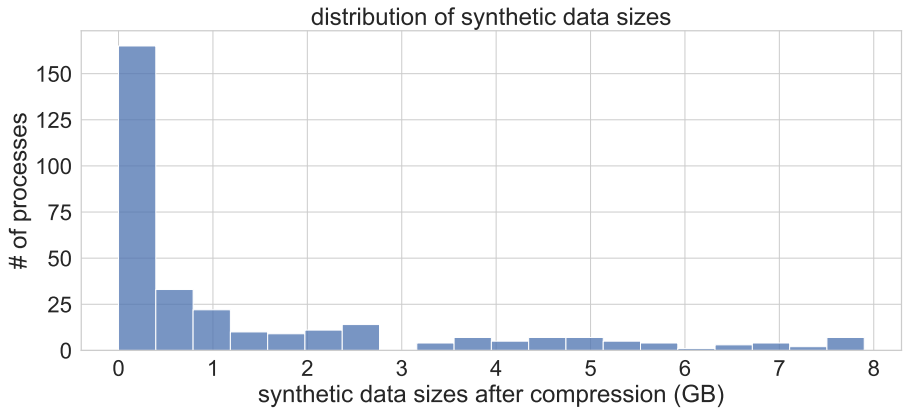


Figure 6.4: Distribution of the synthetically generated data sizes among processes to mimic the effect of lossy compression.

throughput. In order to mitigate the caching effects and fully saturate the I/O bandwidth, instead of using the existing datasets, we developed a code which can synthetically generate data with arbitrarily large sizes for each process.

6.3.2 Impact on Three Common Write Patterns

The impact of unbalanced parallel I/O might be different if the data is written in different patterns. There are three common patterns for writing data in parallel: N-N, N-1-collective, and N-1-independent. "N-N" denotes the code is executed by N processes and each process writes its data to a separate file independently. I/O libraries such as ADIOS2 adopts this pattern. "N-1-collective" denotes N processes write out their data to a single shared file using collective MPI-IO functions, while "N-1-independent" represents the data is written out to a single shared file using independent MPI-IO functions. These two patterns are used by I/O libraries such as HDF5, PnetCDF, etc. To understand which write pattern is more sensitive to the unbalanced parallel I/O, we measure the performance of writing data in all these three patterns in our experiments.

Table 6.1
EXPERIMENTS FOR UNDERSTANDING THE IMPACT OF UNBALANCED PARALLEL I/O LOADS.

Experiment	**Description**
uncompressed-equal	Each process writes the same amount of uncompressed data.
compressed-random	The sizes of the compressed data owned by each process are randomly generated based on certain probability distribution.
compressed-clustered	The per-process compressed data sizes are randomly generated based on the same probability distribution, but they are sorted and assigned to each process in ascending order so that the largest sizes are always assigned to a few processes running on the same compute node.
compressed-equal	The per-process compressed data sizes are randomly generated based on the same probability distribution, but the total size of the compressed data is equally divided by the number of processes and each process writes the same amount of compressed data.

First of all, we measure the overall performance of writing out the uncompressed data in different patterns as the baseline. In this experiment, we assume the uncompressed data on each process is 8GB. We lunch a job on Andes with 10 compute nodes and 32 processes per node, and let each process write out 8GB randomly generated data to the file system using different patterns. This experiment is denoted by "uncompressed-equal" in Table 6.1. Secondly, we randomly generate the sizes of compressed data owned by each process based on a Weibull distribution. The reason for choosing the Weibull distribution is that its probability density function usually has a long tail, which is similar to those shown in Figure 6.2.To verify it, we use Weibull distribution to fit the real XGC data and show Q-Q plot in Figure 6.3. From the plot, we can see that most points perfectly lie on $y = x$, which suggests Wellbull distribution is a good representation. Since the size of the compressed data cannot be less than or equal to zero or greater than or equal to the original data size, we make sure only the valid random numbers are selected. Figure 6.4 shows the sizes of the compressed data synthetically generated for 320 processes using a Weibull distribution whose shape parameter is 0.3 and scale parameter is 5.0. These sizes are randomly assigned to each process and each process writes out certain amount of data based on the assigned size. This experiment is denoted by "compressed-random" in Table 6.1.

Thirdly, as shown in Figure 6.1, the information density of each process' data demonstrates spacial locality. Data blocks that have similar information density are likely to be assigned to processes running on the same compute node or compute nodes that are close to each other in the HPC system's interconnect topology. E.g., in the bottom left corner of the second sub-figure in Figure 6.1, data on those three processes all have dense information content. It is also possible that those three processes run on the same compute node as they need to share the I/O bandwidth of that particular compute node. If the sizes of the compressed data on those three processes are much larger than other processes, the I/O on that compute node would become a bottleneck. To mimic this scenario, we sort the sizes of the compressed data synthetically generated in the "uncompressed-random" experiment. We then assign them to each process in ascending order, so that the largest sizes are always assigned to a few processes running on the same compute node to trigger the bandwidth contention. This experiment is denoted by "compressed-clustered" in Table 6.1.

Finally, we measure the write performance when the sizes of the compressed data on each process are all the same. Although this scenario rarely occurs in practice, we believe the performance numbers can be useful for us to understand how much write time the lossy compression can save ideally. In order to have a fair comparison with numbers measured in other experiments, we also use the sizes of the compressed data synthetically generated in the previous two experiments. We sum up all these synthetic sizes and calculate the mean of them. Then we let each process write out data with the size of that mean value. This experiment is denoted by "compressed-equal" in Table 6.1.

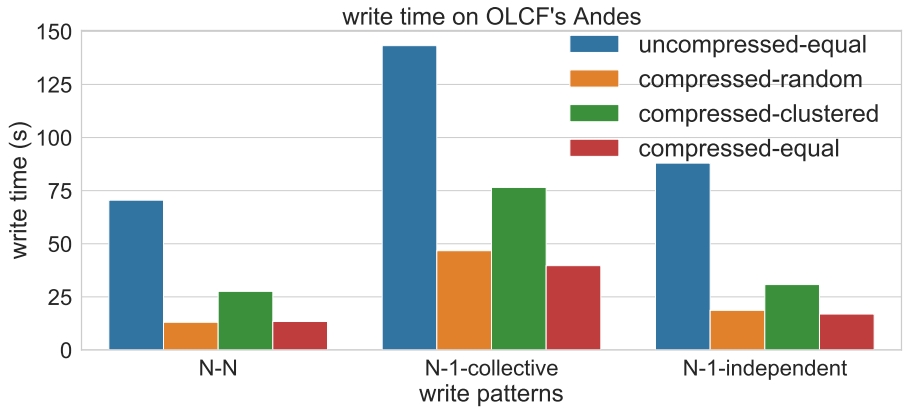


Figure 6.5: The overall write time.

6.3.2.1 Impact on Write Time

The overall write time measured in these four experiments are shown in Figure 6.5. As we can see, adopting lossy compression does reduce the overall write time. The total size of the original data is $8 \times 32 \times 10 = 2560$GB, while the total size of the compressed data is 441GB. Ideally, if the sizes of the compressed data on each process are all the same ("compressed-equal"), we expect the write time is reduced by roughly 6 times. However, for other more realistic scenarios, the amount of write time reduced are less than the ideal case. As expected, "compressed-clustered" reduces the least amount write time. As we mentioned above, if processes running on the same compute node all have much larger data compared to other processes after lossy compression, I/O bandwidth contention on that compute node is likely to happen which makes the I/O on that particular node much slower than other nodes. Even those processes run on different compute nodes but those nodes are close to each other in the network topology, bandwidth contention might also happen on the routers shared by those nodes.

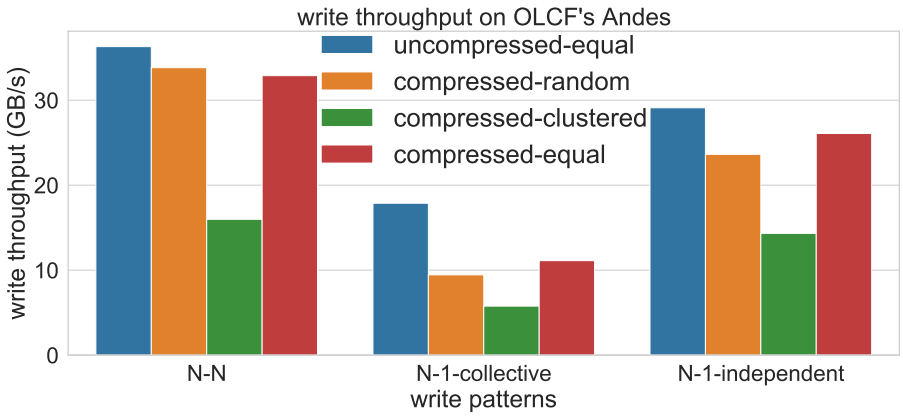


Figure 6.6: The overall write throughput.

6.3.2.2 Impact on Write Throughput

The write throughput for different scenarios are shown in Figure 6.6. Apparently, "compressed-clustered" achieves the lowest write throughput no matter which write pattern is used. "compressed-random" achieves almost the same throughput as "compressed-equal" when each process writes its own data to a separate file. This is because in the "compressed-random" experiment, although the data sizes are very different across processes, the total data size of the 32 processes on each compute node does not show significant imbalance due to the random assignment of synthetic data sizes. If all the processes write data to a single shared file, "compressed-equal" always outperforms "compressed-random". This indicates that writing data to a single shared file is more sensitive to the unbalanced parallel I/O loads.

6.4 Summary

In this dissertation, we focus on an often neglected side effect of lossy scientific data compression: unbalanced parallel I/O. We conduct a comprehensive study by applying three

commonly used lossy compressors MGARD, ZFP, and SZ to data generated by three real-world scientific applications Gray-Scott simulation, WarpX, and XGC. Our study quantifies the data skewness of lossy compressed scientific data across processes due to heterogeneous information density. Further experiments on write performance demonstrates how such data skewness can cause unbalanced parallel I/O and thus impact the parallel I/O performance.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

The main contribution of this dissertation is we explore the unique features of both application and system to seek overlooked optimization opportunities or tackle challenges that are difficult to be addressed by only looking at the application or system individually. A set of models and methodologies are developed in this dissertation to achieve optimal scaling of different workloads and are summarized as follows.

- For memory intensive applications:
  - We propose two scaling methodologies to guarantee application's performance with minimal cost in cloud. More specifically, (i) We develop analytic models to predict applications' performance and cost when various portions of virtual memory are used in public clouds. As a result, users will know how, and by how much, virtual memory will affect the overall performance and cost of their applications before actually executing them. (ii) We propose multiple cost-effective approaches for preventing OOM error in public clouds. Instead of periodically checkpointing memory states, the proposed elevation-migration approaches incur only one batch of memory accesss when encountering memory depletion. (iii) We extensively evaluate the proposed techniques using bench-

marks and real-world applications. Experimental results demonstrate high effectiveness of both techniques on AWS.

- For compute intensive applications:
  - We propose a new deep-learning-based approach to efficiently and accurately detect abnormal cells in electirc area using Kirchhoff analyses. The new approach employs multiple techniques including early pruning of unqualified input data, parallelization of forward labelling, data augmentation through random errors and dimension reduction, all of which collectively enable an efficient and accurate mechanism for large-scale Kirchhoff analyses. We implement the proposed approach with the latest machine learning framework and the parallel computing library, and evaluate it with real-world engineering applications showing promising results.

- For both memory and compute intensive applications:
  - We develop an accurate and efficient emulating system to replay the execution of large-scale blockchain systems on tens of thousands of nodes in the cloud. Existing blockchain frameworks exhibit a technical barrier for many users to modify or test out new research ideas in blockchains. To make it worse, many advantages of blockchain systems can be demonstrated only at large scales, which are not always available to researchers. In this dissertation, we propose BlockLite to emulate this large scale application in cloud.the very first system who can emulate large-scale public blockchains on up to 40,000 nodes. BlockLite achieves such a high scalability through an offline calibration of PoW consensus and distributed queues. In terms of usability, BlockLite provides an

easy-to-use interface to plug in application-specific components such as ad-hoc consensus protocols.

- For I/O intensive applications:
  - We observe one important yet often neglected side effect of lossy scientific data compression. Lossy compression techniques have demonstrated promising results in significantly reducing the scientific data size while guaranteeing the compression error bounds, but the compressed data size is often highly skewed and thus impact the performance of parallel I/O. Therefore, we believe our research community should pay more attention to the unbalanced parallel I/O caused by lossy scientific data compression.

7.2 Future Work

There are several potential extensions to the methodologies and results in this dissertation.

7.2.1 Prediction of No Traits Applications

The performance-cost model proposed in 3 can be verified on more real world applications. And in scenario 2, we will use machine learning methods such as time series model to predict no traits applications.

7.2.2 Mitigation of Imbalance Parallel I/O

We observed the high data skewness of lossy compressed scientific data across processes due to heterogeneous information density in 6. And for future work, we plan to employ a tiered approach to group decomposed data with complimentary information density on the same node to reduce the imbalance in parallel I/O. For example, before writing data to the file, we will allocate some aggregators to collect the data from different ranks, so that the size of data on these aggregators is balanced. Therefore, the total running time will not be affected by the slowest rank as the running time on each aggregator is similar.

7.2.3 Optimized Scheduling of Cross-platform Workloads

We applied multiple scientific applications in 3, 4, 5 and 6. And we know scientific applications are usually composed of many small tasks and there can be heterogeneous computing demands among these tasks causing load imbalance. Existing methods for achieving load balance are usually application specific and require significant amount of manual efforts to make changes to the application code. Elasticity and pay-as-you-go features in cloud computing have potential to address the load imbalance issue through a general resource scheduling and migration approach without changing application code. So especially for these kinds of memory, compute and communication Intensive Applications, there're bottlenecks such as communication overhead, so we want to effectively leverage the unique features of cloud and HPC, in order to find the optimal schedule for different workloads.

BIBLIOGRAPHY

[1] Mark Ainsworth, Ozan Tugluk, Ben Whitney, and Scott Klasky. Multilevel techniques for compression and reduction of scientific data—the univariate case. *Computing and Visualization in Science*, 19(5-6):65–76, 2018.

[2] Mark Ainsworth, Ozan Tugluk, Ben Whitney, and Scott Klasky. Multilevel techniques for compression and reduction of scientific data—the multivariate case. *SIAM Journal on Scientific Computing*, 41(2):A1278–A1303, 2019.

[3] Mark Ainsworth, Ozan Tugluk, Ben Whitney, and Scott Klasky. Multilevel techniques for compression and reduction of scientific data-quantitative control of accuracy in derived quantities. *SIAM Journal on Scientific Computing*, 41(4):A2146–A2171, 2019.

[4] Mark Ainsworth, Ozan Tugluk, Ben Whitney, and Scott Klasky. Multilevel techniques for compression and reduction of scientific data—the unstructured case. *SIAM Journal on Scientific Computing*, 42(2):A1402–A1427, 2020.

[5] Abdullah Al-Mamun, Tonglin Li, Mohammad Sadoghi, and Dongfang Zhao. In-memory blockchain: Toward efficient and trustworthy data provenance for hpc systems. In *IEEE International Conference on Big Data (BigData)*, 2018.

[6] Lindsey Allen et al. Veritas: Shared verifiable databases and tables in the cloud. In *9th Biennial Conference on Innovative Data Systems Research (CIDR)*, 2019.

[7] Amazon EC2. `http://aws.amazon.com/ec2`, Accessed March 6, 2015.

[8] L. Aniello, R. Baldoni, E. Gaetani, F. Lombardi, A. Margheri, and V. Sassone. A prototype evaluation of a tamper-resistant high performance blockchain-based transaction log for a distributed database. In *13th European Dependable Computing Conference (EDCC)*, 2017.

[9] Yusuke Aoki, Kai Otsuki, Takeshi Kaneko, Ryohei Banno, and Kazuyuki Shudo. Simblock: A blockchain network simulator. *CoRR*, abs/1901.09777, 2019.

[10] B. S. Arab, D. Gawlick, V. Krishnaswamy, V. Radhakrishnan, and B. Glavic. Using reenactment to retroactively capture provenance for transactions. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 30(3):599–612, March 2018.

[11] D.C. Barber and B.H. Brown. Applied potential tomography. *Journal of Physics E: Scientific Instruments*, 17:723–733, 1984.

[12] U. Bellur, A. Malani, and N. C. Narendra. Cost optimization in multi-site multi-cloud environments with multiple pricing schemes. In *2014 IEEE 7th International Conference on Cloud Computing (CLOUD)*, pages 689–696, June 2014.

[13] A. Beloglazov and R. Buyya. Energy efficient allocation of virtual machines in cloud data centers. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pages 577–578, May 2010.

[14] BigchainDB. `https://github.com/bigchaindb/bigchaindb`, Accessed 2018.

[15] Bitcoin. `https://bitcoin.org/bitcoin.pdf`, Accessed 2019.

[16] Bitcoin Scale. `https://bitnodes.earn.com`, Accessed 2019.

[17] Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. Practical uc-secure delegatable credentials with attributes and their application to blockchain. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 683–699, 2017.

[18] Franck Cappello, Sheng Di, Sihuan Li, Xin Liang, Ali Murat Gok, Dingwen Tao, Chun Hong Yoon, Xin-Chuan Wu, Yuri Alexeev, and Frederic T Chong. Use cases of lossy compression for floating-point data in scientific data sets. *International Journal of High-Performance Computing Applications*, 33(6), 2018.

[19] Choong-Seock Chang, S Ku, PH Diamond, Z Lin, S Parker, TS Hahm, and N Samatova. Compressed ion temperature gradient turbulence in diverted tokamak edge. *Physics of Plasmas*, 16(5):056108, 2009.

[20] Choong-Seock Chang and Susan Ku. Spontaneous rotation sources in a quiescent tokamak edge plasma. *Physics of Plasmas*, 15(6):062510, 2008.

[21] R. Chard, K. Chard, K. Bubendorfer, L. Lacinski, R. Madduri, and I. Foster. Cost-aware elastic cloud provisioning for scientific workloads. In *2015 IEEE 8th International Conference on Cloud Computing (CLOUD)*, pages 971–974, 2015.

[22] Jing Chen, Shixiong Yao, Quan Yuan, Kun He, Shouling Ji, and Ruiying Du. Certchain: Public and efficient certificate audit based on blockchain for tls connections. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018.

[23] Junjie Chen, Jialin Liu, Philip Roth, and Yong Chen. Using working set reorganization to manage storage systems with hard and solid state disks. In *Proceedings of the 2014 43rd International Conference on Parallel Processing Workshops (ICPP)*, 2014.

[24] Alessandro Cimatti, Sergio Mover, and Mirko Sessa. Smt-based analysis of switching multi-domain linear kirchhoff networks. In *2017 Formal Methods in Computer Aided Design (FMCAD)*, pages 188–195, 2017.

[25] Peter V. Coveney, Trevor L. Hughes, and Philip Fletcher. Using artificial neural networks to predict the quality and performance of oil-field cements. *AI Magazine*, 17(4):41–53, 1996.

[26] D. Dai, Y. Chen, P. Carns, J. Jenkins, and R. Ross. Lightweight provenance service for high-performance computing. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2017.

[27] D. Dai, Y. Chen, D. Kimpe, and R. Ross. Provenance-based object storage prediction scheme for scientific big data applications. In *IEEE International Conference on Big Data (BigData)*, 2014.

[28] Mingjun Dai, Shengli Zhang, Hui Wang, and Shi Jin. A low storage room requirement framework for distributed ledger in blockchain. *IEEE Access*, 6:22970–22975, 2018.

[29] Quang A. Dang and Nguyen Thanh Huong. Existence results and iterative method for solving a nonlinear biharmonic equation of kirchhoff type. *Computers and Mathematics with Applications*, 76:11–22, 2018.

[30] DiDi Data Mining Competition. `http://research.xiaojukeji.com/competition/detail.action?competitionId=DiTech2016`, Accessed February 12, 2018.

[31] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. Blockbench: A framework for analyzing private blockchains. In *ACM International Conference on Management of Data (SIGMOD)*, 2017.

[32] D. J. Dubois, G. Valetto, D. Lucia, and E. Di Nitto. Mycocloud: Elasticity through self-organized service placement in decentralized clouds. In *2015 IEEE 8th International Conference on Cloud Computing (CLOUD)*, pages 629–636, June 2015.

[33] Ethereum. `https://www.ethereum.org/`, Accessed 2019.

[34] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation (NSDI)*, 2016.

[35] Facebook. Zstandard lossless compressor. `http://facebook.github.io/zstd/`. Visited Sep 29, 2021.

[36] M. R. Hoseiny Farahabady, Y. C. Lee, and A. Y. Zomaya. Pareto-optimal cloud bursting. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 25(10):2670–2682, Oct 2014.

[37] Rosa Filgueira, Malcolm Atkinson, Yusuke Tanimura, and Isao Kojima. Applying selectively parallel i/o compression to parallel storage systems. In *European Conference on Parallel Processing*, pages 282–293. Springer, 2014.

[38] Jean-loup Gailly and Mark Adler. Zlib compression library. 2004.

[39] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016.

[40] William F Godoy, Norbert Podhorszki, Ruonan Wang, Chuck Atkins, Greg Eisenhauer, Junmin Gu, Philip Davis, Jong Choi, Kai Germaschewski, Kevin Huck, et al. Adios 2: The adaptable input output system. a framework for high-performance data management. *SoftwareX*, 12:100561, 2020.

[41] Richard M. Golden. Kirchoff law markov fields for analog circuit design. In *Advances in Neural Information Processing Systems (NIPS)*, 2000.

[42] Jianmin Guo, Shiwang Ma, and Guang Zhang. Solutions of the autonomous kirchhoff type equations in r^n. *Applied Mathematics Letters*, 82:14–17, 2018.

[43] Siyuan Han, Zihuan Xu, and Lei Chen. Jupiter: A blockchain platform for mobile devices. In *IEEE International Conference on Data Engineering (ICDE)*, 2018.

[44] P. Hoenisch, S. Schulte, S. Dustdar, and S. Venugopal. Self-adaptive resource allocation for elastic process execution. In *2013 IEEE Sixth International Conference on Cloud Computing (CLOUD)*, pages 220–227, June 2013.

[45] Shengshan Hu, Chengjun Cai, Qian Wang, Cong Wang, Xiangyang Luo, and Kui Ren. Searching an encrypted cloud meets blockchain: A decentralized, reliable

and fair realization. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018.

[46] H. Huang, L. Wang, B. C. Tak, L. Wang, and C. Tang. Cap3: A cloud auto-provisioning framework for parallel processing using on-demand and spot instances. In *2013 IEEE Sixth International Conference on Cloud Computing (CLOUD)*, pages 228–235, June 2013.

[47] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, Sep. 1952.

[48] Hyperledger. `https://www.hyperledger.org/`, Accessed 2018.

[49] Inkchain. `https://github.com/inklabsfoundation/inkchain`, Accessed 2018.

[50] Bahman Javadi, Ruppa K. Thulasiram, and Rajkumar Buyya. Characterizing spot price dynamics in public cloud environments. *Future Gener. Comput. Syst.*, 29(4):988–999, June 2013.

[51] Linhua Jiang, Ke Wang, and Dongfang Zhao. Davram: Distributed virtual memory in user space. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Accepted, 2018.

[52] Keras. `https://keras.io`, 2019.

[53] Kirchhoff Law. `https://en.wikipedia.org/wiki/Kirchhoff_circuit_laws`, 2019.

[54] P. Kokkinos, T. A. Varvarigou, A. Kretsis, P. Soumplis, and E. A. Varvarigos. Cost and utilization optimization of amazon ec2 instances. In *2013 IEEE Sixth International Conference on Cloud Computing (CLOUD)*, pages 518–525, June 2013.

[55] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy (SP)*, 2016.

[56] Y. C. Lee and B. Lian. Cloud bursting scheduler for cost efficiency. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 774–777, June 2017.

[57] A. F. Leite, V. Alves, G. N. Rodrigues, C. Tadonki, C. Eisenbeis, and A. C. M. A. d. Melo. Automating resource selection and configuration in inter-clouds through a software product line method. In *2015 IEEE 8th International Conference on Cloud Computing (CLOUD)*, pages 726–733, June 2015.

[58] L. M. Leslie, Y. C. Lee, P. Lu, and A. Y. Zomaya. Exploiting performance and cost diversity in the cloud. In *2013 IEEE Sixth International Conference on Cloud Computing (CLOUD)*, pages 107–114, June 2013.

[59] Tonglin Li, Kate Keahey, Ke Wang, Dongfang Zhao, and Ioan Raicu. A dynamically scalable cloud data infrastructure for sensor networks. In *Proceedings of the 6th Workshop on Scientific Cloud Computing (ScienceCloud)*, 2015.

[60] Tonglin Li, Xiaobing Zhou, Ke Wang, Dongfang Zhao, Iman Sadooghi, Zhao Zhang, and Ioan Raicu. A convergence of key-value storage systems from clouds to supercomputer. *Concurr. Comput. : Pract. Exper.*, 2016.

[61] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Shaomeng Li, Hanqi Guo, Zizhong Chen, and Franck Cappello. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 438–447. IEEE, 2018.

[62] Gilbert Lim, Mong Li Lee, Wynne Hsu, and Tien Yin Wong. Transformed representations for convolutional neural networks in diabetic retinopathy screening. In *Workshops at the Twenty-Eighth Association for the Advancement of Artificial Intelligence (AAAI) Conference on Artificial Intelligence*, 2014.

[63] Hou Xiang Lin, Li Qi, Chen Cong, and Sun Hong. Study and application of optimization algorithm about sinusoidal steady-state circuit. In *Proceedings of the 34th Chinese Control Conference (CCC)*, 2015.

[64] Peter Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE transactions on visualization and computer graphics*, 20(12):2674–2683, 2014.

[65] M. Mao and M. Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–12, Nov 2011.

[66] Parmita Mehta, Sven Dorkenwald, Dongfang Zhao, Tomer Kaftan, Alvin Cheung, Magdalena Balazinska, Ariel Rokem, Andrew Connolly, Jacob Vanderplas, and Yusra AlSayyad. Comparative evaluation of big-data systems on scientific image analytics workloads. In *Proceedings of the 43rd International Conference on Very Large Data Bases (VLDB)*, 2017.

[67] MPICH. http://www.mpich.org/, Accessed 2018.

[68] Richard Murphy, Arun Rodrigues, Peter Kogge, and Keith Underwood. The implications of working set analysis on supercomputing memory hierarchy design. In *Proceedings of the 19th Annual International Conference on Supercomputing (ICS)*, 2005.

[69] R. Nakazawa, K. Ogata, S. Seelam, and T. Onodera. Taming performance degradation of containers in the case of extreme memory overcommitment. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 196–204, June 2017.

[70] Vivek Prakash Nigam and Daniel Graupe. A neural-network-based detection of epilepsy. *Neurological research*, 26(1):55–60, 2004.

[71] X. Niu, R. Kapoor, B. Glavic, D. Gawlick, Z. H. Liu, V. Krishnaswamy, and V. Radhakrishnan. Provenance-aware query optimization. In *IEEE 33rd International Conference on Data Engineering (ICDE)*, 2017.

[72] Ye Niu, Abdullah Al-Mamun, Hui Lin, Tonglin Li, Yi Zhao, and Dongfang Zhao. Toward scalable analysis of multidimensional scientific data: A case study of electrode arrays. In *IEEE International Conference on Big Data (BigData)*, 2018.

[73] Ye Niu, Lin Qi, Fen Zhang, and Yi Zhao. Geometric screening of core/shell hydrogel microcapsules using a tapered microchannel with interdigitated electrodes. *Biosensors and Bioelectronics*, 112:162–169, 2018.

[74] NS3. https://www.nsnam.org/tutorials/NS-3-LABMEETING-1.pdf, Accessed 2019.

[75] Numpy: adding matrix. https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.add.html, Accessed March 2, 2018.

[76] MFXJ Oberhumer. Lzo-a real-time data compression library. *http://www. oberhumer. com/opensource/lzo/*, 2008.

[77] OLCF. Andes user guide. https://docs.olcf.ornl.gov/systems/andes_user_guide.html. Visited Sep 29, 2021.

[78] John E Pearson. Complex patterns in a simple system. *Science*, 261(5118):189–192, 1993.

[79] David Salomon and Giovanni Motta. *Handbook of Data Compression*. Springer, 2010.

[80] Eric R Schendel, Saurabh V Pendse, John Jenkins, David A Boyuka, Zhenhuan Gong, Sriram Lakshminarasimhan, Qing Liu, Hemanth Kolla, Jackie Chen, Scott Klasky, et al. Isobar hybrid compression-i/o interleaving for large-scale parallel i/o optimization. In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, pages 61–72, 2012.

[81] Scikit-learn. `https://scikit-learn.org`, 2019.

[82] V. Yu. Semenov. A method to find all the roots of the system of nonlinear algebraic equations based on the krawczyk operator. *Cybernetics and Systems Analysis*, 51(819), September 2015.

[83] SHA-256. `https://en.bitcoin.it/wiki/SHA-256`, Accessed 2018.

[84] S. Shi, C. Wu, and Z. Li. Cost-minimizing online vm purchasing for application service providers with arbitrary demands. In *2015 IEEE 8th International Conference on Cloud Computing (CLOUD)*, pages 146–154, 2015.

[85] S. Spinner, N. Herbst, S. Kounev, X. Zhu, L. Lu, M. Uysal, and R. Griffith. Proactive memory scaling of virtualized applications. In *2015 IEEE 8th International Conference on Cloud Computing (CLOUD)*, pages 277–284, June 2015.

[86] Lyubomir Stoykov, Kaiwen Zhang, and Hans-Arno Jacobsen. Vibes: Fast blockchain simulations for large-scale peer-to-peer networks: Demo. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference (Middleware)*, 2017.

[87] A. Strunk. A lightweight model for estimating energy cost of live migration of virtual machines. In *2013 IEEE Sixth International Conference on Cloud Computing (CLOUD)*, pages 510–517, June 2013.

[88] C. Tan, S. Lv, F. Dong, and M. Takei. Image reconstruction based on convolutional neural network for electrical resistance tomography. *IEEE Sensors Journal*, 19(1):196–204, 2019.

[89] Michael David Taylor. Joule counting correction for electric vehicles using artificial neural networks. In *Twenty-Eighth Association for the Advancement of Artificial Intelligence (AAAI) Conference on Artificial Intelligence*, 2014.

[90] TensorFlow. `https://www.tensorflow.org/`, Accessed July 26, 2016.

[91] Prasang Upadhyaya, Magdalena Balazinska, and Dan Suciu. How to price shared optimizations in the cloud. *International Conference on Very Large Data Bases (VLDB)*, 5(6):562–573, February 2012.

[92] Prasang Upadhyaya, Magdalena Balazinska, and Dan Suciu. Price-optimal querying with data apis. *International Conference on Very Large Data Bases (VLDB)*, 9(14):1695–1706, October 2016.

[93] V. D. Valerio, V. Cardellini, and F. L. Presti. Optimal pricing and service provisioning strategies in cloud systems: A stackelberg game approach. In *2013 IEEE Sixth International Conference on Cloud Computing (CLOUD)*, pages 115–122, June 2013.

[94] J-L Vay, A Almgren, J Bell, L Ge, DP Grote, M Hogan, O Kononenko, R Lehe, A Myers, C Ng, et al. Warp-x: A new exascale computing platform for beam–plasma simulations. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 909:476–479, 2018.

[95] G. Wang and T. S. E. Ng. The impact of virtualization on network performance of amazon ec2 data center. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1–9, March 2010.

[96] P. Wang, Y. Qi, D. Hui, L. Rao, and X. Liu. Present or future: Optimal pricing for spot instances. In *2013 IEEE 33rd International Conference on Distributed Computing Systems (ICDCS)*, pages 410–419, July 2013.

[97] Benjamin Welton, Dries Kimpe, Jason Cope, Christina M Patrick, Kamil Iskra, and Robert Ross. Improving i/o forwarding throughput with data compression. In *2011 IEEE International Conference on Cluster Computing*, pages 438–445. IEEE, 2011.

[98] J. Xu and B. Palanisamy. Cost-aware resource management for federated clouds using resource sharing contracts. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 238–245, June 2017.

[99] L. Yazdanov and C. Fetzer. Vscaler: Autonomic virtual machine scaling. In *2013 IEEE Sixth International Conference on Cloud Computing (CLOUD)*, pages 212–219, June 2013.

[100] Z. Yin, H. Chen, J. Sun, and F. Hu. Eaers: An enhanced version of autonomic and elastic resource scheduling framework for cloud applications. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 512–519, June 2017.

[101] Kaiwen Zhang and Hans-Arno Jacobsen. Towards dependable, scalable, and pervasive distributed ledgers with blockchains. In *38th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2018.

[102] L. Zhang, Z. Li, and C. Wu. Dynamic resource provisioning in cloud computing: A randomized auction approach. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 433–441, April 2014.

[103] Dongfang Zhao, Ning Liu, Dries Kimpe, Robert Ross, Xian-He Sun, and Ioan Raicu. Towards exploring data-intensive scientific applications at extreme scales through systems and simulations. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 27(6), 2016.

[104] Dongfang Zhao, Nagapramod Mandagere, Gabriel Alatorre, Mohamed Mohamed, and Heiko Ludwig. Toward locality-aware scheduling for containerized cloud services. In *IEEE International Conference on Big Data (BigData)*, pages 273–280, 2015.

[105] Dongfang Zhao, Kan Qiao, and Ioan Raicu. Hycache+: Towards scalable high-performance caching middleware for parallel file systems. In *Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 267–276, 2014.

[106] Dongfang Zhao, Xu Yang, Iman Sadooghi, Gabriele Garzoglio, Steven Timm, and Ioan Raicu. High-performance storage support for scientific applications on the cloud. In *Proceedings of the 6th Workshop on Scientific Cloud Computing (ScienceCloud)*, 2015.

[107] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.

[108] Hongbo Zou, Yongen Yu, Wei Tang, and Hsuanwei Michelle Chen. Improving i/o performance with adaptive data compression for big data applications. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, pages 1228–1237. IEEE, 2014.