Thermal Management and Data Archiving in Data Centers

by

Yuanqi Chen

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
May 7, 2016

Keywords: Thermal, Modelling, Benchmark, Data Center, Three-tier, Erasure Code, Archiving

Approved by

Xiao Qin, Professor of Computer Science and Software Engineering
Saad Biaz, Professor of Computer Science and Software Engineering
Wei-Shinn (Jeff) Ku, Associate Professor of Computer Science and Software Engineering
Hari Narayanan, Professor of Computer Science and Software Engineering

Abstract

This dissertation is focused on thermal and resource management of data centers. Recognizing that there is the lack of comprehensive benchmarks for thermal management in the context of cluster computing in data centers, we propose a thermal efficiency benchmark - *ThermoBench* - for clusters. *ThermoBench* evaluates the thermal efficiency of computing and storage clusters deployed in data centers. We shed light on the criteria, metrics and challenges of developing a thermal efficiency benchmark. We also pay particular attention to clusters running scalable client-server enterprise applications in data centers. We apply *ThermoBench* to evaluate the thermal efficiency of a real-world cluster by running TPC-W benchmark with changing transactional arrival rate and mix percentage. *ThermalBench* provides a simple yet powerful benchmark solution for assessing thermal behaviours of computing clusters in data centers.

In the second part of this dissertation research, we build a self-adjusting model called *TERN* to predict thermal behaviours of hardware resources for client sessions. Our *TERN* contains two major components: (1) a resource utilization model being responsible for estimating hardware usage based on the number of running client transactions, and (2) a thermal model that discovers correlation between resource utilization and their temperatures. *TERN* is conducive to predicting thermal trends of diverse workload conditions with a changing transaction mix. *TERN* judiciously adjusts the models to maintain prediction accuracy for dynamically changing request patterns. The experimental results show that *TERN* provides a simple yet powerful solution for resource provisioning in thermal-aware data centers where exist rapidly changing workload conditions.

In the last part of this dissertation, we propose an erasure-coded data archival system called *aHDFS* for Hadoop clusters, where $RS(k+r, k)$ Codes are employed to archive rarely accessed replicas in the Hadoop distributed file system or HDFS to achieve storage efficiency in data centers. We develop two archival strategies (i.e., *aHDFS–Grouping* and *aHDFS–Pipeline*) in *aHDFS* to speed up the data archival process. *aHDFS–Grouping* keeps each mapper's intermediate output *key–value* pairs in a local *key–value* store. With the local store in place, *aHDFS–Grouping* merges all the intermediate *key–value* pairs with the same key into one single *key–value* pair, followed by shuffling the single *key–value* pair to reducers to generate final parity blocks. *aHDFS–Pipeline* forms a data archival pipeline using multiple data node in a Hadoop cluster. Unlike *aHDFS–Grouping*'s shuffle and reduce phases, *aHDFS–Pipeline* delivers the merged single *key–value* pair to a subsequent node's local key-value store. Last node in the pipeline is responsible for outputting parity blocks. The experimental results show that *aHDFS* can significantly improve the overall archival performance of the *Baseline* system.

Acknowledgments

This dissertation would not have been completed without invaluable guidance, help and experience sharing from the people who constantly support and encourage me during my study at Auburn University.

It is a great pleasure to thank those who made this dissertation possible.

First and foremost, I would like to give my sincere and deepest gratitude to my advisor, Dr. Xiao Qin, for his encouragement, guidance, and support, trust in me and my work. He is such a nice advisor, teacher and friend who is always equipped with infectious enthusiasm and unbounded energy even when the road ahead seemed long and uncertain. He opens the research door for me and bring me to a brand new, interesting and challenging world, and lead, facilitate and training my research skills and ability with his extensive knowledge, great patience and charm personalities.

I would also gratefully thank for my dissertation members, Dr. Saad Biaz, Dr. Wei-Shinn (Jeff) Ku and Dr. Hari Narayanan who gave me numerous valuable advices not only for this dissertation, but also my entire Ph.D program in Auburn University. I took Advanced Algorithm and Artificial Intelligence lectured by Dr.Hari Narayanan helped me building a strong ability of designing and applying algorithm. Dr. Saad Biaz 's lecture Advanced Computer Architecture cultivates me with more logical and comprehensive ability to know and analyze computer system and architecture. Database System lectured by Dr. Wei-Shinn (Jeff) Ku gave me deeper understanding of relation database which is extremely helpful for my experiments in second part of this dissertation.

Futuremore, I also gratefully thank Dr. Fa Foster Dai from the Department of Electrical and Computer Engineering for serving as my dissertation outside reader, reading my dissertation, and picking a time slot from his busy schedule to join my dissertation defense.

Working with our research group is fantastic. I own my gratitude to formal and current my research group members: Xunfei Jiang, Tausif Muzaffar, Ji Zhang, Chetan Prakash Somani, Ajit Chavan, Yusheng Ding, Zhitao Gong, Xiao Li, Yangyang Liu, Wei Liu, Liang Tang, Shubbhi Taneja, Wenlu Wang and Yi Zhou who helped me with paper writing, experimental result collection and group discussion. I will also like to thank all the professors and students in the Department of Computer Science and Software Engineering, who create and maintain an excellent atmosphere for study and research.

Finally and most important, the endless understanding, love and support from my family and relatives is the most powerful strength that keeps me fighting, growing for my research. I offer them with this dissertation and achievement.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the growth of repaid energy, power and cooling cost in current data centers, numerous of methods on energy, power and thermal efficiency have been proposed in past years. In order to tackling those challenges, various of energy, power and thermal efficiency techniques and models have been introduced to achieve conserving energy consumption and reducing cooling cost. [29][91][47] focus on achieving network energy efficiency in data. JouleSort, an external benchmark for evaluating the energy efficiency of wide range computer systems [73]. Additionally, lots of models have been build as guidance to achieve energy efficiency and energy optimization [20]. [75][62]. In contrast to energy efficiency, cooling cost make another big contribution in nowdays data center which accounts for almost half of total cost. There are a large bodies of study have been paid on, such as Cool Job Allocation obtains cost saving by placing jobs at cooling-efficiency locations in data centers [9]. Most of researches emit cooling cost conservation idea from smart workload scheduling, dynamic resource allocation and management and thermal aware models perspectives [89][82][46][45][34].

In addition to thermal and energy efficiency, investigating resource usage efficiency in data center is also another focused filed of this dissertation. For example, how to make the best use of front-end and back-end servers in a large e-commercial website such as Amazon Web Service to achieve low latency. [43][53][23] paid attention to provide resource for coming requests dynamically. However, another group of researchers obtained resource efficiency by taking advantage of erasure coding mechanism to minimize storage capacity consumption [92][58][67][59].

This dissertation consists of three parts. The first part presents a thermal benchmark-ThermoBench to evaluate thermal efficiency of computing and storage clusters; the second part develops a self-adjusting resource and thermal models; at the last part, a erasure-coding archival system aHDFS is introduced to boost the archival performance in Hadoop clusters. This chapter is organized as follows. Section 1.1 elaborates motivations of our ThermoBench. The illustrate motivation of incorporated resource and thermal models in section 1.2. Section 1.3 reveals our intention of aHDFS. Finally, section 1.4 illustrates the organization of this dissertation.

1.1 Motivation of ThermoBench

There has been no focus on a comprehensive benchmark for thermal management in the context of cluster computing. Thermal efficiency benchmarks for clusters providing services to client-server computing applications become critical. In this chapter, we propose ThermoBench aiming to evaluate the thermal efficiency of clusters running in data centers.

In data center environments, energy cost is one of the significant components of operational costs [48]. For example, a data center containing 1000 racks consumes 10MW total power per year [63]. Much attention has been paid to the development of high performance and high energy efficient clusters in data centers, because designing energy-efficient and environmental friendly clusters is greatly desirable [96]. Unfortunately, previous research on clusters has been primarily focused on energy efficiency improvement. There is a growing demand for thermal management techniques tailored for clusters.

The purpose of optimizing thermal efficiency in data centers is two-fold. First, energy cost in data centers can be significantly reduced by improving thermal efficiency of the centers, which includes a highly efficient cooling infrastructure coupled with advanced thermal management techniques for cluster computing systems. Second, as the number of computing nodes housed in data centers grows, minimizing cooling cost of the data centers becomes increasingly difficult [64][37]. Evidence demonstrates that high cooling cost in data centers could be reduced by minimizing the energy dissipation in cooling systems [51][88][35].

Reducing outlet temperatures or optimizing air recirculation is a practical way of improving energy efficiency [84]. In addition to direct temperature control, thermal-aware load balancing strategies lead to good temperature distribution [51] [88].

To optimize thermal efficiency of data centers, we have to develop benchmarks to assess the effectiveness of thermal management schemes deployed in clusters. Although in a handful of prior studies, benchmarks have been developed for energy efficiency in data centers (see, for example, [44][64]), the exiting benchmarks are focused on energy efficiency rather than thermal efficiency. Furthermore, growing attention has been paid to the improvement of computer thermal efficiency [65][90]; there has been little focus on the thermal efficiency of clusters in the context of client-server computing, which is quite popular in modern data centers. In this study, ThermoBench pays particular attention to thermal efficiency of clusters running scalable client-server enterprise applications in data centers.

ThermoBench applies various types of transactional requests in the TPC-W [3] benchmark (see section 3.3.1) to investigate thermal behaviors of multiple hardware components such as CPU and disk. Each transactional request has its own hardware-resource-demanding characteristics, which exhibits unique thermal behaviors. For example, CPU-intensive transactional requests stressing CPUs boosts up CPU temperatures; in contrast, I/O-intensive transactional requests can lift up disk temperatures. In addition, one unit resource usage of CPUs and disks respectively leads to different temperature increases in CPUs and disks. ThermoBench aims to uncover internal relationships of each request type's CPU and disk thermal behaviors in both front-end and back-end servers under changing arrival rates. With the thermal behaviors information measured by ThermoBench, one may adopt thermal-aware resource management strategies to improve thermal efficiency during the course of transactional request dispatching.

We start this study by investigating the criteria, metrics as well as challenges of developing thermal efficiency benchmarks. We focus on workload scenarios, thermal profiling and performance metrics, which help in investigating thermal efficacy optimizations. We characterize workload conditions in cluster computing environments in forms of client sessions of multiple requests. In the development of ThermoBench, we incorporate the TPC-W benchmark to change transactional requests mix and load over time. In doing so, we are able to resemble real-world applications running in data centers. Then, we apply ThermoBench to evaluate the thermal efficiency of a real-world cluster running various client-server enterprise applications. Our experimental results show that ThermalBench provides a simple yet efficient benchmark solution for assessing thermal efficiency of modern clusters in data centers.

1.2 Motivation of TERN

Thermal management plays a vital role in modern data centers. Traditional thermal models are mainly focused on static and non-changing workloads. To address this issue, we propose a self-adjusting thermal model called TERN to predict thermal behaviors of hardware resources provided for client sessions. We characterize the resource usage and thermal behaviors for various transaction types. We show how to apply TERN to predict thermal trends for three real-world scenarios under changing workload conditions.

Our thermal model is motivated by the following four factors in the context of data centers:

- The growing importance of thermal management in modern data centers.
- The pressing needs of prediction models to guide dynamic resource provisioning.
- Fine granularity modeling for resource provisioning and thermal optimization.
- Self-adjustment for rapidly changing transaction mixes.

Data centers house computing clusters that consist of thousands or even ten thousands of servers. A current trend shows that computing and storage capacity demands of data centers exponentially increase due to big data explosion [6]. Gartner predicted that a big data center's hardware expense will take up 71 percent (almost $126.2 billion) of the worldwide data centers hardware expense [83]. Such a huge infrastructure brings a surge of supplying energy cost, which is in the order of millions of dollars (see also recent reports in [66]). Two major contributing factors of energy cost in data centers: energy consumed by data nodes (e.g., CPU, disk, memory, fans, network) and energy caused by cooling systems (e.g., *Computer Room Air Conditioners (CRACS)*). Cooling cost can be as high as half of the total energy costs depending on data centers' power usage effectiveness (PUE). There is a large body of studies on run-time reduction of cooling and energy cost through thermal-aware task dispatch and migration strategies [10][52].

It is not uncommon for a 1000-rack data center to consume 10MW total power per year [48]. A handful of studies have been focused on modeling energy efficiency of data centers. For example, Allalouf *et al.* introduced an energy model to estimate power consumption of storage cluster in data center [7]. Pan *et al.* simulated data centers using their developed energy models [61]. Increasing attention has been paid to the development of high performance and energy-efficient clusters in data centers [96]. Apart from these energy efficiency studies, there is a large body of thermal management techniques tailored for clusters in data centers.

There are two factors driving us to optimize thermal efficiency in data centers. First, cooling cost can be reduced by minimizing energy consumed by cooling infrastructures coupled with advanced thermal management techniques [14][42]. Second, maintaining low cooling cost for large-scale data centers with an increasing number of nodes becomes a grand challenge [64][37]. Bringing down energy consumption of cooling systems largely depends on thermal behaviors of data centers. Several approaches have been investigated to avoid high temperatures in data centers. For example, preventing server temperatures from moving above a red-line (i.e., *hot spots*) reduces hot air recirculated from outlets into inlets of nodes. Thermal management not only decreases cooling cost, but also improves server reliability. The thermal model developed in our study can be adopted for building thermal management modules. In addition, our model provides a guideline for resource provisioning of applications like web searching and e-commercial under rapidly changing workload conditions.

Internet services have become an indispensable part of people's everyday life. These services are expected to instantly respond to requests while guaranteeing performance requirements. Any delay in request response may cause thousands of dollar losing. Typical internet services tend to exhibit dynamical regular long-term (e.g., weekly or daily) variations as well as short-term fluctuations due to flash crowds (e.g., holiday season sales) [87]. There are numerous documented examples of Internet applications that faced an outage due to an unexpected overload. These outage problems reveal that well predicting patterns of services (e.g., its peak workload) and real-time monitoring are extremely important for dynamic resource provisioning to deal with unexpected overload. For instance, Amazon's website suffered a forty-minute downtime due to an overload during the popular holiday season in November 2000 [1]. Most internet services consist of an assortment of transactions; analyzing each type of transaction request in terms of resource demands will be of help for real-time resource provisioning. Profiling internet services at the request level not only offers fine-granularity resource allocations to requests, but also provides thermal information to improve energy efficiency of data centers.

It is challenging to construct a thermal model for dynamic resource provisioning in the context of internet applications in data centers because of the following three reasons.

- Each workload has its own resource-requirement characteristics (e.g., some are computing intensive whereas others are data-intensive).
- A wide range of hardware settings have various thermal behaviors.
- Integrating multiple models to predict thermal trend may worsen prediction errors.

In this study, we adopt the divide and conquer approach to seamlessly integrate two models into TERN to overcome the aforementioned challenges. The first model is a resource utilization model, which is responsible for building up correlation between hardware utilization and request patterns (e.g., arrival rate). The second one is a thermal model to predict thermal behaviors of processors and disks. We integrate these two models into TERN, which can adjust modeling parameters to achieve low prediction errors for dynamically changing request patterns.

1.3 Motivation of aHDFS

Existing disk-based archival storage systems are inadequate for Hadoop clusters due to the ignorance of data replicas and the map-reduce programming model. To tackle this problem, we develop an erasure-coded data archival system called *aHDFS*, which archives rare accessed data in large-scale data centers to minimize storage cost. aHDFS exploits parallel and pipelined encoding processes to speed up data archival performance in Hadoop distributed file system (HDFS) on Hadoop clusters. In particular, aHDFS leverages the three-way data replicas and the map-reduce programming models in Hadoop cluster to boost the archival performance in HDFS. We show how to accelerate the encoding process in aHDFS by the virtue of data locality of three replicas of blocks in HDFS.

The following three factors motivate us to develop the erasure-code-based archival system for Hadoop clusters:

- a pressing need to lower storage cost of large-scale data centers,
- high cost-effectiveness of erasure-code storage and,
- the popularity of Hadoop computing platforms.

Reducing Storage Cost. Petabytes of data are nowdays stored in large distributed storage systems like the Google File System (GFS) [27], the Hadoop Distributed File System (HDFS) [12], the Windows Azure Storage [16]. In 2012, two millions of search queries received per minute; by 2014, that number has more than double. Every minute, 2.5 millions pieces of content shared in Facebook, 50 thousand applications downloaded by Apple users, 200 million messages sent via Emails [50]. Such a massive amount of data demands large-scale storage systems maintained in data centers. With an increasing number of storage nodes installed, the data storage cost goes up dramatically.

A large number of nodes leads to a high possibility of failures caused by unreliable components, software glitches, machine reboots, maintenance operations and the like. To guarantee high reliability and availability in the presence of various types of failures, data redundancy are commonly used in cluster storage systems. Two widely adopted fault-tolerant solutions are replicating additional data blocks (i.e., 3X-replica redundancy) and storing additional information as parity blocks (i.e., erasure-coded storage). For example, the 3X-replica redundancy is employed in Google's GFS [27], HDFS [12], Amazon S3 [76] to achieve fault tolerance. Also, erasure-coded storage is widely used in cloud storage platforms (e.g., Windows Azure and Facebook HDFS [16] [40] [30]) and data centers [86] [26] [56].

Applying Erasure-Coded Storage. Although 3X-replica redundancy (a.k.a., three-way replication) achieves high performance than erasure-coded schemes, 3X-replica redundancy inevitably leads to low storage utilization. Cost-effective erasure-coded storage systems are deployed in large data centers to achieve high reliability at low storage cost [30] [86]. Reed-Solomon code [71] is a popular family of erasure codes used in Google's ColossusFS [2], Facebook's HDFS [86] [78], and several other storage systems [70] [80].

The Reed-Solomon code (RS) has a general 1.5x storage overhead in ColossusFS [2]; the Facebook's HDFS reduces the storage overhead down to 1.4x by adopting RS(k+r,k) (e.g., (10+4,10)), the overhead of which is half less than that of 3X-replica redundancy schemes. In addition, most of data are accessed within a short duration of the data's lifetime. For example, over 90% of accesses in a Yahoo! M45 Hadoop cluster occur within the first day after data creation [39]. We are motivated to devleop an economically friendly aHDFS to archive data replicas in Hadoop clusters using erasure codes.

Archiving Data in Hadoop. Hadoop is a MapReduce implementation running on clusters, where HDFS stores data to offer high aggregate I/O bandwidth. Hadoop is a simple yet efficient parallel and distributed computing framework providing high scalability and fault tolerance [21]. In past few years, Hadoop has been widely applied to support the development of from both commercial and scientific applications [72]. The popularity of Hadoop inspires us to develop data achival schemes for HDFS to maintain a low data storage cost for Hadoop clusters deployed in large-scale data centers.

Salient Features of aHDFS. We make use of 3X-replica redundancy in HDFS to boost archival performance in clusters by the virtue of the MapReduce programming model in addition to reducing network traffic. We develop aHDFS - a parallel data archival system - for Hadoop clusters. aHDFS has the following five salient features.

- aHDFS manages multiple Map tasks across the data nodes that are archiving their local data in parallel.
- Two archival schemes (i.e., parallel archiving and pipeline archiving) are seamlessly integrated into aHDFS, which switches between the two techniques based on the size and locality of archived files.
- Intermediate parity blocks generated in the Map phrase substantially lower the I/O and computing load during the data reconstruction process.
- aHDFS reduces network traffic among nodes during the course of data archiving, because aHDF takes the advantage of the good data locality of the 3X-replica technique, where there is a high possibility that each file's different blocks are residing in one node.
- We develop aHDFS on the top of Hadoop system, allowing a system administrator to easily and quickly deploy our aHDFS without having to modify and rebuild HDFS in Hadoop clusters.

Contributions. The contributions of this study are summarized as follows:

- We propose two efficient archival techniques - parallel archiving and pipeline archiving. These two schemes make use of the 3X-replica data layout to speed up archival performance.

- We apply the MapReduce programming model to develop the aHDFS system, where the two data archival techniques are implemented for Hadoop clusters.

- we develop a analytical model incorporating of several parameters(e.g., block size, the number of Mapper task, block numbers and so on) to evaluate the two techniques' archival performance.

- We conduct extensive experiment to investigate the impacts of the block size, file size, the number of blocks, and block location on the performance of aHDFS.

1.4 Organization

This dissertation is organized as follows. In Chapter 2, related work reported in the literature is briefly reviewed. We introduce our ThermoBench in chapter 3. Then, in Chapter 4.2, we build a self-adjusting thermal model for dynamic resource provisioning in order to achieve thermal and resource efficiency. The last study presented in this dissertation is an erasure-coded data archival system for Hadoop clusters in chapter 5. Finally, we summarize the contributions of this dissertation and comments for future research in chapter 6.

Chapter 2
Related Work

As Chapter 1 mentioned, numerous researches has been extensively paid on thermal efficiency and resource efficiency during past decades. This chapter briefly reviews existing approaches that related to energy efficiency, thermal efficiency, dynamic resource provisioning, erasure-coding data archiving.

2.1 ThermonBench

There is a large body of prior studies on optimizing energy efficiency of clusters in data centers. For example, Ramapantulu *et al.* analyzed the energy efficiency of mixing high-performance and low-power nodes in a cluster [69]. Their findings suggest that mixing high-performance and low-power nodes improves energy-efficiency of traditional homogeneous datacenter clusters. Feller *et al.* evaluated the performance of Hadoop clusters in the traditional model of collocated data and compute services as well as the impact of separating out the services [25]. They conducted an energy efficiency evaluation of Hadoop on physical and virtual clusters in various configurations.

A few studies have been devoted to power management solutions for energy-efficient storage clusters. Huang *et al.* developed an energy saving technique for erasure-coded storage clusters [33]. Chai *et al.* designed an energy-efficient method to significantly reduce data migration overhead in storage clusters [17]. Manzanares *et al.* proposed an energy-aware prefetching strategy called PRE-BUD for storage clusters [48].

In clusters and data centers, increasing attention has been paid to evaluating the energy efficiency of computing systems. For example, Rivoire *et al.* developed a benchmark system called JouleSort to evaluate the energy efficiency of a wide range of computer systems from clusters to handhelds [74]. Very recently, Schall *et al.* recommended extensions, refinements, and variations for the TPC benchmarks; they proposed a new benchmarking paradigm that includes realistic power measures [79].

Several novel solutions have been proposed to optimize the thermal efficiency of clusters. In 2012, El-Sayed *et al.* conducted a multi-faceted study of temperature management in data centers [24]. They made use of a large collection of field data from various production computing environments to investigate the impact of temperature on the reliability of storage and memory subsystems and server reliability as a whole. Liu *et al.* predicted renewable energy as well as computing resource demand; their predictions are used to guide a workload management plan that schedules workload and allocates computing resources within a data center to improve cooling efficiency [46]. Very recently, Jiang *et al.* investigated the thermal profile of storage servers and developed a thermal model to estimate the outlet temperature of a storage server [37]. Since these existing thermal management systems and models are designed for optimizing thermal efficiency, they are not focused on the development of thermal-efficiency benchmarks. Unlike the prior studies, our ThermoBench aims to offer benchmarking workloads for the purpose of thermal-efficiency evaluations.

2.2 TERN

2.2.1 Resource Management and Provisioning

There are a handful of studies focusing on resource management and provisioning. For instance, Doyle *et al.* proposed a model-based approach to utility resource management for coordinated provisioning of memory and storage resources [22]. This approach enables systems to achieve efficient resource management, including differentiated service quality, performance isolation, storage-aware caching, and proactive allocation of surplus resources. Zhang *et al.* applied a regression-based approximation of the CPU demand of client transactions on a given hardware [95]. Their experimental results confirm that the regression-based approach provides an efficient capacity planning and resource provisioning of multi-tier applications under changing workload conditions. Nae *et al.* developed a dynamic resource provisioning method for massively multiplayer online games or MMOG operation in data centers [54].

2.2.2 Energy Efficiency in Data center

There is a large body of prior studies on optimizing energy efficiency of clusters because of rapidly increasing energy consumption of data centers [8][13]. Total energy used in global data centers in 2010 accounted for approximate 1.5% of total electricity use, for the US that number became 2.2% [41]. Various energy-efficient techniques have been proposed to lower the energy consumption of data centers. For example, Chase *et al.* improved the energy efficiency of server clusters by dynamically resizing active server sets [19]. They provided a solution to respond to power supply disruptions or thermal events by degrading service in accordance with negotiated service level agreements. Rajkumar *et al.* designed energy-efficient resource allocation policies coupled with the scheduling algorithms for managing data center resources for cloud computing [15]. They conducted a set of rigorous performance evaluation; their results reveal that the cloud computing model has immense potential of offering significant cost saving under dynamic workload scenarios. Feller *et al.* evaluated the performance of Hadoop clusters in the traditional model of collocated data and computing services as well as the impact of separating out the services [25]. They conducted an energy efficiency evaluation of Hadoop on physical and virtual clusters in various configurations.

2.2.3 Energy-efficient storage clusters

Some important studies have been devoted to the development of energy-efficient storage clusters. For example, Chavan *et al.* proposed a thermal-aware file assignment technique called TIGER aiming to reduce cooling cost of storage clusters. Huang *et al.* developed an energy-saving technique for erasure-coded storage clusters [33]. Ruan *et al.* designed an energy-efficient cluster storage system - ECOS - to improve energy efficiency and perfomrance of storage clusters [77]. Jiang *et al.* built an energy prediction model that estimates the total energy cost of data migrations [38]. In light of the model, they implemented a predictive energy-aware management system, which automatically selects the most energy-efficient method for data transfers. Chai *et al.* designed an energy-efficient method to significantly reduce data migration overhead in storage clusters [17].

2.2.4 Thermal efficiency in data centers

A few novel solutions have been proposed to optimize the thermal efficiency of clusters. In 2012, El-Sayed *et al.* conducted a multi-faceted study of temperature management in data centers [24]. A large amount of field data were collected from various production computing environments to investigate the impact of temperatures on the reliability of storage and memory subsystems and server reliability as a whole. Liu *et al.* investigated a prediction scheme to guide a workload management plan for renewable energy as well as computing resources in a data center [46]. Their solution schedules workload and allocates computing resources to improve cooling efficiency [46]. Very recently, Jiang *et al.* conducted thermal profiling of storage servers and developed a thermal model to estimate the outlet temperatures of storage servers [37].In 2010, Pakbaznia developed a power and thermal management engine or PTM, which is employed to determine the number and placement of active servers while simultaneously adjusting supplied cold air temperatures in order to minimize total power consumption for servers and air conditioning units) [57].

Although thermal management and provisioning schemes have been proposed for data centers, little attention has been paid to thermal-aware resource provisioning in data centers. The lack of thermal-aware resource provisioning techniques motivates us to focus on resource provisioning strategies to improve thermal efficiency of data centers where there are changing workload conditions. We aim to develop thermal prediction models to meet the needs of resource provisioning in data centers with a dynamic nature.

2.3 aHDFS

This section outlines the prior studies related to data archiving schemes in clusters.

General Data Archival Techniques. There are a handful of studies focusing on data archival techniques to save data storage space. For example, You *et al.* proposed the Deep Store - a large-scale archival storage system - to store immutable data efficiently and reliably for long periods of time [94]. Venti [68] is a network storage system intended for archival data. Venti enforces a write-once policy, preventing accidental or malicious destruction of data, coalescing duplicate copies of block, and reducing storage cost. Schwarz *et al.* investigated a disk-scrubbing process in a large archival storage system where disks are periodically accessed to detect disk failures [81].

Erasure-Coded Data Archival. Apart from the above traditionally techniques used for data archival, erasure codes are widely adopted in cloud storage platforms [18][40] and large-scale data centers [26][56]. Erasure-coded storage systems offer high data reliability and storage efficiency for data centers to achieve important data.

Huang *et al.* proposed an efficient scaling scheme called *Scale-RS* for Reed-Solomon-coded storage clusters [31]. Scale-RS has three outstanding features, namely, uniform data distribution, data movement minimization, and I/O performance improvement. Very recently, Huang *et al.* exploited pipelined encoding process by incorporating two data layouts (i.e., [D+P]cd and[3X]) to boost erasure-coded data archival performance on storage clusters [32]. The experimental results reveal that their archival schemes outperform the other solutions by at least a factor of 3.41 in a 9-node storage cluster. Pamies-Juarez *et al.* developed two solutions - a decentralized erasure coding process [60] and *RapidRAID* [58]. The former one reduces network traffic by up to 56% in the data archival process; whereas the latter one shortens coding times by up to 90% and 20% for archiving single object and multiple objects, respectively.

Unlike aforementioned *Scale-RS* and *RapidRAID* that were designed for clusters and RAID storage, our aHDFS aims to improve data archival performance for Hadoop storage clusters.

Archiving Data in the Hadoop Distributed File System. Although there are a large body of studies aims at data archival in data centers, little attention has been paid to speed up archival process in HDFS. A few pilot projects were kicked off to address this problem. For example, Gupta *et al.* developed an active data archival scheme for data warehouses incorporating HDFS to efficiently store data [28]. While Gupta's approach is focused on the performance optimization of querying archived data, our aHDFS pays attention to improving archiving performance through parallel computing and pipelining. In particular, our aHDFS incorporates parallel archiving and pipeline archiving that take full advantage of the MapReduce programming model and the 3X-replica data layout to boost data archival performance on Hadoop clusters.

Xia *et al.* presented a new erasure-coded storage system called HACFS as an extension to HDFS. Adapting to dynamically changing workloads, HACFS switches on the fly between two different erasure codes, namely, (1) fast code optimizing recovery performance and (2) compact code reducing storage overhead [93]. From the perspective of implementation, aHDFS differs from HACFS in that aHDFS implements the data archival functionality on top of HDFS at the application level rather than at the system level.

Chapter 3

ThermoBench: A Thermal Efficiency Benchmark for Clusters in Data Centers

This chapter proposes a thermal efficiency benchmark for clusters-ThermoBench, which evaluate the thermal efficiency of computing and storage clusters deployed in data centers. We shed light on the criteria, metrics and challenges of developing a thermal efficiency benchmark, and also pay particular attention to clusters running scalable client-server enterprise applications in data centers. Because these applications are quite popular in modern data centers, thermal efficiency benchmarks for clusters providing services to these applications become critical. We characterize workload conditions in such a cluster computing environment in forms of client sessions of multiple transactional requests. To resemble real-world applications, ThermoBench makes use of the TPC-W benchmark to change transactional requests mix and load over time. We apply ThermoBench to evaluate the thermal efficiency of a real-world cluster. Experimental results show that ThermalBench provides a simple yet powerful benchmark solution for assessing thermal behaviours of computing clusters in data centers as well as offers thermal-aware scheduling strategies during the course of requests dispatching.

The rest of this chapter is organized as follows. In Section 3.1, we discuss the criteria, metrics and challenges in the development of ThermoBench - our thermal efficiency benchmark. The design of ThermoBench as well as implementation issues can be found in Section 3.2. Section 3.3 provides an in-depth thermal measurement study and analysis, where ThermoBench is applied to evaluate thermal behavior of a real-world system.

3.1 Criteria, Metrics and Challenges

Recognizing that there is the lack of comprehensive benchmarks for thermal-efficiency evaluation in the context of cluster computing, we start this study by focusing on the criteria, metrics and challenges of the development of thermal-efficiency benchmark tools. The criteria outlined in this Section lay the preliminary foundation for the development of our ThermoBench.

The existing thermal management studies (e.g., [24][46][37]) summarized in Section 2.1 paid attention to the thermal-efficiency evaluation and comparison of novel techniques that aim at reducing cooling cost of data centers. In contrast, this study is focused on thermal-aware benchmark tools that address issues related to thermal profiling, energy efficiency, metrics, cooling cost, and continuously changing workload conditions.

Thermal Profiling. Ideally, a thermal benchmark tool provides an intuitive approach to profiling the temperatures of processors and disks as well as inlet and outlet temperatures of storage nodes. Thermal profiling plays a vital role in our ThermoBench, because temperatures of processors, disks, and storage nodes can significantly affect the cooling costs of data centers.

The thermal profiling feature of ThermoBench exhibits two immediate benefits. First, thermal profiling makes it possible to develop thermal models of processors, disks, and storage servers, which are building blocks of clusters in a data center. Second, thermal profiling allows developers to quantitatively evaluate the thermal behaviors of their newly implemented thermal management systems.

Generally speaking, thermal-aware benchmark tools allow researchers and developers to efficiently investigate thermal impacts of specific computing components (e.g., processors and hard disks) inside a cluster. For example, in this study we show how to use ThermoBench to assess outlet temperatures of storage servers based on both processor and disk utilizations.

Thermal Efficiency. A thermal-aware benchmark should be capable of measuring a computing system's performance in terms of thermal efficiency. To drive practical reductions in cooling cost, one has to pay attention to both system performance and thermal impacts. A computing cluster that has no adverse thermal impact and little cooling cost but takes forever to complete a task is unacceptable for any data center. Hence, the best performance metric should incorporate both performance and cooling cost. For example, one may use a product of execution time and cooling cost as a way to evaluate thermal efficiency. Existing CPU-centric benchmarks (see, for example, [13]) is performance oriented. In this study, we chose to incorporate temperature measures and cooling cost into the performance evaluation.

Metrics. Metrics are of a great importance for a thermal-ware benchmark to evaluate thermal efficiency of clusters. Three major metrics introduced in ThermoBench include the ratio of resource utilization to throughput (i.e., U/Th), the ratio of increased resource temperatures to throughput (i.e., Te/Th), and the ratio of increased resource temperature to its utilization (i.e., Te/U). Instead of adopting power as a metric like some other benchmarks (see, for example, the SPEC Power benchmark), ThermoBench concentrates on thermal perspective benchmarking, where temperatures directly reveal thermal behaviors of CPUs, disks, and various transactional requests, which in turn can offer a guideline in to achieving high thermal efficiency and reducing thermal recirculation.

We are motivated to incorporate the aforementioned three metrics in ThermoBench, because evidence shows that these is no directly relationship between power and temperature. Jiang *et al.* and Bellosa *et al.* discovered that CPU and disk temperatures are not only influenced by utilization and power, but also impacted by time [36][11]. Keeping track of instant and accurate thermal data rather than power is a practical approach to studying thermal efficiency. U/Th can be used to evaluate a clusterâĂŹs ability of processing transactional requests; Te/Th and Te/U represent hardware resource's thermal characteristics regarding throughput and utilization, respectively. The Te/Th metric is adequate to evaluate system thermal efficiency under specific transactional requests. Te/U reveals a cluster's thermal capability without considering any specific request type; rather, Te/U pays attention to resource utilization. With these three metrics in place, ThermoBench is able to evaluate a cluster's thermal efficiency from the perspective of thermal characteristics of hardware components supporting transactional requests. ThermoBench adopts the three new metrics to enable thermal management mechanisms to reduce cooling cost and to achieve high thermal efficiency.

Client-Server Applications. The goal of our benchmark is to resemble real-world client-server applications running on clusters in modern data centers. Therefore, workload conditions created by our ThermoBench should exhibit the following three features. First, the workload must be characterized in forms of client sessions of interdependent requests. Second, transaction mix and load of the workloads are changing over time. Last but not least, the workload should have an emphasis on both front nodes and service end nodes. These two types of nodes must be equally stressed by ThermoBench. The benchmark metrics should incorporate thermal efficiency of front nodes as well as server nodes in a cluster.

Multi-Class Behaviors. A thermal-efficiency benchmark should offer a contained abstraction of a cluster computing system by considering flows of requests in a queueing network. Keeping this goal in mind, we design ThermoBench in a way to capture multi-class behaviors, in which computing resource demands of various request classes are different. In doing so, we address the drawback of existing benchmarks of customers provide single-class workloads (i.e., multiple user behaviors are integrated into a single one).

A Combined Usage. A benchmark tool may measure thermal efficiency of a system when it is idle, busy, or anywhere between. Energy-saving techniques aim to minimize idle-mode power to reduce both energy and cooling cost. We design ThermoBench by considering real-world scenarios, where there is a realistic combination of the idle and busy modes. ThermoBench assesses the average thermal efficiency when a wide range of transactions submitted to servers are completed. This type of measurement allows researchers and developers to reduce cooling cost by designing and configuring constraints for data centers.

Simple and Effective. The ThermoBench tools must be representative of a broad range of workloads in the context of client-server applications running on clusters. Moreover, it should be easy to configure and run ThermoBench. For most existing (unoptimized) thermal profiles of the integrated cluster, a common approach to delineate the thermal map of data center is to deploy temperature sensors at inlet and outlet of servers. On one hand, this boosts up the expense of data center, on the other hand, it is extremely complicated and bothersome to dispose thousands of temperature sensors to all the nodes in a large data center. However, ThermoBench simplifies this situation by making use of inner deployed temperature sensors of CPUs and disks to outline the thermal map of data center. Additionally, ThermoBench also uncovers the relation of transactional requests' arrival rate and hardware temperatures. Therefore, with these relations, ThermoBench can directly obtain thermal map of data center from transactional requests arrival rate perspective instead of using sensors to measure.

3.2 The Design of ThermoBench

In this section, we delineate the design of the ThermoBench tool. We start addressing the design issues by proposing a three-tier architecture. The three-tier design enables us to support multiple users and scale up clusters without wiping out existing data. Next, we describe the thermal managers residing in front-end and back-end nodes. We also show how to manage temperatures and power in a way to improve energy efficiency of clusters.

3.2.1 Three-Tier Architecture

The framework of ThermoBench (see Figure 3.1) seamlessly integrates three tiers of nodes, namely, the pre-end, front-end, and back-end tiers. The three-tier architecture offers ample opportunities to expand the benchmark system size to a large scale, because it is flexible to link each front-end node to multiple back-end servers.

The pre-end tier, which is also referred to as the user tier, is responsible for resembling the behaviours of online users sending requests to be processed by clusters in a data center.

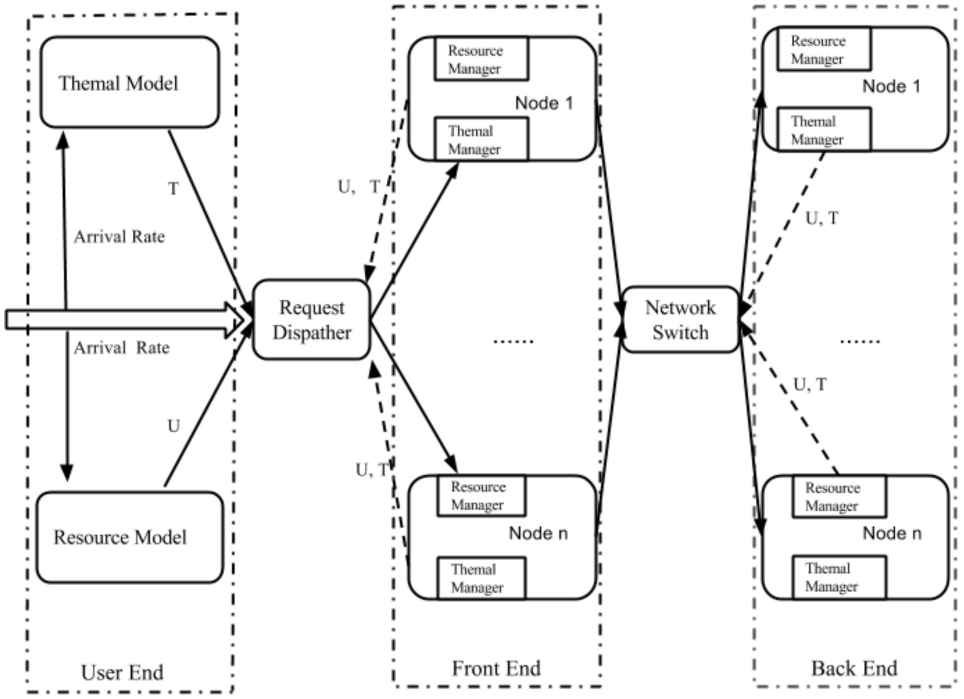


Figure 3.1: The Framework of ThermoBench.

Servers in the back-end tier are in charge of data processing as well as the database management. To maximize system throughput, the servers are independently responding requests in parallel.

The front-end tier becomes an interface between the pre-end and back-end tiers. The front-end tier of ThermoBench simplifies the underlying software components by providing a user-friendly interface for the user-end tier. More importantly, the front-end tier maintains a database containing application objects and links to the databases in the back-end database.

We adopt this three-tier framework in ThermoBench, because a wide range of client-server enterprise applications in data centers separate software components into front and back ends to simplify system development and maintenance. The most popular adoption of the three-tier architecture is the Amazon Web Service (a.k.a, AWS) [85]. Most of current e-Commercial websites apply a similar three-tier architecture as the computing infrastructure, because the three-tier design is flexible to scale up the number of back-end servers connecting to any specific front-end service. From the architecture perspective, one may straightforwardly extend three tiers to multiple tiers. For example, a scheduling tier can be seamlessly injected between user-end and front-end servers for the purpose of dispatching transactional requests. When it comes to workload characteristics in terms of resource demands, the three-tier architecture allows ThermoBench to categorize whether a transactional request is front-end-intensive or back-end-intensive, thereby offering ample opportunities to optimize thermal efficiency in front-end or back-end servers. In addition to the three-tier-based systems, ThermoBench is applicable to a wide range of transactional-requests-based systems (e.g., database system). The current version of ThermoBench is unable to evaluate non-transactional-request-based systems such as Hadoop clusters. As a future research direction, we plan to replace the TPC-W benchmark with a Hadoop benchmark in ThermoBench. The future version of ThermoBench is expected to handle the Hadoop cluster case. In our ThermoBench, the user-end and front-end tiers contain software modules manipulated by users, whereas data processing modules are running on servers in the back-end tiers.

In ThermoBench, we implement the front-end and back-end tiers using an off-the-shelf commodity cluster. It is worth noting that the number of nodes residing in the front-end tier is much smaller than that in the back-end tier. A load balancing module is incorporated in the front-end and back-end tiers, respectively. The load balancer in the front-end cluster evenly distributed requests submitted to the system to all the front-end nodes; the load balancer in the back-end tier aims to balance data processing workload among all the servers.

All requests issued by clients in the user-end tier are handled by a small cluster of front-end nodes. The front-end nodes rely on meta-data information along with request information to determine which server in the back-end tier should process the requests.

One of the most salient features of the user-end tier is controlling workload conditions. The important meta-data of each workload scenario managed by the user-end tier include arrival rate, transaction types (i.e., type-combination percentage), and requested computing resources. The request generator creates a list of requests according to the workload meta-data information.

ThermoBench is transparent to users, who are unaware of ThermoBench collects CPU and disk utilization and temperature information. ThermoBench makes of the deployed sensors to automatically collect performance and thermal data. A scheduling mechanism coordinates with ThermoBench to dispatch requests to front-end servers and back-end servers in a thermal-friendly fashion. Specifically, the scheduler relies on the thermal information (e.g., resource usage and temperatures) captured by ThermoBench to make resource management decisions.

3.2.2 Thermal Management

In addition to the workload control, thermal monitoring is made possible in the user-end tier, which relies on the thermal model, resource model, and the CPU/disk utilization to estimate thermal efficiency of each workload condition.

The CPU/disk utilization of servers are monitored by the front-end and back-end tiers. The resource monitoring module residing in the front-end and back-end nodes keep track of both CPU and disk resource utilization driven by requests issued from the user-end nodes.

The thermal management module is designed in a distributed fashion. In other words, there is one thermal manager implemented in each node of the cluster forming both the front-end and the back-end tiers. The thermal manager measures the CPU and disk temperatures as well as the node's inlet and outlet temperatures. The measured temperature information is locally stored in each server. The thermal managers transfer logged thermal information (e.g., CPU/Disk, inlet and outlet temperatures) back to the user-end tier in a batch manner for the purpose of thermal-efficiency analysis.

In addition to keeping track of temperature information, cooling down database servers in the back-end tier is a second goal achieved by the thermal management module. More specifically, when loads of CPUs and disks in a server have been very heavy (e.g., higher than 90%) for a long period of time (e.g., 10 minutes), the thermal managers in the database server immediately notify such a high load condition to the front-end cluster, which will stop issuing more requests to the hot database server. The thermal managers in the front-end cluster can manipulate temperatures of the back-end cluster, because the CPU and disk temperatures in a database server in the back-end tier can be brought down by decreasing the server's CPU and disk utilization.

Under light workload conditions in which CPUs and disks of some nodes have been sitting idle for a long period of time (e.g., 5 minutes), the thermal managers place these idle nodes into the sleep mode to offer energy savings. The front-end cluster stops issuing requests to any back-end server, which is transitioned into the sleep mode. Similarly, a front-end node in the sleep mode does not handle any requests issued from the user-end tier. In the back-end cluster nodes put into sleep mode will notice *requests assigner* not forwarding requests to this node if current nodes can satisfy all coming requests in safe situations. The sleeping nodes will be calling back again when there exists shortage of current working nodes.

3.3 Experimental Results

In this section, we first give a brief introduction of TPC-W, then we describe our configuration of the testbed on which ThermoBench is installed. At last, we apply ThermBench to evaluate the performance and thermal efficiency of the tested cluster.

3.3.1 TPC-W Benchmark

TPC-W is a transactional web e-Commerce benchmark simulating activities of a retail store website. An emulated user can browse and order books from the website using a remote browser emulator. There are total 14 types of activities accessing web pages of the website in TPC-W. For example, "Home" page resembles browsing home page of a website, "Product Page" represents viewing detail information for a book, and "Order Page" is an activity that orders books. The 14 transactional requests (a.k.a, accessing different web pages) can be grouped into two profiles, namely, WIPSb, WIPSo based on the ratio of load imposed to front-end and back-end servers (see Table 3.1 for two profiles combination percentage). The two profiles are of help in analyzing thermal behaviors in front-end and back-end servers.

Request	**Browsing mix**	**Shopping mix**	**Ordering mix**
WIPSb	95.00%	80.00%	50.00%
Home	29.00%	16.00%	9.12%
New Product	11.00%	5.00%	0.46%
Best Seller	11.00%	5.00%	0.46%
Product Detail	21.00%	17.00%	12.35%
Search Request	12.00%	20.00%	14.53%
Search Results	11.00%	17.00%	13.08%
WIPSo	5.00%	20.00%	50.00%
Shopping Cart	2.00%	11.60%	13.53%
Customer Reg.	0.82%	3.00%	12.86%
Buy Request	0.75%	2.60%	12.73%
Buy Confirm	0.69%	1.20%	10.18%
Order Inquiry	0.30%	0.75%	0.25%
Order Display	0.25%	0.66%	0.22%
Admin Request	0.10%	0.10%	0.12%
Admin Confirm	0.09%	0.09%	0.11%

Table 3.1: TPC-W benchmark combination percentage.

3.3.2 Node Configuration

The tested cluster consists of 19 nodes, among which there are two types of Linux servers connected through the fast Ethernet. We configure two nodes in the front-end tier and 16 nodes in the back-end tier; one node serves as the user-end tier.

Table 3.2 Testbed Configurations

	Node Type 1	**Node Type 2**
CPU	Intel(R) Celeron(R) 450@2.2GHz	
Network	1 GigaBit Ethernet network card	
Disk	WD-500GB Sata disk([5])	WD-160GB Sata disk([4])
Operating System	Ubuntu 10.04(lucid) Linux kernel 2.6.32-43	Ubuntu 10.04(lucid) Linux kernel 2.6.32-38

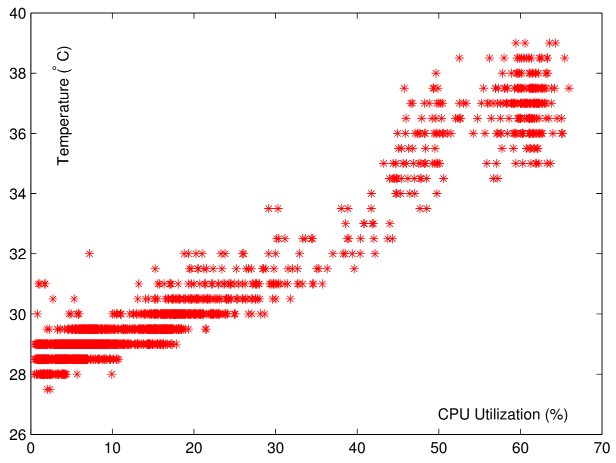
Table 3.2 summarizes the configuration details of these two types of servers performing as nodes of our testbed. In our experiments, CPU and disk temperatures are collected by ThermoBench using embedded device sensors. Inlet and outlet temperatures of the database nodes are monitored by four sensors attached to the nodes.

3.3.3 Hardware Resource Thermal Profiling

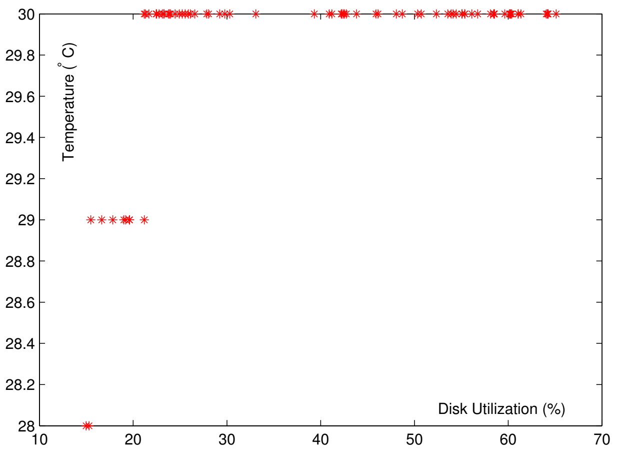
CPU Thermal Profiling

In the first experiment, we apply ThermoBench to conduct a CPU thermal profiling test. Fig. 3.2(a) shows average CPU temperature as a function of CPU utilization in the tested cluster.

The ThermoBench tool issues an multi-thread matrix process, which has CPU-intensive workload, to the server nodes of the cluster. ThermoBench manipulates CPU utilization of the nodes by tuning the request idle time period. Fig. 3.2(a) reveals that the average CPU temperature is 28 ℃ when CPUs are idle. This temperature equals to the ambient temperature. Fig. 3.2(a) also shows that the CPU temperature goes up when the CPU utilization increases from 1% to 65%. The preliminary results offered by ThermoBench suggest that the CPU temperature is very sensitive to CPU utilization. The thermal manager of a node is capable of reducing the node's CPU temperature by reducing CPU utilization.



(a) CPU Thermal Profiling



(b) Disk thermal profiling

Figure 3.2: CPU and Disk Thermal Profiling.

Disk Thermal Profiling

Now we employ ThermoBench to conduct a thermal profiling test on disks in the nodes. In this experiment, ThermoBench creates the disk-intensive workload using IOzone - a filesystem benchmark. ThermoBench gradually stresses disks by increasing disk utilization and measuring disk temperatures. Fig. 3.2(b) shows the impact of disk utilization on the average temperature of disks in the cluster.

Comparing Fig. 3.2(b) with Fig. 3.2(a), we observe that disks are less thermal sensitive than CPUs. When disks are sitting idle, disk temperatures are 28 ℃, which is the ambient temperature. The disk temperature increases from 28℃ to 29℃ when the disk utilization is anywhere between 15% and 21%. After the disk utilization is above the 21% mark, the disk temperature quickly goes up to 30℃. Interestingly, the disk temperature stays at the level of 30℃ when the disk utilization is ranging from 22% to 65%.

CPU and Disk Thermal Profile Analysis.

By comparing CPU and disk thermal profiles, we observe that CPUs have more significant impacts to temperature than disks. For example, with utilization increasing from 0 to 70%, CPU's temperature increases by 11℃ to 39 ℃ (see Fig. 3.2(a)); in contrast, disk

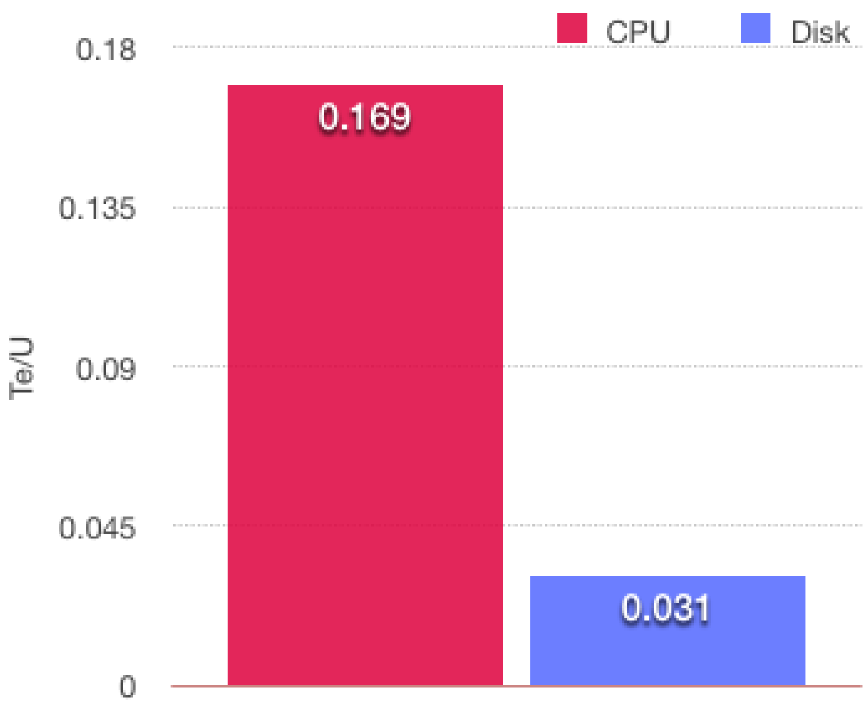
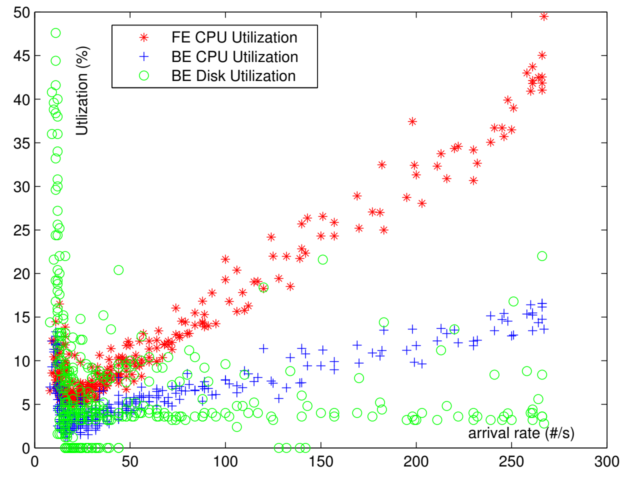
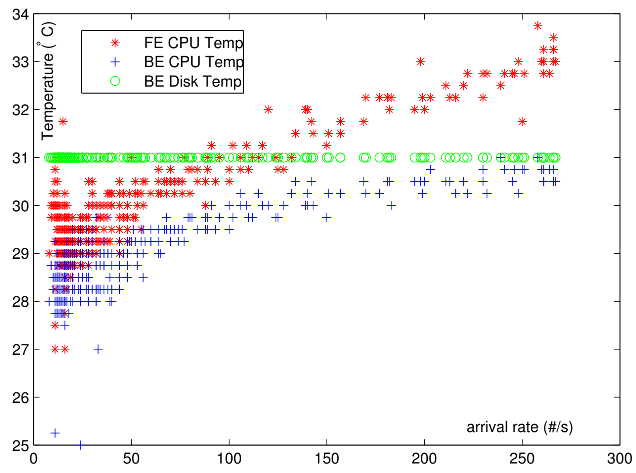


Figure 3.3: Comparison of Te/U of CPU and Disk

temperature only increases by 2 ℃ to 30 ℃ (see Fig. 3.2(b)). From the Te/U metric perspective, with one unit of utilization, CPUs have five fold thermal impact on clusters than disks (see Fig. 3.3). These results obtained from ThermoBench suggest that CPU-intensive transactional requests should be dispatched to multiple servers to avoid heating up a single server. Additionally, I/O-intensive requests can be dispatched to these servers that fully use hardware resources without substantially increasing server temperatures. The results from ThermoBench also suggest that servers handling CPU-intensive transactional requests should be located in a rack that makes the least contribution to heat recirculation in a data center.



(a) Resource Utilization



(b) Thermal Profiling

Figure 3.4: Resource Utilization and Thermal Profiling of *Home* Requests. FE: Front-End, BE: Back-End.

3.3.4 Thermal Profiling of Individual Transactional Requests

Thermal Profiling of Home Requests

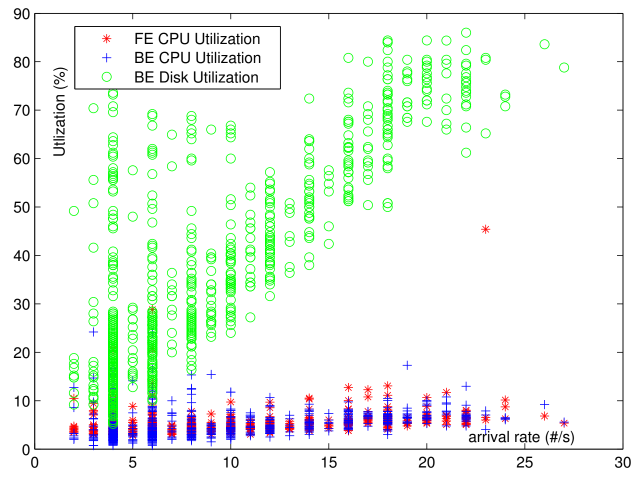
Now we make use of ThermoBench to investigate the thermal behaviours of home requests, which access homepages of websites. Each home request issued to a front-end web server seeks to access a homepage.

In ThermoBench, a homepage is the first webpage of the online book store in the TPC-W benchmark. The homepage contains the TPC (a.k.a., Transaction Processing Performance Council) logo, promotional books, and navigation options to the best-selling books and new books, search pages, a shopping cart, and an order status. A user may send a request to the web server to browse pages, which is comprised of a list of new or best sellers grouped by subject. A user may also search books using a title, an author name, or a subject. To process a home request, ThermoBench has to retrieve book information data from the back-end database server.

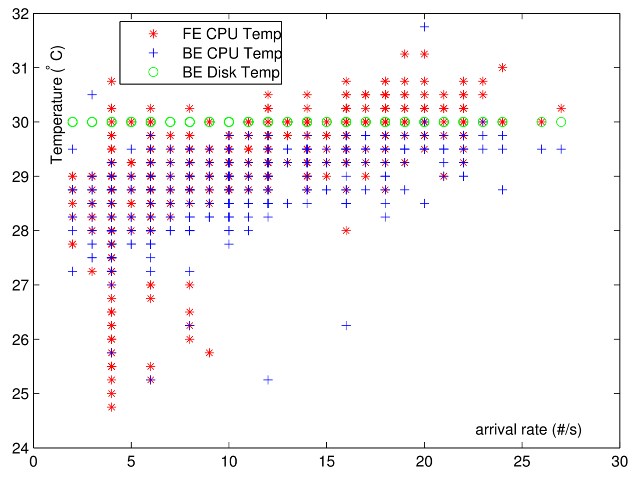
Fig. 3.4 plots the resource utilization and thermal behaviors of the front-end and back-end nodes processing home requests in the tested cluster. To investigate the impact of load on CPU and disk utilization, we increase the request arrival rate in ThermoBench from 5 No./Sec. to 270 No./Sec. with an increment of 5 No./Sec.

We observe from Fig. 3.4(a) that the front-end nodes are more sensitive to the arrival rate than the back-end nodes from the perspective of CPU utilization. One way to lower the CPU utilization of the front nodes under heavy load conditions is to wake up more front-end nodes in the cluster. Fig. 3.4(a) illustrates that the disk utilization of the back-end database servers is very high (e.g., 45%) when the arrival rate is anywhere between 10 No./Sec. and 50 No./Sec. The high disk utilization under a low arrival rate is attributed to the thermal manager that places a number of database servers into the sleep mode to conserve the energy, which in turn pushes up the resource utilization of the limited number of active servers. After an increasing number of servers are waked up, the servers' disk utilization stays at the low level of 5% when the arrival rate is in the range between 50 to 200 No./Sec. When the request arrival rate is as high as 200 to 270 No./Sec., the disk utilization of the database servers goes up to 22%. This is because after all the servers are in the active mode, the disk utilization of the database servers proportionally increases with the increasing arrival rate.

Fig. 3.4(b) shows the thermal behaviors of front-end and back-end nodes in the cluster when the arrival rate increases from 5 to 270 No./Sec. in ThermoBench. We observe that disk temperature is insensitive to the arrival rate, because in most cases the disk utilization in the back-end servers is very low (see also Fig. 3.4(a)). When the arrival rate is a small value (e.g., 50 No./Sec.), the CPUs in both front-end and back-end nodes share similar thermal behaviors. In contrast, the CPU temperature of the front-end nodes becomes much higher than that of the back-end nodes due to front-end nodes' high CPU utilization. This result suggests that one should pay particular attention to reducing the temperatures of the front-end nodes under heavy load conditions.



(a) Resource Utilization



(b) Thermal Profiling

Figure 3.5: Resource Utilization and Thermal Profiling of *Customer-Registration* Requests.

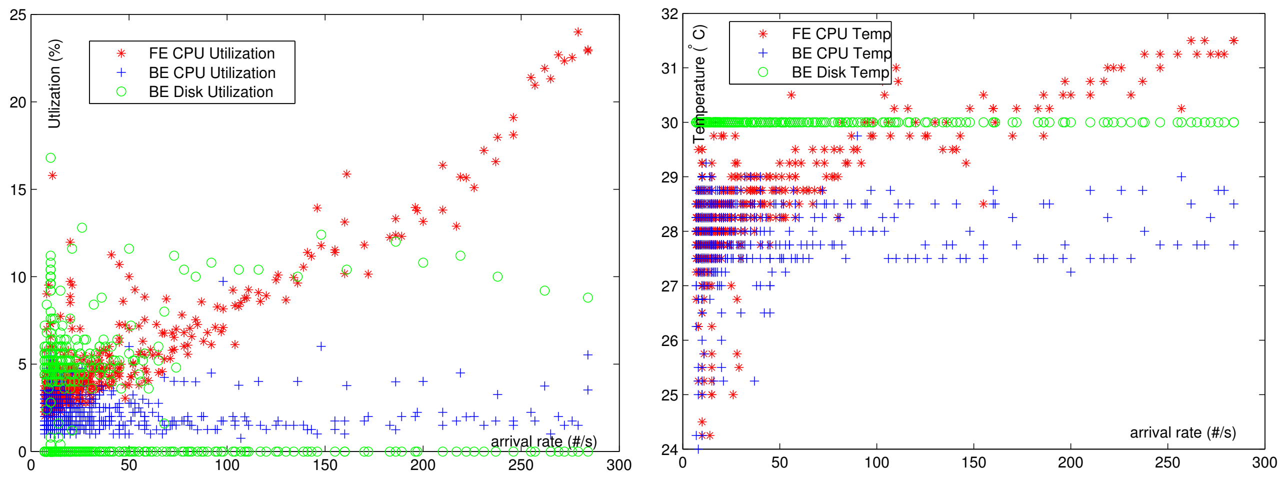
Thermal Profiling of Customer-Registration Requests

In this group of experiments, we focus on applying ThermoBench to investigate the thermal behaviors of *Customer-Registration* requests. This type of request allows users to submit registration information to register for website visits.

Each *Customer-Registration* request issued to a front-end web server seeks to access a homepage. In ThermoBench, we maintain a customer registration webpage, where a customer may login with an existing account or register with the user's birth date, name, address, phone number, email address, username, and passowrd. This registration webpage also offers search and back to home options. To respond customer register requests, ThermoBench records customer information to database servers in the back-end nodes.

Fig. 3.5 illustrates the resource utilization and thermal profiles of the front-end and back-end nodes processing customer register requests in the tested cluster. We set the arrival rate from 2 No./Sec. to 28 No./Sec.

We observe from Fig. 3.5(a) that the CPU utilization of both the front-end and back-end nodes are insensitive to the arrival rate. For example, when the arrival rate increases from 5 to 25 No./Sec., the CPU utilization only varies in a range between 2% and 12%. In contrast, the arrival rate has a significant impact on the disk utilization of the back-end nodes. For



(a) Resource Utilization (b) Thermal Profiling

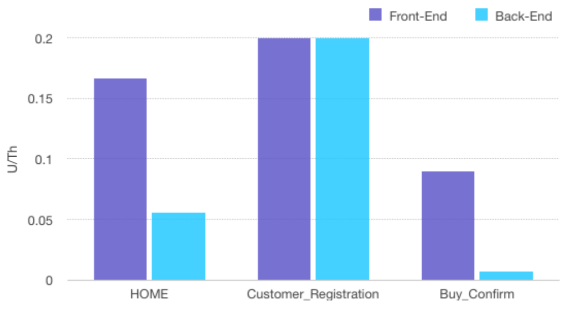
Figure 3.6: Resource Utilization and Thermal Profiling of *Buy-Confirm* Requests.

instance, when the arrival rate is 20 No./Sec., the back-end nodes' disk utilization is at a high level of 68-84%. These results from ThermoBench indicate that the customer registration requests form disk-intensive workload scenarios.

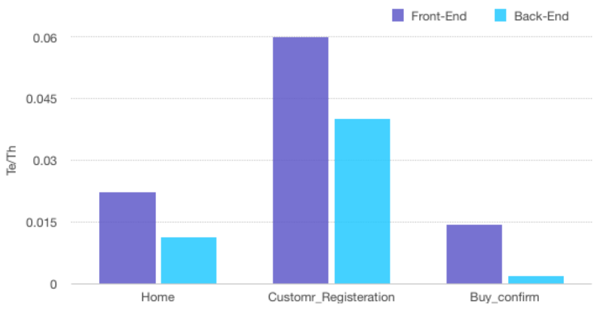
Fig. 3.5(b) shows the cluster's thermal profiles in terms of front-end and back-end nodes when the arrival rate increases from 2 to 28 No./Sec. The experimental results reveal that CPU temperatures of the front-end and back-end nodes increase when the arrival rate goes up. Interestingly, the arrival rate has a marginal impact on the disk temperatures of the back-end nodes. The results suggest that under workload conditions where *Customer-Registration* requests are a dominating factor, thermal management should be focused on CPU temperatures of clusters.

Thermal Profiling of Buy-Confirm Requests

Now we are in a position to study the thermal behaviours of *Buy-Confirm* requests. A *Buy-Confirm* request contains order information entered from the *Buy-Confirm* webpage, which shows what items a customer bought, total price, and an order number. This *Buy-Confirm* webpage provides the search and back-to-homepage options.



(a) Comparison of U/Th



(b) Comparison of Te/Th

Figure 3.7: CPU's U/Th, CPU's Te/Th Comparision of Three Types Individual Requests

Fig. 3.6 plots the resource utilization and thermal profiles of the front-end and back-end nodes handling *Buy-Confirm* requests when the arrival rate is set from 5 to 290 No./Sec. The experimental results plotted in Fig. 3.6(a) indicate that the CPU utilization of the front-end nodes is significantly affected by the arrival rate. For example, the CPU utilization increases from 2% to 24% when the arrival rate goes up to 290 No./Sec. from 5 No./Sec. Unlike the front-end nodes, the back-end nodes' CPU utilization is relatively independent of the arrival rate. Similarly, the disk utilization of the back-end nodes remains at a low level even when the arrival rate is increasing to as high as 290 No./Sec. These findings offered by ThermoBench imply that the *Buy-Confirm* requests are likely to make the CPU resources in front-end nodes busy.

Fig. 3.6(b) illustrates the cluster's thermal profiles when the arrival rate increases from 5 to 290 No./Sec. The experimental results show that the front-end nodes' CPU temperatures are noticeably affected by the arrival rate. In contrast, the arrival rate has very little impact on the back-end nodes' CPU and disk temperatures. We conclude from this group of experiments that when it comes to *Buy-Confirm* requests, one should pay attention to reducing CPU temperatures of front-end nodes in a cluster.

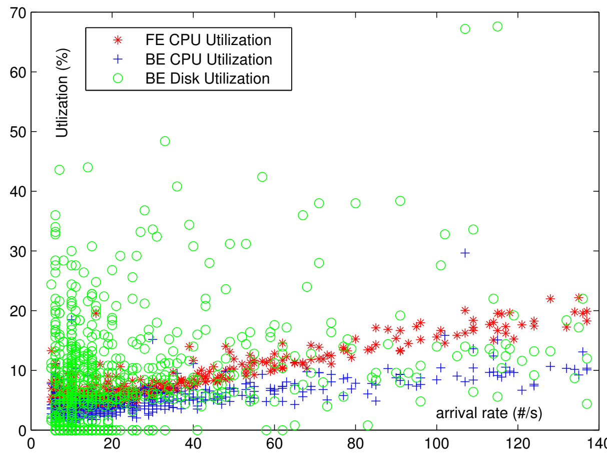
Individual Transactional Requests Thermal Profiling Analysis

Comparing the thermal profiles of the three request types, we observe that both the *Home* request and *Buy-Confirm* request have bigger thermal impacts to the front-end server than the *Customer-Registration* request. From the Te/Th perspective, the thermal contributions on CPU of these three types of transactional requests are 0.02, 0.06, and 0.014 with one request per second in the front end. We conclude that from the Te/Th metric perspective (see Fig. 3.7(b)), each *Customer-Registration* transactional request makes more contribution to CPU temperature increase than the other two requests no matter on front-end servers or back-end servers; in the back end, the *Buy-Confirm* requests have the least impact to CPU temperature among the three transactional requests. Fig. 3.7(a) illustrates that the CPU U/Th values of the *Home*, *Customer-Registration*, *Buy-Confirm* requests are 0.16, 0.2, 0.1 in the front-end server; the three requestsâĂŹ U/Th values in the back-end servers are 0.04, 0.2, 0.008 respectively. *Home* and *Buy-Confirm* requests more CPU load on the front-end servers than the back-end servers. The experimental results obtained from ThermoBench indicate that it is thermal friendly to dispatch these three types of requests to front-end servers that make little thermal impact to data centers. On the other hand, only the *Customer-Registration* requests make high disk load in the back-end servers (see Fig. 3.5(a)); whereas the other two requests have little thermal impact on disks in the back-end servers. We believe that grouping *Customer-Registration* with the other two types of requests can optimize disk usage while achieving thermal balance.

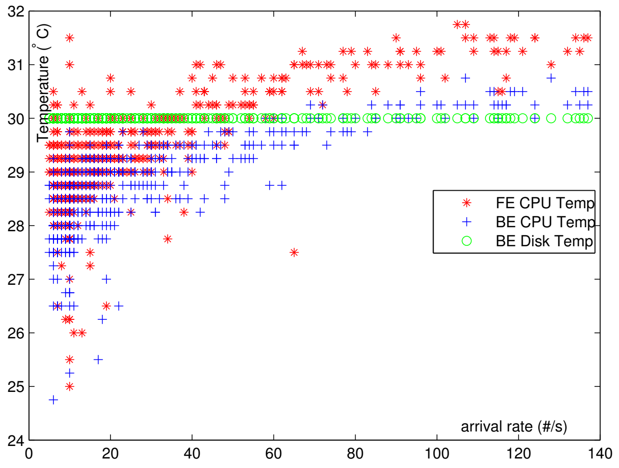
3.3.5 Thermal profiling of Three Type of Workload Mix

Thermal Profiling of the Web Browsing Workload

In this group of experiments, we use ThermoBench to investigate the thermal profiles of the cluster in mixed workload scenarios. The Web browsing load in ThermoBench aims to simulate a website access traffic where the majority (i.e., 95%) of online customers are



(a) Resource Utilization

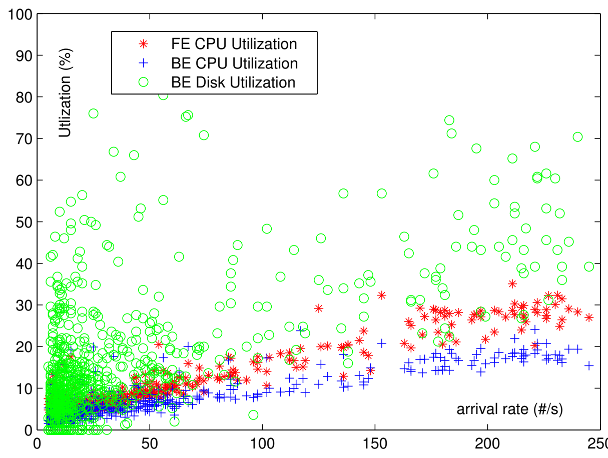


(b) Thermal Characteristics

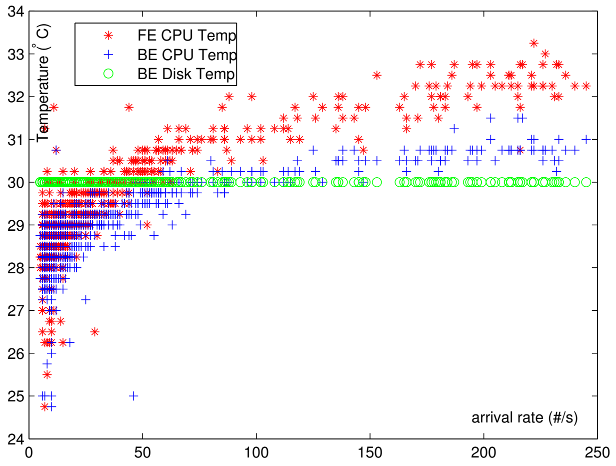
Figure 3.8: Resource Utilization and Thermal Characteristics of the Browsing Mixed Workload. FE: Front-End Nodes, BE: Back-End Nodes.

browsing the website. This type of mixed workload is a stress test on front-end nodes of a cluster.

Fig. 3.8 shows the resource utilization and thermal characteristics of the test cluster under the browsing mixed workload. We observe from Fig. 3.8(a) that the arrival rate has more noticeable impacts on the front-end nodes' CPU utilization than the back-end nodes' CPU and disk utilization. This result is expected because the browsing mixed workload imposes a stress test on the front-end nodes rather than the back-end ones. Consequently, Fig. 3.8(b) reveals that the temperatures of CPUs in the front-end nodes noticeably increase with the increasing value of the arrival rate. Fig. 3.8(b) also shows that the disk temperatures of the back-end nodes change very little with the increasing arrival rate; the CPU utilization of the back-end nodes rises from 25℃ to 30.2℃ when the arrival rate increases from 10 to 135 No./Sec. The results plotted in Fig. 3.8(b) suggest that the thermal management should be focused on CPU resources of clusters running the browsing workload.



(a) Resource Utilization



(b) Thermal Characteristics

Figure 3.9: Resource Utilization and Thermal Characteristics of the Shopping Mixed Workload. FE: Front-End Nodes, BE: Back-End Nodes.

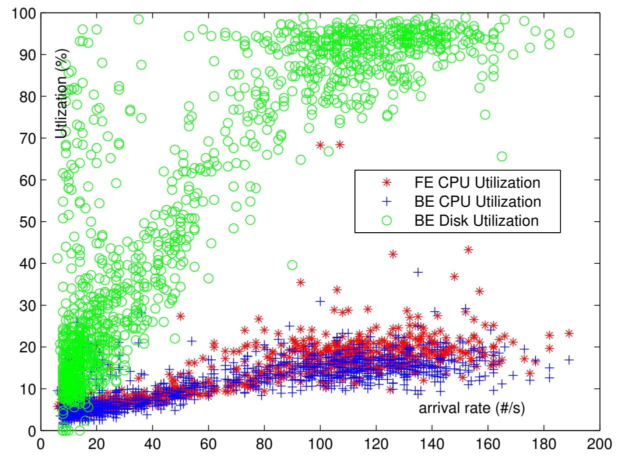
Thermal Profiling of the Shopping Workload

Now we run ThermoBench to simulate the shopping workload, where 80% of web requests are browsing and 20% of web requests are ordering items. The resource utilization and CPU/disk temperatures in the cluster are collected by the ThermoBench tool when the request arrival rate is set from 5 to 250 No./Sec.

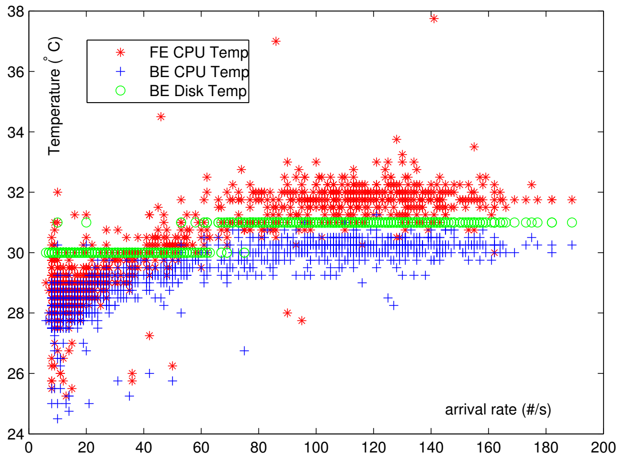
The resource utilization values plotted in Fig. 3.9(a) are very close to those shown in Fig. 3.8(a), except that disk load in Fig. 3.9(a) is heavier than that in Fig. 3.8(a). More interestingly, the heavy disk load imposed by the shopping workload does not cause any noticeable increases in disk temperatures (see Fig. 3.9(b)). The temperature trends of Fig. 3.9(b) are similar to those of Fig. 3.8(b). Again, the results shown in Fig. 3.9(b) imply that the cluster's CPU resources should be the focus of the thermal management under the shopping workload.

Thermal Profiling of the Ordering Workload

In this set of experiments, ThermoBench simulates the ordering workload, where 50% of the requests are browsing the webpages and 20% of the requests are ordering items from



(a) Resource Utilization



(b) Thermal Characteristics

Figure 3.10: Resource Utilization and Thermal Characteristics of the Ordering Mixed Workload. FE: Front-End Nodes, BE: Back-End Nodes.

the website managed by the cluster. ThermoBench measures the resource utilization and CPU/disk temperatures when the arrival rate is configured in a range between 5 to 190 No./Sec.

Unlike Figs. 3.8(a) and 3.9(a), Fig. 3.10(a) clearly indicates that the disk load of the back-end nodes is heavy when the arrival rate is larger than 100 No./Sec. For example, when the arrival rate equals to 120 No./Sec., the average disk utilization is 90%. Consequently, Fig. 3.10(b) shows that when the arrival rate becomes higher than 60 No./Sec., the average disk temperatures of the back-end nodes jump from 30℃ to 31℃. We conclude from Fig. 3.10 that when it comes to the ordering workload, cooling cost can be reduced by minimizing disk load in back-end nodes in addition to keeping down CPU load in both front-end and back-end nodes.

Chapter 4

TERN: A Self-Adjusting Thermal Model for Dynamic Resource Provisioning in Data Centers

In this chapter, we propose a self-adjusting model called TERN to predict thermal behaviors of hardware resources for client sessions. Our TERN contains two major components: (1) a resource utilization model being responsible for estimating hardware usage based on the number of running client transactions, and (2) a thermal model that discovers correlation between resource utilization and their temperatures. TERN is conducive to predicting thermal trends of diverse workload conditions with a changing transaction mix. We apply the TPC-W benchmark to characterize the resource demands of each type of transactions. Then, we conduct a thermal profiling study of the benchmark with various transaction mixes. TERN judiciously adjusts the models to maintain prediction accuracy for dynamically changing request patterns. Experimental results show that TERN provides a simple yet powerful solution for resource provisioning in thermal-aware data centers where exist rapidly changing workload conditions.

The rest of this chapter is organized as follows. Experimental testbed is described in Section 4.1. In Section 4.2, we delineate the design of TERN in a three-tier architecture. Then, we discuss the development details of TERN in Section 4.3. Section 4.4 shows how to make TERN adjust modeling parameters to minimize prediction errors. The models integrated in TERN are validated with the three workload mixes in Section 4.5.

4.1 Experimental Setup

In this section, we first outline our three-tier testbed; then, we discuss the configuration parameters of the testbed as well as the TPC-W benchmark.

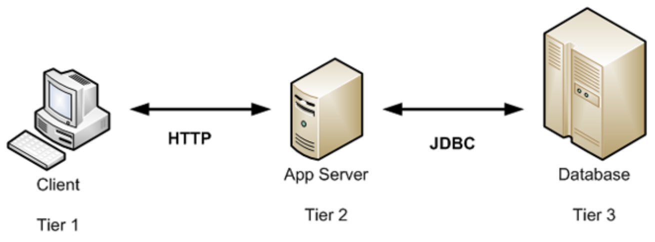


Figure 4.1: In the three-tier architecture, the presentation tier allows users to access web pages; the application tier is a logic tier offering services; and the data tier provides data management.

Hardware	
Computer	HP Proliant ML110 G6
CPU	Intel(R) Celeron(R) 450@2.2GHz
Memory	2 GB
Network	1 GigaBit Ethernet network card
Disk	WD-500GB Sata disk([2])
Software	
Operating System	Ubuntu 13.10 Linux Kernel 3.11.0-26
Web Server	Jboss 4.0.2
Database	MySQL 5.1.72

Table 4.1: Hardware and software configurations of the testbed

The three-tier application architecture shown in Fig. 4.1 is widely adopted in the industry for the implementation of scalable client-server applications. Such an architecture supports multiple users and can be scaled up under different settings. The presentation tier (a.k.a., tier 1 or client end) is the topmost level of applications, from which users directly access web pages. The application tier (a.k.a., tier 2 or front-end web server) is a logic tier, which is responsible for performing processes of a service. The data tier (a.k.a., tier 3 or back-end database server) is used to house and manage data. We built a three-tier testbed, the hardware and software configuration of which are summarized in Table 5.1.

TPC Benchmark W or TPC-W is a web server and database performance benchmark. TPC-W emulates e-commerce transactions where users browse and place orders on a bookstore web site [3][49]. In the TPC-W benchmark, there are 14 types of transaction requests

Tables	Fields	**Default Size**	**Modified Size**
ITEM	I_TITLE	60	1200
	I_PUBLISHER	60	2000
	I_DESC	500	5500
AUTHOUR	A_FNAME	20	200
	A_LNAME	20	200
	A_MNANE	20	200
CUSTOMER	C_UNAME	20	400
	C_UPASSWD	20	400
	C_LNAME	20	400
ADDRESS	ADDR_STREET1	40	400
	ADDR_STREET2	40	400
	ADDR_CITY	30	300
COUNTRY	—	—	—
ORDERS	—	—	—
ORDER_LINE	—	—	—
CC_XACTS	—	—	—

Table 4.2: Orignal and modified data tables. We manipulate the data tables to study the impacts of memory and disk activities on the thermal and performance impacts of the testbed.

(see table 3.1) and 8 data tables (see table 4.2). These requests are categorized into two camps, namely, (1) WIPSb - stress tests for web servers and (2) WIPSo - stress tests for database servers. The WIPSb workload is composed of a few database operations and a majority of web page browsing operations. On the other hand, a significant portion of operations in the WIPSo workload are database operations. TPC-W relies on a set of probabilities to simulate the changing behaviors of users. Using the weight of each activity type, TPC-W defines three web workload mixes including web *Browsing*, web *Shopping*, and web *Ordering*.

Table 3.1 outlines the detailed composition of each workload mix. For each experiment representing a workload mix, we run three hours, in which the first 15 minutes and the last 15 minutes are the warm-up and cool-down phases. The measures collected during the warm-up and cool-down phases are omitted in our evaluation.

Given the eight data tables, we modified the field size of table ITEM, AUTHOR, ADDRESS and CUSTOMER in order to study the impacts of memory and disk activities on the thermal and performance impacts of the testbed. Table 4.2 summarizes the characteristics of the modified datasets.

4.2 The Design of TERN

We start addressing the design issues by delineating the three-tier architecture, which supports multiple users in e-commerce applications. The three-tier TERN makes it possible to scale up the testbed without wiping out existing data. We also discuss the resource and thermal models used in TERN.

4.2.1 The TERN Framework

The TERN framework (see fig. 4.2) seamlessly integrates three tiers (i.e., the user-end, front-end and back-end tiers) of servers to resemble the behaviors of real-world client-server applications. The three-tier architecture offers ample opportunities to scale up systems, because it is flexible to deploy multiple nodes into the three tiers and to connect servers residing in different tiers.

In the design of TERN, we deploy a pre-end layer sitting between the user-end and front-end tiers. The pre-end layer incorporates the resource and thermal models (see Section 4.2.2) offering thermal and performance prediction services in the TERN framework. The important meta-data of each workload scenario managed in the resource and thermal models of the pre-end layer includes arrival rate, transaction types (i.e., type-combination percentage), and requested computing resources. One of the most salient features of the pre-end layer is the dynamic control of workload conditions. This feature is achieved by the

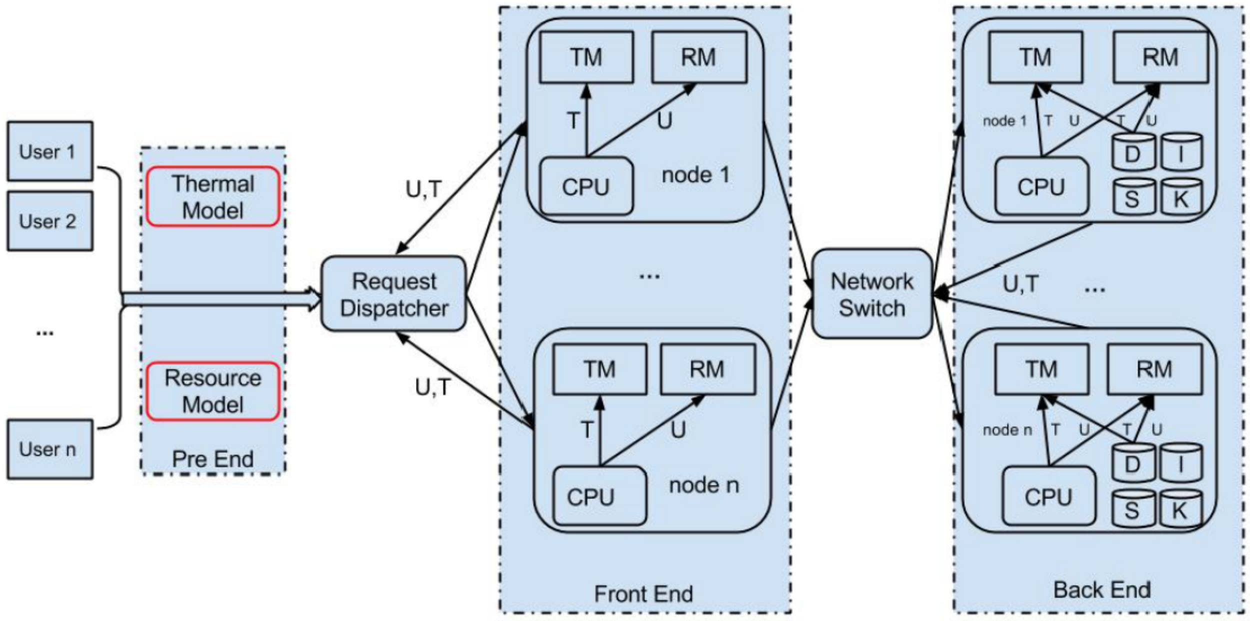


Figure 4.2: The TERN Framework Integrates the User-end, Front-end and Back-end Tiers to Resembling the Behaviors of Client-Server Applications. A Pre-end Layer is Connecting the User-end and Front-end Tiers. The Pre-end Layer Incorporates the Resource and Thermal Models (see Section 4.2.2) Offering Thermal and Performance Prediction Services.

virtue of a request generator resemble user behaviors by creating a list of requests according to the workload meta-data information.

We adopt the three-tier framework in TERN, because software components of a wide range of client-server applications running in data centers are partitioned into the front-end and back-end portions to simplify system development and maintenance. The user-end tier contains software modules manipulated by users, whereas the front-end and back-end tiers are comprised of data processing modules running on servers in data centers.

The user-end tier in TERN is mainly responsible for emulating the behaviours of online users, who are sending requests in client-server applications running on clusters in a data center. Before user requests are forwarded to the front-end web servers in TERN, these requests are processed by the thermal and resource modelling modules residing in the pre-end layer. The goal of the pre-end layer is to manage access patterns like requests arrival rate and request types. In addition, the pre-end layer predicts the amount of resources for the coming requests and the thermal behaviours of the testbed. The prediction information regarding resources and thermal profiles will provide guidelines for request dispatchers to manage workload among servers in the back-end tier in a thermally-efficient way.

The front-end tier of TERN facilitates a user-friendly interface for the user-end tier. More importantly, the front-end tier maintains links to the databases managed by servers in the back-end tier. The servers in the back-end tier are in charge of data processing as well as the database management. All the servers are independently responding requests in parallel to maximize system throughput.

Each server or computing node in the front-end and back-end tiers is equipped with a resource manager and a thermal manager, which are responsible for keeping track of available computing resources (e.g., processors and disks) and thermal profiles (e.g., CPU and disk temperatures). The resource and thermal information are transmitted to a request dispatcher, which manages system workload in the back-end tier by assigning requests to the back-end servers. It is worth mentioning that the resource managers may place a set of under utilized servers into the low-power mode to achieve energy savings.

We implement the front-end and back-end tiers in TERN using an off-the-shelf commodity cluster. It is noteworthy that the number of nodes residing in the front-end tier is much smaller than that in the back-end tier. We incorporate in TERN a load balancing module in the front-end and back-end tiers, respectively. The load balancer in the front-end tier evenly distributes requests submitted to the TERN system to all available front-end nodes; the load balancer in the back-end tier aims at balancing data processing workload among all the active servers.

All requests issued by clients in the user-end tier are handled by a small cluster of front-end nodes, which make use of meta-data information coupled with request information to assign the requests to the most appropriate servers in the back-end tier for further processing.

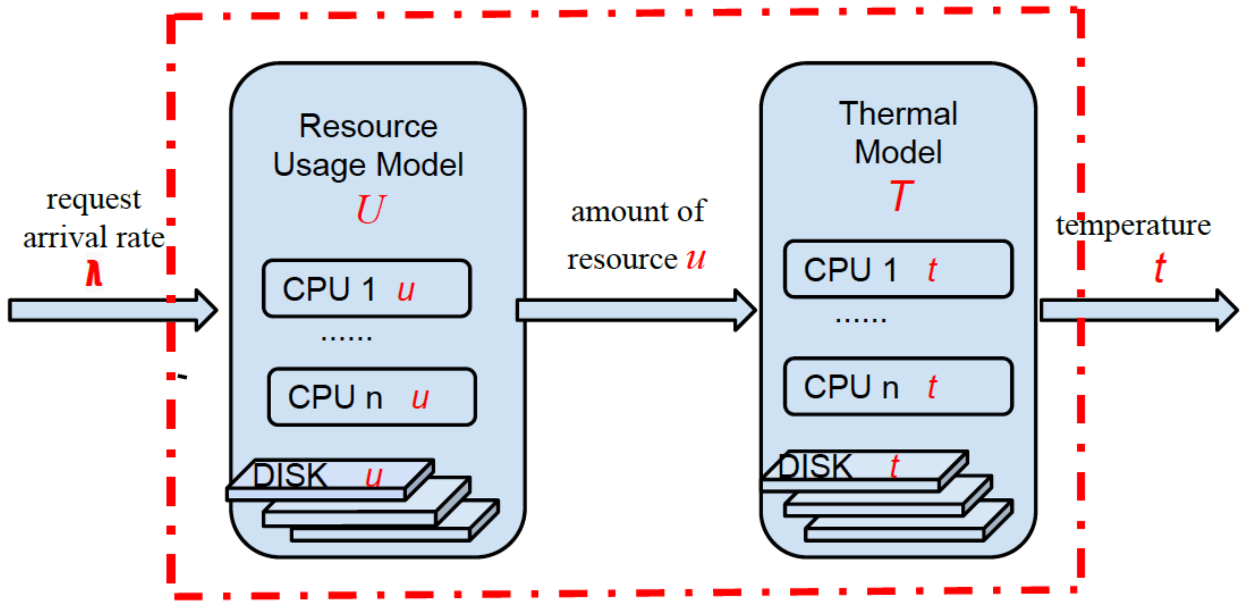


Figure 4.3: The Resource and Thermal Models Make Use of Access Patterns of A Web Service to Offer An Approximation of Resource Demands and Thermal Profiles of Various Transaction Types.

4.2.2 Resource and Thermal Models

Given access patterns of a web service (i.e., request metadata information), the resource and thermal models residing in the pre-end layer are applied to offer an approximation of resource demands and thermal profiles of various transaction types.

Fig. 4.3 illustrates the correlation between the resource model and the thermal model. More specifically, the resource model relies on access patterns to estimate CPU and disk resource demands for incoming web requests issued to the front-end and back-end servers. Access patterns of any transaction mix are characterized by two factors: request arrival rate λ and the transaction types. Intuitively, resource demands are proportional to arrival rates; a high arrival rate leads to heavily utilized CPU and disk resources. Each transaction type exhibits unique characteristics in terms of resource demands for web and database servers (see also Section 4.3.1). We build a resource usage profile for each transaction type. After individually profiling all the candidate transaction types, we construct a resource model by composing the resource profiles of modified transaction mixes.

With the knowledge of CPU and disk demands offered by the aforementioned resource model, the thermal model yields an approximation of CPU and disk temperatures. Combining the resource model and the thermal model, one can directly derive thermal trends from access patterns. We seamlessly integrate the resource model and the thermal model in a way that we can easily investigate the prediction accuracy of each model. Our preliminary findings show that prediction errors introduced by the resource model can be further magnified in the thermal model. In this study, we address this challenge by making TERN adjust model parameters to mitigate prediction errors under dynamically changing workload conditions.

The two models outlined in Fig. 4.3 have the following three strengths. First, the resource model characterizes CPU and disk usages from the arrival rate of each transaction type. Second, the thermal model can speculate the temperature trends of front-end and back-end servers. Third, the models can be integrated to offer thermal profiles of servers in a data center.

4.3 Modelling

4.3.1 Transaction Resource Profiling

We conduct experiments to characterize resource usages as a function of the arrival rate of a transaction type. Resource usages demonstrates resources demands of a transaction mix in front-end and back-end servers.

Table 4.3 reveals that all the types of transactions have CPU activities on both the front-end and back-end servers except the *Buy Confirm* transaction type, which exhibits no CPU activity on the back-end server. In addition, the *Buy Confirm* transaction issues no request to disks in the front-end and back-end servers. Among the evaluated transaction types, only the *Customer Reg.* and *Buy Request* transaction types impose disk-intensive

Request	**FE CPU**	**FE IO**	**BE CPU**	**BE IO**
Home	✓	✗	✓	✓
New Product	✓	✗	✓	✓
Best Seller	✓	✗	✓	✓
Product Detail	✓	✗	✓	✓
Search Request	✓	✗	✓	✓
Search Results	✓	✗	✓	✓
Shopping Cart	✓	✗	✓	✓
Customer Reg.	✓	✗	✓	✓
Buy Request	✓	✗	✓	✓
Buy Confirm	✓	✗	✗	✗

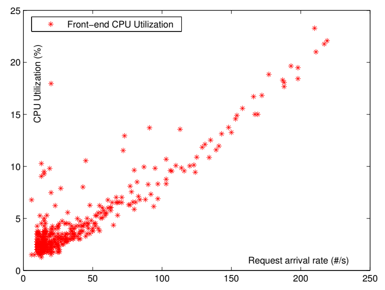
Table 4.3: Resource Demands of Transaction Types.

workload to the back-end servers. Regardless of arrival rates, no transaction type issues disk requests to the front-end servers.

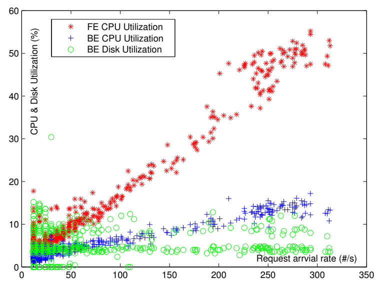
We observe from Figs. 4.4(a) and 4.4(b) that the front-end nodes are more sensitive to the arrival rate than the back-end nodes from the perspective of CPU utilization. For example, the *Buy Confirm* requests impose no CPU activity on the back-end nodes regardless of the arrival rate. When the arrival rate of *Home* requests increases from 10 to 320 No./Sec, the CPU utilization goes up from 5% to 55% on the front-end node, whereas the CPU utilization of the back-end counterparts only changes from 5% to 12%. Fig. 4.4(c) reveals that the CPU utilization of the front-end and back-end nodes all approximately increase by 8% when the arrival rate reaches 30 No./Sec; in contrast, the disk utilization climbs up to 95% in this case. We conclude that disks in the back-end nodes are a performance bottleneck that prevents CPUs from being busy on the front-end and back-end nodes even under heavy load (see, for example, Fig. 4.4(c) when the arrival rate is 33 No./Sec, Figs. 4.4(a) and fig. 4.4(b) when the arrival rate is 230 No./Sec and 320 No./Sec, receptively).

Interestingly, Fig. 4.4(b) shows that the disk utilization of the back-end nodes is initially as high as 15%, then it drops to 5% when the arrival rate increased to 50 No./Sec. After that point, the disk utilization gradually and slightly decreases even when arrival rate keeps going up to 320 No./Sec. This intriguing trend can be attributed to contents and operations included in *Home* requests. A *Home* request is issued to retrieve a homepage, which is the first webpage of the online bookstore in the TPC-W benchmark. The homepage contains information of promotional books that triggers database query operations. At an early stage, the disk utilization is high because of retrieving all promotional book information from disks; gradually, the book information loaded from the disks are cached in main memory, thereby making a substantial drop in disk utilization. In order to keep our model simple yet practical, we ignore the impact of this type of transactions on the back-end nodes' disks.

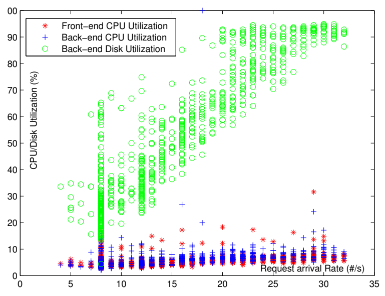
Each type of transactions shows various resource-demanding characteristics on front-end and back-end nodes; therefore, we individually construct a model for each transaction type. By investigating the relation of resource utilization and arrival rate of the three representative transactions (see Fig. 4.4), we observe that such a relationship follows linear or quadratic polynomial functions.



(a) The buy confirm request arrival rate and resource utilization.



(b) The home request arrival rate and resource utilization.



(c) The customer register request arrival rate and resource utilization.

Figure 4.4: Request Arrival Rate and Resource Utilization in the Front-end and Back-end Nodes

4.3.2 Resource Usage Profilings

Recall that the linear and quadratic polynomial functions are two candidates to model resource demanding characteristics (see also the previous section). Before incorporating an appropriate function into our models, let us briefly introduce a statistic concept *Residual sum of squares (RSS)* [55].

RSS - the sum of squares of residuals - is a measure of discrepancy between real data and an estimation model. A small RSS value indicates that a model can well fit real data. In (4.1), U_i^r represents measured CPU or disk utilization when arrival rate is i, U_i^m is CPU or disk utilization estimated at arrival rate i. RSS, which describes the sum of discrepancy square, is not a straightforward way of showing each modeled point's deviation from real measured data. To address this problem, we derive the square root of average of RSS ($sraRSS$) $\sqrt{RSS_{average}^{utilization}}$ (see (4.2)) to quantify the average discrepancy of our model to each measured data at a specific arrival rate.

$$RSS = \sum_{i=1}^{n}(U_i^m - U_i^r)^2 \tag{4.1}$$

$$sraRSS = \sqrt{RSS_{average}^{utilization}} = \sqrt{\frac{\sum_{i=1}^{n}(U_i^m - U_i^r)^2}{n}} \tag{4.2}$$

$$R_{sraRSS} = \frac{sraRSS}{(\sum_{i=1}^{n} U_i^r)/n} \tag{4.3}$$

We apply the curve fitting method to obtain the linear and quadratic polynomial function models of all the ten types of transactions. Then, we calculate the $sraRSS$ and R_{sraRSS} of these models in terms of front-end CPU utilization. The $sraRSS$ values listed in the second column of Table 4.4 show that both the linear and quadratic polynomial functions well fit front-end CPU utilization, the discrepancy of which oscillates only within 3%. The quadratic polynomial models are slightly better than the linear ones; the maximum difference between the two types of models is 0.203% (see New Product). For *Home* requests, the difference of the linear model and the quadratic polynomial model is as little as 0.074%, which accounts for about 0.264% to the average of its front-end CPU utilization (see Fig. 4.4(b)).

We observe from the third column of Table 4.4 that R_{sraRSS} values of the seven transaction requests are below 15% percent except for the *Search Result*, *Customer Reg.*, and *Buy* requests. This is because the CPU utilization of these transaction requests in the front-end nodes are relatively low. For example, Fig. 4.4(c) shows that the front-end node's CPU utilization is under 8%. Comparing the R_{sraRSS} values of the *Home* and *Buy Confirm* requests in Table 4.4 and front-end CPU utilization in Fig. 4.4, we discover that high CPU utilization leads to improved model accuracy.

Unfortunately, the above trends are not applicable for disks. Back-end CPU utilization in the case of *Customer Reg.* and *Buy Request* transactions climbs as high as 100%; the R_{sraRSS} values for both models are larger than 20% (see Table 4.6).

After comparing absolute and relative discrepancies of the linear and quadratic polynomial models for all the ten transaction requests, we decide to apply the linear polynomial functions as our resource usage models, because in this approach estimated CPU and disk utilizations are extremely close to the quadratic functions. In addition, the linear models are less expensive in computing time than the quadratic models. A modern data center normally has to process thousands of or even millions of requests per second; a computing resource usage model has to predict resource usage for a large number of requests in a timely manner. Low computation overhead of usage models play becomes indispensable for real-time and dynamic resource provisioning.

Our linear models are expressed as (4.4), where λ_{r_i} is arrival rate of transaction request r_i, $U^{r_i}_{cpu/io}$ is CPU or disk utilization of request r_i when arrival rate is λ_{r_i}. a_{r_i} and b_{r_i} are two coefficients of corresponding transaction request's linear polynomial model. We use the Matlab curve fitting tool to obtain these important coefficients listed in Table 4.5.

$$U^{r_i}_{cpu/io} = a_{r_i} \times \lambda_{r_i} + b_{r_i} \tag{4.4}$$

4.3.3 Thermal Profiling

In this section, we first investigate CPU and disk thermal behaviors in the two groups of experiments. Then, we uncover the relation of CPU/disk temperatures and inlet/outlet

	sraRSS		R_{sraRSS}	
	Linear	*Quadratic*	*Linear*	*Quadratic*
Home	2.287	2.213	0.084	0.081
New Product	3.203	3.0	0.121	0.113
Best Seller	3.080	2.958	0.126	0.121
Product Detail	2.121	1.969	0.147	0.136
Search Request	1.967	1.969	0.087	0.087
Search Result	1.896	1.890	0.203	0.202
Shopping Cart	2.372	2.322	0.106	0.103
Customer Register	1.977	1.979	0.363	0.363
Buy Request	1.945	1.944	0.263	0.262
Buy Confirm	1.481	1.406	0.135	0.128

Table 4.4: *sraRSS* and R_{sraRSS} of the CPU Models

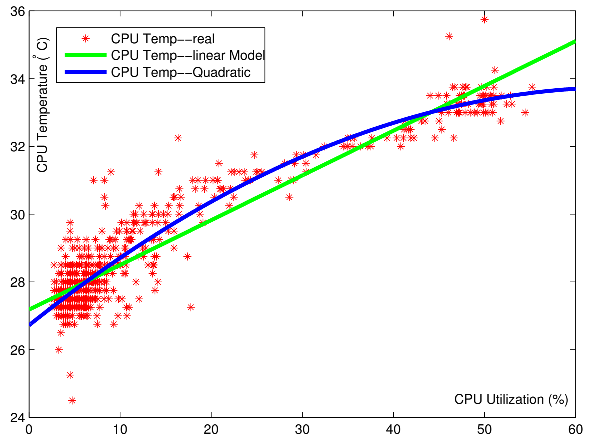
	FE CPU		**BE CPU**		**BE Disk**	
	a	b	a	b	a	b
Home	0.170	1,793	0.044	1.814	—	—
New Product	0.163	3.56	0.057	1.977	—	—
Best Seller	0.119	3.476	0.049	2.036	—	—
Product Detail	0.038	3.935	0.019	2.225	—	—
Search Request	0.096	3.469	0.049	2.212	—	—
Search Result	0.156	3.025	0.098	2.864	—	—
Shopping Cart	0.161	2.234	0.044	1.451	—	—
Customer Register	0.124	3.529	0.184	3.31	3.199	1.609
Buy Request	0.109	2.978	0.056	3.943	0.976	10.78
Buy Confirm	0.084	1.267	—	—	—	—

Table 4.5: Coefficient Values of the Linear Polynomial Models.

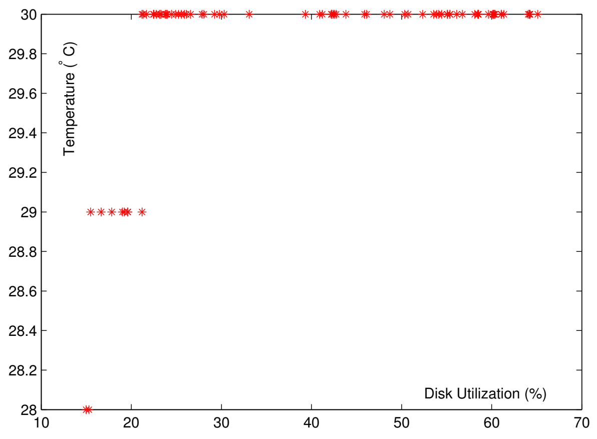
	$sraRSS$		R_{sraRSS}	
	$Linear$	$Quadratic$	$Linear$	$Quadratic$
Customer Register	10.006	9.895	0.208	0.206
Buy Request	12.884	12.728	0.286	0.283

Table 4.6: $sraRSS$ and R_{sraRSS} of the Disk Models

temperature gap of a node. In the first group of experiments, we run on the server nodes a multi-thread matrix process, which exhibits CPU-intensive workload. We manipulate CPU utilization of the nodes by tuning request idle time periods.



(a) CPU Thermal Profiling and Model.



(b) Disk Thermal Profiling.

Figure 4.5: CPU and Disk Thermal Profiling

Fig. 4.5(a) plots the average CPU temperature as a function of CPU utilization. The red stars represent real measured CPU temperatures, and the green and blue curves are the linear and quadratic models for CPU temperature estimations. The average CPU temperature is 27 ℃when CPU utilizations are 3%. This temperature equals to the CPU internal temperature. Fig. 4.5(a) shows that the CPU temperature goes up from 27 ℃ to 34 ℃ when the CPU utilization increases from 3% to 55%. The linear model underestimates the CPU temperature with CPU utilization ranging from 15% to 45%. In contrast, the quadratic model has better fitting performance than the linear model during this CPU utilization range.

The *sraRSS* of the liner and quadratic models are 0.66 and 0.61 ℃, respectively. The discrepancy difference of the two models is 0.05 ℃, meaning that the quadratic model is 7.5% more accurate than the linear model. Nevertheless, for all the ten types of transactions, the quadratic model is only on average 3% more accurate than the linear model. As a consequence, while making a tradeoff between accuracy and computing overhead, we choose the linear model for estimating transaction request resource usage and the quadratic model for predicting CPU temperatures.

The CPU thermal model is expressed as (4.5), where T_{r_i} is CPU temperature when CPU Utilization is $U_{cpu/io}^{r_i}$, and *a, b, c* are three coefficients.

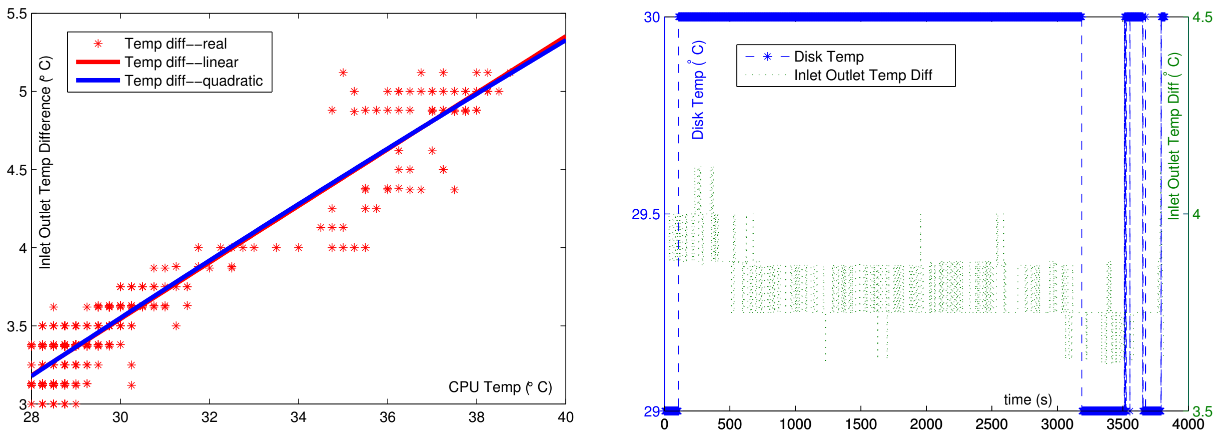
$$T_{r_i} = a \times {U_{cpu/io}^{r_i}}^2 + b \times U_{cpu/io}^{r_i} + c \tag{4.5}$$

where a=-0.001541, b=0.2076, c=27.86.

Now we conduct a thermal profiling test on disks in the computing nodes. In this group of experiments, we create disk-intensive workload using IOzone - a filesystem benchmark. By gradually stressing disks, we measure disk temperatures when disk utilization increases. Fig. 4.5(b) shows the impact of disk utilization on the average disk temperature in the cluster of nodes. We observe that the disks are less thermal sensitive than the CPUs in the nodes. Besides, the relationship between disk temperatures and utilization does not match neither the linear nor quadratic models. However, the piecewise function (see Equation (4.6)) can be used to depict the relation.

$$T_{r_i} = \begin{cases} 28\text{ ℃} & U_{io} \in (0, 15) \\ 29\text{ ℃} & U_{io} \in [15, 25] \\ 30\text{ ℃} & U_{io} \in (25, 100) \end{cases} \tag{4.6}$$

In the third set of experiments, we conduct stressing test on CPUs using CPU-intensive workload. The inlet and outlet temperatures are collected by deploying two sensors at the



(a) Inlet and Outlet Temperatures.   (b) Inlet/Outlet Temperature Difference.

Figure 4.6: CPU and Disk Inlet/Outlet Temperature Difference

node's inlet and outlet positions. Fig. 4.6(a) shows that inlet/outlet temperature difference behaves as a function of CPU temperature. The inlet/outlet temperature difference increases by 2.5 ℃ from 3 to 5.5 ℃ when the CPU temperature goes up from 28 to 39 ℃. Every 4 ℃ temperature increase in CPU contributes approximately 1 ℃ increase in outlet temperatures. However, when CPU's temperature climbs up from 33 to 37 ℃, the inlet/outlet temperature difference does not increase with an anticipated range from 4.25 to 5.25 ℃. The linear and quadratic models overestimate by 0.25 ℃ during this temperature range.

Fig. 4.7 shows the Inlet/Outlet temperature linear model (see equation (4.7))'s $sraRSS$ is around 0.15 ℃ and R_{sraRSS} is about 1.25%. However, the values of that are 0.33 ℃ and 8% when temperature increases from 37 to 39 ℃ which is 2 fold and 5 fold of that value when temperature increases from 28 to 39 ℃. The model estimates well from 28 to 33 ℃ with least $sraRSS$ about 0.12 ℃.

$$T_{r_i}^{inlet/outlet} = a \times T_{cpu} + b \tag{4.7}$$

where a=0.1809, b=-1.883

Fig. 4.6(b) plots the thermal profiling results of the fourth experiment revealing that the disk temperature changes have no significant impact on inlet/outlet temperature difference. For example, at time 100 Sec., when disk temperature jumps up 1 degree to 30 ℃, inlet/outlet temperature difference does not change at all. However, at time 600 Sec., inlet/outlet temperature difference does drop by 0.2 ℃, whereas disk temperature is still 30 ℃. Interestingly, 1 degree decreasing of disk temperature directly causes 0.2 ℃ drop in inlet/outlet temperature difference at time 3200 Sec. Comparing with Fig. 4.6(a), we observe that 1 degree temperature increase in CPU or disk changes inlet/outlet temperature difference by anywhere between 0.2 to 0.25 ℃. Because disks have a narrow temperature variant range, disks have little impact on inlet/outlet temperature difference.

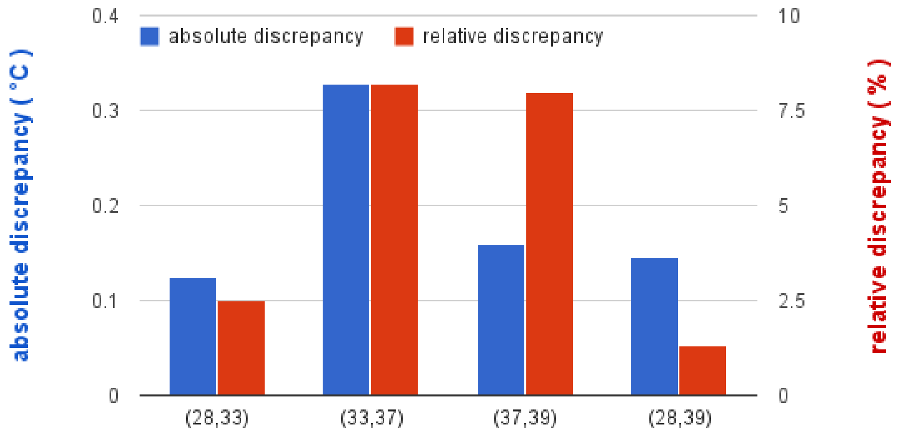


Figure 4.7: Inlet/Outlet Temperature Model $sraRSS$ and R_{sraRSS}

4.3.4 Integrated Models

In the previous sections, we have modeled relations between each transaction type's arrival rate and CPU and disk utilizations (see (4.4)) for CPU utilization, see (4.5)) for thermal modeling, see (4.7) for inlet/outlet temperature difference).

Integrating (4.6), (4.5), and (4.4), we build a thermal model in form of a function of the requests arrival rate as

$$T_{r_i} = f\left(\lambda_{r_i}\right) \tag{4.8}$$

When putting (4.4),(4.6), (4.5),and (4.7) together, we obtain an inlet/outlet temperature difference model as a function of arrival rate below.

$$T_{r_i}^{inlet/outlet} = f\left(\lambda_{r_i}\right) \tag{4.9}$$

In Section 4.1, we describe three types of workload mixes, each of which has a specific percentage combination of individual transaction requests. Therefore, CPU and disk utilization model of a workload mix can be derived as

$$U_{mix} = \sum_{r_i \in R} P_{r_i} \times U_{r_i} \tag{4.10}$$

where P_{r_i} is transaction request r_i's combination percentage in the workload mix and U_{r_i} is the transaction request's resource usage model.

Similarly, we construct the thermal model of a workload mix as (4.11) by incorporating the CPU and disk thermal models of multiple transaction types.

$$T_{mix} = \sum_{r_i \in R} P_{r_i} \times T_{r_i} \tag{4.11}$$

$$T_{r_i}^{adjusted} = T_{r_i} \pm \sigma^{AD} \tag{4.12}$$

4.4 A Self-Adjustment Strategy

4.4.1 Deriving Algorithm

The CPU temperature model and the inlet/outlet temperature difference model proposed in the previous section do not fit well to real data when CPU temperature is in the range between 33 ℃ to 37 ℃. In this section, we develop a self-adjustment strategy (see Algorithm 1) to automatically adjust the models according to the models' absolute and relative discrepancies.

In the self-adjustment strategy, we partition a model's CPU/disk temperature range into an array of small consecutive ranges. We set initial absolute and relative discrepancies to 0, and set the initial model to using the original model (see Lines 1-6). Next, in an iterative process, if previous CPU/disk temperature range has a R_{sraRSS} exceeding the specified threshold, then the model will self-adjust by increasing or decreasing previous range's $sraRSS$ depending on the model's overestimation or underestimation. If the model is accurate for the current range, there will be no need to adjust the model's parameters. Obviously, the model applied for the first range is the original model where $RD_0 = 0 < RD_{threshold}$ (see Lines 7-12). In the last step, the algorithm calculates the current range's absolute and relative discrepancies and determines whether or not the model should adjust itself for the next range. Then, the algorithm repeatedly processes the next range until all the ranges are processed. (see Lines 13-17). The model powered by the self-adjustment strategy is comprised of multiple models for all the individual ranges.

4.4.2 Evaluation of Self Adjustment

In this section, we apply Algorithm 1 to model (4.7), where N is set to 11 and the range of CPU temperature is set to 1. For instance, when CPU temperature is 28 ℃, we apply the original model to estimate thermal behavior as well as absolute and relative discrepancies.

```
Input: $\langle M, X_{min}, X_{max}, AD, RD^{threshold}, N\rangle$
$M$: original model
$X_{min}$: minimum X value of Model M
$X_{max}$: maximum X value of Model M
$AD$: Model M's overall $sraRSS$ $RD^{threshold}$: predefined $R_{sraRSS}$
$N$: the amount of divided ranges of x axis
Output: $M^{adjusted}$ the adjusted model
1  if $N = 1$ then
2      $M^{adjusted} = M + sign \times AD$
3  else
4      $X_{incre} \leftarrow (X_{max} – X_{min})/N$
5      $sign = 1$
6      $i = 0$
7      $AD_i = 0$
8      $RD_i = 0$
9      $M_i = M$
10     while $X_{min} < X_{max}$ do
11         if $RD_i > RD_{threshold}$ then
12             $M_{i+1} = M_i + sign \times AD_i$
13         else
14             $M_{i+1} = M_i$
15         end
16         $i \leftarrow i+1$
17         $RD_i = R_{sraRSS}{}_{X_{min}}^{X_{min}+X_{incre}}$
18         $sign = \sum_{X_{min}}^{X_{min}+X_{incre}} (R – M)$
19         $AD_i = sraRSS_{X_{min}}^{X_{min}+X_{incre}}$
20         $X_{min} = X_{min} + X_{incre}$
21     end
22 end
23 return $M^{adjusted} = \langle M_1, M_2, \ldots, M_N\rangle$
```

Algorithm 1: A Self-Adjustment Strategy

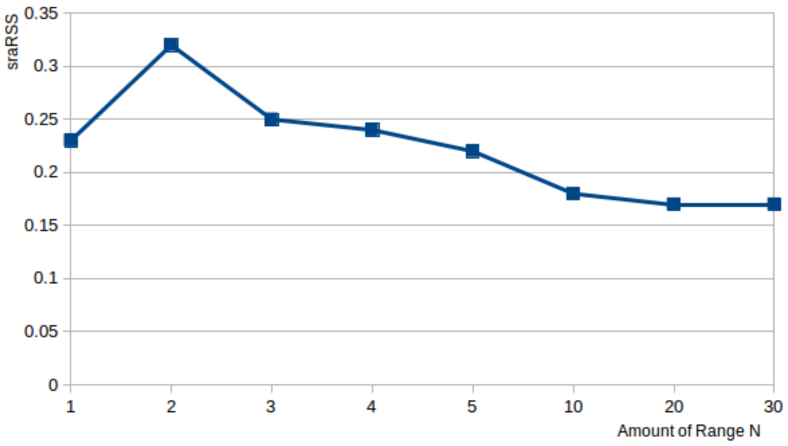


Figure 4.8: The impact of the number of ranges N on self-adjusting algorithm

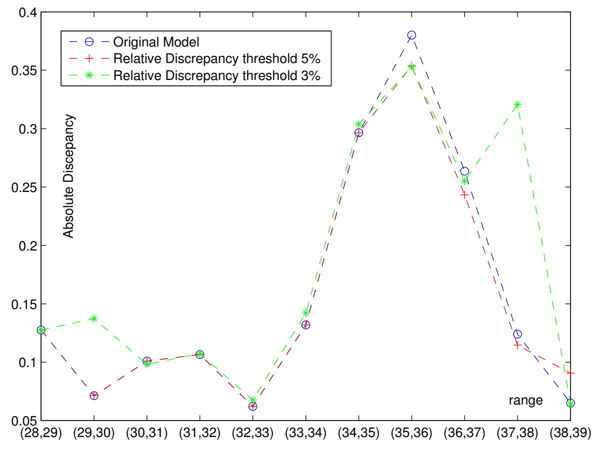
Then, these parameters can be adjusted for the next CPU temperature range from 29 to 30 ℃.

Fig. 4.9(a) shows the $sraRSS$ of the three models (i.e., the original mode, the model with self-adjustment where the predefined R_{sraRSS} threshold is set to 5% and 3%, respectively). Fig. 4.9(b) shows the corresponding relative discrepancies. Absolute discrepancies plotted in Fig. 4.9(a) are below 0.15 from 28 ℃ to 35 ℃ for all the three models. Then, the discrepancies jump from 0.125 to 0.3 when CPU temperature increases from range (33,34) to range (34,35). This trend is consistent with that shown in Fig. 4.6(a).

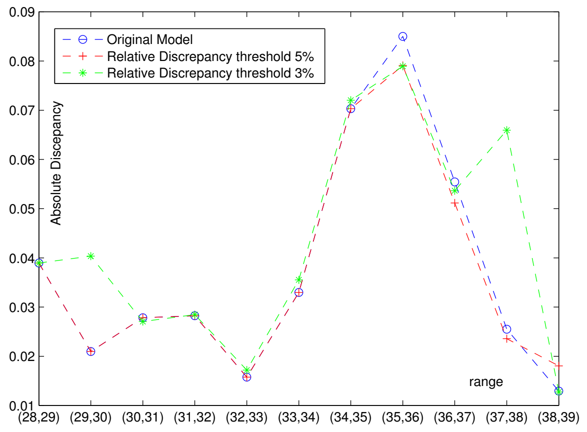
For the self-adjustment strategy where the threshold is set to 5%, the R_{sraRSS} increases from 0.032 to 0.07 when the range moves from (33,34) to (34,35). In this case, the model starts adjusting the parameters. As a result, the $sraRSS$ of the model in range (35,36) decreases from 0.38 to 0.35 (see Fig. 4.9(b)), and the R_{sraRSS} is reduced from 0.085 to 0.078, which is approximately 10% more accurate than the original mode. From range (36, 37) to (37, 38), the 5%-threshold model also reduces $sraRSS$. However, the 3%-threshold model begins to adjust itself at range (29,30) in which the $sraRSS$ noticeably increases. Additionally, its $sraRSS$ is larger than those of the original model and the 5%-threshold model in ranges (36,37) and (37,38).

Fig. 4.8 illustrates the impact of the number of ranges N in self-adjusting algorithm on the accuracy of models where the threshold is set to 5%. The $sraRSS$ is about 0.23 ℃ when set N as 1. The model becomes less accurate with $sraRSS$ climbs from 0.23 to 0.32 when N increase from 1 to 2. Then $sraRSS$ begins to decrease from 0.25 to 0.17 when enlarge the number of range N from 3 to 30.

The models with our self-adjustment strategy improve estimate accuracy of the traditional modeling approach. The new models' accuracy is affected by the predefined R_{sraRSS} threshold. Specifically, if this threshold is set too high, the model will be degraded to its original model. On the other hand, if the threshold is set too low, the self-adjustment strategy will be very sensitive to R_{sraRSS}. The self-adjustment strategy coupled with a low threshold is inadequate for models in which R_{sraRSS} is frequently and dramatically changing.



(a) $sraRSS$.



(b) R_{sraRSS}

Figure 4.9: $sraRSS$ and R_{sraRSS}

4.5 Model Validation

In this section, we validate the accuracy of the resource usage models as well as the thermal models in TERN. Section 4.5.1 evaluates the resource usage models using three representative transaction requests (i.e., *Home*, *New Product* and *Buy Request*). Then, we apply the resource usage models to predict resource utilization in real-world scenarios of three workload mixes. We discuss in Section 4.5.3 the validation of the thermal models for various transaction requests in addition to the three workload mixes.

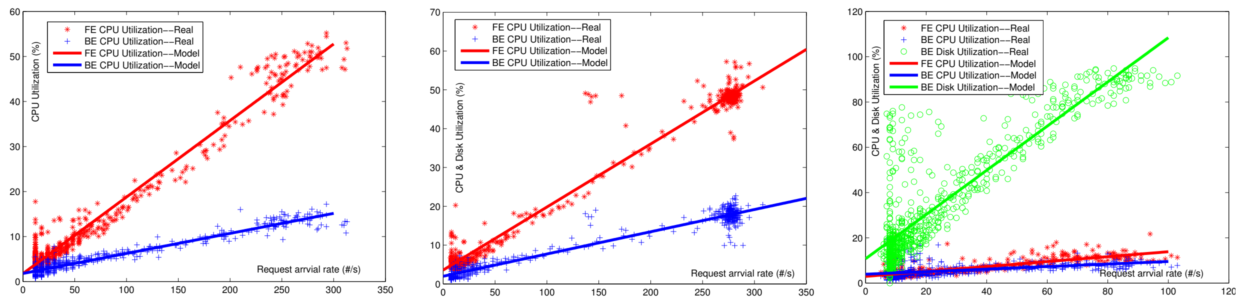
4.5.1 Resource Usage Model Validation

Fig. 4.10 shows that regardless of the three types of transactions, our resource usage models accurately estimate resource demands of front-end CPUs, back-end CPUs, and back-end disks. For the *Home* request, the model slightly overestimates front-end CPU usage by less than 3% when arrival rate increases from 10 No./Sec. to 130 No./Sec. And, the overestimation goes up to approximately 5% when arrival rate is in the range between 130

No./Sec. to 250 No./Sec. On the other hand, the model underestimates front-end CPU usage by anywhere between 5% to 10% when arrival rate increases from 250 No./Sec. to 300 No./Sec.

We observe from Fig. 4.10 that in the case of *Home* request, the back-end CPU model is superior to the front-end CPU model. A similar trend can be found for the resource usage models in the cases of *New Products* and *Buy Request* requests. In the *Buy Request* case, the disk resource usage model has a high *sraRSS* of 13. Nevertheless, Fig. 4.10(c) reveals that real measured data are evenly distributed among the estimated data except for the cases where arrival rate is below 30 No./Sec.

Fig. 4.11(a) illustrates the absolute discrepancies of the models in the three transaction types. For the *Home* transaction request, the *sraRSS* of the back-end nodes is more than 2 fold of that of the front-end node. A similar observation can be drawn for the *New Product* transaction request. Unlike the models for the *Home* and *New Product* requests, the *Buy Request* model's *sraRSS* of back-end nodes is larger than than that of front-end one; this pattern is not intuitively observed in Fig. 4.10(c).

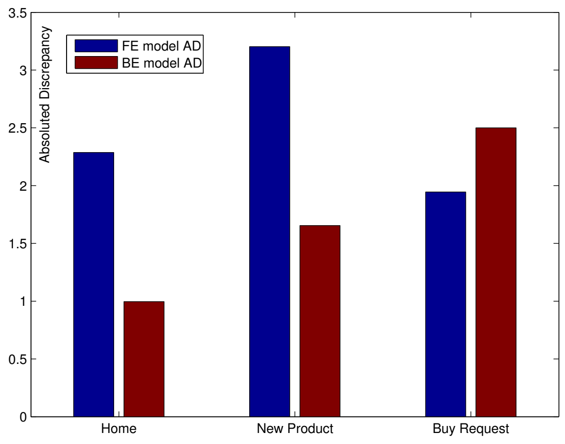


(a) *Home* (b) *New Product* (c) *Buy Request*

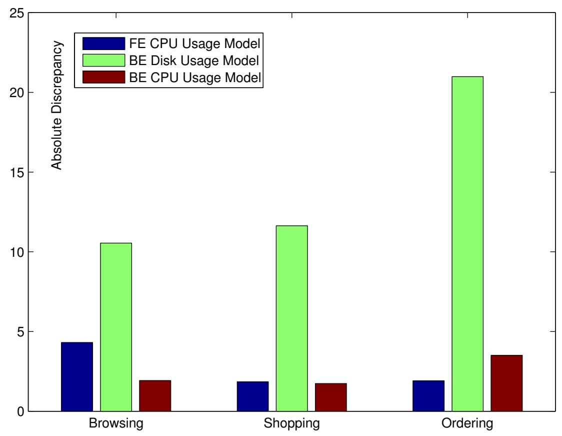
Figure 4.10: Accuracy of the resource usage models for front-end CPUs, back-end CPUs, and back-end disks in the cases of the Home, New Product, and Buy Request transactions.

4.5.2 Workload with Transaction Mixes

Using different transaction mixes, we investigate performance of the model that integrates multiple resource usage models. Fig. 4.12 illustrates that the proposed model (see

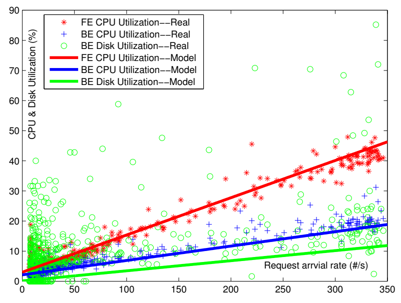


(a) *sraRSS* of the Resource Usage Models in the Three Types of Transactions.

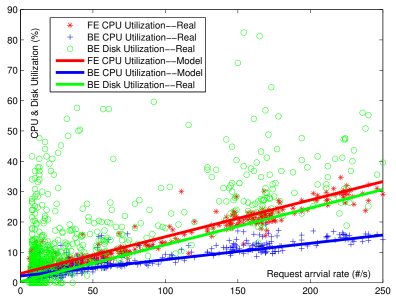


(b) Resource Usage Model *sraRSS*

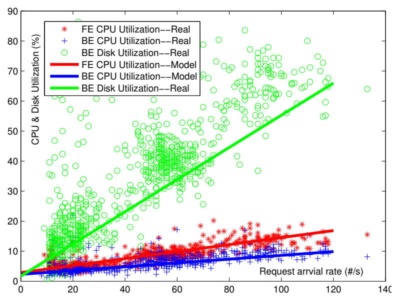
Figure 4.11: Resource Usage Model *sraRSS*



(a) *Browsing*

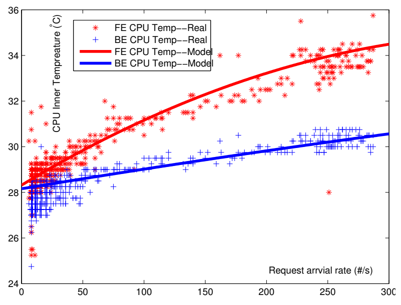


(b) *Shopping*

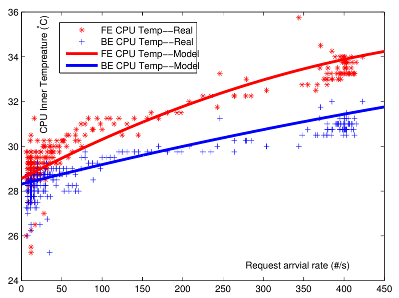


(c) *Ordering*

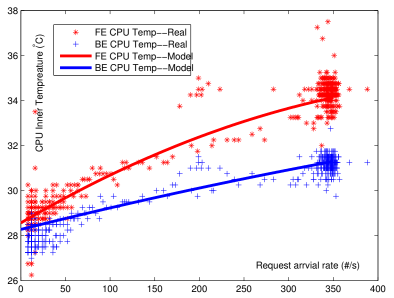
Figure 4.12: Workload mix's CPU & Disk Utilization



(a) *Shopping Cart*

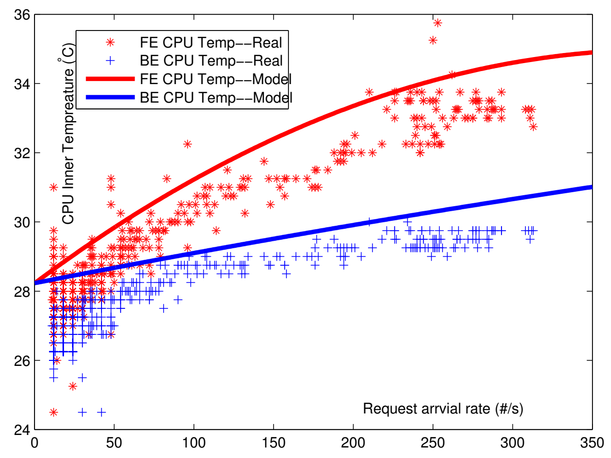


(b) *Search Request*

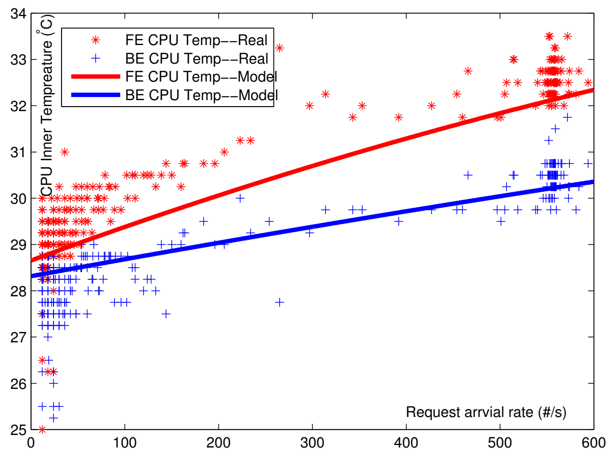


(c) *Best Seller*

Figure 4.13: Individual Transaction Request's CPU Inner Temperature

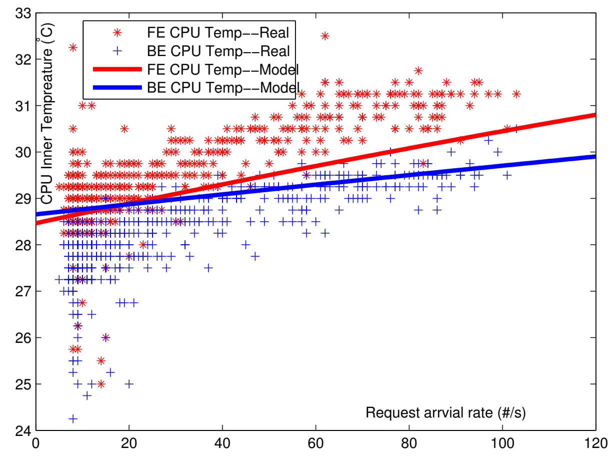


(a) *Home*

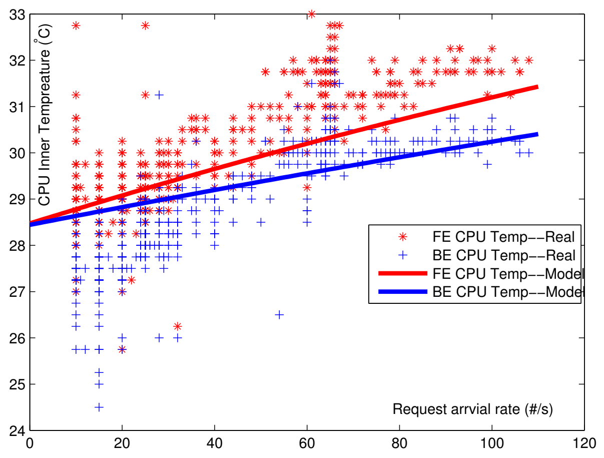


(b) *Product Detail*

Figure 4.14: Individual Transaction Request's CPU Inner Temperature



(a) *Buy Request*



(b) *Search Result*

Figure 4.15: Individual Transaction Request's CPU Inner Temperature

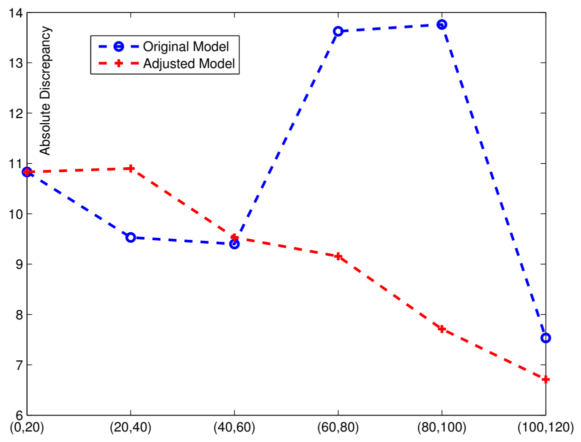
(4.10)) can effectively predict CPU utilization of the front-end and back-end nodes under diverse workloads with a changing transaction mix.

Fig. 4.11(b) shows the absolute discrepancies of our model when it is applied to provide resource provisioning for the *Browsing*, *Shopping*, and *Ordering* workload mixes. The CPU-usage discrepancies of the model for all the three workload mixes models are below 5%. The back-end-disk-usage discrepancies in the evaluated workload mixes are higher than all the CPU-usage discrepancies. For example, the back-end-disk-usage discrepancies are 11%, 12%, and 21% for the *Browsing*, *Shopping*, and *Ordering* workload mixes, respectively. Interestingly, when back-end disk load becomes heavy, the $sraRSS$ of disk-usage prediction increases; in contrast, the model's prediction accuracy is improved for front-end CPU usage. This accuracy trend can be attributed to the fact that increasing back-end disk load makes an I/O performance bottleneck preventing front-end nodes from handling an increased number of transaction requests. Our model's prediction accuracy is high (i.e., low $sraRSS$) when resource utilization is at a very low level.

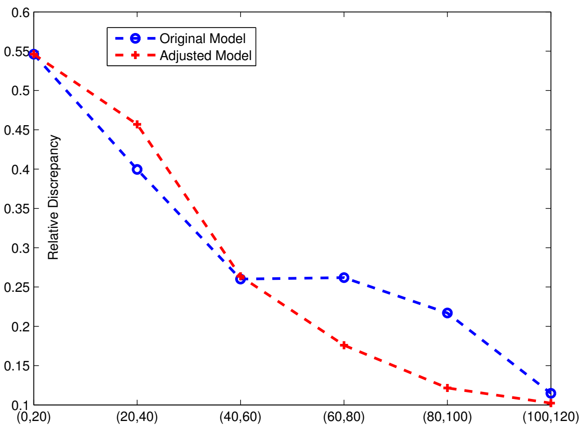
To address the model's accuracy problem with respect to disk usage prediction for the three workload mixes (e.g., the *Ordering* mixes), we apply the self-adjustment strategy (see Section 4.4) to improve the model's prediction performance for the *Ordering* workload. Figs 4.16(a) and 4.16(b) show the performance of the original model and the model powered by the self-adjustment strategy. In this group of experiments, we set N to 6 and pre-defined R_{sraRSS} to 10% in the self-adjustment strategy.

Fig. 4.12(c) shows that CPU utilization ranges from 5% to 25% when arrival rate is in a range between 0 to 20 No./Sec. In this scenario, the absolute and relative discrepancies are 11% (see Fig. 4.16(a)) and 55% (see Fig. 4.16(b)), respectively. When arrival rate reaches the level of 60%, the $sraRSS$ drops to 9.5%, which represents an accuracy improvement of 13.6%. We observe from Fig. 4.16(b) a similar pattern with respect to the R_{sraRSS}.

More importantly, the self-adjustment strategy substantially improves the prediction performance of the original model under heavy workload conditions. For example, the original model's discrepancy climbs up to 13.7% when arrival rate increases to the range between 60 and 80 No./Sec. The self-adjustment strategy significantly improves the accuracy of our original model by 34.3%, making the $sraRSS$ drop down to 9%. When the load reaches a very high level (e.g., arrival rate is in the range between 80 and 100 No./Sec), the self-adjustment strategy is capable of improving the model's accuracy by more than 50%. Fig. 4.16(a) reveals that the accuracy improvement offered by the self-adjustment strategy becomes more pronounced when arrival rate and disk utilization are high.



(a) Prediction Performance Measured in Terms of $sraRSS$. The self-adjustment Strategy Improves the Model's Performance under heavy workload conditions.



(b) Prediction performance measured in terms of R_{sraRSS}. The self-adjustment strategy improves the model's performance under heavy workload conditions.

Figure 4.16: Prediction performance measured in terms of $sraRSS$ and R_{sraRSS}. The self-adjustment strategy improves the model's performance under heavy workload conditions.

4.5.3 Performance of the Thermal Models

Recall that the thermal models proposed in Section 4.3.4 (see also (4.8)) make use of arrival rate to predict CPU temperatures. In this subsection, we focus on evaluating the prediction performance of the thermal models. Figs. 4.13 and 4.15 show comparisons between predicted CPU temperatures and the measured ones when the arrival rate is gradually increased.

The experimental results suggest that the thermal models can predict CPU temperatures of the nodes processing the three types of transactions requests under a wide range of arrival rates. Fig. 4.15 shows that for the four transaction requests, the models either overestimate or underestimate CPU temperatures of the front-end and back-end nodes. For example, in the *Home* request case, the models overestimate the temperatures by 1.23 ℃ and 1.46℃ for the front-end and back-end nodes, respectively. On the other hand, the models underestimate the CPU temperatures of the front-end and back-end nodes by 0.71 and 0.70 ℃ for the *Product Detail* workload, 0.73 and 1.03 ℃ for the *Buy Request* workload, and 0.65 and 1.3 ℃ for the *Search Request* workload.

We adjust the models (see (4.12) in Section 4.3.4) to optimize the prediction accuracy of the thermal models, where parameter N is set to 1 and σ^{AD} equals to a corresponding $sraRSS$ (i.e., an underestimation or overestimation value). Fig. 4.18 illustrates that the thermal models with adjustment offer improved prediction accuracy.

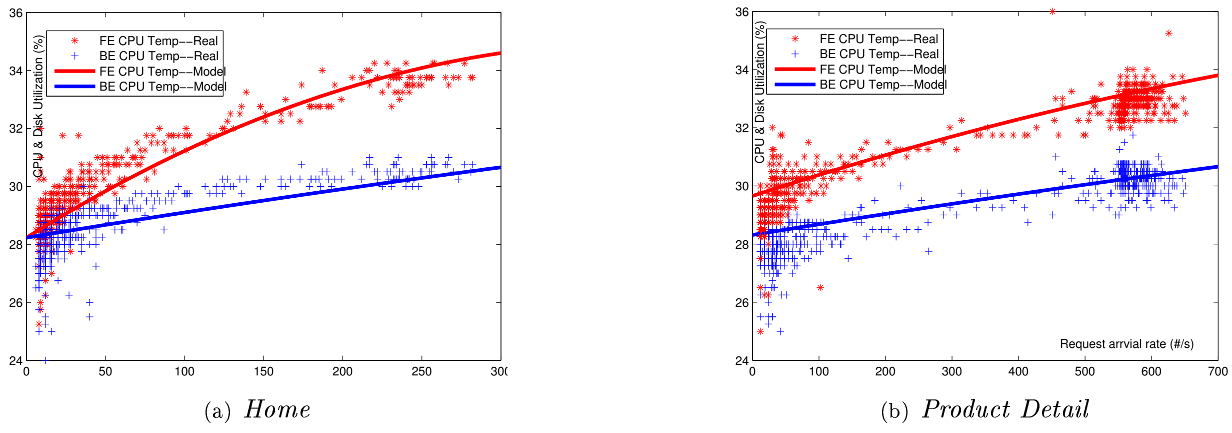
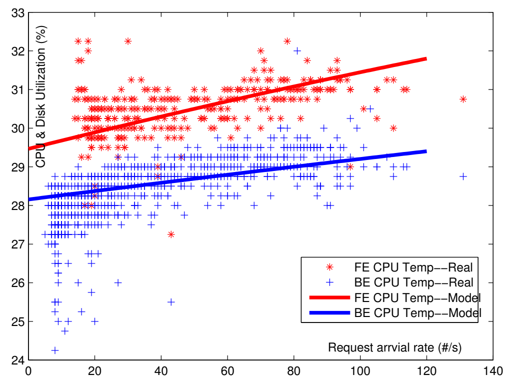
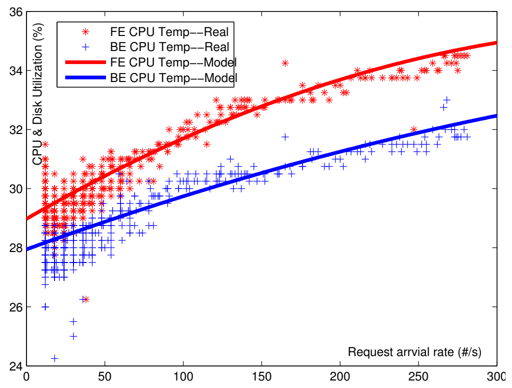


Figure 4.17: Performance of the thermal models for the individual transaction request.



(a) *Buy Request*

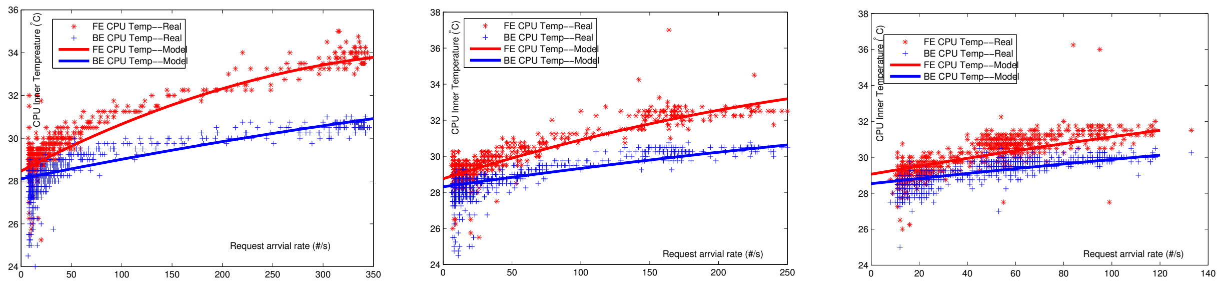


(b) *Search Result*

Figure 4.18: Performance of the thermal models for the individual transaction request.

4.5.4 Thermal Models for Workload Mixes

Now, we are in a position to evaluate the performance of the thermal models (see (4.11)) in the context of workload mixes. Fig. 4.19 shows the prediction performance of the models under the three types of workload mixes running on the front-end and back-end nodes. The thermal model built for the *Browsing* workload (see Fig. 4.19(a)) slightly underestimates CPU temperatures of the front-end node when arrival rate increases from 10 No./Sec to 170 No./Sec. Then, the model offers good prediction performance when arrival rate climbs up to 350 No./Sec. A similar performance trend is observed for the models that predict the thermal behaviors of back-end nodes. The experimental results also show that the models built for the *Shopping* and *Ordering* workload mixes exhibit better performance than the model for the *Browsing* workload mix.



(a) *Browsing* (b) *Shopping* (c) *Ordering*

Figure 4.19: Performance of the thermal models for workload mixes.

Chapter 5

aHDFS: an Erasure-Coded Data Archival System for Hadoop Cluster

In this chapter, we propose an erasure-coded data archival system called *aHDFS* for Hadoop clusters, where $RS(k{+}r, k)$ erasure codes are employed to archive rarely accessed replicas in the Hadoop distributed file system or HDFS. We explore two archival strategies (i.e., *parallel* data archiving and *pipelined* data archiving) in aHDFS to speed up the data archival process in HDFS. Instead of directly shuffling intermediate output from mappers to reducers in default *MapReduce* programming model (a.k.a, *DefaultMR*), *parallel* archiving scheme will, (1) store each mapper's intermediate output *Key-Value* pairs in a data server; (2) merge all intermediate *Key-Value* pairs with the same key into one single *Key-Value* pair on data server; (3) shuffle the merged single *Key-Value* pair to reducers for future computing to generate final parity blocks. Unlike *parallel* archiving in step (3), *pipeline* archiving scheme delivers the merged single *Key-Value* pair to its subsequent node's data server instead of shuffling to reducers. All the nodes repeat the same process until last node which is responsible for write the final *Key-Value* (a.k.a, parity blocks) to HDFS. aHDFS leverages the triplication redundancy to boost archiving performance by handling encoding operations in parallel as well as reducing network traffic during shuffling phase . We implement aHDFS in a real-world Hadoop cluster. Experimental results show that our aHDFS outperforms default MapReduce programming model (a.k.a, *DefaultMR*)in terms of archiving time by upto 4X and network traffic by upto 10X in a 10-node storage cluster.

The remainder of this chapter is organized as follows. Section 5.1 introduces the background of this study. The basic idea of our data archival system using the *MapReduce* programming model is discussed in section 5.2. Section 5.3 presents design and implementation of aHDFS. Then, in section 5.4, we analyse preliminary results from default archival strategy. Before conclusion, we evaluate aHDFS performance in section 5.5.

5.1 Background

We start this section by introducing Reed-Solomon code or RS, a popular erasure-coding scheme. Then, we discuss scenarios of adopting RS in the Hadoop distributed file system (HDFS). Next, we give a glimpse of RS reconstruction in case of encountering data failures. Finally, we shed some light on the internal data flows of the *MapReduce* framework.

5.1.1 Erasure-Coded Data Storage

When data becomes corrupted in storage systems, erasure-coded data can be reconstructed by using data stored in a disk array or distributed storage systems. Erasure coding has two salient features, namely, improving fault-tolerance performance and minimizing the storage-capacity overhead in date centers. For instance, Huang *et al.* adopted an erasure-coded or RS codes to build an erasure-coded storage cluster [33] to improve both data storage reliability and efficiency.

$RS(k{+}r, k)$ encodes source data with *RS-generated-coding matrix* (see the $k{+}r$ by k matrix on the left-hand side of the equation in Fig. 5.1). It is worth mentioning that the RS-generated-coding matrix contains the following two matrices:

- **Identical Matrix.** This is a $k \times k$ matrix, where all the diagonal elements are set to 1 and the other elements are set to 0 (see, for example, the k by k sub-matrix on the top portion of the left-hand side of the equation in Fig. 5.1).

- **Redundancy Matrix.** This is a $k \times r$ matrix used to generate parity blocks (see, for example, the r by k matrix on the bottom part on the left-hand side of the equation in Fig. 5.1).

With a simple linear algebra calculation, the parity data blocks (see the bottom part on the right-hand side of the equation in Fig. 5.1) are derived from the $k \times r$ *redundancy matrix*. In a data archival scenario, we simply apply the redundancy matrix to obtain calculate r parity blocks, which can be employed to reconstruct blocks for data failures (see also Section 5.1.2 for detailed discussions). A $RS(k+r, r)$ code scheme can sustain up to r numbers of concurrent blocks failures.

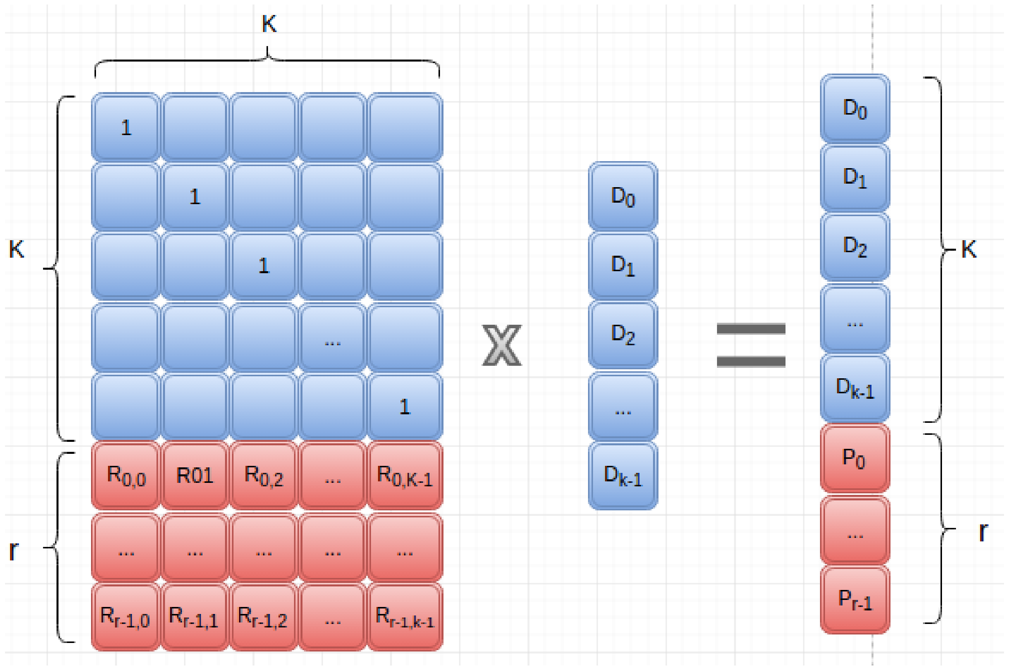


Figure 5.1: With a simple linear algebra calculation, the parity data blocks (see the bottom part on the right-hand side of the equation are derived from the $k \times r$ redundancy matrix (see the bottom part on the left-hand side of the equation)

5.1.2 Data Reconstruction with Reed-Solomon Coding

Now we elaborate the idea of data reconstruction using the Reed-Solomon coding. $RS(k+r, k)$ code can facilitate data recovery from a failed storage systems with up to r concurrent corrupted data blocks. Fig. 5.2 outlines a data reconstruction process, where RS coding is deployed to reconstructs source data blocks from surviving blocks and reconstruction matrix, which are defined as follows. We also introduce the definition of an inverse matrix used in the data reconstruction process.

- **Surviving Blocks.** This is archived data blocks which are not corrupted (see, for example,$k \times 1$ matrix on right-most-hand side of the equation in Fig. 5.2).

- **Reconstruction Matrix T.** This is a $k \times k$ matrix which are derived from *RS-generated-coding matrix* by getting rid of corresponding rows of corrupted blocks in *Surviving Blocks* (see, for example, the k by k matrix at the right middle of the left-hand side equation in Fig. 5.2).

- **Inverse Matrix T^{-1}.** This $k \times k$ matrix is an inverse matrix of *Reconstruction Matrix* T (see for example, the left-most matrix on left-hand side of equation and right-hand side of equation in Fig. 5.2).

Fig. 5.2 shows that the corrupted source data (see $D_0, ..., D_{k-1}$ on the left-hand side of the equation in Fig. 5.2) can be reconstructed by multiplying inverse matrix T^{-1} with surviving blocks (see the two items on the right-hand side of the equation in Fig. 5.2).

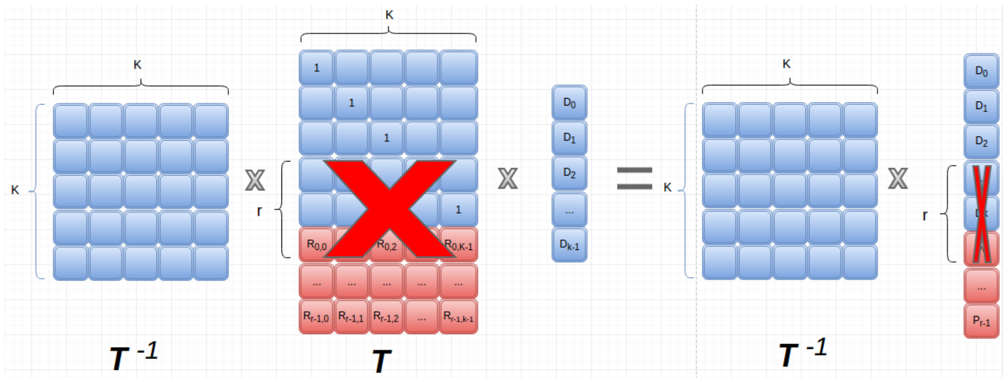


Figure 5.2: Data reconstruction using the Reed-Solomon coding scheme $RS(k{+}r, k)$. The failed source data (see $D_0, ..., D_{k-1}$ on the left-hand side of the equation) can be reconstructed by multiplying inverse matrix T^{-1} with surviving blocks (see the two items on the right-hand side of the equation.

5.1.3 MapReduce Programming Model

Now we introduce the workflow of a *MapReduce* program which consists of mappers, shufflers, and reducers. Fig. 5.3 delineates the data flows of the map phase (see the module on the left-hand side of shuffle) and the reduce phase (see the module on the right-hand side of shuffle), which are connected by the shuffle phase.

The *MapReduce* program starts with blocks fed into predefined *RecordReader* residing in the mapper to emit *Key-Value* pairs (see Step 1 in Fig. 5.3). In Step 2, given predefined *Key-Value* pairs, mappers generate their own *Key-Value* pairs (a.k.a., *record* output). Before written to an output buffer in main memory, *Key-Value* pairs are headed with a partition identification used for the shuffling purpose. In the case of fix-sized blocks, a large number of *Key-Value* pairs created by the mappers trigger a huge amount of data written into the main memory and disks, as well as shuffled to reducers. After the data buffered into the output buffer reach a specific percentage (e.e., 80%) of the configured size (e.g., the default size is 100MB), these data will be spilled into a local disk (see Step 7 in Fig. 5.3).

Unlike large blocks that tend to be kept in the disk, small blocks that are placed in the output buffer are sorted in accordance with their keys. The sorting time complexity is $O(n \log_2^n)$, which is proportional to the number n of *Key-Value* pairs. In Step 6, these sorted records of the small blocks are buffered into the local disk. Files that storing *Key-Value* pairs are merge to a final file to be shuffled to reducers (see Step 8 in Fig. 5.3).

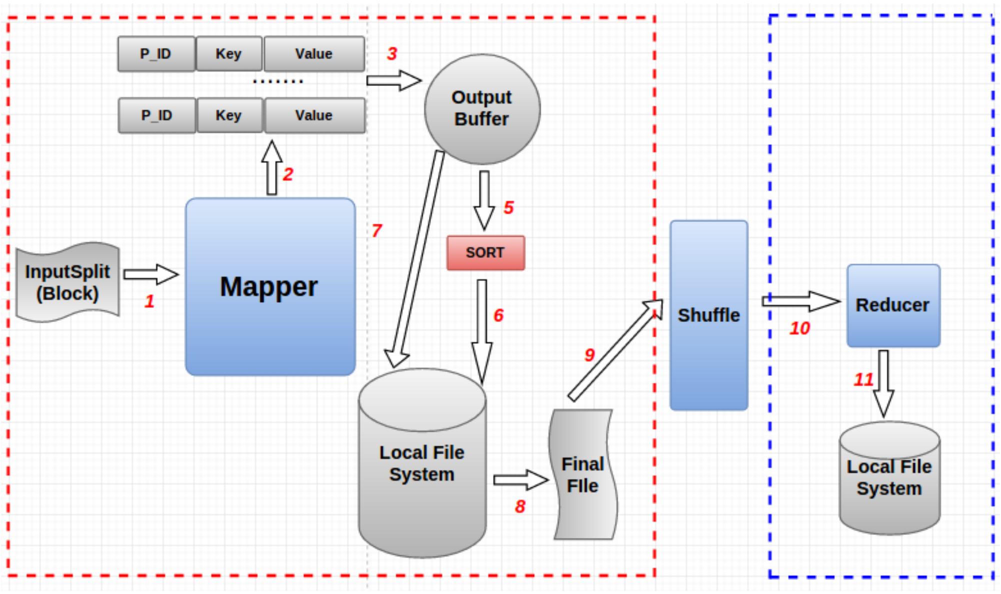


Figure 5.3: The MapReduce programming model. The data flows of the map phase and the reduce phase, which are connected by the shuffle phase.

During the shuffling stage, the size of pre-generated files in the map phase dominates both network traffic and shuffling time. In order to boost the shuffling performance, one may minimize data overhead (i.e., P_ID and Key) and compress shuffled data in the map phase. Increasing the number of reducers also helps in lowering network traffic occurred in the shuffling phase.

As each reducer is responsible for processing multiple groups of *Key-Value* pairs, the reducer merges these data in the main memory; the processed data are written to the local disk after the reducer completes the process. If large data exceed memory capacity, the data will be written to the disk and be merged on the disk (see Step 11 in Fig. 5.3).

The performance of a *MapReduce* application largely depends on

- data overhead (i.e., P_ID and Key) incurred by mappers yielding new *Key-Value* pairs,
- I/O operations in the map and reduce phases when data size exceed main memory capacity, and
- network traffic during the shuffling phase.

5.2 The Basic Idea of aHDFS

Before presenting the design and implementation of *aHDFS* (see Section 5.3), we shed some light on the basic idea of our data archival system using the MapReduce programming model. We start this section with the discussion on the application of Reed-Solomon coding to Hadoop distribute file system or HDFS (see Section 5.2.1). Then, we show the key-value design in the context of erasure-coded data archival systems (see Section 5.2.2).

5.2.1 Reed-Solomon (RS) coding in HDFS

Although replicated data mechanisms have been widely adopted in cluster storage systems to achieve data with fault tolerance and high I/O parallelisms, redundant data (e.g., two duplicated data or three) inevitably impose huge capacity overhead to storage systems. Reed-Solomon (RS) coding has gains its popularity in modern distributed file systems like Google's ColossusFS [2] and Facebook's HDFS [30]; the reason is three-fold. First, a vast majority of newly generated data (e.g., more than 90%) are less likely to be accessed after 24 hours upon their creation. Second, RS coding exhibits fault tolerance characteristics. Third, RS coding improves efficiency of data storage capacity. Thanks to the strengths of RS coding, in this study we focusing the development of our *aHDFS*, which applies RS to achieve data in Hadoop distributed file system or HDFS.

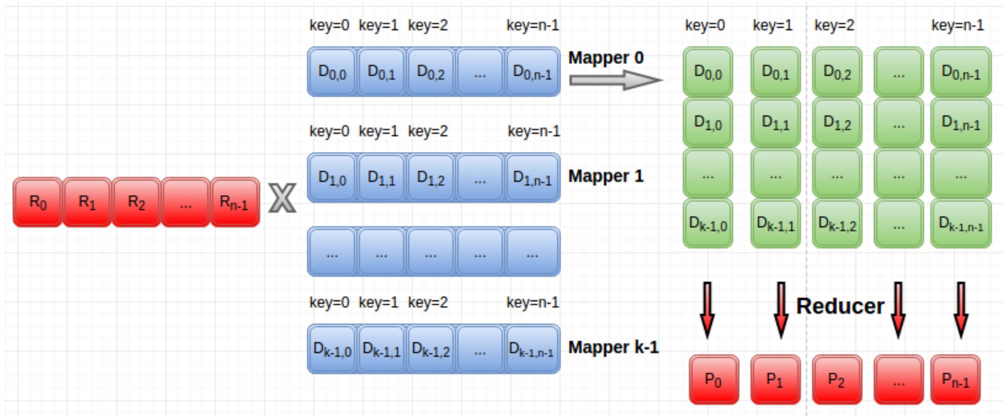

key=0 key=1 key=2 key=n-1
$D_{0,0}$ $D_{0,1}$ $D_{0,2}$ ... $D_{0,n-1}$
Mapper 0
$D_{1,0}$ $D_{1,1}$ $D_{1,2}$ ... $D_{1,n-1}$
Mapper 1
R_0 R_1 R_2 ... R_{n-1}
X
$D_{k-1,0}$ $D_{k-1,1}$ $D_{k-1,2}$ ... $D_{k-1,n-1}$
Mapper k-1
Reducer
P_0 P_1 P_2 ... P_{n-1}

Figure 5.4: The basic idea of $aHDFS$: the key-value design in the context of erasure-coded data archival systems.

5.2.2 The Basic Idea of Applying MapReduce

Fig. 5.4 illustrates the key idea behind *aHDFS* leveraging the *MapReduce* programming model to generate parity data blocks with the *Key-Value* pair design. We refer to the vector $((R_0, ..., R_n))$ on the left-hand side in Fig. 5.4 as a *redundancy vector*, which is one row from the *redundancy matrix* in Fig. 5.1. For simplicity, we only show one *redundancy vector* in Fig. 5.4; the redundancy vector can be extended to a *Redundancy Matrix* in our *aHDFS* design. Data to be archived are considered as source data of mappers (see the k by n source data matrix on the right-hand side of Fig. 5.4). In the example presented in Fig. 5.4, there are a total of k source data blocks to be archived; each data block is represented as a row in the source data matrix. The top portion on the right-hand side of Fig. 5.4 is the intermediate output data from the mappers; the bottom part (i.e., $(P_0, ...P_{n-1})$ on the right-hand side of Fig. 5.4 is the parity data blocks produced by reducers.

In the map phase, the number of spawned mappers depends on the number of *data blocks*. For example, k mappers are originated to be in charge of archiving k data blocks. In our system, we equally divide each block to n sub-blocks represented as $D_{i,0}$ $D_{i,1}$, ... , $D_{i,n-1}$. We assign one key to each sub-block; thus, the keys sequentially assigned to the n sub-blocks are $0, 1, ...$, and n–1. The value of each key is the data of each sub-block. One may tune sub-block size in order to optimize mapper performance (see also Section 5.4.4). By multiplying the sub-blocks with the *redundancy matrix*, the intermediate sub-blocks outputted from the multiple mappers are grouped by keys and shuffled to the reducers for further processing.

In the reduce phase, sub-blocks with the same key are grouped and calculated to derive one parity sub-block (e.g., $P_i, 0 \leq i \leq n$–1). Thus, all the intermediate sub-blocks with keys *0*, *1*, ..., and n are reduced to parity sub-blocks P_0, P_1, ..., P_{n-1}, respectively.

5.3 The Design of aHDFS

In this section, we delineate the design details of *aHDFS*. We address the design issues by using the $RS(k{+}r, k)$ coding scheme (see, for example, $RS(6{+}2, 6)$) to facilitate the data archiving process on a Hadoop cluster. We propose two Hadoop-based data archival strategies, namely, the *aHDFS-Grouping* and *aHDFS-Pipeline* schemes. Both the two strategies are developed using the MapReduce programming model. Please refer to Section 5.1.3 for the introduction of MapReduce; the basic design idea governed by the MapReduce programming model can be found in Section 5.2.2. We decide to use the HDFS default settings. Thus, in *aHDFS*, the number of the replicas of each block is three. Among the three replicas, two replicas are residing in two different nodes located in the same rack; the third replica is kept in another rack.

5.3.1 aHDFS-Grouping: Parallel Data Archiving

We start this section by illustrating how the *parallel data archiving* scheme works from the perspective of the *MapReduce* programming model. Then, we incorporate the key idea of the *Key-Value* pair design into parallel data archival.

Overview of Parallel Archiving

Let us make use of a concrete example (see Fig. 5.5) to demonstrate the idea of applying the MapReduce programming model to design our parallel data archival system. For simplicity, we consider a small file containing six (6) blocks denoted as $D_{i,j}$, where $i \in \{1, 6\}$ and $j \in \{1, 3\}$. Here $D_{i,j}$ represents the jth replica of the ith block in the file; for example, $D_{4,2}$ signifies the second replica of the 4th block. Recall that two out of the three replicas of each block are residing in the same rack, whereas the third one is stored in another rack. For instance, block 1's two replicas (i.e., $D_{1,1}$ and $D_{1,2}$) are residing in rack 1; block 1's third replica $D_{1,3}$ is hosted in a second rack (i.e., rack 2). Fig. 5.5 reveals two representative run-time cases of the parallel data archiving in *aHDFS*. Let us elaborate the two cases below.

- **Case 1.** There are three map tasks spawned on nodes 1 and 3, respectively. More specifically, the three mappers are concurrently processing $D_{1,1}$, $D_{2,3}$, and $D_{6,2}$ on node 1; whereas the other three mappers are handling $D_{2,3}$, $D4,1$, and $D_{5,2}$), respectively. The intermediate results from the six mapper tasks running on nodes 1 and 3 are grouped in form of a single block referred to as an intermediate parity block (see, for example, $P1$ and $P2$ in node 1 and node 3, respectively).

- **Case 2.** There are two mappers spawned on each of nodes 4, n–1, and n. Thus, node 4 runs two mappers to process blocks $D_{2,2}$ and $D_{1,3}$; node n–1 manages two mappers to deal with blocks $D_{4,3}$ and $D_{3,1}$; node n governs two mappers to treat blocks $D5,1$ and $D_{6,1}$. The intermediate output from each group of the two map tasks on the same node is merged as one block also referred to as an intermediate parity block (see, for example, $P1$, $P2$, and $P3$ on node 4, n–1, and n, respectively). The intermediate parity blocks are delivered to reducers to generate final parity blocks P and P' in node P and P', respectively.

In a real-world scenario, mapper tasks may be distributed among the nodes in an unbalanced fashion. Nevertheless, the key idea illustrated in the aforementioned two cases remains unchanged. In summary, the idea behind our the parallel archiving scheme is to group intermediate results yielded from map tasks to form one intermediate parity block to be delivered to a reducer for further computing.

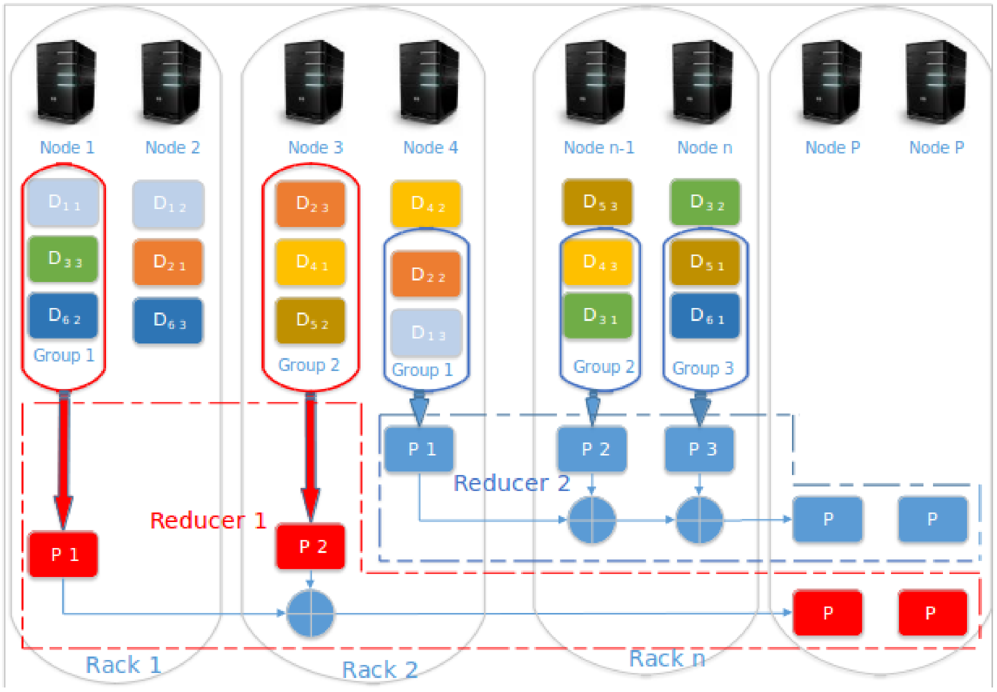


Figure 5.5: Applying the MapReduce programming model to design our parallel data archiving scheme for the $RS(6{+}2, 6)$-coded storage.

MapReduce-based Parallel Archiving

Now we design a parallel archiving scheme using the MapReduce programming model. Fig. 5.6 illustrates our MapReduce-based parallel archiving scheme, detailing the activities of several mappers and one reducer in node 1. The behaviors of the other nodes are identical as that of node 1.

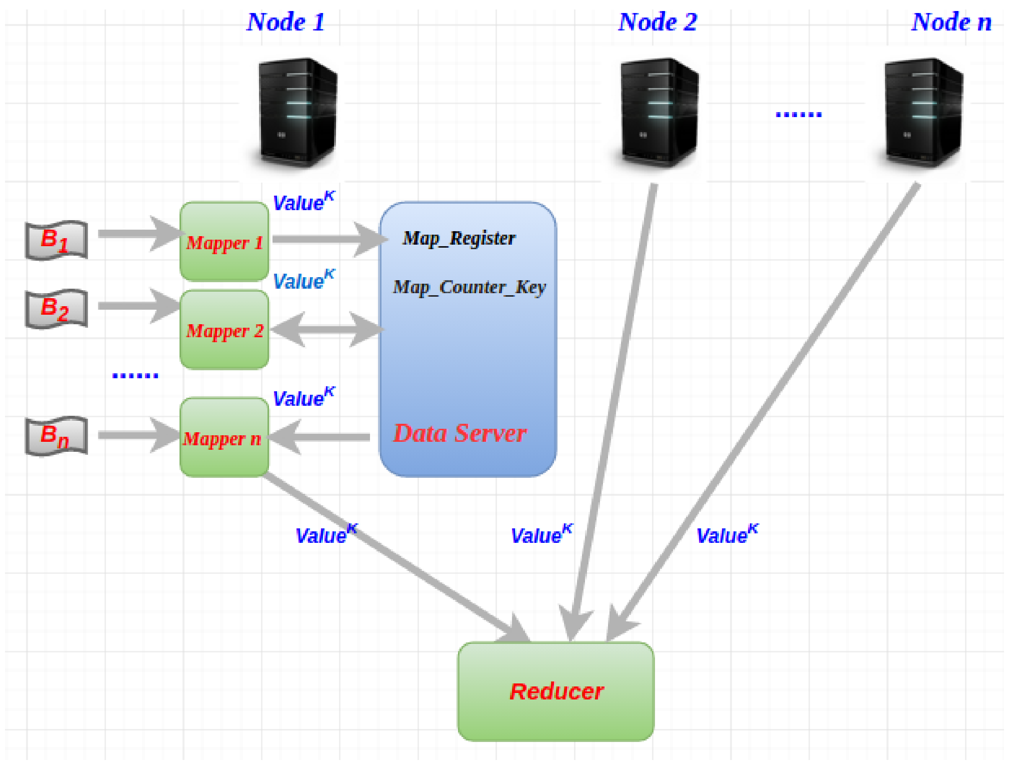


Figure 5.6: The MapReduce-based parallel archiving scheme.

We delineate the MapReduce-based parallel archiving algorithm in Section 15. To optimize the performance of our parallel archiving algorithm, we propose a new grouping strategy (see Section 5.3.1) to reduce the number of intermediate key-value pairs. We also design a local key-value store (see Section 5.3.1) to minimize the disk I/O load imposed in the map and reduce phases. The analysis of network traffic and disk I/Os can be found in Section 15.

Grouping Strategy

Fig. 5.6 illustrates our MapReduce-based strategy of grouping intermediate output from the multiple mappers (i.e., mappers 1–n). A conventional wisdom is to deliver an intermediate result created by each mapper to a reducer through the shuffling phase. To optimize the performance of our parallel archiving scheme, we group multiple intermediate results sharing the same *key* into one *Key-Value* pair to be transferred to the reducer. During the course of grouping, of course, the XOR operations are performed to generate the value in the *Key-Value* pair. The advantage of our new group strategy makes it possible to shift the grouping and computing load traditionally handled by reducers to mappers. In doing so, we alleviate the potential performance bottleneck problem incurred in the reducer.

Local Key-Value Store

At the heart of our MapReduce-based parallel archiving scheme is a *local Key-Value store* managed in each node (see the large block sitting in the center of Fig. 5.6). Fig. 5.6 shows that a local key-value store manipulated by node 1 enables the above grouping strategy to merge multiple intermediate results sharing the same *key* into one *Key-Value* pair.

Let us first introduce the notation used in Fig. 5.6, followed by the functionality of the local key-value stores. The arrows in Fig. 5.6 represent data flows among mappers, reducers, and the key-value data store. It is worth noting that the transferred data are in the form of $Value^k$s, where k is a key. The local key-value store residing in node 1 is responsible for temporally storing intermediate *Key-Value* pairs created by the mappers. n number of data servers, distributed to nodes 1–n nodes, are booted up ahead of spawned mappers. Each local key-value store, which processes local data, independently functions without collaborating with the other key-value stores on remote nodes.

The metadata of a local key-value store includes two counters, namely, the *Map_Register* and *Map_Counter_Key* counters (see also the local key-value store in Fig. 5.6). The first counter (i.e., *Map_Register*) is used to keep track of the total number of spawned mapper tasks on its local node. The second counter (i.e., *Map_Counter_Key*) monitors the number of mappers that have already stored their intermediate *Key-Value* pairs with respect to specific keys to the local key-value store.

The Parallel Archiving Algorithm

Now we present the algorithm of the MapReduce-based parallel archiving scheme. Algorithm 2 illustrates the detailed process of parallel archiving, where the Map-Register and

```
   Input: Map-Register, Map-Counter-Key, Val-Buffer
   Map-Register: spawned Mapper tasks on each node
   Map-Counter-Key: amount of mapper tasks which have already stored the key's value into data server
   Val-Buffer: store the value of Key-Value pairs
1  if Val-Buffer.get($KEY_k$) == null then
2      Val-Buffer.put($KEY_k$, VALUE)
3      Map-Counter-$KEY_k$++
4  else
5      Return Val-Buffer.get($KEY_k$) to Mapper
6      Mapper executes exclusive or calculation to returned value and value of $KEY_k$ emitted
7      Map-Counter-$KEY_k$++
8      if Map-Counter-$KEY_k$==Map-Register then
9          write calculated result to reducer
10         Val-Buffer remove $Key_k$
11     else
12         store calculated result to data server
13         Map-Counter-$KEY_k$++
14     end
15 end
```

Algorithm 2: The parallel archiving algorithm.

Map-Counter-Key values keep track of the total number of spawned mappers and the number of mappers that have stored intermediate results, respectively. Recall that Map-Register and Map-Counter-Key can be found in the local key-value store.

When a mapper delivers key K's value to the local key-value store, the algorithm checks whether the local store buffers a copy of key K's value (see Line 1 in Algorithm 2). If the value of key K is not residing in the local store, the algorithm places key K and its value into buffer *Val_ Buffer*, followed by increasing the counter of key *Key* by 1 (see Lines 1-3 in Algorithm 2 and mapper 1 in Fig. 5.6).

If key K's is found in the local key-value store, then the stored value will be forwarded back to the mapper, which will in turn be XORed with the same key's value emitted by the mapper (see Lines 5-6 in Algorithm 2 and Mapper 2 in Fig. 5.6).

After key K's counter is increased by 1, the algorithm checks if the mapper is the last one storing key K's value to the local key-value store. The algorithm achieves this goal by comparing *Map_ Counter_ Key* with *Map_ Register*. If the mapper is the last one, this mapper writes the key-value pair to a reducer rather than storing this key-value pair in the local store (see Lines 9-10 in Algorithm 2 and Mapper n in Fig. 5.6).

Unlike the last mapper, non-last mappers buffer the key-value pairs to the local store while increasing the counters of their key by 1 (see Lines 12-13 in Algorithm 2 and Mapper 2 in Fig. 5.6).

Reducing Network Traffic and I/O Load

Intuitively, we show that only one copy of each key K's value is shuffled to a reducer. As a result, regardless of the number of spawned mappers on each node, only one intermediate key-value pair is transferred on the network during the shuffling phase. Suppose there are m mappers on a node, our algorithm is capable of reducing the shuffling network load by $\frac{m-1}{m}\times 100\%$. For example, given a node running eight mappers, our algorithm reduces the network traffic by 87.5%.

Similarly, our algorithm significantly lowers the I/O load, because the total amount of data written to a local disk is reduced by $\frac{m-1}{m}\times 100\%$, where m is the number of running mappers on a node. Our algorithm is conducive to suppressing the I/O load of the map and reduce phases, because the local key-value store (see Fig. 5.6) buffers intermediate *Key-Value* pairs in the main memory rather than the hard drive.

5.3.2 aHDFS-Pipeline: Data Archiving with a Pipeline

Overview of Parallel Archiving with a Pipeline

To boost the performance of the parallel data archiving scheme proposed in Section 5.3.1, we design a pipeline and incorporate the pipelining technique into our *aHDFS* system. Fig. 5.7 shows a way of incorporating a pipeline into our parallel data archiving scheme for the $RS(6{+}2, 6)$-coded storage.

In the parallel archiving scheme described in Section 5.3.1 (see Fig. 5.6), intermediate parity blocks created by mappers running on a node are delivers to reducers. Unlike this process, the pipelined data archiving scheme delivers intermediate parity blocks (see blocks $P1$ and $P2$ in Fig. 5.7) to subsequent nodes (for example, node 3 and node n–1 in Fig. 5.7).

Next, the subsequent nodes repeatedly deliver their intermediate parity blocks to their subsequent nodes (see, for example, node n is the subsequent one of node n–1). Finally, the last node that has no subsequent node writes its parity blocks to the file system (i.e., HDFS).

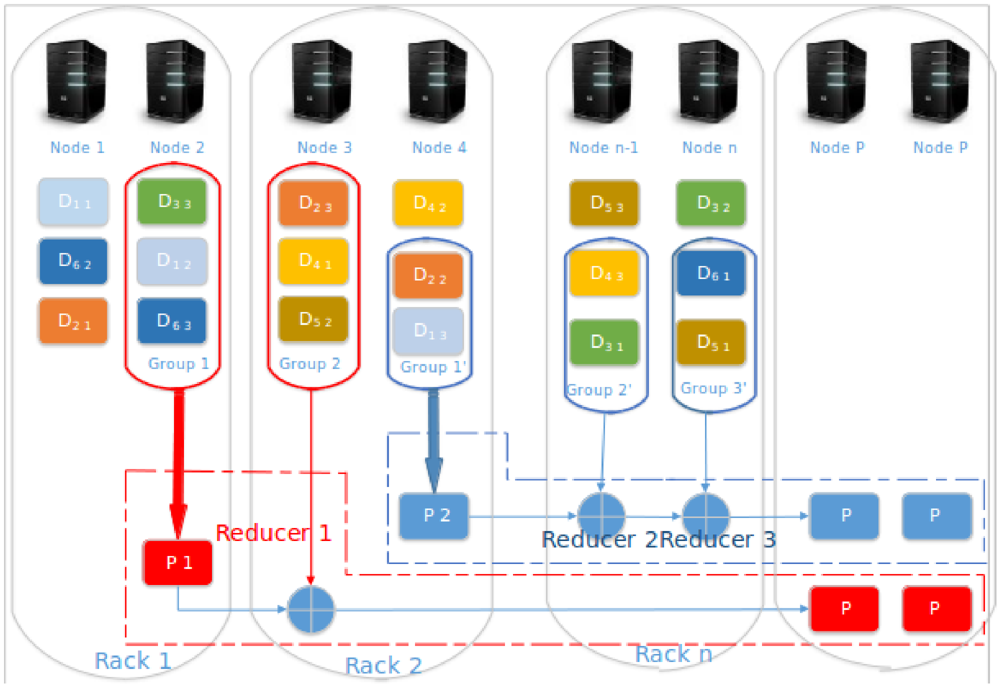


Figure 5.7: Incorporating a pipeline into our parallel data archiving scheme for the $RS(6+2,6)$-coded storage.

Constructing Data Archiving Pipelines

According to data placement governed by the layout of HDFS, all nodes can be divided into multiple groups, each of which handles data archiving operations in a pipelined manner. For example, Fig. 5.7 shows that node group 1 consists of nodes 2 and 3, whereas node group 2 contains nodes 4, n–1, and n. In this example, *aHDFS* builds an archiving pipeline between nodes 2 and 3 in node group 1; *aHDFS* constructs another archiving pipeline among nodes 4, n–1, and n in node group 2. These two archiving pipelines perform data archival operations in parallel. Please refer to the two run-time cases discussed in Section 5.3.1 for details on how to form multiple node groups to conduct parallel data archiving.

Multiple data archiving pipelines are capable of simultaneously carrying out encoding processes for separate data sets, thereby delivering high data-archival performance through improved archival parallelism.

```
Input: Map-Register, Map-Counter-Key, Val-Buffer
Map-Register: spawned Mapper tasks on each node
Map-Counter-Key: amount of mapper tasks which have already stored the key's value into data server
Val-Buffer: store the value of Key-Value pairs
if Val-Buffer.get($KEY_k$) == null then
    Val-Buffer.put($KEY_k$, VALUE)
    Map-Counter-$KEY_k$++
else
    Return Val-Buffer.get($KEY_k$) to Mapper
    Mapper executes exclusive or calculation to returned value and value of $KEY_k$ emitted
    Map-Counter-$KEY_k$++
    if Nodes is first node && Map-Counter-$KEY_k$==Map-Register then
        write calculated result to subsequential node
    else if node is not last node && Map-Counter-$KEY_k$==Map-Register+1 then
        write calculated result to subsequential node
    else if node is not last node && Map-Counter-$KEY_k$==Map-Register+1 then
        write calculated result to reducer
    Val-Buffer remove $Key_k$
    else
        store calculated result to data server
        Map-Counter-$KEY_k$++
    end
end
```

Algorithm 3: *aHDFS* PIPELINE ARCHIVING

MapReduce-based Pipelined Data Archiving

Our pipelined data archiving strategy is an extension of the parallel data archiving scheme proposed in Section 5.3.1. Fig. 5.8 plots that in the pipelined archiving strategy, the last mapper writes *key-value* pairs to its subsequent node's local *key–value* store (see Section 5.3.1) rather than a reducer. The last mapper in each node propagates *key-value* pairs to the node's subsequent node's key-value store. The last node in an archiving pipeline has no subsequent node, it is the last node's responsibility to write a parity block to HDFS.

Compared with the parallel archiving scheme (see also Section 15), this pipelined data archiving approach evenly distributes network I/O load among multiple nodes in each pipeline. A salient feature of our pipelined archiving scheme lies in its improvement of the parallel archiving scheme in terms of the archiving performance in the shuffling and reduce phases.

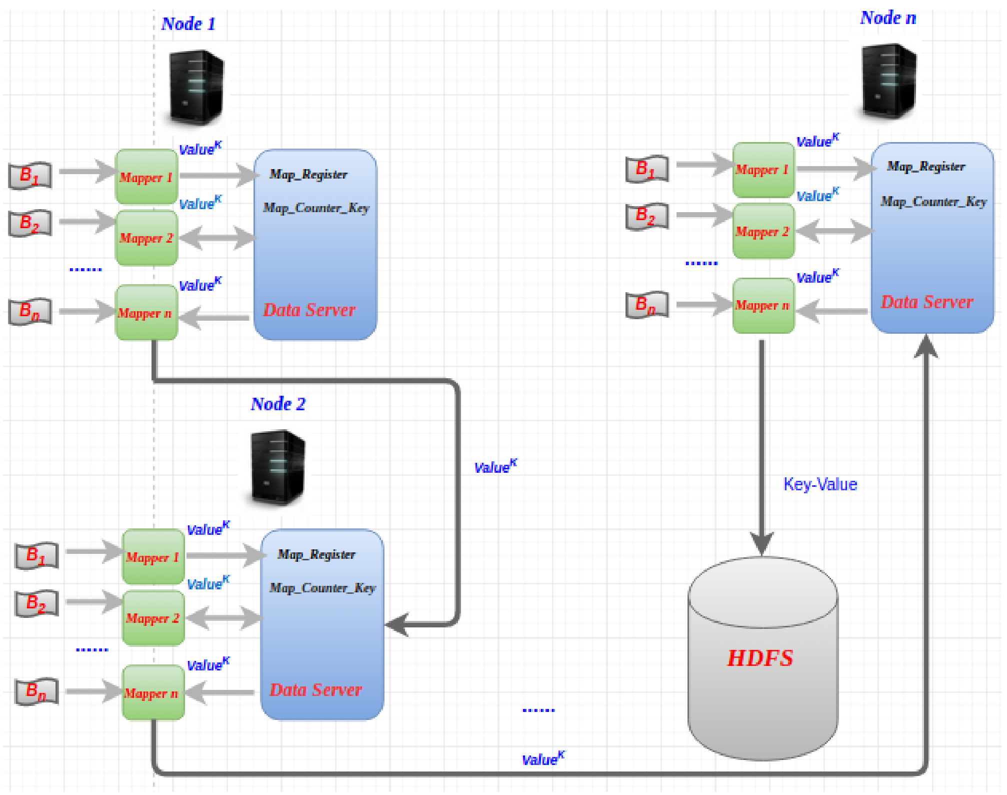


Figure 5.8: The MapReduce-based parallel archiving scheme coupled with a pipeline.

Algorithm 3 outlines the procedure of MapReduce-based pipelined data archiving. Algorithm 3) has to figure out whether a node is the first or the last node in an archiving pipeline (see Lines 8 and 10 in Algorithm 3), because the first node has no predecessor node whereas the last node has no subsequent node. The intermediate data are deliver from one node to its subsequent node (see Lines 9 and 11 in Algorithm 3). The last node in an archiving pipeline is in charge of writing a parity block to reducer (see Lines 12 and 13 in Algorithm 3). To check if all the mappers has stored intermediate *Key-Value* pairs to the local *key–value* store, all the nodes except the first node in a pipeline has to add 1 to *Register-Map*, indicating that these non-first nodes receive one more value from their predecessors (see Lines 10-13 in Algorithm 3).

Improving I/O Performance

Block sizes of HDFS are large; the default block size of HDFS [12] is 128MB. A storage request unit is an access unit from the I/O path's standpoint, where each block is accessed via a sequence of storage request units. I/O accesses tend to be non-sequential when storage request units are located on different blocks; as a result, such non-sequential access patterns downgrade the write bandwidth of our *aHDFS*.

The aforementioned non-sequential I/O access patterns may occur in *aHDFS* if the following two conditions are met: (1) Two parity blocks generated by two data archiving pipelines are written to an identical node and (2) the two parity blocks are written to different disk regions. We incorporate the *write aggregation* technique into *aHDFS*. It is noteworthy that the *write aggregation* technique is a popular I/O optimization scheme, because large storage request units help in achieving high write bandwidth [32][31], The write aggregation technique consolidates multiple request units into a single write request. A node storing parity blocks buffers storage request units delivered from multiple data archiving pipelines; then, the buffered parity-block requests are written to the node in form of a large sequential write request.

5.4 Case Studies

We conduct three case studies to uncover the factors affecting data archival performance. The case studies offer an insightful guidance on the development of the performance optimization strategies presented in Section 5.3.

5.4.1 Experimental Setup

We setup a Hadoop cluster; Table 5.1 summarizes the cluster's hardware and software configuration. The number of parity is configured to 3; and the number of nodes is configured as 10. We set the default value size, block size, and file size to 1KB, 128MB, and 10GB,

Hardware	
CPU	Intel Genuine 62@ 2.2GHz,20 CPU
Memory	128 GB
Network	1 GigaBit Ethernet network card
Disk	1 TB
Software	
Operating System	CentOS 6.5 Linux 2.6.32-431.el6.x86_64 x86_64
Hadoop	2.6.0

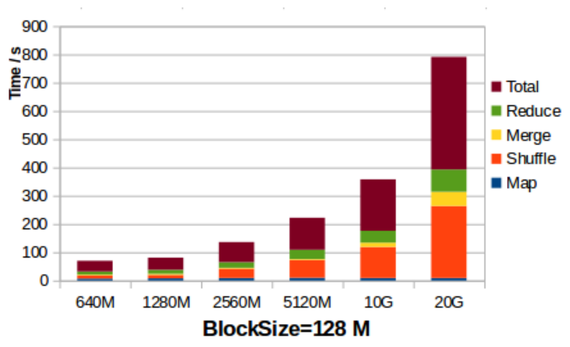
Table 5.1: Hardware and software configurations of the testbed.

respectively. We measure data archival performance in terms of the execution times in the map, shuffle, and reduce phases as well as the total execution time of each archiving process.

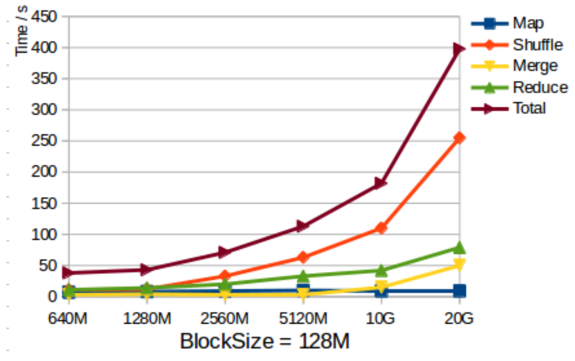
We investigate the impacts of file size, block size, and *key–value* pair size on the archival performance of a baseline system, where we disable the proposed grouping strategy (see Section 5.3.1), local key-value store (see Section 5.3.1), and archiving pipeline (see Section 5.3.2).

5.4.2 Case Study 1: File Size

We conduct six groups of experiments by varying file size, which is an affecting factor to archival performance. The file sizes chosen in the experiments are 640MB, 1280MB, 2560MB, 5120MB, 10GB, and 20GB. We set the block size and the value size to their default values.



(a)



(b)

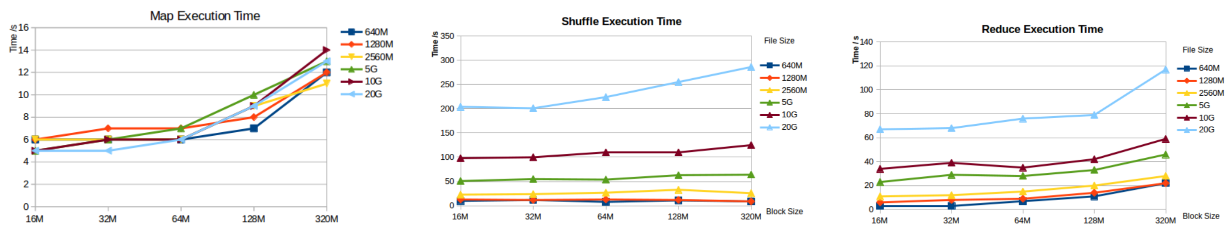
Figure 5.9: The execution time of the four MapReduce phases under the various file size

We aim to pinpoint data-archival-performance bottleneck among the four MapReduce phases, namely, map, shuffle, merge, and reduce. Fig. 5.9 illustrates the execution time of the four MapReduce phases under the various file size. Fig. 5.9(b) shows that with the file size increased from 640MB to 20GB, the total execution time significantly increases from 50 to 400 Sec. Such a trend is more pronounced when the file size is doubled from 10GB up to 20GB.

The results reveal that the map phase contributes a tiny portion to the total execution time (see Fig. 5.9(a)). Additionally, the map execution time does not change noticeably when the file size dramatically increase from 640MB to 20GB. In contrast to the map tasks, the shuffle tasks play a major role in determining the total execution time. More specifically, the shuffling time accounts for more than half of the total execution time (see Fig. 5.9(a)). Fig. 5.9(b) shows that the archival-performance trend, measured in total execution time, largely depends on the shuffling execution time. Following the shuffle tasks, the reduce tasks consume approximately 25% of the total execution time (see Fig. 5.9(a)).

Optimization Guidance. Because the shuffle and reduce phases account for 60% and 25% of the entire data archiving time, this convincing evidence suggests that shortening the time spent in the shuffle and reduce phases can greatly optimize the archival performance of *aHDFS*. Please refer to Section 5.1.3 for the discussions on the fact that an excessive amount of intermediate results from the mappers makes the shuffle phase impose a significant impact on the archival performance. The execution time of the reduce phase is also determined by the amount of data shuffled to the reducers.

We take the following two approaches to optimizing the data archival performance. First, because the load of the mappers is fairly light thanks to the high parallelism, we optimize the *aHDFS*'s performance by lowering the workload of the reducers, the computing load of which is reallocated to the mappers. Second, we minimize the amount of data shuffled to the reduce tasks. This goal can be achieved by our grouping strategy (see Section 5.3.1) and data-archiving pipelines (see Section 5.3.2).



(a) *Map* (b) *Shuffle* (c) *Reduce*

Figure 5.10: The execution times of the map, shuffle, and reduce phases under various block size

5.4.3 Case Study 2: Block Size

In this section, we conduct five groups of experiments to evaluate the impacts of block size on archival performance. Again, we set the file size to 640MB, 1280MB, 2560MB, 5120MB, 10GB, and 20GB, respectively.

Fig. 5.12 reveals that the execution times of the map, shuffle, and reduce phases, which are obviously affected by block size. Let us first consider the map phase (see Fig. 5.10(a)). When the block size increases from 16MB to 320MB, the execution time of the mappers doubles regardless of the file size. For example, when we set block size to 16 and 320MB, the time spent in the map phase is 6 and 12 Sec., respectively. Interestingly, keeping the blocks size a constant, we observe that the map execution time discrepancy among various file size is within two seconds.

The trend shown in Fig. 5.10(a) is attributed to fact that each spawned mapper is in charge of one block serving as an input split. A small block size implies that the amount of data to be loaded and calculated by each map task is small. Moreover, a small block size of a fixed file size means a large number of mappers, which pushes the parallelism level up. For example, Fig. 5.11 shows that when we increase the number of mappers from 16 to 90, the average map execution time is correspondingly cut down from 13 to 7.

In contrast, the mapper performance is marginally improved by one second when the number of mappers is changed from 90 to 180 and from 180 to 360, respectively. Such a limited performance improvement lies in the heavy load in terms of the number of mappers on each data node of the Hadoop cluster. In our experiments, the average number of mappers assigned to each node is 1.6, 4.5, 9, 18, and 36, respectively. The nodes are lightly loaded if the average number of mappers per node is below nine. Under light workload condition, increasing the number of mappers on each node enhances the parallelism level. On the other hand, assigning too many mappers to each node can undoubtedly overload the node, thereby slowing down the speed of the mappers.

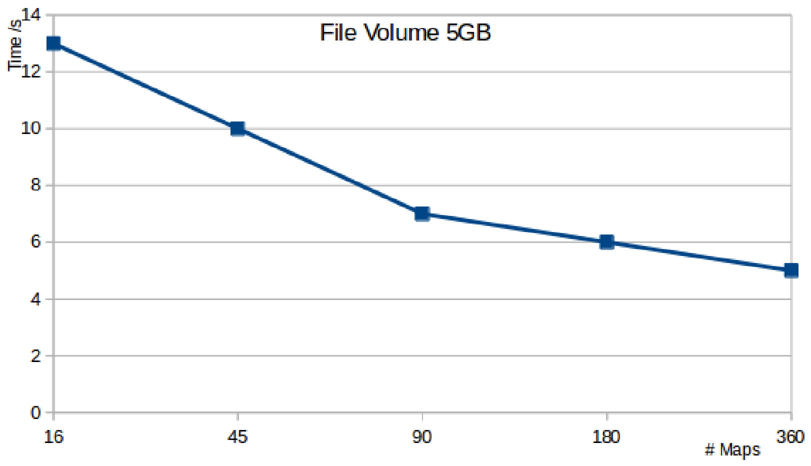


Figure 5.11: The impact of the number of map tasks on time spent in the map phase. File size = 5GB.

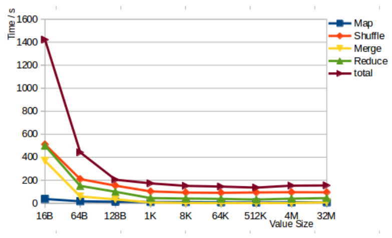
Fig. 5.10(b) shows that when the block size is increased from 16 to 320MB, the shuffling time is enlarged. This trend becomes more pronounced when the file size is large (e.g., 20GB). Similarly, Fig. 5.10(c) reveals that a large block size leads to a long reduce phase. Comparing Figs. 5.10(b) and 5.10(c), we observe that reduce execution time is more sensitive to block size than shuffle execution time. For instance, in the 20GB case, the shuffle and reduce execution times jump from 200 to 280 seconds and from 65 upto 120 seconds, respectively. In the same case, the map execution time only increases from 8 to 13 seconds.

Optimization Guidance. There are two approaches to tuning the performance of *aHDFS*. First, block size has a noticeable impact on map, shuffle, and reduce phases. We can speed up data archival performance by applying a small block size, which gives rise to high level of parallelisms. Second, to significantly shorten the reduce execution time, we make the data archiving process less reduce-intensive by shifting computing load from the reducers to the mappers in *aHDFS*.

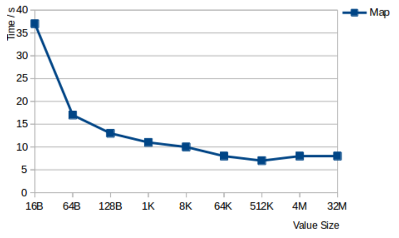
5.4.4 Case Study 3: Key-Value Pair Size

Now we demonstrate the impacts of *key-value* pair size on mappers in data archiving. In this group of experiments, we test nine value sizes (i.e., 16B, 64B, ..., 32M). Fig. 5.12(a) plots the correlation between execution times and the value size.

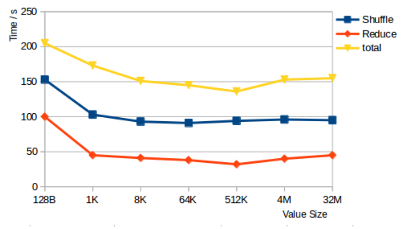
The results show that a large value size lowers the execution times. For example, when value size is 16B, the total execution time is 1422 seconds (see Tab. 5.2), which is approximately three times larger than the case of value size being 64B. Similarly, the shuffling time dramatically changes from 511 to 210 seconds; the merging time declines from 365 to 59 seconds; and the reducing time significantly shrinks 509 to 151 seconds. Not surprisingly, the total execution time drops from 422 to 205 seconds when the value size is doubled from 64 to 128B. Fig. 5.12(b) shows a large value size helps in reducing the mapping time. For example, increasing the value size from 16 to 64B cuts the mapping time from 37 to 17 seconds.



(a) *Total*



(b) *Map*



(c) *Other*

Figure 5.12: The execution times of the map, shuffle, and reduce phases under various value size

Figs. 5.12(a)-5.12(c) shows that when the value size is smaller than 128 KB, the archival performance is extremely sensitive to the value size. When the value size goes beyond 128 KB, further increasing the value size can hardly lower the archiving time. We discover that

512 KB is an optimal value size, which leads to the smallest map and reduce phases (i.e., 7 and 32 seconds in Table reftab:tm).

The aforementioned performance trends are attributed to the data overhead introduced to *key-value* pairs (See Section 5.1.3). Recall that each *key-value* pair has a eight-byte key and a four-byte partition identification (see Section 5.1.3). When the value size is 16B, the data overhead represents 75% of the value size; such overhead percentage drops down to 18.75%, if the value size is as large as 64B. Moreover, increasing the number of *key-value* pairs inevitably imposes data overhead used to maintain the partition identifications.

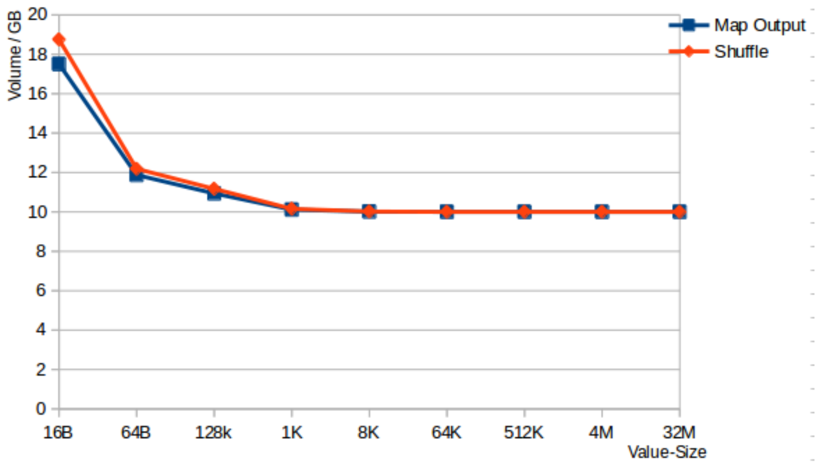


Figure 5.13: The amount of data in map phase and shuffle phase, File size = 10GB

Fig. 5.13 illustrates that a large value size leads to a small amount of data emitted from the mappers. Let us consider an example, where the input data size is 10 GB (see Tab. 5.3). When the value size is 16B, the amounts of map emitted data, output and input data are 17.5, 67.15, and 48.75 GB, respectively (see Fig. 5.14 and Tab. 5.3). Tab. 5.3 shows that network I/O traffic during the shuffling stage is 18.75 GB when the value size is set to 16B. Fortunately, an increasing value size helps to cut the amount of map emitted data as well as to alleviate the heavy network traffic (i.e., approximately to 10 GB, see also Fig 5.13). The I/O load in terms of writes and reads is gradually lowered down to the levels of 30 and 20 GB, respectively.

Optimization Guidance. This group of experiments suggests that the benefit of a large value size is four-fold: (1) minimizing data overhead; (2) reducing the number of I/O operations; (3) lowering network traffic; and (4) conserving time spent in sorting key-value pairs (see Step 5 in Section 5.1.3). Nevertheless, an extremely large value size may lead to the heap overflow problem, thereby downgrading archival performance (see Fig. 5.12(b) 5.12(c)).

Value Size	**Total**	**Map**	**Shuffle**	**Merge**	**Reduce**
16 Byte	1422	37	511	365	500
64 Byte	442	17	210	59	151
128 Byte	205	13	153	34	100
1 KB	173	11	103	7	45
8 KB	151	10	93	3	41
64 KB	145	8	91	2	38
512 KB	136	7	94	2	32
4 MB	153	8	96	2	40
32 MB	155	8	95	8	45

Table 5.2: The execution times of the map, shuffle, merge, and reduce phases as well ass total execution time under various value size

Value Size	**IO/W**	**IO/R**	**HDFS/R**	**Map/Out**	**Shuffling**
16 Byte	48.75	67.15	10.00	17.50	18.75
64 Byte	26.51	38.70	10.00	11.88	12.19
128 Byte	24.30	35.48	10.00	10.94	11.17
1 KB	20.31	30.48	10.00	10.12	10.16
8 KB	20.04	30.07	10.00	10.01	10.02
64 KB	20.01	30.02	10.00	10.01	10.00
512 KB	20.00	30.01	10.00	10.00	10.00
4 MB	20.00	30.01	10.00	10.00	10.00
32 MB	20.00	30.01	10.00	10.00	10.00

Table 5.3: I/O Write, Read, HDFS I/O Read, Mapper Output and Network Traffic I/O under various vaue size, File size = 10GB.

5.5 Performance Evaluation

We conduct extensive experiments to quantitative evaluate the archival performance of *aHDFS*. We also compare *aHDFS* with the baseline system, the performance of which

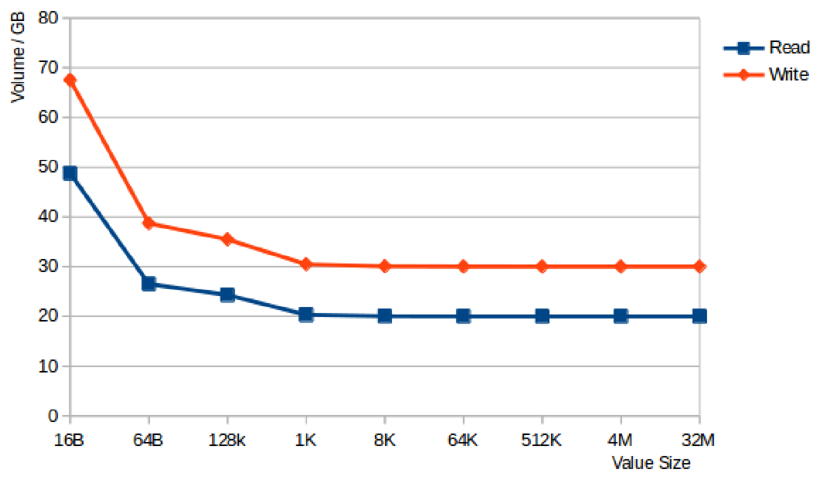


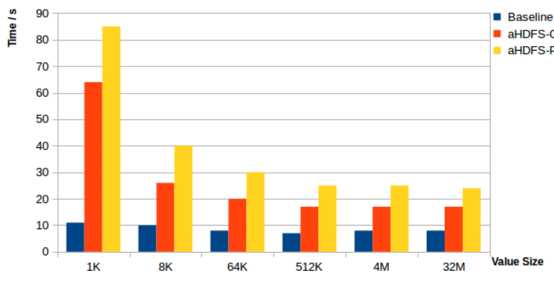
Figure 5.14: The amount of Input and Output data during the MapReduce' all phases. File size = 10GB

is analyzed in Section 5.4. We denote *aHDFS-Grouping* as the *aHDFS* system, where the grouping and local key-value store strategies are incorporated. We denote *aHDFS-Pipeline* as the *aHDFS* system, where the archiving pipeline is enabled. In the tested baseline system (i.e., referred to as *Baseline*), none of the three optimization schemes is applied. The details on the testbed can be found in Section 5.4.1.

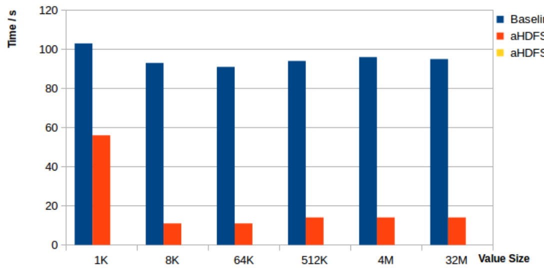
5.5.1 Key-Value Pair Size

We examine the impacts of key-value pair size on the archival performance of *aHDFS-Grouping*, *aHDFS-Pipeline*, and *Baseline*. We test six optimal value sizes (i.e., 1KB, 8KB, ..., 32M) (see Case Study 3 in Section 5.4.4 for the optimal value sizes). The file size and block size are set to 10GB and 128MB, respectively.

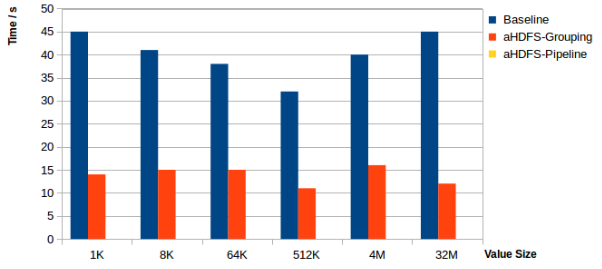
Fig. 5.15 shows execution times of the map, shuffle, reduce phases as well as the total time of the three systems under various value sizes. The results show that *Baseline* outperforms *aHDFS-Grouping* and *aHDFS-Pipeline* during the map phase when the value size is 1KB (see Fig. 5.15(a)). For example, the mapper execution times of *aHDFS-Grouping* and *aHDFS-Pipeline* are 65 and 85 seconds, which are approximately 6.5 and 8.5 times longer than that of*Baseline*. Interestingly, the discrepancy of the mapper execution times among



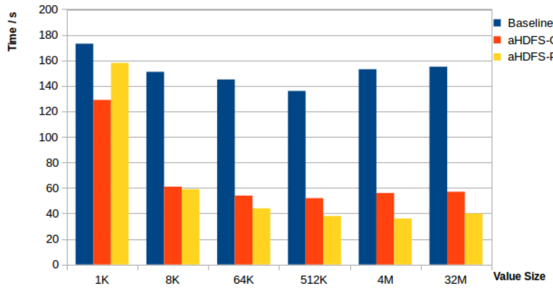
(a) *Map*



(b) *Shuffle*



(c) *Reduce*



(d) *Total*

Figure 5.15: The execution times of the map, shuffle, reduce phases as well as the total time of the three systems under various value sizes

aHDFS-Grouping, *aHDFS-Pipeline*, and *Baseline* is gradually diminishing when the value size increases. For instance, if the value size is 8KB, *aHDFS-Grouping* and *aHDFS-Pipeline*'s mapper execution times are approximately 2.5 and 4 times longer than that of *Baseline*. The mapper time differences among the three system is further narrowed down when the value size exceeds 512KB.

The aforementioned performance trends with respect to the map phase are attributed to the fact that heavy computation load is reallocated from reducers to mappers in *aHDFS-Grouping* and *aHDFS-Pipeline*. Each mapper in *aHDFS* should store temporal *key-value* pairs into the local *key–value* store; then, *aHDFS* performs the XOR operation on the emitted *key-value* pair with the previous stored *key-value* pairs in the local *key–value* store. Given fix-sized blocks, a small value size leads to a large number of *key-value* pairs, which in turn push up the number of I/O accesses to the local *key–value* store. Although the mapper execution time of *aHDFS* is longer compared to *Basedline*, *aHDFS-Grouping* is superior to *Baseline* in terms of shuffle and reduce execution times. For example, *aHDFS-Grouping* significantly shorten *Baseline*'s shuffle and reduce phase by approximate 90% and 60% when the value size is larger than 1KB. (see Figs. 5.15(b) and 5.15(c)).

Fig. 5.15(d) shows that when the value size is large than 1KB, *aHDFS-Grouping* and *aHDFS-Pipeline* are capable of cutting *Baseline*'s the overall archival time by up 60% and 75%, respectively.

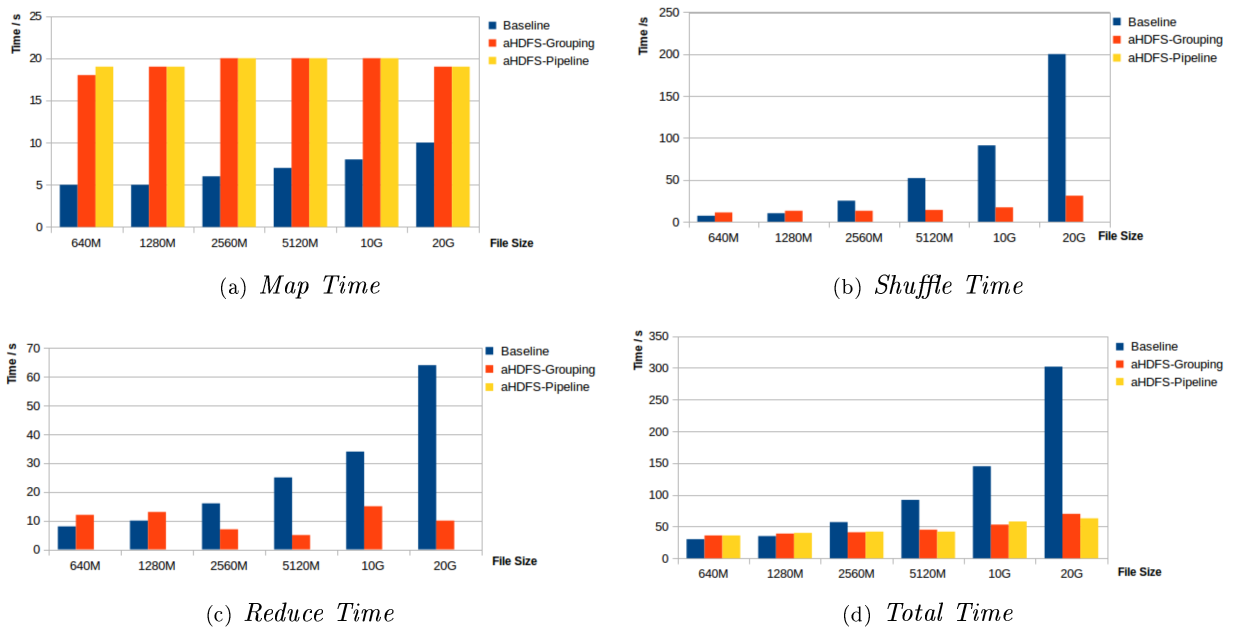


Figure 5.16: The execution time of the map, shuffle, and reduce phases along with the total execution time of the three systems under various file size

5.5.2 File Size

To evaluate the impacts of file size on the three data archival systems, we set the file size to 640MB, 1280MB, 2560MB, 5120MB, 10GB, and 20GB, respectively.

Fig. 5.16 illustrates the execution time of the map, shuffle, and reduce phases along with the total execution time. Fig. 5.16(a) shows that *Baseline*'s mapper execution time rises from 5 to 10 seconds when we increase the file size from 640MB to 20GB. In contrast, *aHDFS-Grouping* and *aHDFS-Pipeline*'s mapper times are almost kept at the level of 19 seconds.

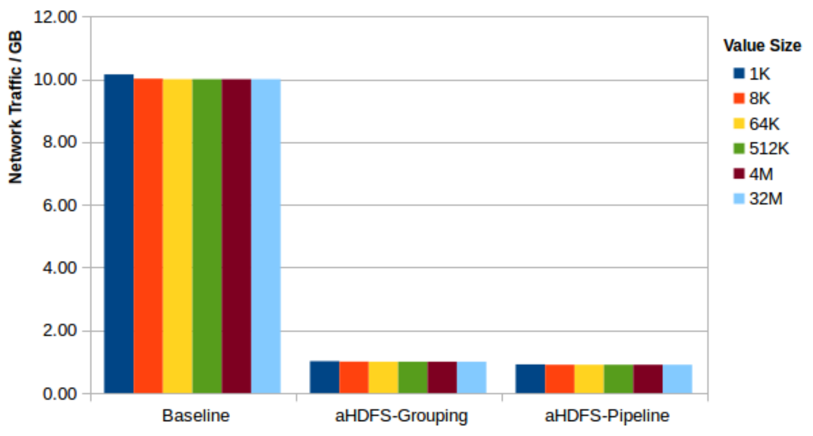


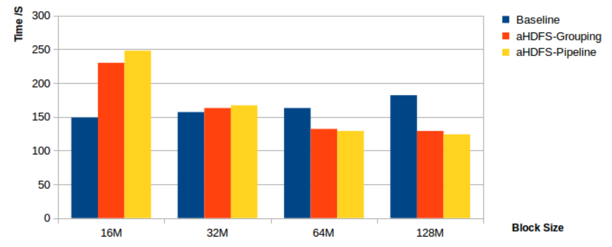
Figure 5.17: Network Traffic

Such a downside of *aHDFS-Grouping* and *aHDFS-Pipeline* is attributed to frequent accesses of the mappers to the local *key–value* store (see Section 5.3.1 and 5.3.2).

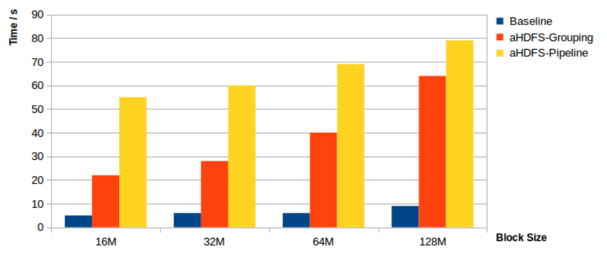
Figs. 5.16(b) and 5.16(c) shows that *aHDFS-Pipeline* embraces neither shuffle nor reduce phases. Recall that *Baseline* and *aHDFS-Grouping* have a reduce phase being responsible for receiving output from mappers and computing final results to be written into HDFS. In *aHDFS-Pipeline*, the last finished mapper residing in the last node of a pipeline is responsible for writing parities to HDFS.

In contrast to mappers, the shuffle and reduce tasks in *aHDFS-Grouping* take less time than those in *Baseline* in the large-file-size cases. For example, *aHDFS-Grouping* shortens the shuffle execution time of *Baseline* by up to 85.7% when the file size is 20GB (see Fig. 5.16(b)). A similar trend in the reduce phase can also be observed from Fig. 5.16(c). Such a performance improvement of *aHDFS-Grouping* over *Baseline* is more pronounced when we increase file size, because *key-value* pairs with same keys emitted from mappers on each node are grouped by *aHDFS-Grouping* as a single final *key-value* pair shuffled to reducers.

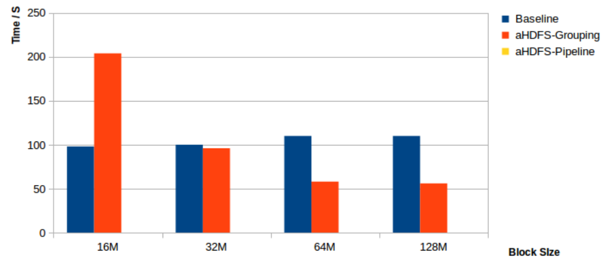
aHDFS-Grouping achieves the above performance improvement by (1) reducing network I/O traffic and (2) cutting down the data amount of *key–value* pairs processed by the reducers (see Section 15). The evidence shown in Fig. 5.17 indicates that *aHDFS-Grouping* and *aHDFS-Pipeline* are conducive to lowering the network I/O traffic of *Baseline* by approximately to 87% and 89%, respectively.



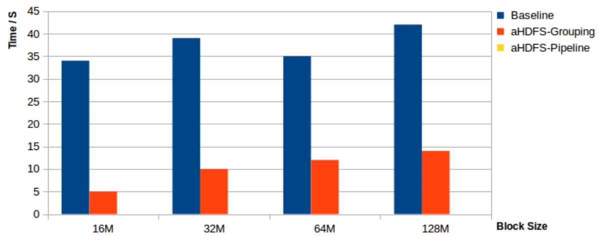
(a) *Total Time*



(b) *Map Time*



(c) *Shuffle Time*



(d) *Reduce Time*

Figure 5.18: The execution time of the map, shuffle, and reduce phases along with the total execution time of the three systems under various block size

5.5.3 Block Size

To evaluate the archival performance impacts of block size on *aHDFS*, we set the block size to 16MB, 32MB, 64MB and 128MB, respectively. Fig. 5.18(a) shows that *aHDFS-Grouping* and *aHDFS-Pipeline* can cut *Baseline*'s total archiving time by approximately 20% when the bock size is set to 64 and 128MB. In what follows, we show the comparison of the three schemes with respect to execution times of the map, shuffle, and reduce phase.

Fig. 5.18(b) shows that block size has a big impact on mapper performance. For example, when block size increases from 16 to 128MB, the map execution times of *Baseline*, *aHDFS-Grouping*, and *aHDFS-Pipeline* are enlarged by 60%, 300%, and 60%, respectively. Given files with fixed size, a small block size triggers a large number of mappers spawned, which in turn push up the parallelism level. *aHDFS-Grouping* outperforms *aHDFS-Pipeline* at the map phase, because mappers at the downstream nodes of a pipeline has to wait for predecessor nodes to deliver *key-value* pairs. Delaying mappers in the predecessor nodes slows down the process of the entire pipeline.

Fig. 5.18(c) reveals that *aHDFS-Grouping*'s shuffle time is two times longer than that of *Baseline* when block size is 16MB. Nevertheless, increasing block size helps in shortening *aHDFS-Grouping*'s shuffle time; in contrast, the increased block size slightly worsens the shuffle time of *Baseline*. Consequently, when the block size is larger than or equal to 32MB, *aHDFS-Grouping* is superior to *Baseline* in terms of shuffle time. Fig. 5.18(d) shows that a large block size leads to long reduce execution time. *aHDFS-Grouping* exhibits small reduce time in the small-block-size case, because each node only shuffles one block to reducers, the processing time of which depends on block size.

Chapter 6

Conclusion and Future Work

In this dissertation, we proposed two novel strategies to achieve thermal and resource efficiency in data center. *Thermobench* provides a simple yet powerful benchmark solution for assessing thermal behaviours of computing clusters in data centers. TERN running one ThermoBnech is conducive to predicting thermal and resource trends of diverse workload conditions with a changing transaction mix. aHDFS focus more on achieving storage efficiency in data center. This chapter concludes the dissertation study by summarizing the contributions and future work.

6.1 Self-Adjusting Thermal Model for Dynamic Resource Provisioning in Data Centers

We have proposed a modeling system called TERN, which dynamically adjusts itself to predict thermal behaviors of hardware resources allocated to applications running on clusters. TERN is conducive to predicting thermal patterns of diverse workload conditions with changing transac- tion mixes. More precisely, TERN seamlessly integrates a resource utilization model and a thermal model, which establish correlation between resource utilization and ther- mal trends of processors and hard drives in servers. We made use of the TPC-W benchmark to conduct thermal profiling studies of workload with various transaction mixes. A salient feature of TERN lies in its capability of adjusting the modeling parameters to improve prediction accuracy by approximately 10terns. We showed through the experiments performed on a real-world cluster that TERN offers a simple yet pow- erful solution for thermal-aware resource provisioning in data centers. The main drawback of TERN is that it is focused on CPU and disk resources. For large-scale data centers, we have to consider other shared system resources like main memory and network interconnections. We plan to extend TERN by incorporating the other resource types supporting various applications on clusters. Furthermore, we will exploit a prediction model for thermal-aware Hadoop clusters in the context of big data analytics.

6.2 An Erasure-Coded Data Archival System for Hadoop Clusters

We presents an erasure-coded data archival system - aHDFS - in the realm of Hadoop cluster computing. We proposed two archiving strategies called aHDFS-Grouping and aHDFS-Pipeline to speed up archival performance in Hadoop distributed file system or HDFS. Both the archiving schemes adopt the MapReduce-based grouping strategy, which wraps up multiple intermediate key-value pairs sharing same key into one key-value pair on each node. aHDFS-Grouping transfers the single key-value pair to a reducer, whereas aHDFS-Pipeline delivers this key-value pair to the subsequent node in the archiving pipeline. We implemented these two archiving strategies, which were compared against the conventional MapReduce-based archiving strategy referred to as Baseline. The experimental results show that aHDFS-Grouping and aHDFS-Pipeline can improve the overall archival performance of Baseline by a factor of 4. In particular, aHDFS-Grouping and aHDFS- Pipeline speed up Baseline's shuffle and reduce phases by a factor of 10 and 5, respectively. In addition, aHDFS- Grouping and aHDFS-Pipeline significantly lower the net- work I/O traffic by 87% and 89%, respectively. As a future research direction, we will develop a data reconstruction system to deal with block failure issues on Hadoop clusters. We plan to apply the grouping and pipelining strategies to the reconstruction system to speed up the reconstruction process. To optimize reconstruction performance, we will investigate a way of choosing inter- mediate parity blocks to be kept in the local key-value store.

Bibliography

[1] Amazon 2000. `http://www.fool.com/news/2000/wmt001127.htm`.

[2] Colossus: Successor to the google file system (gfs). `http://www.highlyscalablesystems.com/3202/colossus-successor-to-google-file-system-gfs/`.

[3] Tpc-w benckmark specification. `http://www.tpc.org/tpcw/spec/tpcw_v1.8.pdf`.

[4] Wd1600aajs specification. `http://www.wdc.com/wdproducts/library/SpecSheet/ENG/2879-701277.pdf`.

[5] Wd5000aaks specification. `http://www.wdc.com/wdproducts/library/SpecSheet/ENG/2879-701277.pdf`.

[6] B. G. H. T. S. A. Beyond the data deluge. In *Computer Science*, volume 323, pages 1297–1298, March 2009.

[7] M. Allalouf, Y. Arbitman, M. Factor, R. I. Kat, K. Meth, and D. Naor. Storage modeling for power estimation. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, SYSTOR '09, pages 3:1–3:10, New York, NY, USA, 2009. ACM.

[8] L. A. Barroso and U. HÃűlzle. The case for energy-proportional computing. *IEEE Computer*, 40, 2007.

[9] C. Bash and G. Forman. Cool job allocation: Measuring the power savings of placing jobs at cooling-efficient locations in the data center. In *USENIX Annual Technical Conference*, volume 138, page 140, 2007.

[10] C. Bash and G. Forman. Cool job allocation: Measuring the power savings of placing jobs at cooling-efficient locations in the data center. In *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, ATC'07, pages 29:1–29:6, Berkeley, CA, USA, 2007. USENIX Association.

[11] F. Bellosa, A. Weissel, M. Waitz, and S. Kellner. Event-driven energy accounting for dynamic thermal management. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLPâĂŹ03)*, volume 22, 2003.

[12] D. Borthakur. The hadoop distributed file system: Architecture and design, 2007. *Apache Software Foundation*, 2012.

[13] D. J. Brown and C. Reams. Toward energy-efficient computing. *Commun. ACM*, 53(3):50–58, Mar. 2010.

[14] R. E. Brown, E. R. Masanet, B. Nordman, W. F. Tschudi, A. Shehabi, J. Stanley, J. G. Koomey, D. A. Sartor, and P. T. Chan. Report to congress on server and data center energy efficiency: Public law 109-431. 06/2008 2008.

[15] R. Buyya, A. Beloglazov, and J. H. Abawajy. Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. *CoRR*, abs/1006.0308, 2010.

[16] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, et al. Windows azure storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 143–157. ACM, 2011.

[17] Y. Chai, Z. Du, D. A. Bader, and X. Qin. Efficient data migration to conserve energy in streaming media storage systems. *Parallel and Distributed Systems, IEEE Transactions on*, 23(11):2081–2093, 2012.

[18] J. C. Chan, Q. Ding, P. P. Lee, and H. H. Chan. Parity logging with reserved space: towards efficient updates and recovery in erasure-coded clustered storage. In *FAST*, pages 163–176, 2014.

[19] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, pages 103–116, New York, NY, USA, 2001. ACM.

[20] T. Daim, J. Justice, M. Krampits, M. Letts, G. Subramanian, and M. Thirumalai. Data center metrics: An energy efficiency model for information technology managers. *Management of Environmental Quality: An International Journal*, 20(6):712–731, 2009.

[21] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[22] R. P. Doyle. *Model-based Adaptive Resource Provisioning in a Web Service Utility.* PhD thesis, Durham, NC, USA, 2003. AAI3135127.

[23] R. P. Doyle, J. S. Chase, O. M. Asad, W. Jin, and A. Vahdat. Model-based resource provisioning in a web service utility. In *USENIX Symposium on Internet Technologies and Systems*, volume 4, pages 5–5, 2003.

[24] N. El-Sayed, I. A. Stefanovici, G. Amvrosiadis, A. A. Hwang, and B. Schroeder. Temperature management in data centers: why some (might) like it hot. *ACM SIGMETRICS Performance Evaluation Review*, 40(1):163–174, 2012.

[25] E. Feller, L. Ramakrishnan, and C. Morin. Performance and energy efficiency of big data applications in cloud environments: A hadoop case study. *Journal of Parallel and Distributed Computing*, 2015.

[26] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in globally distributed storage systems. In *OSDI*, pages 61–74, 2010.

[27] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM, 2003.

[28] R. Gupta, H. Gupta, U. Nambiar, and M. Mohania. Efficiently querying archived data using hadoop. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1301–1304. ACM, 2010.

[29] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. Elastictree: Saving energy in data center networks. In *NSDI*, volume 10, pages 249–264, 2010.

[30] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, S. Yekhanin, et al. Erasure coding in windows azure storage. In *Usenix annual technical conference*, pages 15–26. Boston, MA, 2012.

[31] J. Huang, X. Liang, X. Qin, P. Xie, and C. Xie. Scale-rs: An efficient scaling scheme for rs-coded storage clusters. 2015.

[32] J. Huang, Y. Wang, X. Qin, X. Liang, S. Yin, and C. Xie. Exploiting pipelined encoding process to boost erasure-coded data archival.

[33] J. Huang, F. Zhang, X. Qin, and C. Xie. Exploiting redundancies and deferred writes to conserve energy in erasure-coded storage clusters. *ACM Transactions on Storage (TOS)*, 9(2):4, 2013.

[34] M. Ibrahim, S. Gondipalli, S. Bhopte, B. Sammakia, B. Murray, K. Ghose, M. K. Iyengar, and R. Schmidt. Numerical modeling approach to dynamic data center cooling. In *Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2010 12th IEEE Intersociety Conference on*, pages 1–7. IEEE, 2010.

[35] X. Jiang, M. M. Al Assaf, J. Zhang, M. I. Alghamdi, X. Ruan, T. Muzaffar, and X. Qin. Thermal modeling of hybrid storage clusters. *Journal of Signal Processing Systems*, 72(3):181–196, 2013.

[36] X. Jiang, M. I. Alghamdi, M. M. Al Assaf, X. Ruan, J. Zhang, M. Qiu, and X. Qin. Thermal modeling and analysis of cloud data storage systems. *Journal of Communications*, 9(4), 2014.

[37] X. Jiang, M. I. Alghamdi, J. Zhang, M. Assaf, X. Ruan, T. Muzaffar, and X. Qin. Thermal modeling and analysis of storage systems. In *Performance Computing and Communications Conference (IPCCC), 2012 IEEE 31st International*, pages 31–40. IEEE, 2012.

[38] X. Jiang, J. Zhang, M. Alghamdi, X. Qin, M. Jiang, and J. Zhang. Peam: Predictive energy-aware management for storage systems. In *Networking, Architecture and Storage (NAS), 2013 IEEE Eighth International Conference on*, pages 105–114, July 2013.

[39] R. T. Kaushik and K. Nahrstedt. T: a data-centric cooling energy costs reduction approach for big data analytics cloud. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 52. IEEE Computer Society Press, 2012.

[40] O. Khan, R. C. Burns, J. S. Plank, W. Pierce, and C. Huang. Rethinking erasure codes for cloud file systems: minimizing i/o for recovery and degraded reads. In *FAST*, page 20, 2012.

[41] J. Koomey. *Growth in data center electricity use 2005 to 2010*. Oakland, CA: Analytics Press, 2011.

[42] J. G. Koomey. Estimating total power consumption by servers in the u.s. ad the world. Technical report, Lawrence Berkeley National Laboratory and Consulting Professor, Stanford University, 2007.

[43] D. Kusic and N. Kandasamy. Risk-aware limited lookahead control for dynamic resource provisioning in enterprise computing systems. *Cluster Computing*, 10(4):395–408, 2007.

[44] J. Laudon. Performance/watt: the new server focus. *ACM SIGARCH Computer Architecture News*, 33(4):5–13, 2005.

[45] K. Le, R. Bianchini, M. Martonosi, and T. D. Nguyen. Cost-and energy-aware load distribution across data centers. *Proceedings of HotPower*, pages 1–5, 2009.

[46] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hyser. Renewable and cooling aware workload management for sustainable data centers. In *ACM SIGMETRICS Performance Evaluation Review*, volume 40, pages 175–186. ACM, 2012.

[47] P. Mahadevan, S. Banerjee, P. Sharma, A. Shah, and P. Ranganathan. On energy efficiency for enterprise and data center networks. *IEEE Communications Magazine*, 49(8):94–100, August 2011.

[48] A. Manzanares, X. Qin, X. Ruan, and S. Yin. Pre-bud: Prefetching for energy-efficient parallel i/o systems with buffer disks. *ACM Transactions on Storage (TOS)*, 7(1):3, 2011.

[49] D. Menasce. Tpc-w: a benchmark for e-commerce. volume 6, pages 83–87, May 2002.

[50] S. S. Miller, M. S. Shaalan, and L. E. Ross. Correspondent-centric management email system uses message-correspondent relationship data table for automatically linking a single stored message with its correspondents, Sept. 2 2003. US Patent 6,615,241.

[51] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling "cool": temperature-aware workload placement in data centers. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, pages 5–5, 2005.

[52] J. Moore and J. S. Chase. Weatherman: Automated, online, and predictive thermal mapping and management for data centers. In *In International Conference on Autonomic Computing*, 2006.

[53] V. Nae, A. Iosup, and R. Prodan. Dynamic resource provisioning in massively multiplayer online games. *Parallel and Distributed Systems, IEEE Transactions on*, 22(3):380–395, 2011.

[54] V. Nae, A. Iosup, and R. Prodan. Dynamic resource provisioning in massively multiplayer online games. *Parallel and Distributed Systems, IEEE Transactions on*, 22(3):380–395, March 2011.

[55] H. S. Norman R. Draper. *Applied Regression Analysis*. Wiley-Interscience, 3rd edition, April 1998.

[56] M. Ovsiannikov, S. Rus, D. Reeves, P. Sutter, S. Rao, and J. Kelly. The quantcast file system. *Proceedings of the VLDB Endowment*, 6(11):1092–1101, 2013.

[57] E. Pakbaznia, M. Ghasemazar, and M. Pedram. Temperature-aware dynamic resource provisioning in a power-optimized datacenter. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 124–129, 3001 Leuven, Belgium, Belgium, 2010. European Design and Automation Association.

[58] L. Pamies-Juarez, A. Datta, and F. Oggier. Rapidraid: Pipelined erasure codes for fast data archival in distributed storage systems. In *INFOCOM, 2013 Proceedings IEEE*, pages 1294–1302. IEEE, 2013.

[59] L. Pamies-Juarez, F. Oggier, and A. Datta. Data insertion and archiving in erasure-coding based large-scale storage systems. In *Distributed Computing and Internet Technology*, pages 47–68. Springer, 2013.

[60] L. Pamies-Juarez, F. Oggier, and A. Datta. Decentralized erasure coding for efficient data archival in distributed storage systems. In *Distributed Computing and Networking*, pages 42–56. Springer, 2013.

[61] Y. Pan, R. Yin, and Z. Huang. Energy modeling of two office buildings with data center for green building design. *Energy and Buildings*, 40(7):1145 – 1152, 2008.

[62] L. Parolini, B. Sinopoli, B. H. Krogh, and Z. Wang. A cyber–physical systems approach to data center modeling and control for energy efficiency. *Proceedings of the IEEE*, 100(1):254–268, 2012.

[63] C. Patel and P. Ranganathan. Enterprise power and cooling. *ASPLOS tutorial*, 2006.

[64] C. D. Patel. A vision of energy aware computing from chips to data centers. In *The International Symposium on Micro-Mechanical Engineering*, 2003.

[65] C. D. Patel, R. Sharma, C. E. Bash, and A. Beitelmal. Thermal considerations in cooling large scale high compute density data centers. In *Thermal and Thermomechanical Phenomena in Electronic Systems, 2002. ITHERM 2002. The Eighth Intersociety Conference on*, pages 767–776. IEEE, 2002.

[66] C. L. B. P.E. In the data center, power and cooling costs more than the it equipment it supports. In *Electronics Cooling*, February 2007.

[67] M. Pitkanen, R. Moussa, M. Swany, and T. Niemi. Erasure codes for increasing the availability of grid data storage. In *Telecommunications, 2006. AICT-ICIW'06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*, pages 185–185. IEEE, 2006.

[68] S. Quinlan and S. Dorward. Venti: A new approach to archival storage. In *FAST*, volume 2, pages 89–101, 2002.

[69] L. Ramapantulu, B. M. Tudor, D. Loghin, T. Vu, and Y. M. Teo. Modeling the energy efficiency of heterogeneous clusters. In *Parallel Processing (ICPP), 2014 43rd International Conference on*, pages 321–330. IEEE, 2014.

[70] K. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran. A hitchhiker's guide to fast and efficient data reconstruction in erasure-coded data centers. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 331–342. ACM, 2014.

[71] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[72] Z. Ren, J. Wan, W. Shi, X. Xu, and M. Zhou. Workload analysis, implications, and optimization on a production hadoop cluster: A case study on taobao. *Services Computing, IEEE Transactions on*, 7(2):307–321, 2014.

[73] S. Rivoire, M. A. Shah, P. Ranganathan, and C. Kozyrakis. Joulesort: A balanced energy-efficiency benchmark. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 365–376, New York, NY, USA, 2007. ACM.

[74] S. Rivoire, M. A. Shah, P. Ranganathan, and C. Kozyrakis. Joulesort: a balanced energy-efficiency benchmark. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 365–376. ACM, 2007.

[75] S. Rivoire, M. A. Shah, P. Ranganathan, C. Kozyrakis, and J. Meza. Models and metrics to enable energy-efficiency optimizations. 2007.

[76] D. Robinson. *Amazon Web Services Made Simple: Learn how Amazon EC2, S3, SimpleDB and SQS Web Services enables you to reach business goals faster.* Emereo Pty Ltd, 2008.

[77] X. Ruan, S. Yin, A. Manzanares, J. Xie, Z. Ding, J. Majors, and X. Qin. Ecos: An energy-efficient cluster storage system. In *Performance Computing and Communications Conference (IPCCC), 2009 IEEE 28th International*, pages 79–86, Dec 2009.

[78] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur. Xoring elephants: Novel erasure codes for big data. 6(5):325–336, 2013.

[79] D. Schall, V. Hoefner, and M. Kern. Towards an enhanced benchmark advocating energy-efficient systems. In *Topics in Performance Evaluation, Measurement and Characterization*, pages 31–45. Springer, 2012.

[80] F. B. Schmuck and R. L. Haskin. Gpfs: A shared-disk file system for large computing clusters. In *FAST*, volume 2, page 19, 2002.

[81] T. J. Schwarz, Q. Xin, E. L. Miller, D. D. Long, A. Hospodor, and S. Ng. Disk scrubbing in large archival storage systems. In *Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004.(MASCOTS 2004). Proceedings. The IEEE Computer Society's 12th Annual International Symposium on*, pages 409–418. IEEE, 2004.

[82] A. Shah, C. Patel, C. Bash, R. Sharma, and R. Shih. Impact of rack-level compaction on the data center cooling ensemble. In *Thermal and Thermomechanical Phenomena in Electronic Systems, 2008. ITHERM 2008. 11th Intersociety Conference on*, pages 1175–1182. IEEE, 2008.

[83] J. T. Slow but steady growth for data centers through 2015. October 2011.

[84] Q. Tang, S. Gupta, and G. Varsamopoulos. Thermal-aware task scheduling for data centers through minimizing heat recirculation. In *Cluster Computing, 2007 IEEE International Conference on*, pages 129 –138, sept. 2007.

[85] M. Tavis and P. Fitzsimons. Web application hosting in the aws cloud: Best practices. *Amazon AWS Whitebooks*, 2010.

[86] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu. Data warehousing and analytics infrastructure at facebook. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1013–1020. ACM, 2010.

[87] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. Agile dynamic provisioning of multi-tier internet applications. *ACM Trans. Auton. Adapt. Syst.*, 3(1):1:1–1:39, Mar. 2008.

[88] N. Vasic, T. Scherer, and W. Schott. Thermal-aware workload scheduling for energy efficient data centers. In *Proceedings of the 7th international conference on Autonomic computing*, ICAC '10, pages 169–174, 2010.

[89] M. A. Viredaz, S. Pradhan, M. Mesarina, and G. M. Lyon. Method and system for dynamically controlling cooling resources in a data center, May 13 2008. US Patent 7,373,268.

[90] M. vor dem Berge, G. Da Costa, M. Jarus, A. Oleksiak, W. Piatek, and E. Volk. Modeling data center building blocks for energy-efficiency and thermal simulations. In *Energy-Efficient Data Centers*, pages 66–82. Springer, 2014.

[91] L. Wang, F. Zhang, J. A. Aroca, A. V. Vasilakos, K. Zheng, C. Hou, D. Li, and Z. Liu. Greendcn: A general framework for achieving energy efficiency in data center networks. *IEEE Journal on Selected Areas in Communications*, 32(1):4–15, January 2014.

[92] H. Weatherspoon and J. D. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Peer-to-Peer Systems*, pages 328–337. Springer, 2002.

[93] M. Xia, M. Saxena, M. Blaum, and D. A. Pease. A tale of two erasure codes in hdfs. In *To appear in Proceedings of 13th Usenix Conference on File and Storage Technologies*, 2015.

[94] L. L. You, K. T. Pollack, and D. D. Long. Deep store: An archival storage system architecture. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 804–815. IEEE, 2005.

[95] Q. Zhang, L. Cherkasova, and E. Smirni. A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In *Autonomic Computing, 2007. ICAC '07. Fourth International Conference on*, pages 27–27, June 2007.

[96] Z. Zong, A. Manzanares, X. Ruan, and X. Qin. Ead and pebd: two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters. *Computers, IEEE Transactions on*, 60(3):360–374, 2011.