**Improving Energy Efficiency and Security in Cluster Computing Systems**

by

Xiaojun Ruan

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
Aug 6th, 2011

Keywords: Energy-Efficient, Cluster Computing Systems, Parallel and Distributed Computing, Real-Time Systems, Storage Systems, Computer Security



Approved by

Xiao Qin, Chair, Associate Professor of Computer Science and Software Engineering
David A. Umphress, Associate Professor of Computer Science and Software Engineering
Richard O. Chapman, Associate Professor of Computer Science and Software Engineering

Abstract

Cluster computing systems are widely used in parallel and distributed computing research. Besides performance, energy cost and security should also be carefully concerned in large scale cluster computing systems to reduce budget and to avoid information leak. In this dissertation, I proposed a Time Aware Dynamic Voltage Scaling scheduling algorithm to conserve energy cost of processors in parallel computing systems and a design of an energy-efficient I/O System with write buffer disks to conserve energy cost of large scale storage systems. To explain when the energy consumption could be reduced in cluster computing systems, I analyzed the CPU and I/O system performance in a security-aware storage system. Security is another issue which has not been well explored in cluster computing systems. I implemented a transparent encryption/decryption layer in a popular Message Passing Interface implementation: MPICH2. Then I quantitatively evaluate the system performance on two cluster computing systems.

Acknowledgments

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

First and foremost, I am heartily thankful to my advisor, Dr. Xiao Qin, whose encouragement, guidance, and support from the preliminary to the concluding level enabled me to complete this Ph.D. dissertation. I have been working for Dr. Qin for almost five years since we met in 2006 when we were still in New Mexico Institute of Mining and Technology. Then I following him moved to Auburn University in 2007. Under his supervision, I learned how to do research and how to write technical papers from scratch. Without him, it would be impossible for me to finish this dissertation.

I would gratefully thank my dissertation committee members, Dr. David A. Umphress and Dr. Richard O. Chapman, who gave me valuable advices not only for this dissertation, but also my entire Ph.D. program in Auburn University. I took Software Engineering for Internet Applications lectured by Dr. Umphress in summer 2008. From the course I learned the knowledge related to the topic and also the way of being a good teacher. Dr. Umphress gave me a lot of advices for my research and job hunting. I really appreciate the help I gained from him in the past 4 years. I met Dr. Chapman in the Software for Space and Satellite Systems class from where I learned what cube satellites are and how to build one. Dr. Chapman brought the question about how to validate the storage system design. The question made me rethink about the experiments on our storage system.

I also gratefully thank Dr. Guofu Niu 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.

I have been working in a great research group.

I would like to thank the formal group members: Ziliang Zong, Kiranmai Bellam, and Adam Manzanares who have helped me a lot in my research and study. Ziliang helped me to know the process of writing a paper and how to organize data. Kiran and I worked on several papers together. And I have worked with Adam for about 3 years and collaborated in several projects supported by NSF.

I would also like to thank my current group members: Shu Yin, Zhiyang Ding, James Majors, Jiong Xie, Yun Tian, Yixian Yang, and Jianguo Lu. Working with them is beneficial and pleasant. I enjoyed my life in this research group every day. This group provides both happiness and research productivity which rarely coexist.

Furthermore, I gained help from other students also from the Department of Computer Science and Software Engineering. Hence, I would like to thank Qing Yang, Chengjun Wang, and Haiquan Chen for their help in my research.

My deepest gratitude goes to my parents Tongjun Ruan and Shumin Li for their selfless support.

Table of Contents

List of Figures

List of Tables

# Chapter 1

# Introduction

This dissertation addresses two major issues in cluster computing systems: energy efficiency and security. Cluster Computing systems cost a significant amount of energy every year. According to the information from Dell's Texas Data Center, 37% electricity is cost by storage systems (mainly hard drive disks) and 40% is cost by processors including CPUs and GPUs. Hence, I designed a Time Aware Dynamic Voltage Scaling algorithm to conserve energy in parallel applications and design a storage system called BUD to reduce energy cost in parallel storage systems. In the research, I observed that I/O is reduced to improve data confidentiality in security-aware storage systems. I analyzed the possibility of energy conservation in a security-aware storage system. Then I implemented two cryptographic algorithms in a Message Passing Interface implementation and evaluated the system performance downgrade.

In the past decade, cluster computing platforms have been deployed to support a variety of parallel computing applications. Scheduling parallel applications on large-scale clusters is technically challenging due to communication latencies and high energy consumption. As such, shortening schedule lengths and saving energy are two major concerns in the design of economical and environmentally friendly clusters. Although the existing dynamic voltage scaling technique (a.k.a., DVS) can be employed to reducing energy consumption of parallel applications running on clusters, DVS can inevitably lead to increased execution times of parallel tasks by lowering processor voltages. To solve this performance problem while improving energy efficiency of clusters, I propose in this dissertation a scheduling algorithm called TADVS to judiciously exploiting processor idle times among parallel tasks to provide energy savings for both high-performance clusters and mobile clusters. The TADVS

algorithm first aims to discover idle time intervals incurred by tasks precedence constraints. Then, TADVS applies DVS to lower voltages when processors are sitting idle due to the precedence constraints. Therefore, TADVS makes use of DVS to conserve energy provided that the schedule lengths of parallel applications are not increased. Experimental results clearly show that TADVS is capable of reducing energy dissipation in large-scale clusters without adversely affecting system performance.

Parallel disk systems have been developed to address the problem of I/O performance in the past years. A critical challenge with modern parallel I/O systems is that parallel disks consume a significant amount of energy in servers and high performance computers. To conserve energy consumption in parallel I/O systems, one can immediately spin down disks when disk are idle; however, spinning down disks might not be able to produce energy savings due to penalties of spinning operations. Unlike powering up CPUs, spinning down and up disks need physical movements. Therefore, energy savings provided by spinning down operations must offset energy penalties of the disk spinning operations. To substantially reduce the penalties incurred by disk spinning operations, I developed a novel approach to conserving energy of parallel I/O systems with write buffer disks, which are used to accumulate small writes using a log file system. Data sets buffered in the log file system can be transferred to target data disks in a batch way. Thus, buffer disks aim to serve a majority of incoming write requests, attempting to reduce the large number of disk spinning operations by keeping data disks in standby for long period times. Interestingly, the write buffer disks not only can achieve high energy efficiency in parallel I/O systems, but also can shorten response times of write requests. To evaluate the performance and energy efficiency of our parallel I/O systems with buffer disks, I implemented a prototype using a cluster storage system as a test bed. Experimental results show that under light and moderate I/O load, buffer disks can be employed to significantly reduce energy dissipation in parallel I/O systems without adverse impacts on I/O performance.

Cluster storage systems are essential building blocks for many high-end computing infrastructures. Although energy conservation techniques have been intensively studied in the context of clusters and disk arrays, improving energy efficiency of cluster storage systems remains an open issue. To address this problem, I describe in this dissertation an approach to implementing an energy-efficient cluster storage system or ECOS for short. ECOS relies on the architecture of cluster storage systems in which each I/O node manages multiple disks - one buffer disk and several data disks. Given an I/O node, the key idea behind ECOS is to redirect disk requests from data disks to the buffer disk. To balance I/O load among I/O nodes, ECOS might redirect requests from one I/O node into the others. Redirecting requests is a driving force of energy saving, and the reason is two-fold. First, ECOS makes an effort to keep buffer disks active while placing data disks into standby in a long time period to conserve energy. Second, ECOS reduces the number of disk spin downs/ups in I/O nodes. The idea of ECOS was implemented in a Linux cluster, where each I/O node contains one buffer disk and two data disks. Experimental results show that ECOS improves the energy efficiency of traditional cluster storage systems where buffer disks are not employed. Adding one extra buffer disk into each I/O node seemingly has negative impact on energy saving. Interestingly, our results indicate that ECOS equipped with extra buffer disks is more energy efficient than the same cluster storage system without the buffer disks. The implication of the experiments is that using existing data disks in I/O nodes to perform as buffer disks can achieve even higher energy efficiency.

Improving energy efficiency of security-aware storage systems is challenging, because security and energy efficiency are often two conflicting goals. The first step toward making the best tradeoffs between high security and energy efficiency is to profile encryption algorithms to decide if storage systems would be able to produce energy savings for security mechanisms. I was focused on encryption algorithms rather than other types of security services, because encryption algorithms are usually computation-intensive. In this study, I used the XySSL libraries and profiled operations of several test problems using Conky -

a lightweight system monitor that is highly configurable. Using our profiling techniques I concluded that although 3DES is much slower than AES encryption, it more likely to save energy in security-aware storage systems using 3DES than AES. The CPU is the bottleneck in 3DES, allowing us to take advantage of dynamic power management schemes to conserve energy at the disk level. After profiling several hash functions, I noticed that the CPU is not the bottleneck for any of these functions, indicating that it is difficult to leverage the dynamic power management technique to conserve energy of a single disk where hash functions are implemented for integrity checking.

An increasing number of commodity clusters are connected to each other by public networks, which have become a potential threat to security sensitive parallel applications running on the clusters. To address this security issue, I developed an MPI (Message Passing Interface) implementation to preserve confidentiality of messages communicated among nodes of clusters in an unsecured network. I focus on MPI rather than other protocols, because MPI is one of the most popular communication protocols for parallel computing on clusters. Our MPI implementation - called ES-MPICH2 - was built based on MPICH2 developed by the Argonne National Laboratory. Like MPICH2, ES-MPICH2 aims at supporting a large variety of computation and communication platforms like commodity clusters and high-speed networks. I integrated encryption and decryption algorithms into the MPICH2 library with the standard MPI interface and; thus, data confidentiality of MPI applications can be readily preserved without a need to change the source codes of the MPI applications. MPI-application programmers can fully configure any confidentiality services in MPICHI2, because a secured configuration file in ES-MPICH2 offers the programmers flexibility in choosing any cryptographic schemes and keys seamlessly incorporated in ES-MPICH2. I used the Sandia Micro Benchmark and Intel MPI Benchmark suites to evaluate and compare the performance of ES-MPICH2 with the original MPICH2 version. Our experiments show that overhead incurred by the confidentiality services in ESMPICH2 is marginal for small messages. The security overhead in ES-MPICH2 becomes more pronounced with

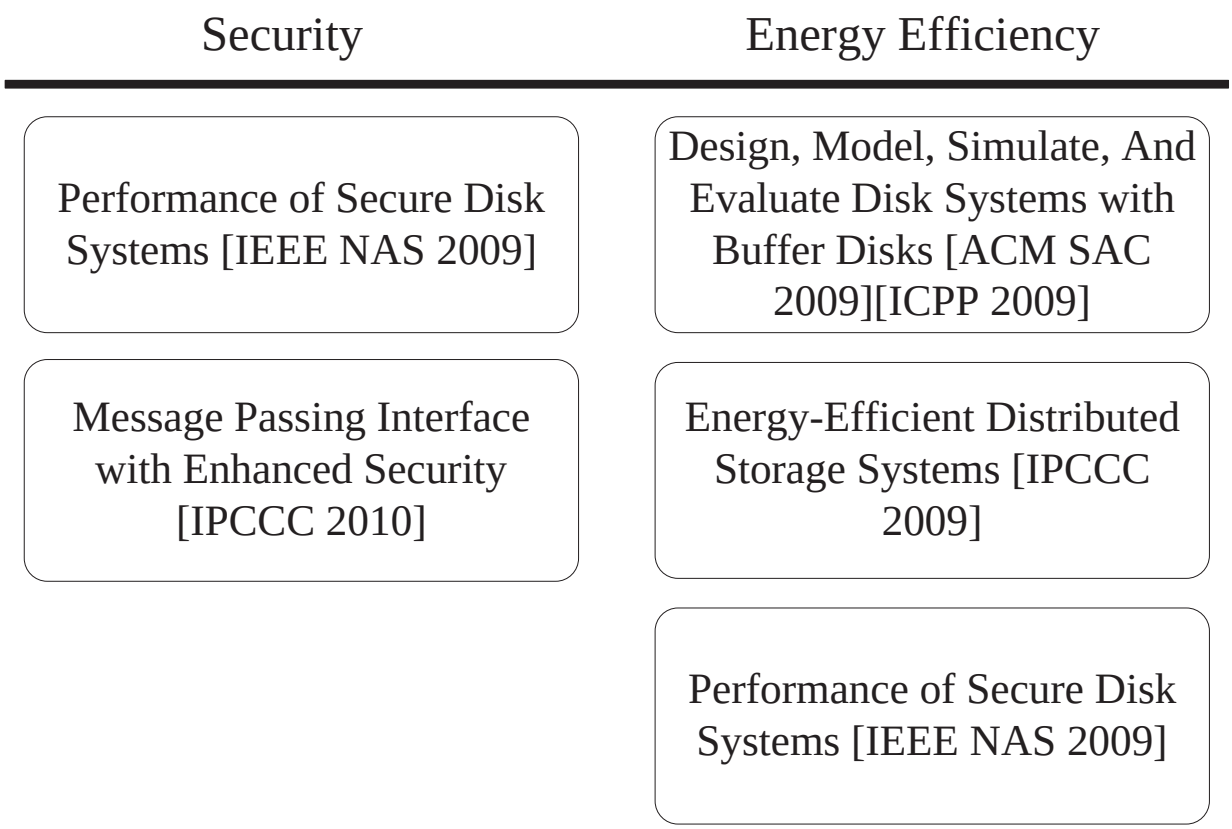


Figure 1.1: Contributions

larger messages. Our results also show that security overhead can be significantly reduced in ES-MPICH2 by high-performance clusters.

Fig. 1.1 presents the research covered in this dissertation. Both security and energy efficiency issues will be discussed in this dissertation. And the two issues are related to each other based on different performance impacts.

This dissertation is organized into the following chapters: Chapter 2 introduces the previous research related to this dissertation. Chapter 3 proposes a Time Aware Dynamic Voltage Scaling scheduling algorithm to conserve energy cost of processors in Cluster systems; Chapter 4 presents a design of an energy-efficient I/O system with write buffer disks; a clustered energy-efficient storage system prototype is implemented in Chapter 5; to explain

when the energy consumption could be reduced in I/O systems, I analyze the CPU and I/O system performance in a security-aware storage system in Chapter 6; Chapter 7 presents a Message Passing Interface with Enhanced Security; Chapter 8 presents conclusion and future work.

Chapter 2

Literature Review

In this chapter, I briefly summarize the previous literatures which are most relevant to the research in terms of energy efficiency and security in cluster computing systems. Section 2.1 introduces related work on energy efficient parallel scheduling; Section 2.2 presents the related work on energy efficient parallel storage systems; Section 2.3 introduces the research related to energy-efficient cluster storage systems; Section 2.4 presents the related work on security-aware cluster storage systems; Section 2.5 presents the related work on enhanced security MPICH2.

## 2.1 Related Work on Time-Aware Dynamic Voltage Scaling

A cluster is a group of computers, which are connected together to provide fast and reliable services. [52] There are a wide range of parallel applications developed to be running on cluster computing platforms. In the last decade, energy saving techniques has made it possible to develop energy-efficient cluster computing platforms. The issue of energy conservation for parallel application running on large-scale clusters has attracted little attention. Recently, researchers have started to pay attention to energy conservation techniques for clusters. For example, Kim et al. investigated the dynamic voltage scaling scheme (DVS). Doing so, they developed a novel algorithm called dynamic link shutdown (DLS), which makes use of an appropriate adaptive routing algorithm to shut down links in a judicious way [58]. Kim et al. proposed an optimized buffer design to reduce energy consumption in cluster interconnects [16]. Juan et al. designed a protocol called cluster-based Energy-Saving Routing Algorithm (CERA), which allows mobile computing nodes to autonomously create

clusters that minimize energy dissipation in mobile nodes based on the clusters layout in wired or wireless network [10].

Scheduling strategies deployed in clusters have a large impact on overall system performance. Scheduling algorithms for clusters can be generally classified into three camps: priority-based scheduling [10], group-based scheduling [94], and duplication-based scheduling. In a priority-based scheduling algorithm, tasks are assigned priorities before being mapped to the computing nodes of a cluster according to their assigned priorities. In contrast, group-based scheduling algorithms group tasks communicating frequently with one another into a set. This task set is allocated to a computing node, thereby eliminating communication overheads [82]. Duplication-based scheduling schemes exhibit high performance by redundantly executing tasks to eliminate communication overheads.

Very recently, I developed several duplication-based scheduling algorithms to reduce energy dissipation in cluster interconnects. Our scheduling approaches are especially beneficial for communication-intensive parallel applications [122].

The communication-to-computation ratio or CCR plays an important role in achieving high performance of clusters running parallel applications. The CCR of a parallel application is defined as the ratio between the average communication cost and the average computation cost of the application on a given cluster. Thus, CCR quantitatively expresses t he relationship between communication cost and computing time of a parallel application. It is worth noting that the computing times of parallel tasks largely depend on processor frequencies (see Eq. 1). Without loss of generality, in this study I use the highest processor frequency to determine the computing times of parallel tasks. Once the computing times are measured, I can quantify the CCR values of parallel applications.

There are some studies focusing on DVS techniques on applied in real-time systems [7] and high performance clusters. [36]

In the previous research, the DVS or Power Scalable technique is considered as one of the most efficient ways to conserve energy on cluster systems. [50][35] In CMOS circuits, by

completing the same quantity of CPU circles, the energy consumption of applying a lower voltage level in a longer time is much less than the energy consumption of applying a higher energy consumption in a short time [65]. Schmitz proposed an approach for energy efficient mapping and scheduling for DVS-enabled distributed embedded system [101]. However, for parallel applications in clusters, precedence-constraint issues must be addressed since in parallel applications could not start without their prerequisites are completed. Similarly, one task itself could also be the prerequisite of other tasks. Zhang proposed a similar idea, which applies DVS to reduce energy consumption by lowering down voltage levels. [117] Zhang's approach achieves a significant energy conservation rate at the cost of increased scheduling lengths. In a short length of tasks this is not a problem; however, in a huge cluster working on millions of tasks, this issue may lead to a significant performance reduction because each task has its corresponding prerequisites and it also is the others' prerequisites. TADVS will not affect the performance of the cluster system.

## 2.2 Related Work on Energy-Efficient Storage Systems

Energy conservation techniques for disk systems have attracted much attention in the past few years. For example, energy dissipation in disk I/O can be efficiently reduced by applying multi-speed disks as the power-state transition penalties are relatively small [28]. Song and Kandemir developed novel energy-aware compilers for multi-speed disks [106]. Although next-generation disks are likely having multiple speeds, most disks utilized today are non-multi-speed disks. It is expected that future generation multi-speed disks are more expensive than conventional disks. The energy conservation technique investigated in this study does not rely on multi-speed disks. Nevertheless, further energy savings can be achieved by integrating our approach with the above techniques based on multi-speed disks.

Modern disks make use of cache to substantially improve disk I/O performance [34]. Our storage system architecture use disks as I/O buffer. Compared with cache, disks are slower and less energy efficient. However, disks are very cost effective and could buffer much

more data than cache. Moreover, disks are non-volatile storage, meaning that once data is buffered on disks, it could be considered as safe even a power failure occurs. A research for non-volatile caches is done by Gill and Modha [38]; the research focused on single disk, RAID-10 and RAID-5. It is possible to expend the research to energy-aware parallel storage systems.

To improve parallel disk buffer management, Kallahalla and Varman leveraged a shared buffer to improve I/O performance [57]. Rangaswami *et al.* investigated a way of employing disks to buffer data for streaming media servers in order to bridge the widening performance gap between dynamic random access memory and disk drives in the memory hierarchy [95]. Goyal *et al.* explored the issue of quality of service in the context of storage system caches [39]. The fundamental difference between our research and the above three studies is that the goal of our approach is reducing energy consumption in parallel I/O systems.

If the data size of each request is so large that it is worth to spin up and spin down disks for each request, the traditional power management strategy is an efficient energy conservation technique. However, small and sequential data requests in modern scientific applications are very prevalent [47]. Moreover, small writes cause not only an energy consumption problem but also an efficiency problem [9]. Hence, it is imperative for us to develop an energy saving technique that is suitable to small writes issued to parallel I/O systems.

Please note that our approach can be readily applied to distributed network storage systems, where storage nodes are aggregated together into a larger cohesive storage system [20].

### 2.3 Related Work on Energy-Efficient Cluster Storage Systems

An array of techniques were developed to save energy in single disk systems storage systems. Energy consumption of single disks can be reduced at either I/O level (e.g., dynamic power management [27] [66] and multi-speed disks [43] [46] [60]) or operating system level (e.g., power-aware cache management strategies [118], power-aware prefetching schemes

[105]). Apart from energy-saving techniques at the levels of I/O and operating systems, energy efficiency can be optimized at the application level [107] [112]. For example, Weiel et al. developed an I/O semantics called Cooperative I/O for energy-aware applications. The design of our ECOS is orthogonal to the aforementioned schemes and; therefore, incorporating these techniques in ECOS can ultimately improve the energy efficiency of cluster storage systems.

Buffer management has been widely used to boost performance of parallel disk systems [1] [109]. Previous studies showed that data buffers significantly reduce the number of disk accesses in parallel disk systems [114]. More importantly, it is observed from the previous studies that traffic of small reads and writes becomes a performance bottleneck of disk systems, especially when RAM sizes for data buffers are increased rapidly [114]. It is expected that small disk requests dominate energy dissipation in cluster storage systems supporting data-intensive applications like remote-sensing applications and on-line transaction processing systems [64] [108]. Our approach differs itself from the traditional buffer management schemes in the sense that the goal of ECOS is to leverage buffer disks to reduce energy dissipation in cluster storage systems.

Colarelli and Grunwald proposed the Massive Array of Idle Disks (MAID) as a replacement for old tape backup archives with hundreds or even thousands of tapes [21]. It is observed that only a small part of the archive would be active at a time, the idea behind MAID is to copy the required data to a set of cache diskswhile placing all the other disks in the standby mode to conserve energy. I/O accesses to the archive may retrieve data from the cache disks rather than from standby disks. Pinheiro and Bianchini designed a Popular Data Concentration (PDC) scheme to reduce energy consumption in a network server by skewing I/O load toward a few of all the disks in the server [85]. The design of PDC is based on an observation that network server workloads in many cases exhibit files with widely different popularities (e.g., Web server workloads exhibit highly skewed popularity

towards a small set of files.). Taking into account the skewed data popularity, PDC dynamically migrates frequently accessed data to a subset of disks in a disk array. However, the bandwidth of disks limits the performance of PDC system. The reason is that PDC moves popular data to one disk which means the workload is extremely unbalanced. I observed from our experiments that data migration overhead is non-negligible in a highly dynamic I/O workloads [97] [74]. In addition, the popular data disk is has a strong likelihood to become I/O bottleneck. ECOS attempts to balance I/O load by evenly distributing popular data among multiple buffer disks. It is worth noting that both MAID and PDC are focused on energy efficiency issues in tightly-coupled parallel disks like disk arrays. Unlike MIAD and PDC, ECOS is an energy-efficient cluster storage system where I/O nodes are loosely connected to provide high aggregate I/O bandwidth.

Flash drives can be employed to buffer and cache popular data for I/O nodes in clusters [51] [56]. Flash-drive-based cluster storage systems are likely to be the most energy efficient storage systems for cluster computing infrastructures, because flash drives have a very low power consumption compared to hard drives. However, due to limited access times, the reliability of flash drives will have to be addressed when one integrates flash drives into cluster storage systems. A recent study shows that intensive and dynamic accesses for a long period of time can substantially shorten the lifetime of a flash drive [77].

## 2.4 Related Work on Security-Aware Storage Systems

Chandramouli et al. investigated battery power-aware Encryption algorithms [1]. The main conclusions they reached was that the power consumption changes linearly with the number of rounds of several popular cryptographic algorithms. Their experimental test bed had a laptop connected to a power supply. The power supply was connected to a computer running the Lab VIEW software to graph changes in voltage and current from the power supply. These changes were graphed during the life of the encryption algorithms [17]. Chandramouli et al. paid attention on improving energy efficiency of security mechanisms

in mobile devices systems [17]. Our research is radically different from theirs in the sense that I focused on energy-efficient disk systems without modifying existing security mechanisms. Potlapally et. al characterized the energy consumption of cryptographic algorithms and security protocols [88]. The work undertaken by Potlapally et. al [88] was very similar to the research project conducted by Chandramouli et al. [17] Potlapally et. al used IPAQ PDA's instead of using a laptop to measure power consumption [88]. They connected the IPAQ to a power supply that was connected to a computer running the Lab VIEW software. This allowed them to measure the energy differences between various cryptographic algorithms. Potlapally et. al obtained an array of interesting results related to the SSL Protocol processing. For example, they determined that for small data sizes asymmetric algorithms dominated symmetric algorithms in terms of energy consumption. For large data sizes symmetric algorithms consume significantly more energy than asymmetric algorithms. Network protocol energy consumption, defined as non-cryptographic processing necessary to establish the SSL protocol, does not vary much for different data sizes. Their main observations are that asymmetric algorithms consume the most energy with hash algorithms having the smallest energy footprint. Potlapally et. al also stated that asymmetric algorithms energy consumption is dependent on the key size used. They also determined that the level of security and energy consumption can be tuned using key size and number of rounds [88].

Our research differs from the aforementioned research because I focused on the energy impact of encryption algorithms and hash functions on disk systems rather than mobile devices. Furthermore, the goal of our work is to characterize the I/O behavior of encryption algorithms and hash functions in the context of parallel disk systems (e.g., the BUD architecture). I intended to find the existing security mechanisms that produce the most energy savings in an energy-efficient parallel disk system. Rather than implementing energy-efficient security mechanisms by updating the existing security services, I made the first step towards seamlessly integrating security services with energy-efficient disk systems without changing

the source code of the existing security mechanisms. More importantly, our research is orthogonal to the above work in that energy-efficient security mechanisms can be incorporated into our energy-efficient parallel disk architecture to achieve both high security and energy efficiency for parallel disk systems.

## 2.5 Related Work on Enhanced Security MPICH2

**Message Passing Interface.** The Message Passing Interface standard (MPI) is a message passing library standard used for the development of message-passing parallel programs [42]. The goal of MPI is to facilitate an efficient, portable, and flexible standard for parallel programs using message passing. MPICH2 - developed by the Argonne National Laboratory - is one of the most popular and widely deployed MPI implementations in cluster computing environments. MPICH2 provides an implementation of the MPI standard while supporting a large variety of computation and communication platforms like commodity clusters, high-performance computing systems, and high-speed networks [41].

As early as 1997, Brightwell *et al.* from the Sandia National Laboratory insightfully pointed out barriers to creating a secure MPI framework [14]. The barriers include control and data in addition to cryptographic issues. In a secure MPI, both control and data messages must be protected from unauthorized access of attackers and malicious users. Although there is a wide range of implementations of the MPI and MPI-2 standards (e.g., MPICH and MPICH2 are two freely -available implementations from the Argonne National Laboratory), there is a lack of secure MPI frameworks developed for large-scale clusters distributed across wide area networks.

**Data confidentiality in MPI-I/O.** Prabhakar *et al.* designed and implemented a secure interface called MPISec I/O for the MPI-I/O framework [90]. MPISec I/O preserves the advantages of both parallel I/O and data confidentiality without significantly impacting performance of MPI applications. It is flexible for MPI programmers to manually set encryption rules in MPISec I/O. Data can be encrypted and written onto disks in MPISec I/O, then

encrypted data can be read from the disks before being decrypted. There are two interesting features of MPISec I/O. First, MPISec I/O programmers need to carefully set up encryption and decryption rules in their MPI programs. Otherwise, some data may be either stored on disks without encryption or read without decryption and as a result, the MPI programs are unable to function properly until the rules are set in a correct way. Second, MPISec is not completely compatible with non-secure MPI libraries. In other words, preserving data confidentiality in MPISec I/O is not transparent to MPI application programmers. One has to modify the source code of conventional MPI programs to improve security of the MPI programs. Apart from updating the source code of the MPI programs before MPISec I/O can be used properly, disk-resident data must be marked as encrypted or unencrypted.

**Block ciphers.** The Data Encryption Standard (DES) provides a relatively simple method of encryption. 3DES encrypts data three times instead of one using the DES standard [22]. 3DES is a block and symmetric cipher chosen by the U.S. National Bureau of Standards as an official Federal Information Processing Standard in 1976. 3DES increases the key size of DES to protect against brute force attacks without relying on any new block cipher algorithm. A hardware implementation of 3DES is significantly faster than the best software implementations of 3DES [30] [45]. A software implementation of 3DES was integrated in ES-MPICH2. A hardware 3DES can substantially improve performance of 3DES-based ES-MPICH2.

In November 2001, the symmetric block cipher Rijndael was standardized by the National Institute of Standards and Technology as the Advanced Encryption Standard (AES) [24]. AES - the successor of the Data Encryption Standard (DES) - has been widely employed to prevent confidential data from being disclosed by unauthorized attackers. AES can be used in high-performance servers as well as small and mobile consumer products. AES is the preferred cryptographic algorithm to be implemented in ES-MPICH2, which was built based on symmetric block ciphers. Although AES introduces overhead due to additional security operations in ES-MPICH2, the overhead caused by AES in ES-MPICH2 can be significantly

reduced by AES hardware architectures (see [72] for details of a highly regular and scalable AES hardware architecture).

**Security enhancement in clusters.** There are several research works focusing on security enhancement in commodity clusters. For example, Lee and Kim developed a security framework in the InfiniBand architecture (IBA) [63]. For confidentiality and authentication, Lee and Kim proposed the partition-level and QP-level secret key management schemes. The security in IBA is improved with minor modifications to the IBA specification. Ramsurrun and Soyjaudah constructed a highly available transparent Linux cluster security model, which offers a new approach to enhancing cluster security [93]. Koenig *et. al.* implemented a tool that monitors processes across computing nodes in a cluster [59]. The tool delivers real-time alerts when there are immediate threats. Similarly, Pourzandi *et al.* investigated the security issues of detecting threats and hazards in distributed clusters [89]. The aforementioned security solutions developed for clusters are inadequate to directly support security-sensitive MPI programs, because the existing security solutions generally require application developers to implement security functionality in their MPI programs.

## 2.6 Summary

The objective of this dissertation is to present the solutions to reduce energy consumption and to improve security in cluster computing systems. This chapter overviewed a variety of existing techniques related to dynamic voltage scaling, energy-efficient storage systems, security-aware storage system, and enhanced security MPI. The strategies presented in this dissertation have been modelled, simulated, or real implemented on a cluster computing platform. The experimental results have been quantitatively evaluated.

# Chapter 3

# Scheduling Parallel Applications on Dynamic Voltage Scaling-Enabled Clusters

## 3.1 Introduction

With the advancement of computers and networks, large-scale clusters have been widely applied to support scientific and commercial applications, among which many happen to be parallel applications. Examples of this type of parallel applications include, but are not limited to, 3-D perspective rendering [49], molecular dynamics simulation [44], quantum chemical reaction dynamics simulations [33], and 2-D fluid flow using the vortex [100]. In the past decade, most research work has focused on achieving high performance of clusters. Scheduling parallel applications on large-scale clusters is technically challenging due to significant communication latencies and high energy consumption rates of clusters. Reducing schedule lengths and conserving energy are two important goals in the design of high-performance and energy-efficient cluster computing platforms.

### 3.1.1 Motivation

One effective approach to saving energy consumption in clusters is to make use of cutting-edge energy-efficient processors, because processors in computing nodes are consumers of large amounts of energy. Among the various energy conservation techniques for processors, the dynamic voltage scaling scheme or DVS is one of the more attractive ways to provide significant energy savings [15]. DVS has been implemented in modern processors like AMD's Mobile Athlon [53] and Intel's XScale [54]. The basic idea behind the DVS technique is to dynamically reduce a processor's supply voltage while guaranteeing proper

operations. Significant energy savings can be achieved by DVS, because the power dissipation in a processor is proportional to the product of total circuit capacity c, supply voltage V, and system clock frequency f. Thus, the energy dissipation P can be expressed as:

$$P = c \cdot V^2 \cdot f \tag{3.1}$$

Let us consider tasks to be executed on a processor during the time interval [t1, t2]. Suppose P(t) is the power dissipation at time $t \in [t1, t2]$, I can express the total energy consumption $E$ in the processor as

$$E = \int_{t_1}^{t_2} P(t) \cdot dt \tag{3.2}$$

A variety of DVS-based scheduling algorithms have been developed for real-time systems (see, for example [84] and [8]). Existing DVS-based real-time scheduling algorithms dynamically determine when and to which level operating voltages must be scaled. The DVS-based schedulers aim to conserve energy by running real-time systems at the lowest possible voltage provided that deadlines of real-time tasks can be guaranteed. The existing DVS-based real-time scheduling algorithms - proven to be effective in achieving substantial energy savings (from 1.4% to as high as 90% [115]) - are normally implemented in real-time operating systems. Although DVS is a promising and powerful technique providing significant energy savings in real-time systems without sacrificing real-time performance, the energy savings achieved by DVS come at the cost of reduced performance in non-real-time systems. More importantly, performance degradation incurred by DVS is unacceptable for parallel applications running on high-end computing systems like clusters. Conserving energy in clusters at the cost of decreased performance is an undesirable approach, because improving performance of parallel applications is the primary goal of high-performance cluster design. This research is motivated by a challenging issue - can we reduce energy dissipation in DVS-enabled clusters without degrading performance of parallel applications? This Chapter is

the first step towards tackling this problem. I seek to answer fundamental questions such as how to discover idle times among a group of parallel tasks, what idle times can be utilized by DVS, and how to determine the supply voltage of processors in computing nodes of a cluster without affecting starting times of parallel tasks. This research is also motivated by an unrealistic assumption (i.e., supply voltages can be changed simultaneously with no physical constraints) commonly used in theoretical studies [48][19]. To implement an energy-efficient scheduler for parallel applications running on practical DVS-enabled clusters, I focus on multiple-voltage-based clusters, where only a number of predesigned operating voltages are readily available in computing nodes of the clusters. Such DVS-enabled clusters only allow the computing nodes to choose any one of these voltages and clock frequency at run time. It is worth nothing that we are unable to realize the full energy-saving potential of DVS in the context of practical multiple-voltage-based clusters.

### 3.1.2 Contributions and Paper Organization

The issue of applying the dynamic voltage scaling technique or DVS to parallel applications running on large-scale clusters has not been well addressed, mainly because the primary goal of building clusters is to achieve high performance rather than energy efficiency for parallel applications. However, the lack of energy conservation techniques for clusters supporting parallel applications is a critical problem. Without energy saving techniques for parallel applications with precedence constraints, the development of energy-efficient clusters seems unlikely.

In this Chapter, I address the issue of conserving energy consumption in DVS-enabled clusters without sacrificing the performance of parallel applications running on the clusters. The major contributions of this study are:

•An energy-efficient scheduling algorithm (TADVS). Existing DVS-based scheduling algorithms designed for real-time systems are inadequate for clusters, because the existing DVS algorithms are incapable of improving energy efficiency of clusters without degrading

system performance. Our TADVS aims at (1) exploiting idle times among parallel tasks with precedence constraints, (2) choosing idle intervals that can be used by DVS to conserve energy, and (3) determining the supply voltage of processors without having negative impacts on completion times of parallel applications. I also quantitatively evaluate energy savings provided by the TADVS algorithm.

•Energy consumption modeling. I developed an energy consumption model for multiple-voltage-based computing nodes in modern clusters. This model relaxes the unrealistic assumption - supply voltages in computing nodes can be simultaneously adjusted with no physical constraints. The model can be widely employed to estimate energy dissipation in a cluster's computing nodes where a number of predesigned operating voltages are physically available.

•Second these DVS-enabled systems are simulated in a secure wireless communication network. The results demonstrate that: 1) all DVS systems are effective in energy saving over the fixed voltage system; 2) multiple, pessimistic feasible, optimistic feasible, and ideal DVS systems, in that order, become more energy efficient as the physical constraints on how voltage may vary are relaxed; and 3) optimistic and pessimistic feasible DVS systems only consume a little more energy than the ideal DVS system, which suggests that the full potential of DVS in energy saving can (almost) be reached.

The rest of the Chapter is organized as follows. Section 3.2 describes a way to measure energy dissipation in a cluster running parallel applications. Section 3.3 presents the TADVS scheduling algorithm using the DVS technique. In Section 3.4, I evaluate the performance of the proposed scheduling algorithm by comparing it with a baseline scheduling algorithm where the DVS technique is not employed. Finally, Section 3.5 concludes the research with future research opportunities.

## 3.2 Modeling Energy Consumption

In this section, I model the energy dissipation of a cluster running a parallel application represented by a *DirectedAcyclicGraph* (DAG) [120][92]. Throughout this Chapter, a DAG is denoted by a pair $(U, E)$. $U = \{u1, u2, ..., un\}$ represents a set of precedence constrained parallel tasks, and $t_i$ is the $i$th task's computation requirement showing the number of time units to compute $v_i$, $1 \leq$ i $\leq$ n. It is assumed that all the tasks in U are non-preemptive and are indivisible work units [122].

Let $en_i$ be the energy consumption caused by task $v_i$ running on a computing node. The computing node energy consumption rate is $P = c \cdot V^2 \cdot f$ (see Eq. 3.1), and the energy dissipation of task $v_i$ can be expressed by Eq. 3.2. Note that $t_i$ in Eq. 3.3 depends on frequency $f_i$. In order to avoid confusion of $V$, in all following equations, $U$ refers to set $U$; $V_i$ refers to the voltage for the $i$th task; $u_i$ refers to the $i$th task in the precedence constrained parallel tasks' set $U$. $n$ is for the number of tasks in set $U$, and $m$ is for the number of all computing nodes in the cluster system.

$$en_i = c \cdot V_i^2 \cdot f_i \cdot t_i \tag{3.3}$$

Given a parallel application with a task set $U$ and allocation matrix $X$, we can calculate the total energy consumed by all the tasks in $U$ using Eq. 3.4, where $n$ is the number of tasks, $m$ is the number of computing nodes in the cluster system.

$$\begin{aligned} EN_{active} = \sum_{i=1}^{|U|} en_i \ &= \sum_{i=1}^{n} (c \cdot V_i^2 \cdot f_i \cdot t_i) \\ &= c \cdot \sum_{i=1}^{n} (V_i^2 \cdot f_i \cdot t_i) \end{aligned} \tag{3.4}$$

Let $PN_{idle}$ be the energy consumption rate of a computational node when it is inactive, and $\phi_i$ be the completion time of task $u_i$. The energy consumed by an inactive node is a

product of the idle energy consumption rate $PN_{idle}$ and an idle period. Thus, we can make use of Eq. 3.5 to measure the energy consumed by the jth computing node in a cluster when the node is sitting idle.

$$\begin{aligned} EN_{active} = \sum_{i=1}^{|U|} en_i &= \sum_{i=1}^{n} (c \cdot V_i^2 \cdot f_i \cdot t_i) \\ &= c \cdot \sum_{i=1}^{n} (V_i^2 \cdot f_i \cdot t_i) \end{aligned} \tag{3.5}$$

where $\max_{i=1}^{n} (\varphi_i)$ is the schedule length, and $\max_{i=1}^{n} (\varphi_i) - \sum_{i=1}^{n} x_{ij} \cdot t_i$ is the total idle time on the $j$th node. $\max_{i=1}^{n} (\varphi_i)$ represents the completion time of the task that is completed last. Thus, is the completion time of tasks in set $U$. Thus $\sum_{i=1}^{n} (x_{ij} \cdot t_i)$ computes the run time summation of all tasks allocated to the $j$th computing node. $x_{ij}$ is set to 1 if task $v_i$ is allocation to the $j$th computing node; otherwise, $x_{ij}$ is set to 0. The total idle energy consumption of the cluster is written as

$$\begin{aligned} EN_{idle} &= \sum_{j=1}^{m} en_{idle}^{j} \\ &= PN_{idle} \cdot \sum_{j=1}^{m} \left( \max_{i=1}^{n} (\varphi_i) - \sum_{i=1}^{n} (x_{ij} \cdot t_i) \right) \\ &= PN_{idle} \cdot \left( m \cdot \max_{i=1}^{n} (\varphi_i) - \sum_{j=1}^{m} \sum_{i=1}^{n} (x_{ij} \cdot t_i) \right). \end{aligned} \tag{3.6}$$

The total energy consumption $EN$ incurred by the computing nodes of the cluster running the parallel application is the summation of the nodes' energy when the nodes are active and idle. Consequently, energy consumption $EN$ can be derived from Eqs. 3.4 and 3.6 as

$$EN = EN_{active} + EN_{idle} \tag{3.7}$$

We denote $el_{ij}$ as the energy consumed by the transmission of message $(ti, tj) \in E$. We can compute the energy consumption of the message as below

$$el_{ij} = PL_{active} \cdot w_{ij} \tag{3.8}$$

where $PL_{active}$ is the power of the link when it is active, and $w_{ij}$ is the transmission time. In this study, I assume that cluster interconnects are homogeneous. All messages are transmitted over the interconnection network at the same transmission rate. The energy consumed by a network link between $p_a$ and $p_b$ is the cumulative energy consumption total caused by all messages transmitted over the link. Then, the link's energy consumption is obtained as follows, where $L_{ab}$ is a set of messages delivered on the link, and Lab can be written as $L_{ab} = \{\forall(u_i, u_j) \in E,\ 1 \leq a,\ b \leq m \mid x_{ia} = 1 \wedge x_{jb} = 1\}$.

$$\begin{aligned} EL^{ab}_{active} &= \sum_{(u_i,u_j)\in L_{ab}} el_{ij} \ = \sum_{(u_i,u_j)\in L_{ab}} (PL_{active} \cdot w_{ij}) \\ &= \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} (x_{ia} \cdot x_{jb} \cdot PL_{active} \cdot w_{ij}) \end{aligned} \tag{3.9}$$

The energy consumption of the whole interconnection network is expressed by Eq. 3.10, which is the summation of all the links' energy consumption. Thus, we have

$$\begin{aligned} EL_{active} &= \sum_{a=1}^{m} \sum_{b=1, b\neq a}^{m} EL^{ab}_{active} \\ &= \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \sum_{a=1}^{m} \sum_{b=1, b\neq a}^{m} (x_{ia} \cdot x_{jb} \cdot PL_{active} \cdot w_{ij}) \,. \end{aligned} \tag{3.10}$$

We can express the energy consumed by a link when it is inactive as a product of the consumption rate and the idle period of the link.

$$EL_{idle}^{ab} = PL_{idle} \cdot \left( \max_{i}^{n} (\varphi_i) - \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} (x_{ia} \cdot x_{jb} \cdot w_{ij}) \right) \tag{3.11}$$

where $PL_{idle}$ is the power of the link when it is inactive, and $\max_{i}^{n} (\varphi_i) - \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} (x_{ia} \cdot x_{jb} \cdot w_{ij})$ is the total idle time of link. The energy incurred by the whole interconnection network during the idle periods is expressed by

$$EL_{idle} = \sum_{a=1}^{m} \sum_{b=1, b \neq a}^{m} EL_{idle}^{ab} \tag{3.12}$$

The total energy consumption $EL$ exhibited by the cluster interconnect can be straightforwardly derived from Eqs. 3.9 and 3.11. Hence, we have

$$EL = EL_{active} + EL_{idle} \tag{3.13}$$

The energy dissipation $E$ of a cluster running a parallel application is the summation of the energy caused by computing nodes and interconnects. Therefore, energy $E$ can be derived from Eqs. 3.7 and 3.13. Thus, we have

$$EL = EL_{active} + EL_{idle} \tag{3.14}$$

Furthermore, in the experimental result part, we are going to use one parameter called CCR which is mentioned in Section 3.2 also. CCR is the acronym for the Communication-to-Computation Ratio. The communication-to-computation ratio or CCR of a parallel application is defined as the ratio between the average communication cost of the application and the average computation cost on a given cluster. Formally, the CCR of an application $(V, E)$ is given by the following Eq. 3.15. [121]

$$CCR(U,E) = \frac{\frac{1}{|E|}\sum_{(u_i,u_j)\in E} c_{ij}}{\frac{1}{|U|}\sum_{i=1}^{|U|} t_i} \tag{3.15}$$

### 3.3 Voltage Scheduling for Parallel Applications

Now I present the energy-efficient TADVS scheduling algorithm. Due to precedence constraints, parallel tasks are unable to start execution until their parent tasks have been completed. In each computing node of a cluster, there is a high probability that there are some idle time intervals among tasks allocated to this node. In other words, after having finished executing a task, a computing node may not immediately start the execution of the next task. The idle period is caused by precedence constraints forcing the computing node to sit idle for a period of time. The basic premise of the TADVS scheduling algorithm is to exploit idle processor time intervals in each computing node of a cluster. This makes it possible to leverage idle time intervals to dynamically reduce the supply voltage of the computing node. This allows us to substantially conserve energy. More specifically, computing nodes are not supposed to be running with the highest frequency when the nodes are sitting idle. Given a set of idle processor time intervals, it is unnecessary for the computing nodes to process all allocated parallel task as soon as possible using the highest frequency. Rather, processor frequencies can be judiciously lowered during idle periods, provided that the overall schedule lengths of parallel applications are not adversely affected.

Let us first introduce three important parameters to be used in TADVS. The first parameter for each parallel task is the earliest start time (EST). EST of an entry task is 0. The EST of all the other tasks can be calculated in a top-down manner by recursively applying the second term on the right side of Eq. 3.16.

$$EST(u_i) = \begin{cases} 0, \text{ if } \forall 1 \leq j \leq n : (u_j, u_i) \notin E \\ \min\limits_{(u_j,u_i)\in E} \left( \max\limits_{(u_k,u_i)\in E, u_k \neq u_j} (ECT(u_j), ECT(u_k) + w_{ki}) \right), \text{ otherwise} \end{cases} \tag{3.16}$$

where $ECT(u_i)$ is the second important parameter, which quantifies the earliest completion time of task $u_i$. $ECT(u_i)$ can be derived from Eq. 3.17 as the summation of its earliest start time and execution time.

$$ECT(u_i) = EST(u_i) + t_i \tag{3.17}$$

The third parameter $LACT(u_i)$ is the latest allowable completion times of all the other tasks. $LACT(u_i)$ can be calculated in a top-down manner by recursively applying the second term on the right side of Eq. 3.18.

$$LACT(u_i) = \begin{cases} ECT(u_i), \text{ if } \forall 1 \leq j \leq n : (u_i, u_j) \notin E \\ \min \left( \min\limits_{\substack{(u_i,u_j)\in E \\ u_i \neq FP(u_j)}} (LAST(u_j) - w_{ij}), \min\limits_{\substack{(u_i,u_j)\in E \\ u_i = FP(u_j)}} (LAST(u_j)) \right), \text{ otherwise} \end{cases} \tag{3.18}$$

We can make use of these three parameters to dynamically choose processor frequency and supply voltage for each computing node. Note that the execution time of a task lies in its complexity and the corresponding processor frequency. The processor frequency determined on the fly can be derived from the aforementioned three parameters. Given a task $v_i$, its shortest execution time $t_i^{min}$ is the time interval between $ECT(u_i)$ and $EST(u_i)$. Task $u_i$ will experience the shortest execution time if its computing node is processing the task with the highest frequency. In contrast, the execution time of task $v_i$ can be increased up to $t_i^{max} = LACT(u_i) - EST(u_i)$ if the processor frequency is dynamically reduced with respect to task $u_i$. Now our goal is to choose the processor frequency that can lead to the

maximum execution time $t_i^{max}$ without adversely affecting system performance. Determining the optimal frequency is a challenge because processors frequencies are not continuously tunable. Task $u_i$ must be accomplished before $LACT(u_i)$ therefore, the frequency $f_i$ must be higher than or equal to $f_{best}$. As such, the most appropriate frequency can be determined by Eq. 3.19.

$$f_{best} = \frac{f_{Highest}(ECT - EST)}{LACT - EST} \tag{3.19}$$

Eq. 3.20 measures the energy savings gained by TADVS for a given parallel application.

$$Power\ Rate = \frac{\sum_{i=1}^{n} cV_i^2 f_i (LACT_i - EST_i)\frac{f_{best\_i}}{f_i}}{\sum_{i=1}^{n} c(V_{Highest})^2 f_{Highest}(ECT_i - EST_i)} \tag{3.20}$$

Given processor power $P_i$ at different supply voltages, we can substitute $P_i$ and $P_{Highest}$ for $cV_i^2 f_i$ and $c(V_{Highest})^2 f_{Highest}$ respectively, in Eq. 3.20 to further simply Eq. 3.20. Thus, we have

$$Power\ Rate = \frac{\sum_{i=1}^{n} P_i (LACT_i - EST_i)\frac{f_{best\_i}}{f_i}}{\sum_{i=1}^{n} P_{Highest}(ECT_i - EST_i)} \tag{3.21}$$

Given the maximum power, the highest frequency, and voltage of processors in computing nodes, we can readily derive the constant $c$ (i.e., the capacity) using Eq. 3.1. Provided supply voltages and processor frequencies, we can use the constant $c$ and Eq. 3.1 compute energy consumption rate $P_i$. Now we calculate processing time of the computing nodes using Eq. 3.22 as follows.

$$Running\ Time_{TADVS} = \sum_{i=1}^{n} (LACT - EST)\frac{f_{best\_i}}{f_i} \tag{3.22}$$

It is insufficient to demonstrate cluster energy savings only using processors. We also consider energy dissipation in network interconnects. Eq. 3.23 below quantifies the overall energy savings achieved by our TADVS compared with a non-TADVS-based scheme.

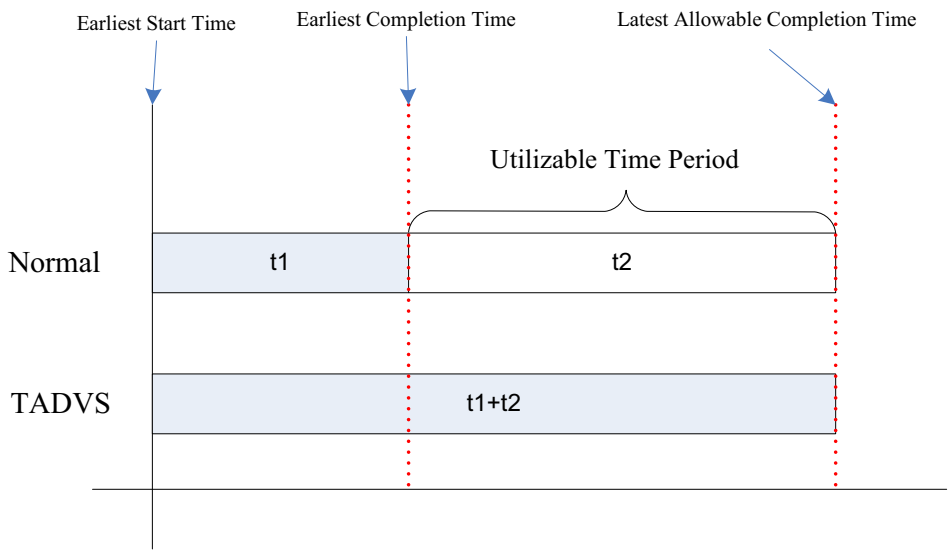


Figure 3.1: Utilizable Time Period

$$TPCSR = \frac{\sum_{i=1}^{n} P_i(LACT_i - EST_i)\frac{f_{best\_i}}{f_i} + w + CPU_{idle} * (t_{all} - running_{Non-TADVS})}{\sum_{i=1}^{n} P_{Highest}(ECT_i - EST_i) + w + \sum_{j=1}^{m} EN_{idle}^{j}} \tag{3.23}$$

where $t_{all}$ is $\sum_{j=1}^{m}(EN_{idle}^{j})$, $w$ represents communication energy consumption. Since our algorithm does not affect the data size and transmitting speed, the transmitting energy consumption in our algorithm is as same as the transmitting energy consumption before.

After introducing the algorithm, let us to prove the correctness of this strategy. The strategy of TADVS is illustrated in Fig. 3.1. In existing algorithm, if one task could be completed as quickly as in $t_1$, CPU will use the highest working frequency to finish the task, and then turn off CPU in $t_2$ in order to conserve energy consumption. Although this is a very straightforward way to conserve energy, there is a better way to do this. According to the definition of LACT, the CPU will not reduce the performance of entire cluster system

if it could complete the current task before LACT. In another word, we can utilize $t_2$ to increase working time and choose a lower frequency and voltage level to conserve energy. This strategy is not very straightforward; hence, I am going to prove its correctness in Eq. 3.23 for each task.

Table 3.1: Symbolic Notations

| System Parameters | Values |
|---|---|
| $W_{N,T}$ | Energy consumption by normal way for T |
| $W_{TA,T}$ | Energy Consumption by TADVS for T |
| $c$ | Capacity of CPU, constant value |
| $V_h$ | Highest working voltage of CPU |
| $V_l$ | lower working voltage TADVS chose |
| $f_h$ | corresponding working frequency of $v_h$ |
| $f_a$ | frequency level we finally apply |
| $f_l$ | corresponding working frequency of $v_l$ |
| $t_1$ | Shortest working time period |
| $t_2$ | extendable time period |
| $T$ | Task No. |
| $L_T$ | Work load of the task T |
| $EST$ | Earliest Start Time |
| $ECT$ | Earliest Completion Time |
| $LACT$ | Latest Allowable Completion Time |

The following two theorems illustrate that our strategy will reduce the energy consumption when there is a gap, which is t2 in Fig. 3.1, and equals to Latest Allowable Completion time minus Earliest Completion Time.

**Theorem 3.1.** *For a task $U_i$ on a processor, strategy 1 is that the processor choose the highest frequency to complete the task $U_i$ as quickly as possible, then turn the processor off until LACT; strategy 2 is that the processor choose a lower voltage level whose corresponding frequency can let the processor complete $U_i$ just before LACT. The energy consumption of strategy 2 is always less than the energy consumption of strategy 1 if $t_i$ is not 0.*

*Proof.* Given task $U_i$, $t_1$–the shortest period of working time, or (Earliest Completion Time - Earliest Start Time); $t_2$–extendable period of time, or (Latest Allowable Completion Time - Earliest Start Time).

In order to prove that the energy consumption $W_{TA}$ of TADVS is always smaller than the energy consumption $W_N$ of normal strategy, we need to prove the quotient of $W_{TA}$ and $W_N$ is always smaller than 1. Energy consumption is the product of Power and time. Hence, the proof starts from q. 3.24 (24).

$$\frac{W_{TA}}{W_N} = \frac{P_l(t_1+t_2)}{P_h t_1} \tag{3.24}$$

According to Eq. 3.1, the power of CPU can be computed by capacity, frequency and voltage, so I come up with Eq. 3.25.

$$\frac{P_l(t_1+t_2)}{P_h t_1} = \frac{cV_l^2 f_l(t_1+t_2)}{cV_h^2 f_h t_1} \tag{3.25}$$

Because the workload for different strategy is the same, the necessary CPU cycles are the same. So we can get Eq. 3.26 and get Eq. 3.27 furthermore.

$$f_h t_1 = f_l(t_1+t_2) \tag{3.26}$$

$$f_l = \frac{t_1}{(t_1+t_2)} f_h \tag{3.27}$$

Use Eq. (27) to replace f in Eq. (25) and then get Eq. (28).

$$\frac{cV_l^2 f_l(t_1+t_2)}{cV_h^2 f_h t_1} = \frac{cV_l^2 f_h \left(\frac{t_1}{t_1+t_2}\right)(t_1+t_2)}{cV_h^2 f_h t_1} \tag{3.28}$$

Eliminate the same elements in right part of Eq. 3.28 and get Eq. 3.29.

$$\frac{cV_l^2 f_h \left(\frac{t_1}{t_1+t_2}\right)(t_1+t_2)}{cV_h^2 f_h t_1} = \left(\frac{V_l}{V_h}\right)^2 \tag{3.29}$$

Hence, the ratio of energy consumption between strategy 1 and strategy 2 only depends on the voltage levels of the two strategies. Since $V_l$ is defined as the lower voltage level, $V_l < V_h$.

Therefore, $\frac{W_{TA}}{W_N} = \left(\frac{V_l}{V_h}\right)^2 < 1$. So finally we obtain that $W_{TA} < W_A$ which completes the proof of Theorem 1. □

In Theorem 1, I proved that for each task, once it has extra time space between ECT and LACT, we could conserve energy. Since each task has its own ECT, EST, and LACT, we could promote Theorem 1 from one task case to all tasks. Then we need to prove Theorem 2.

**Theorem 3.2.** *if* $\exists t_i \in (LACT_i - ECT_i)$ *and* $t_i > 0$*, then applying strategy 2 consumes less energy than applying strategy 2 to complete all* $v_i$ *in set U.*

*Proof.* Since $\forall u_i \in (U, E)$, $W_{TA,u} < W_{N,u}$, and $\exists t_i \in (LACT_i - ECT_i)$. therefore,

$$\left( \frac{\sum_{i=1}^{n} W_{TA,u_i}}{\sum_{i=1}^{n} W_{N,u_i}} \right) < 1 \tag{3.30}$$

□

Hence, for all task in set $U$, total energy consumed by strategy 2 to complete all tasks in cluster system is less than the energy consumed by strategy 1.

Therefore, energy consumption could be reduced for all tasks the cluster needs to work on if there is a free time between ECT and LACT. This is not a problem since in parallel cluster system there are plenty of free time period between ECT and LACT because of the restrains of prerequisite conditions of each task. Furthermore, since LACT($u_i$) is defined as latest allowable completion time which means if task $u_i$ can be completed before LACT($u_i$) the completion time of set $U$ will not be affect, the performance of cluster system is not going to be reduced by applying TADVS because TADVS requires all tasks must be completed before LACT($u_i$). For a single $u_i$, the completion time might be delayed. Hence, how much energy does TADVS could conserve for one task $u_i$ depends on how much free time we have between LACT($u_i$) and ECT($u_i$), but no matter how much energy we could save or there is

even no energy could be conserved at all, the performance of the cluster for entire set $U$ will not be affected at all.

The following algorithm represents the preferred CPU voltage decision process. Earliest Start Time, Earliest Completion Time and Latest Allowable Time are assumed as known or predicted before the frequency level decision process. If Latest Allowable Completion Time is later than Earliest Completion time, we could utilize the time difference between them to conserve energy consumption. Then Eq. 3.19 will be applied to compute $f_{best}$ which is a theoretically optimized CPU frequency level which can fit the time gap between LACT and ECT perfectly. However, CPUs only have fixed number of frequency levels; hence, we need to choose the frequency level which is just higher than $f_{best}$. In this case, the real completion time of task after scaling will be earlier than LACT, and it could equals ECT if the gap is too short for CPU to scaling.

**Algorithm 1** Task Allocating Dynamic Voltage

```
for Each Task i in set V do
  Collect LACT(v_i), ECT(v_i) and EST(v_i)
  if (LACT(v_i) − ECT(v_i)) ≥ 0 then
    use Eq. (19) to compute f_best
    choose the CPU frequency level just higher than f_best
    f_a = the frequency level just higher than f_best
  else
    apply the highest working frequency
    f_a = f_highest
  end if
end for
```

### 3.4 Performance Evaluation and Simulation Results

To evaluate the performance of TADVS, I conduct extensive experiments using various parallel applications. Further, I compare TADVS with an existing energy aware scheduling algorithm NDS [122]. The system parameters are shown in Table 3.2. The performance metrics used to evaluate system performance include: (1) Processor energy: Energy consumption incurred by computing nodes. (2) Total Energy: Energy caused by a set of parallel tasks.

Table 3.2: Pentium 4 System Parameters

| System Parameters | Values |
|---|---|
| CPU | Pentium 4, 1.4GHz |
| Idle power | 5W-22W |
| Frequency | 1400MHz,1200MHz,1000MHz, 800MHz, 600MHz |
| Voltage | 1.484V,1.463V,1.308V, 1.180V, 0.956V |
| Execution time for DFA | 3s, 3s, 4s, 2s, 1s, 10s, 20s, 7s, 5s, 8s |
| Execution time for the Guassian application | 5s, 4s, 1s, 1s, 1s, 1s, 10s, 2s, 3s, 3s, 3s, 7s, 8s, 6s, 6s, 20s, 30s, 30s |
| Network busy | 33.6w |

Figs. 1 and 2 show the synthetic parallel application (SPA) [67] and the Gaussian application [61], which were used to evaluate the performance of TADVS algorithm.

Table 3.3: Intel XScale System Parameters

| System Parameters | Values |
|---|---|
| CPU | Intel XScale |
| Idle power | 80mW |
| Frequency | 1000Hz, 800Hz, 600Hz, 400Hz, 150Hz |
| Voltage | 1.8V, 1.6V, 1.3V, 1.0V, 0.75V |
| Execution time for DFA | 3s, 3s, 4s, 2s, 1s, 10s, 20s, 7s, 5s, 8s |
| Execution time for the Guassian application | 5s, 4s, 1s, 1s, 1s, 1s, 10s, 2s, 3s, 3s, 3s, 7s, 8s, 6s, 6s, 20s, 30s, 30s |
| Network busy | 0.66w |

In the first set of experiments, I varied CCR from 0.1 to 1 to examine the performance impacts of communication intensity on our TADVS scheduling strategy. Figs. 3 and 4 demonstrate that compared with the NDS scheme, TADVS consistently consumes less energy regardless of the value of CCR. For example, TADVS conserves the energy consumption for the SPA application by up to 16.8% with an average of 10.7%. When one increases CCR from 0.1 to 1, the energy consumption gradually goes up. This can be explained by the fact that a high CCR results in high communication cost, which in turn leads to the increased total energy consumption. More interestingly, I observe from Fig. 4 that energy savings achieved

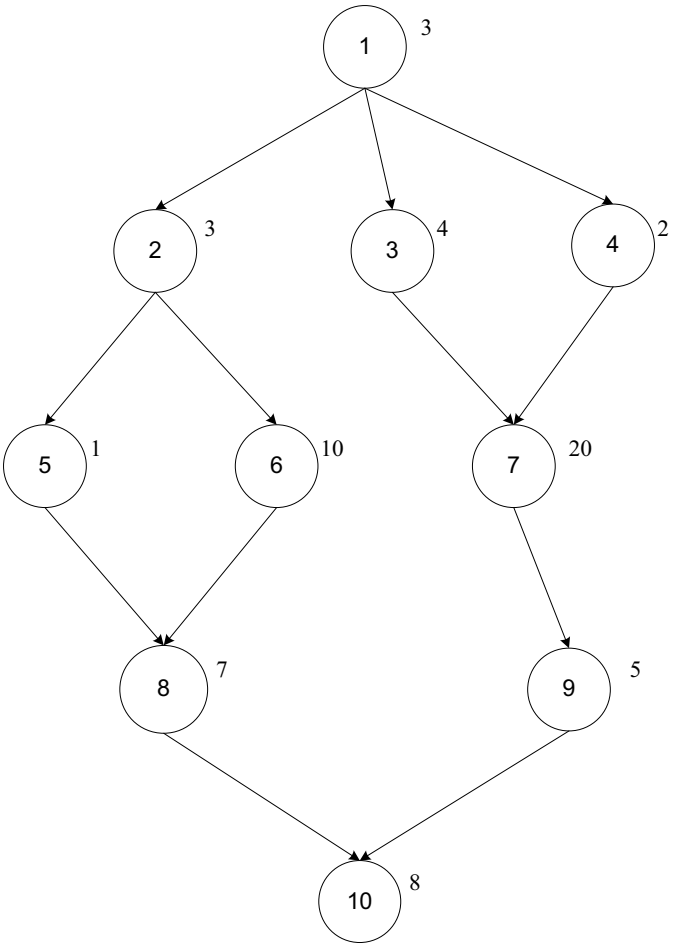


Figure 3.2: Task Graph of SPA

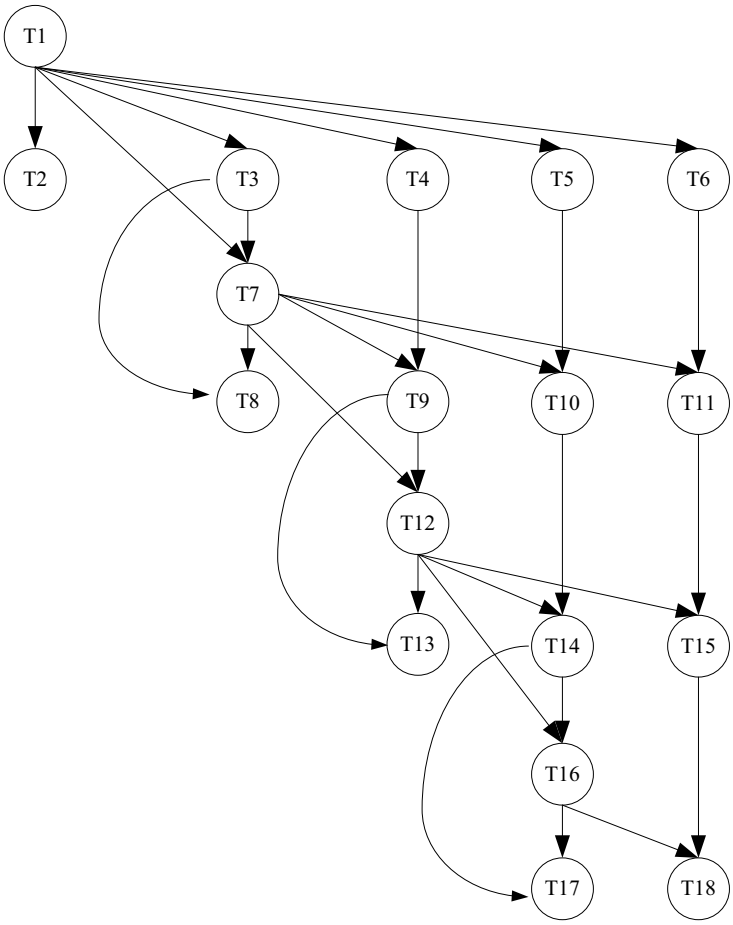


Figure 3.3: The task graph of the Gaussian application

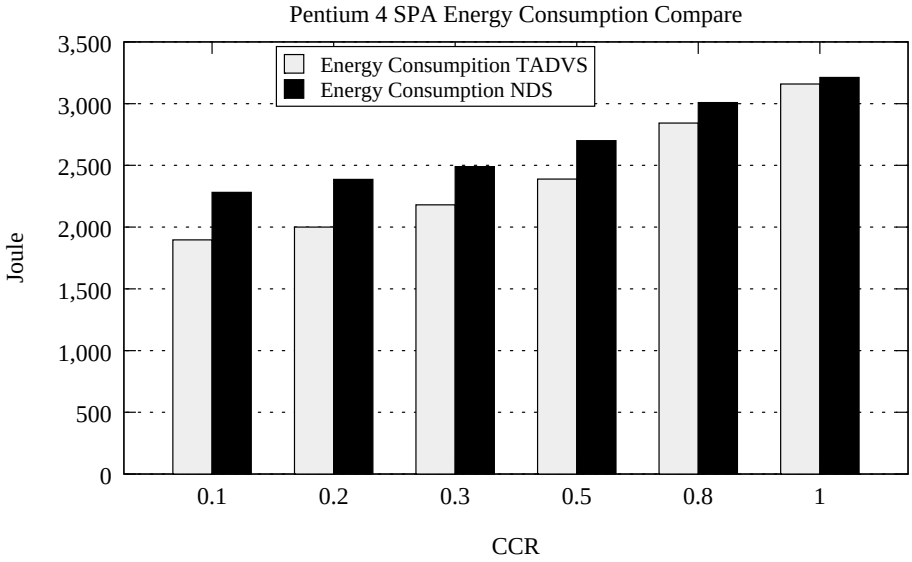


Figure 3.4: Energy consumption of SPA on the cluster with Intel Pentium 4 processors

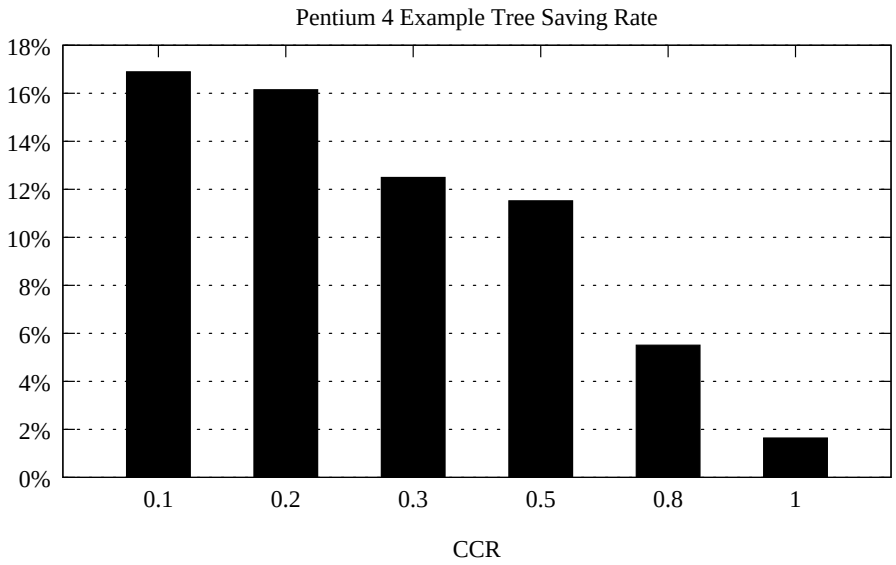


Figure 3.5: Energy saving rates for SPA on the cluster with Intel Pentium 4 processors

by the TADVS strategy become more pronounced when the communication intensity is relatively low. This result clearly indicates that low communication intensity offers more space for TADVS to reduce voltage supplies of computing nodes to significantly conserve energy. In other words, applications with low communication intensities can greatly benefit from the TADVS scheduling scheme.

Fig. 3.6 shows the energy consumption caused by the Gaussian application (see Fig. 3.3 on the cluster with Intel Pentium 4 processors, whereas Fig. 3.7 reveals the energy savings offered by TADVS with respect to the Gaussian application on the cluster uses the TADVS and NDS scheduling algorithms. First of all, the experimental results reveal that TADVS can save energy consumption for the Gaussian application by up to 14.8% with an average of 9.6%. Second, the results plotted in Figs. 3.6 and 3.7 shows that compared the Gaussian application with the SPA application, the energy saving rate of TADVS is less sensitive to the communication intensity. The empirical results suggest that the sensitivity of the energy saving rate of TADVS on communication intensity partially relies on the characteristics of parallel applications. Note that parallel applications' characteristics include parallelism degrees, number of messages, average message size, and the like. Compared with the SPA

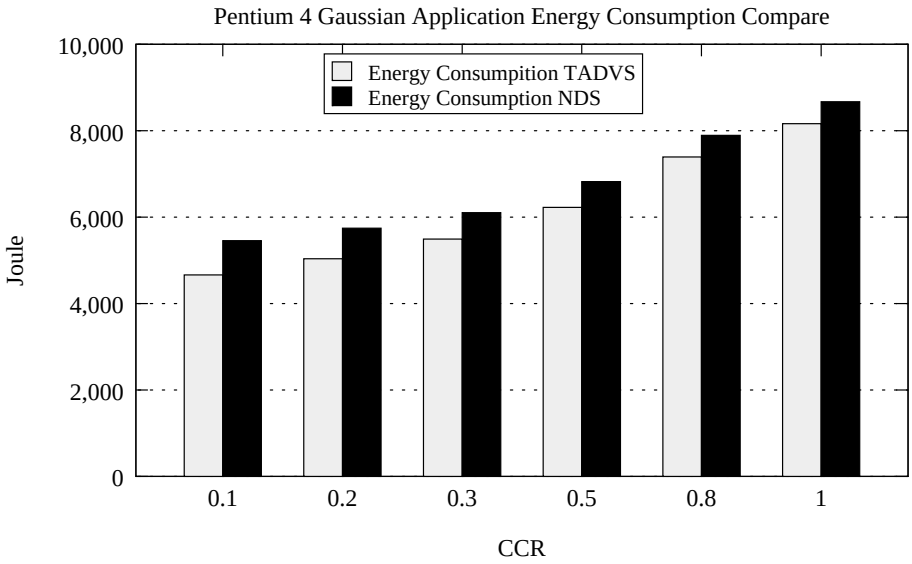


Figure 3.6: Energy consumption of the Gaussian application on the cluster with Intel Pentium 4 Processors

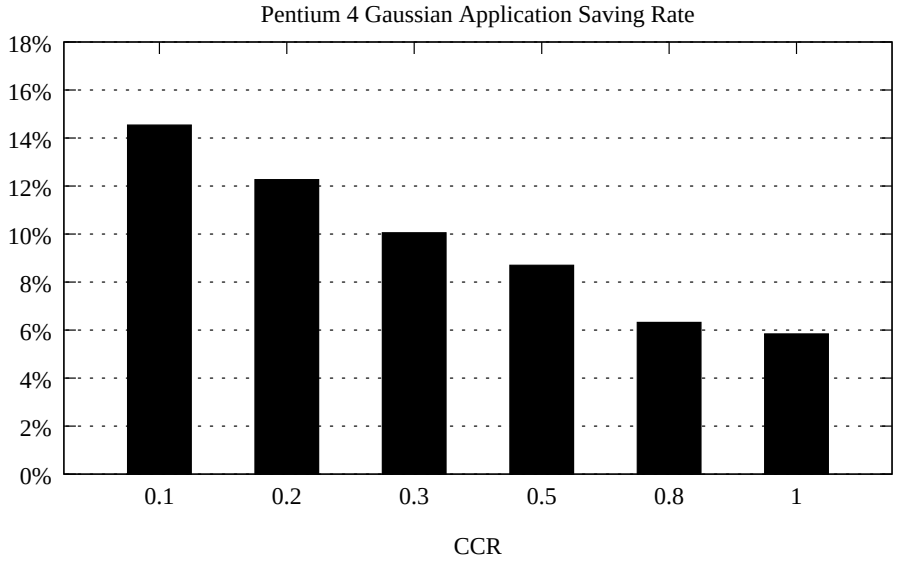


Figure 3.7: Energy saving rates for the Gaussian application on the cluster with Intel Pentium 4 Processors

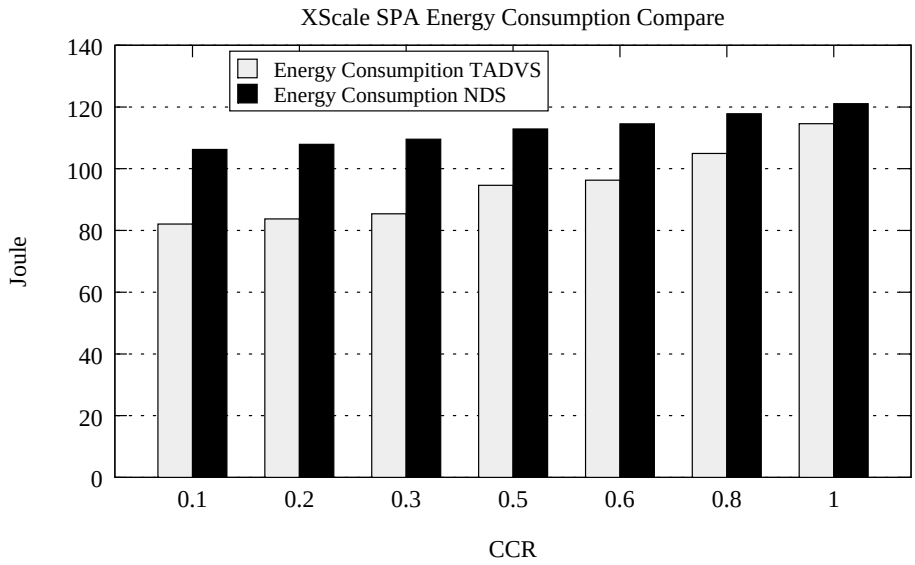


Figure 3.8: Energy consumption of SPA on a cluster with Intel XScale processors

application (Fig. 3.2), the Gaussian application has a higher parallelism degree. More specifically, I concluded from the experimental results shown in Figs. 3.5 and 3.7 that the energy saving rate of TADVS is less sensitive to the communication intensity of parallel applications with higher parallel degrees. Moreover, parallel applications with higher parallelism degrees are able to take more advantages from the TADVS in terms of energy conservation. A practical implication of this observation is that although high communication intensities of parallel applications tends to reduce energy saving rates of TADVS, increasing parallel degrees of the parallel applications can potentially and noticeably boost up the energy saving rates.

Figs. 3.7, 3.8, 3.9, 3.10 plot experimental results for both the SPA and Gaussian applications running on a cluster with Intel XScale processors in each computing node. The goal of this set of experiments is to evaluate the performance of TADVS using embedded processors. XScale processors are very suitable for next-generate mobile clusters (see, for example, [121]), where it is indispensable to conserve energy since mobile clusters are powered by batteries. Now I focus on the performance impacts of TADVS on mobile clusters. The empirical results shown in Figs. 3.7, 3.8, 3.9, 3.10 suggest that TADVS is conducive

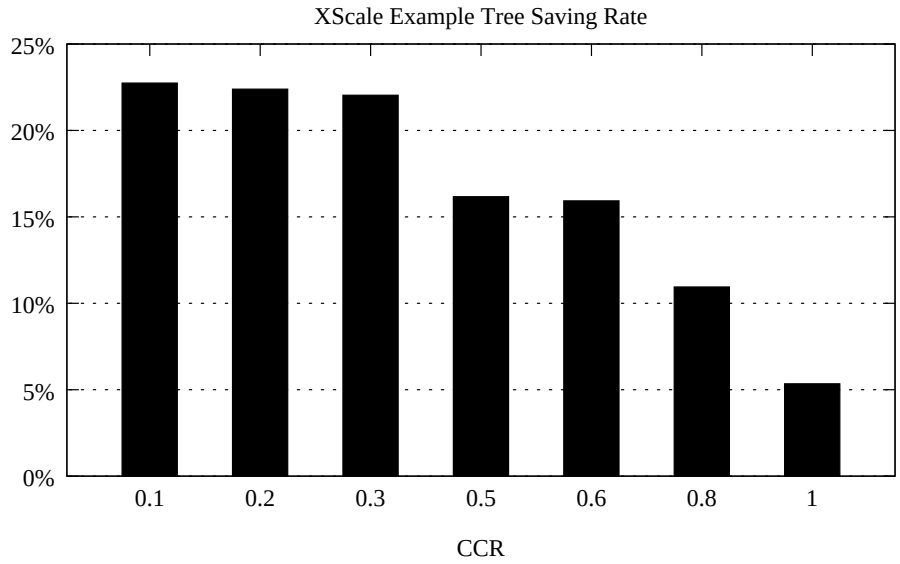


Figure 3.9: Energy saving rates for SPA on a cluster with Intel XScale processors

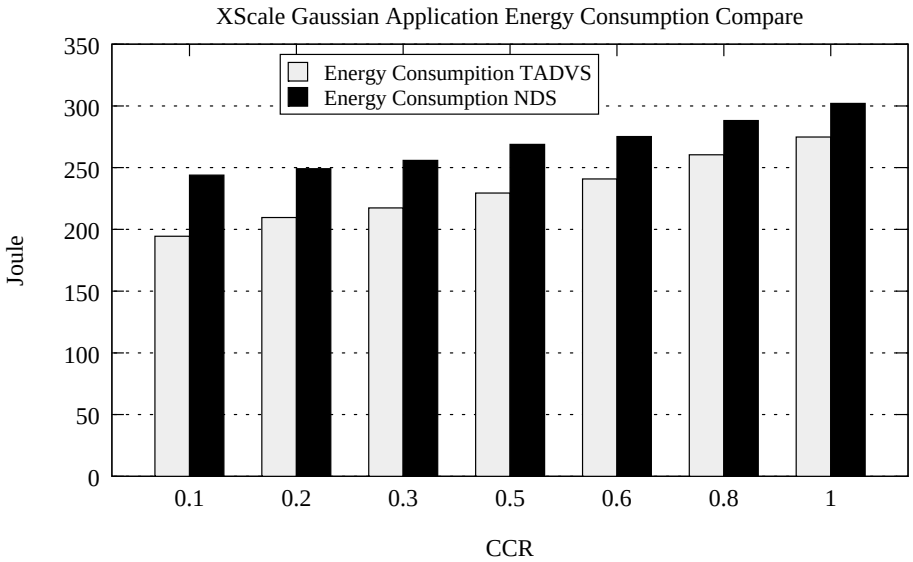


Figure 3.10: Energy consumption of the Gaussian application on a cluster with Intel XScale processors

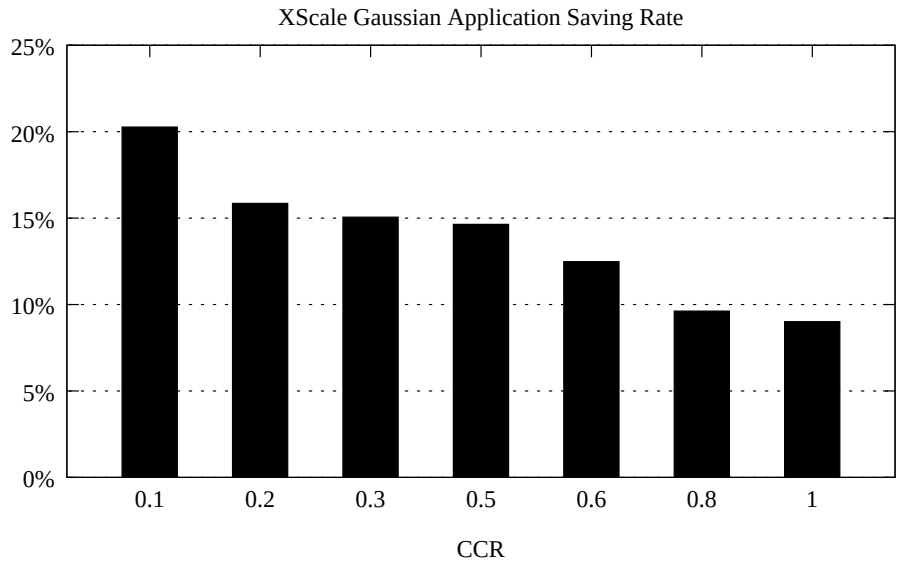


Figure 3.11: Energy saving rates for the Guassian application on a cluster with Intel

to conserving energy consumption in mobile clusters equipped with Intel Xscale processors. For example, TADVS saves energy for the SPA and Gaussian applications by up to 22.6% and 20.4% with averages of 19.2% and 16.2%. Again, I observe from Figs. 3.9 and 3.11 when the value of CCR increases, the energy saving rate drops gradually. This trend is consistent with the results plotted in Figs. 3.5 and 3.7. It is worth noting that the energy saving rates of TADVS are noticeably high, meaning that employing TADVS to schedule parallel applications on mobile clusters can ultimately save a huge amount of energy. The evidence from these experiments implies that TADVS is capable of considerably enlarging battery life. More importantly, the results (see Table 3.4) confirms that compared with traditional high-performance clusters, mobile clusters powered by batteries can enjoy a lot more energy-saving benefits provided by TADVS. Thus, the results suggest that clusters built with computing nodes with low CPU voltage supplies can fully leverage the dynamic voltage scaling technique to reduce energy dissipation in the clusters.

Figs. 3.11, 3.12, 3.13, 3.14 plot experimental results for the LU decomposition application, which is modeled using a task graph. The goal of this set of experiments is to evaluate

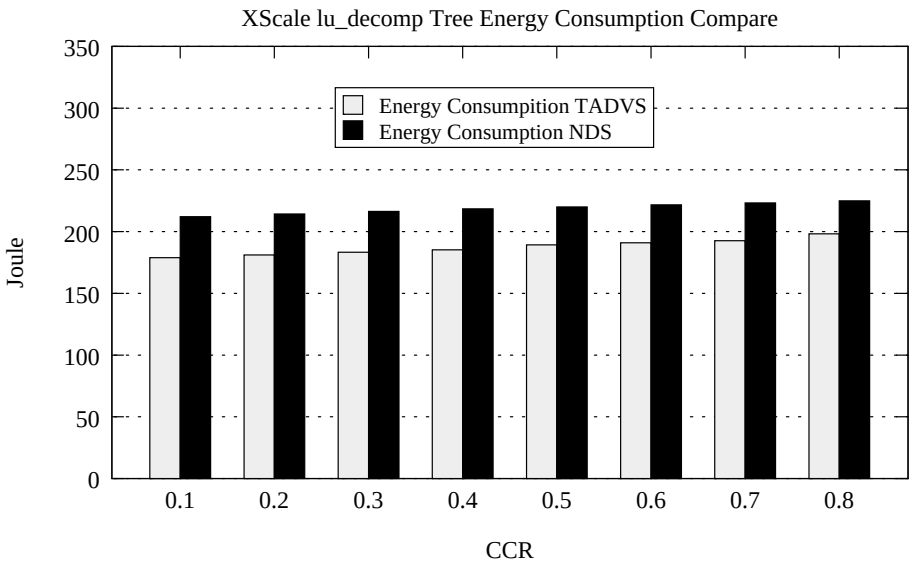


Figure 3.12: Energy consumption of the lu_decomp application on the cluster with Intel XScale processors

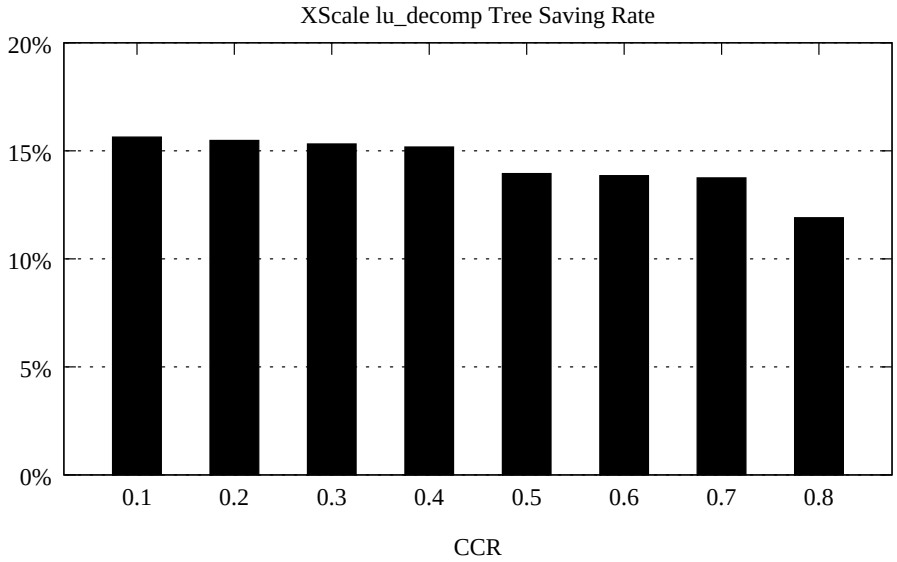


Figure 3.13: Energy saving rates for the lu_decomp application on the cluster with Intel XScale processors

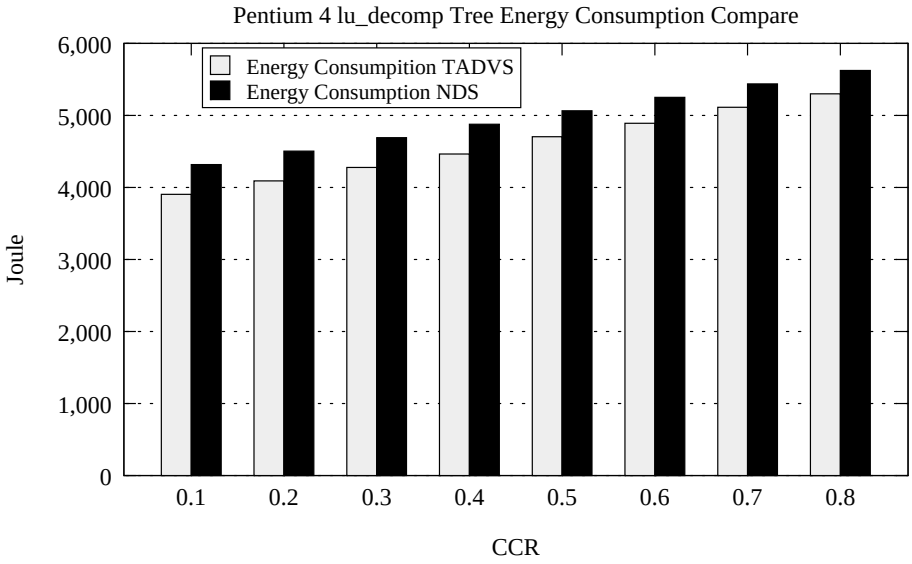


Figure 3.14: Energy consumption of the lu_decomp application on the cluster with Intel Pentium 4M processors

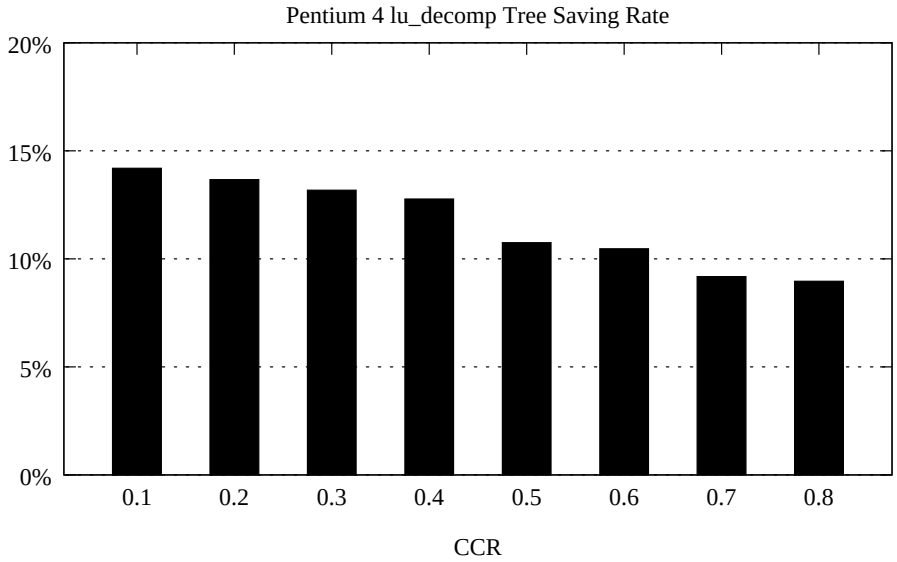


Figure 3.15: Energy saving rates for the lu_decomp application on the cluster with Intel Pentium 4 processors

Table 3.4: High-Performance Clusters vs. Mobile Clusters Energy Saving Rate for the lu_decomp Applications

| lu_decomp | High-Performance Clusters | Mobile Clusters |
|---|---|---|
| Maximum Energy Saving Rate | 14.8% | 20.4% |
| Minimum Energy Saving Rate | 9.6% | 16.2% |

the performance of TADVS on a mobile cluster with the Intel XScale processors and a high-performance cluster with the Intel Pentium 4 processors. Specifically, Figs. 3.12 and 3.13 show the results gathered using the XScale processor, whereas Figs. 3.14 and 3.15 show the results when the applications are running on a high-performance cluster where Intel Pentium processors are used. With respect to the LU decomposition application, the TADVS strategy achieves more energy savings on the mobile cluster than on the high-performance cluster (see Table 3.4 for details). Mobile clusters with XScale processors benefit better from the TADVS scheme because the voltage supplies for XScale processors are lower than those for Pentium processors. In order to show the generality of this algorithm, I added an extra parameter, which is called mul-plus parameter (mpp or calculating parameter), to calculate the computing time for each task. This parameter was added to generate different power consumption data for each task, so I can estimate the effect of TADVS algorithm on energy consumption. I use Eq. 3.23 to calculate the new computing time for each task. Specifically, when Eq. 3.31 is applied, the total communication cost for each test is set to be a constant number.

$$New\ Computing\ Time_i = Old\ Computing\ time_i * mpp + mpp \tag{3.31}$$

Figs. 3.16, 3.17, 3.18, 3.19, 3.20, 3.21, 3.22, 3.23 show the experimental results when I apply the "mul-plus parameter" to calculate computing times of tasks in the two parallel applications. The results shown in these eight figures are for the Gaussian and SPA applications. Again, I evaluate TADVS's performance in terms of energy efficiency on the two clusters with Intel XScale and Intel Pentium 4 processors, respectively. Note that the eight

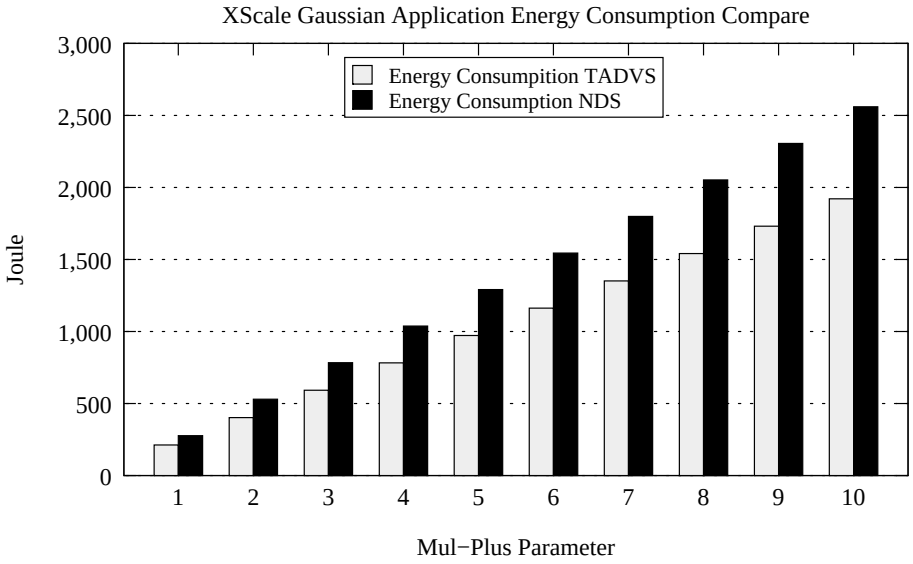


Figure 3.16: Energy consumption of the Gaussian application on the cluster with Intel XScale processors using mul-plus parameters to calculate

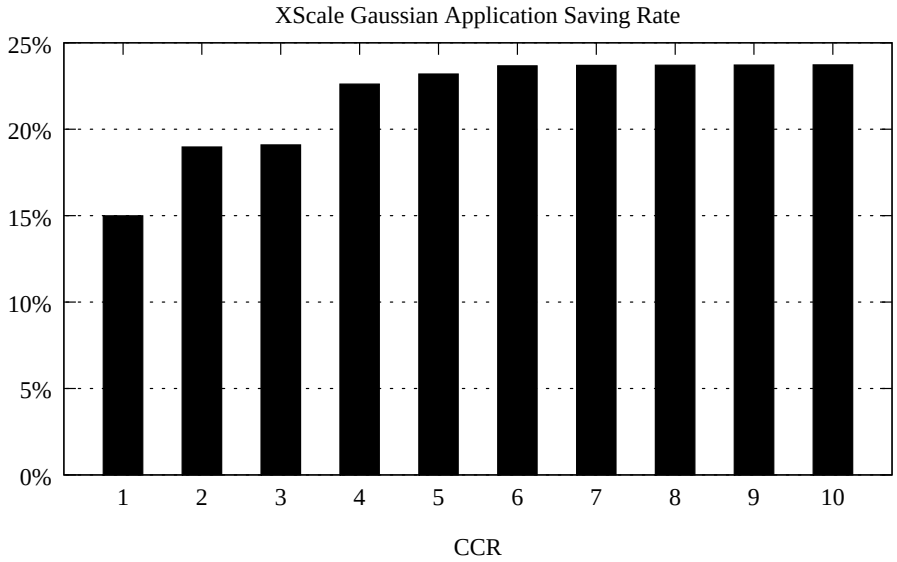


Figure 3.17: Energy saving rate of the Gaussian application on the cluster with Intel XScale processors using mul-plus parameters to calculate

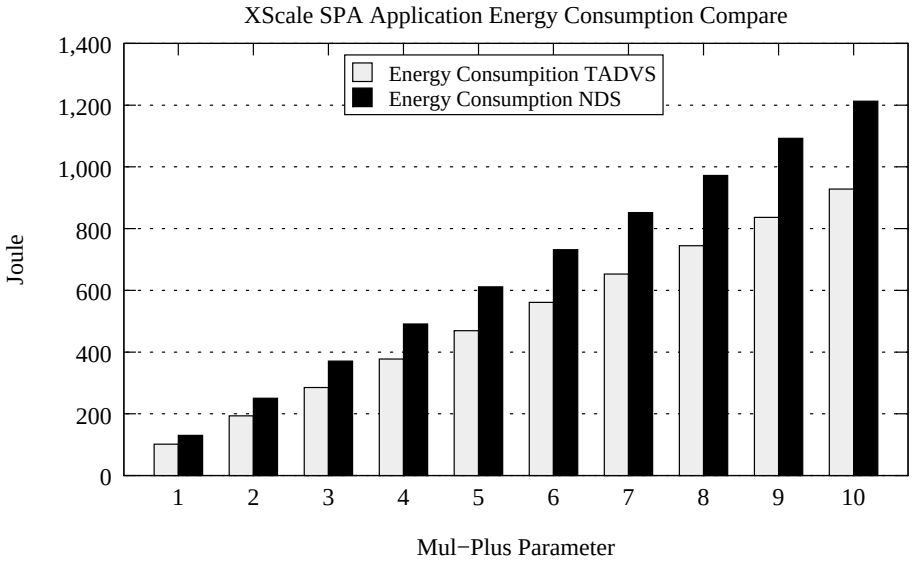


Figure 3.18: Energy consumption of the SPA application on the cluster with Intel XScale processors using mul-plus parameters to calculate

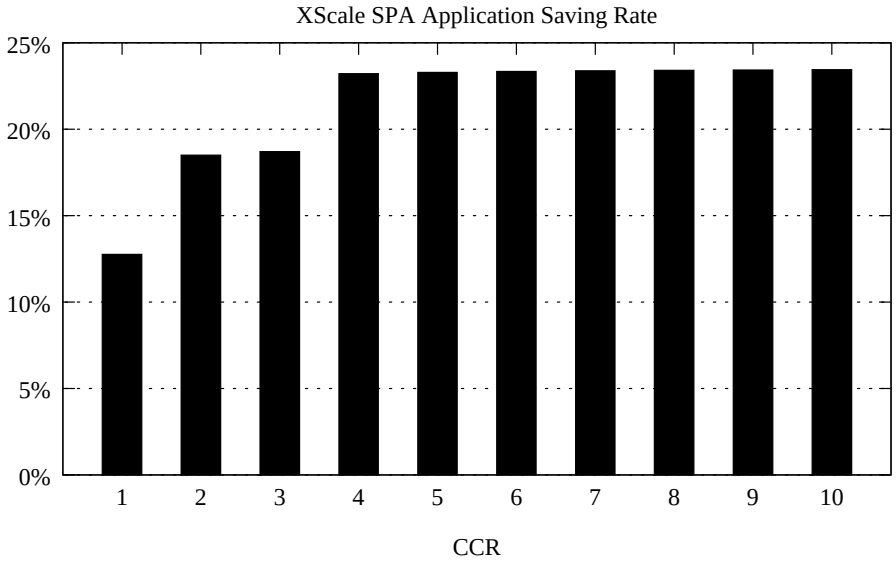


Figure 3.19: Energy saving rate of the SPA application on the cluster with Intel XScale processors using mul-plus parameters to calculate

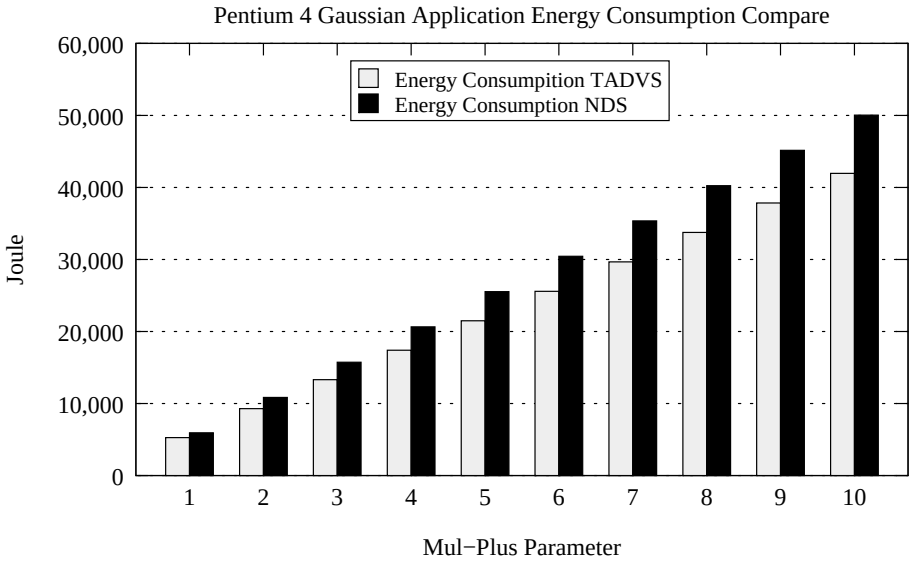


Figure 3.20: Energy consumption of the Gaussian application on the cluster with Intel Pentium 4 processors using mul-plus parameters to calculate

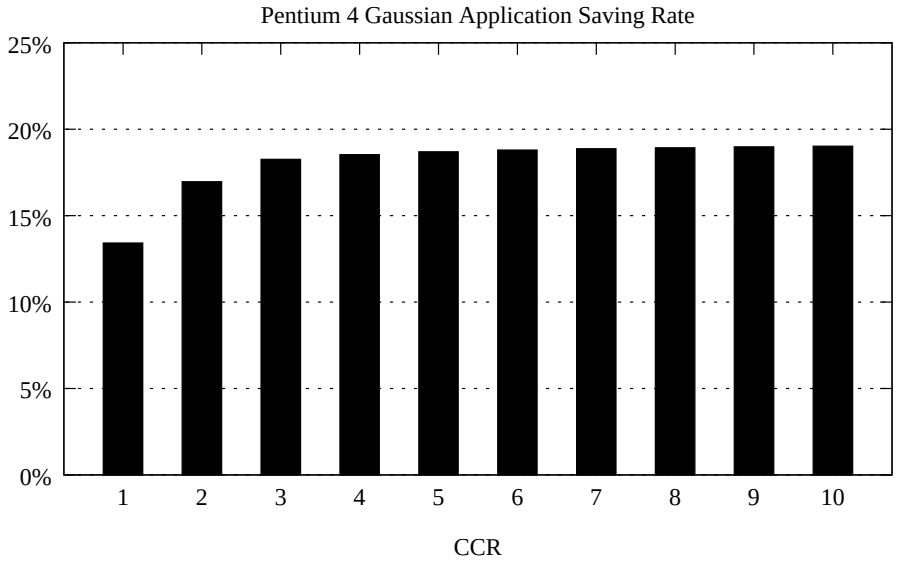


Figure 3.21: Energy saving rate of the Gaussian application on the cluster with Intel Pentium 4 processors using mul-plus parameters to calculate

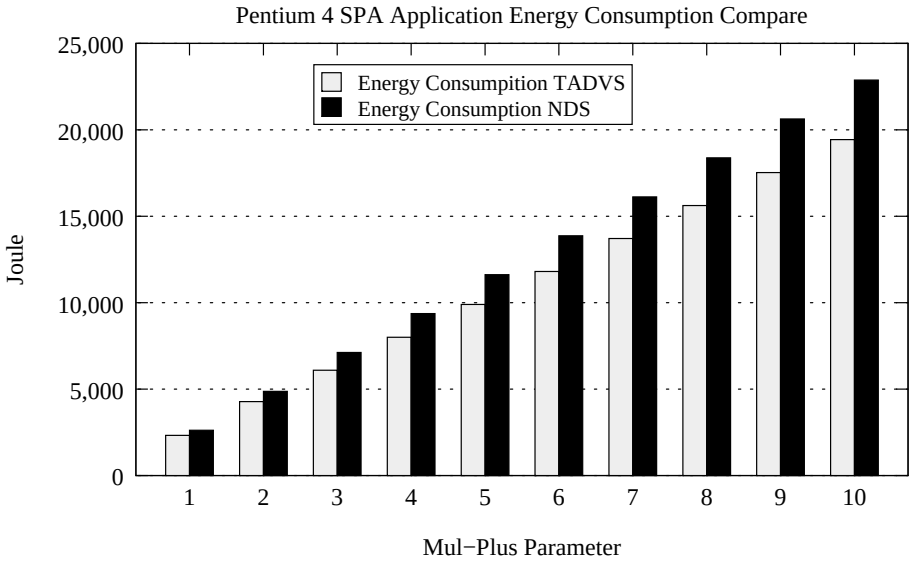


Figure 3.22: Energy consumption of the SPA application on the cluster with Intel Pentium 4 processors using mul-plus parameters to calculate

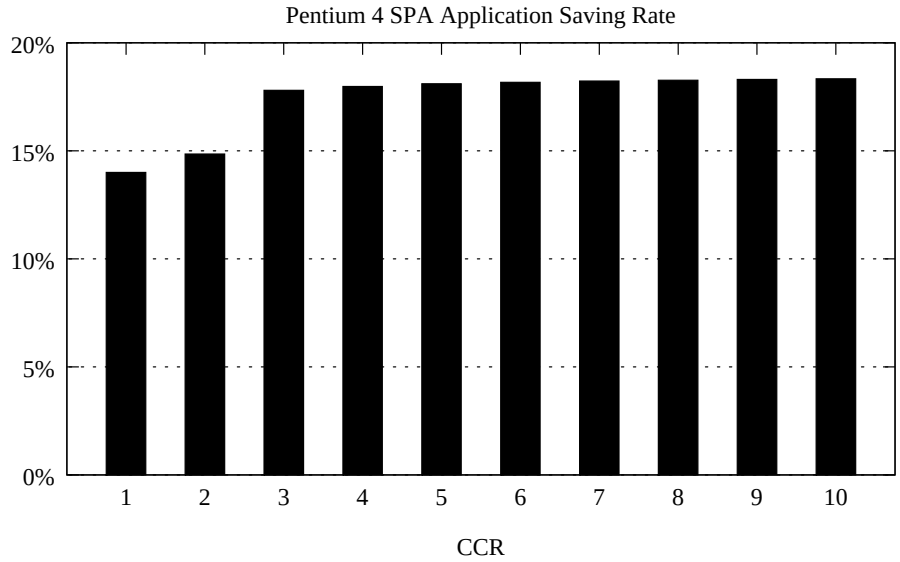


Figure 3.23: Energy saving rate of the SPA application on the cluster with Intel Pentium 4 processors using mul-plus parameters to calculate

figures have four groups: the Gaussian application on the cluster with Intel XScale processors (see Figs. 3.16, 3.17); the SPA application on the cluster with Intel XScale processors (see Figs. 3.18, 3.19); the SPA application on the cluster with Intel XScale processors (see Figs. 3.20, 3.21); and the Gaussian application on the cluster with Pentium 4 processors (see Figs. 3.22, 3.23). I make the following observations from the results plotted. First, Figs. 3.16, 3.18, 3.20, and 3.22 show that regardless of the applications and processor types, energy dissipation in clusters soars when the calculating parameter increases. This is mainly because a high calculating parameter gives rise to long CPU processing times, which in turn lead to a high CPU energy demand. Second, after comparing the results shown in Figs. 3.17 and 3.21, I conclude that energy savings provided by TADVS is more significant on mobile clusters with Intel XScale processors than on high-performance clusters with Intel Pentium 4 processors. The results depicted in Figs. 3.19 and 3.23 can further confirm this observation. Third, when the calculating parameter is smaller than 6, the saving rate of TADVS gradually rises when one increase the calculating parameter (see Figs. 3.17, 3.19, 3.21, 3.23). However, the every saving rate is stabilized after the calculating parameter exceeds 5. In other words, though the computing time of each task is changed significantly, the energy saving rates literally remain unchanged. It is worth noting that the total amount of energy conserved by TADVS continues growing with an increasing value of the calculating parameter.

In the last set of experiments, I evaluate the processor frequency levels for each parallel task using the Gaussian application. For comparison purposes Fig. 3.24 includes NDS, where the highest frequency level is chosen to finish tasks as soon as possible. I also show the optimal frequency levels which maximize the energy saving rate. Since processor frequencies are not continuously tunable, TADVS is unable to achieve the optimal energy savings. However, Fig. 3.24 demonstrates that energy savings achieved using TADVS is close to the optimal solution. These results reveal that TADVS produces sub-optimal energy savings for parallel applications on both high-performance clusters using Intel Pentium 4 processors and mobile clusters using Intel XScale processors.

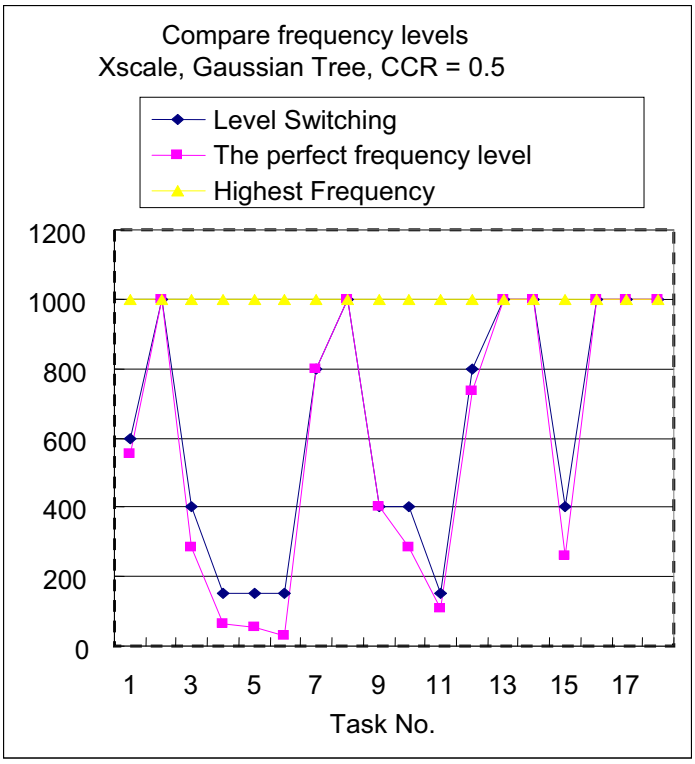


Figure 3.24: Processor frequencies of the Intel XScale processors under the TADVS scheduling algorithm when the CCR value is set to 0.5

## 3.5 Summary

In the past decade clusters have been applied to a variety of parallel applications. Scheduling parallel applications on both clusters, especially large-scale clusters, is challenging. This is due to significant communication latencies and high energy consumption rates. Reducing schedule lengths and energy conservation are two major factors in the design of environmentally and economical friendly clusters. In this Chapter, I proposed a scheduling algorithm call TADVS. It makes use of the dynamic voltage scaling technique (DVS) to provide significant energy savings for both high-performance and mobile clusters. Our TADVS scheme aims to judiciously leverage processor idle times to lower processor voltages. This in turn, reduces the energy consumption of parallel applications running on clusters. However, decreasing supply voltages can inevitably lead to increased execution times. The novel feature of TADVS is that it employs DVS only to parallel tasks that are followed by idle processor times. This decreases energy consumption without increasing the schedule lengths of parallel applications. Experimental results show that TADVS is capable of conserving energy without adversely affecting performance of high-performance clusters and mobile clusters.

# Chapter 4

# Design and Performance Evaluation of Energy-Efficient Parallel I/O Systems With Write Buffer Disks

## 4.1 Introduction

In the past few years, large-scale storage systems have been developed to achieve high I/O performance and large storage capacity for a wide variety of data-intensive applications [91] [62] [71] [47]. Much attention has been paid to the issues of performance and security in storage systems [116] [103] [11]. Making data disks active even when they are sitting idle is an important avenue to maintain high performance, because disks can immediately start serving newly arriving disk requests. This is a practical approach in some cases where there are high-end computing servers require extremely high performance at the cost of high energy consumption. However, this approach can waste a huge amount of energy in large-scale parallel disk systems; this is true especially when there are many long idle periods. Traditional energy conservation techniques (e.g., dynamic power management) improve disk I/O energy efficiency by turning disks into the low-power state if the disks are sitting idle. Unfortunately, the conventional dynamic power management strategies for single disk systems are inadequate for parallel disk systems because of the following three reasons. First, idle periods under some workload conditions are too short to turn disks into a low-power state to conserve energy. Second, although energy can be conserved by frequently place disks into the low-power state, an excessive number of power-state transitions inevitably have adverse impacts on the reliability of parallel I/O systems. Third, numerous power-state transitions impose significant energy overhead as well as response time penalties. Making the traditional strategy energy effective largely depends on workload conditions. If workload of disk requests is relatively low and energy consumption of spinning operations

is much lower, then it makes sense to apply the traditional power management strategy to aggressively spin down idle disks. Unfortunately, disk request arrival rates of most parallel I/O systems are usually high, leaving few opportunities for the dynamic power management strategies to conserve energy in most cases.

It is evident that the existing dynamic power management strategies ultimately encounter the problem of long power-state transition times and noticeable power state transition energy overhead. Although disk active times in the parallel storage system can be shortened, energy dissipation in the storage system may not necessarily be reduced. This is due the fact that power-state transitions introduce a significant amount of energy overhead, which is refers to as penalties of spin-up and spin-down operations. Turning on a disk in a low-power mode does not only need to power the disk up, but also requires the disk to speed up its rotation speed - a physical movement, which in turn consumes much more energy than electrical operations. In addition, if a new request arrives when the disk has been recently shut down, the new request has to wait for an unnecessary period of spin-up time. To remedy this deficiency, in this research I aim to design an innovative approach to significantly reducing unnecessary spinning operations while shortening response times of disk requests.

Recognizing that energy overhead and response time penalties induced by power-state transitions negatively affect energy efficiency of parallel I/O, in this study I seek to reduce the number of power-state transitions for writes processed by a parallel disk system. I focus on write requests, because there are a considerable number of write-intensive applications like transaction processing, log file updates, and data collection [81]. In this research, I present the design and implementation of parallel storage systems with buffer disks processing write requests. Specifically, I aim to develop a dynamic request allocation algorithm for writes or DARAW, which dynamically and energy efficiently allocates buffer disks or data disks to serve write requests. Request allocations depend on not only data sets residing in buffer disks contain but also the power states of data disks. Data sets cached in buffer disks will be transferred to corresponding data disks when a set of conditions are satisfied.

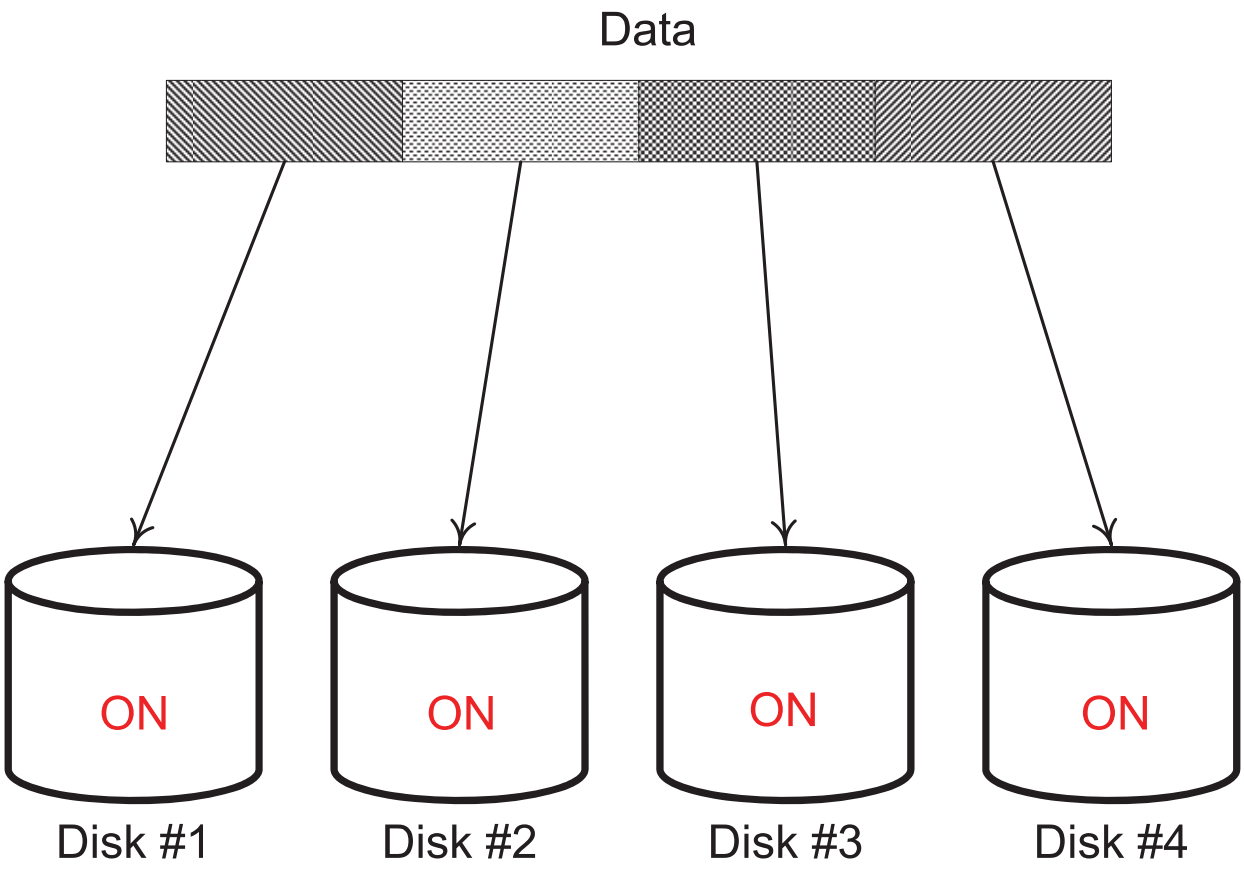


Figure 4.1: Classic Data Stripping

These conditions may be configured by system administrators to tune the performance of storage systems. Experimental results show that DARAW is conducive to conserving energy consumption in parallel storage systems while efficiently reducing response times of write requests.

## 4.2 Architecture with Write Buffers

In this section, I first introduce our energy-efficient disk architecture. Then, I present a dynamic request allocation algorithm for writes or DARAW. Finally, I build an energy consumption model to quantify energy dissipation in parallel I/O systems.

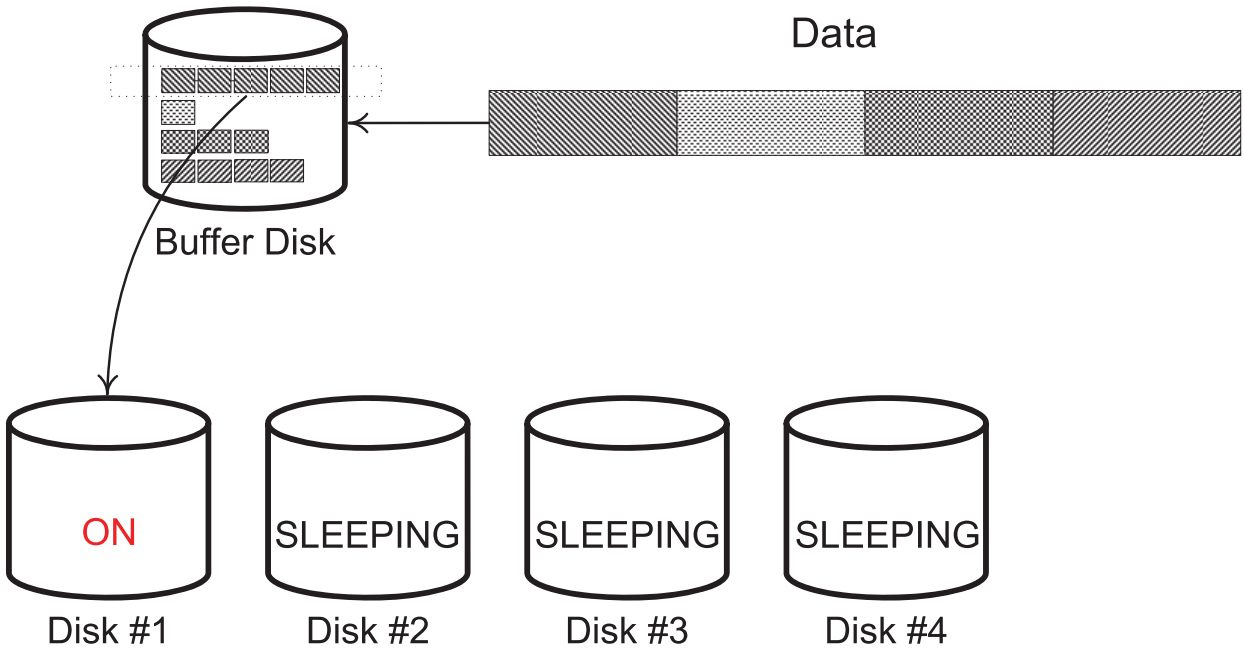


Figure 4.2: Basic BUD Idea

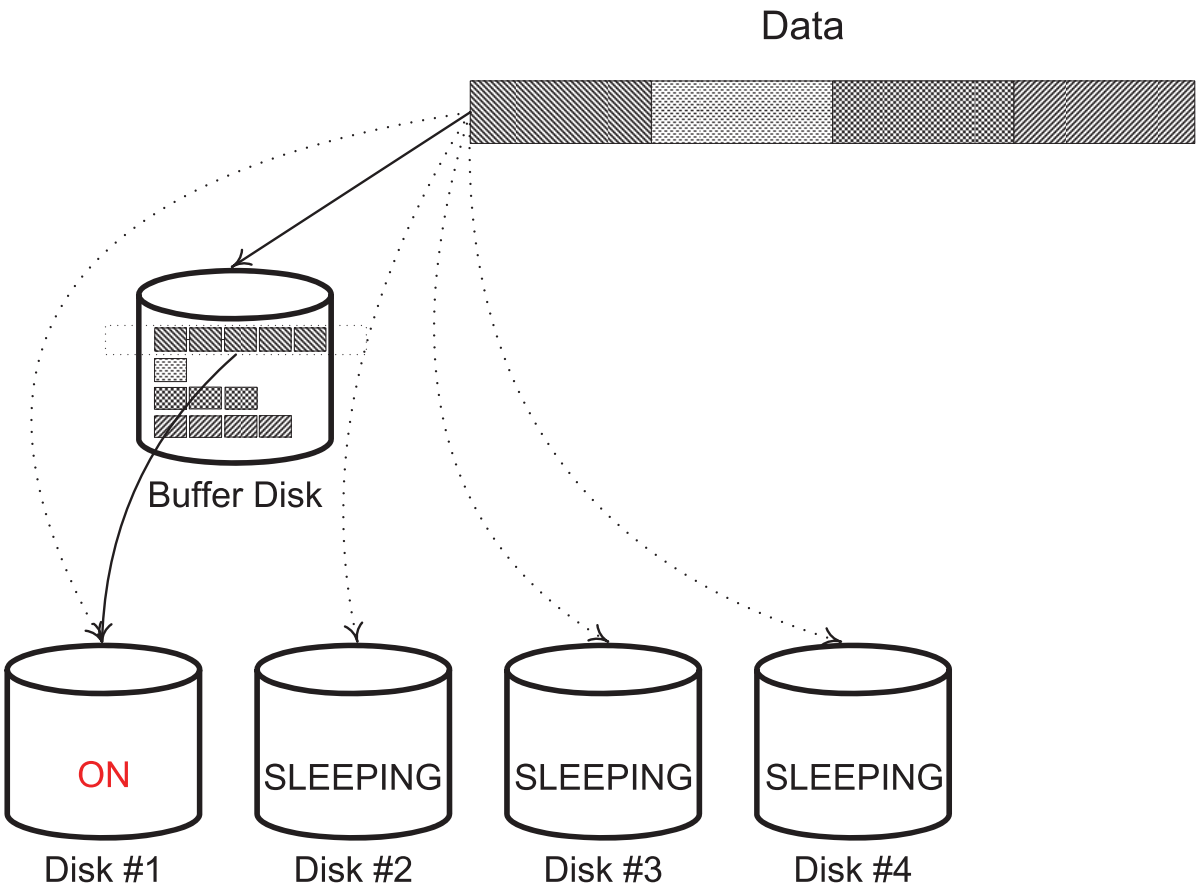


Figure 4.3: BUD Data Flow

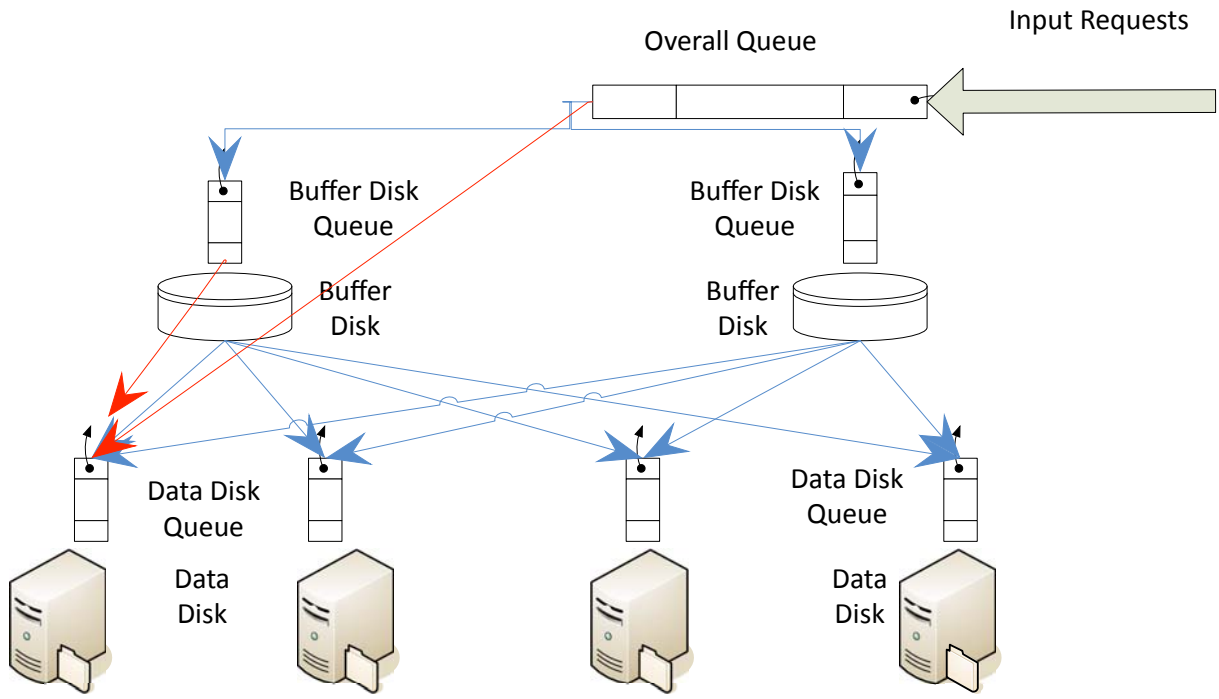


Figure 4.4: The architecture of parallel storage system with buffer disks

### 4.2.1 Parallel Storage Systems with Buffer Disks

Traditionally, data striping is deployed to enhance inter-disk parallelism for higher data transfer rate. Fig. 4.1 presents the classic data striping technology in which design all disks are active all the time. Fig. 4.2 presents the basic design of BUD. A small number of buffer disks buffer incoming writes for data disks. When workload is high, data disks are active most of time, so requests could go to data disks directly without going through buffer disks (Fig. 4.3).

Let us present our energy-efficient disk architecture with buffer disks (see Fig. 4.4). This architecture is unique when compared to traditional parallel storage system architectures. I classify disks in a parallel storage system into two categories: buffer disks and data disks. All disks in the system are separated into two distinct layers. Requests issued to the parallel storage system are written temporally into buffer disks first and then be transferred into data disks at appropriate time periods.

Each disk, regardless of buffer or data disks, has its own queue to store incoming requests. In addition, there is an overall request queue, in which all requests enter when they are submitted to the storage system. In most cases, the number of buffer disks is less than the number of data disks. This is because our target goal in this study is to save energy by keeping a small number of active buffer disks while placing a large number of data disks into the low-power state. The ratio of the number of data disks and the number of buffer disks can largely affect the energy efficiency of the parallel storage system. Ideally, the ratio needs to be adjusted on the fly in accordance with workload conditions. In this study, I evaluate impacts of this ratio on energy efficiency of parallel I/O systems.

### 4.2.2 The DARAW Algorithm

Now I describe the dynamic request allocation algorithm for writes or DARAW, which was designed in light of the novel disk architecture depicted in Fig. 4.4 DARAW is an on-line algorithm, which can handle input disk requests without knowing disk access patterns in a priori. In a parallel I/O system with buffer disks, there is a buffer-disk layer and a data disk layer. This indicates that the first phase in the DARAW algorithm is to decide a buffer disk by which a request should be served. After requests are responded by a buffer disk, it is essential to determine when to transfer the data set from the buffer disk to a corresponding data disk. Therefore, DARAW contains two parts: a buffer-disk layer scheduling scheme and a data-disk layer scheduling scheme. Buffer-disk layer scheduling in DARAW focuses on choosing the most appropriate buffer disks to serve writes. DARAW is an on-line algorithm that has to make a scheduling decision based on the current status of the system. Therefore, the decision made by DARAW might not be the optimal one. Scheduling decisions made in the buffer-disk layer are affected by the status of the data disks. For example, if a targeted data disk of a write request is active, the request can be responded by the targeted data disk. In doing so, unnecessary data transfer from the buffer disk to the target data disk can be eliminated.

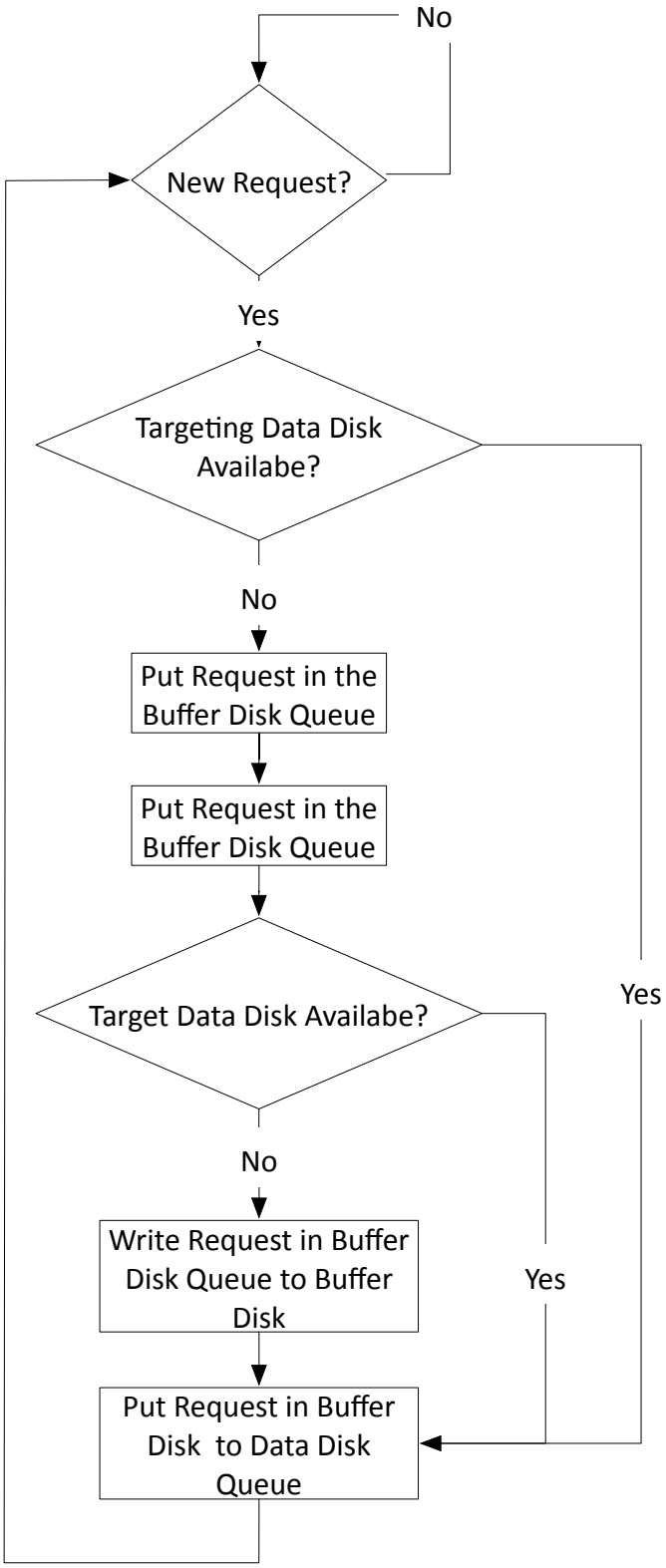


Figure 4.5: Buffer-disk layer scheduling in the dynamic request allocation algorithm for writes

Fig. 4.5 outlines the buffer-disk layer scheduling scheme in DARAW and Fig. 4.6 shows . When a write request enters into a buffer-disk queue or is about to be served by a buffer disk, DARAW can process the request in two ways: the request can be served by the buffer disk; or the request can be allocated to and served by the corresponding data disk. DARAW directly allocates the request to the data disk without having the data buffered in the buffer disk if the data disk is active, thereby keeping the data disk in the active-power state without turning off the data disk until all requests in its queue are completed. Hence, the requests targeting at this data disk could be written in the data disk neither going through buffer disk layer nor affording transition penalty. However, if the targeting data disk of a request is sleeping when the request needs to be served, DARAW has to either put the requests into the queue of a buffer disk or have the buffer disk immediately process the request. In this case, DARAW picks up a buffer disk that contains a list of pending requests targeting at the same data disk as the current request. The goal of choosing a buffer disk for the current request in this way is to make the data movement from buffer disks to data disks more efficient. In other words, buffering requests with the same target data disk into one buffer disk makes it possible to move data back to the target data disk in a batch manner. If there is no such a buffer disk, DARAW will pick a buffer disk that has the lightest workload, which is quantified by the data size of queued requests. In Fig. 4.5, there is a dashed arrow between "write buffer disk" and "write data disk" because this is where Data-Disk scheduling works.

To facilitate the development of DARAW, in what follows I define an important scheduling-control parameter called *Sum of Requests in Buffer*, which is referred to as SRB throughout this research. Note that incoming write requests are separated into two groups with different writing paths which are illustrated in Fig. 4.4. The first processing path, shown by the blue arrows, illustrate that requests are served by buffer disks and then data sets are transferred to data disks. The second processing path, shown by red arrows, indicates that requests are directly handled by the data disk layer. When a request going through the first path is written on a buffer disk, I say it is buffered. Each data disk has its own SRB which contains

the number of buffered requests targeting the data disk. When a request is buffered, the corresponding SRB will be increased by 1. When requests are transferred from a buffer disk to a data disk, the corresponding SRB will be decreased. It is clear that requests going through the second path will not be counted in SRB, because these requests are not buffered. I set up a threshold value $SRB_{th}$ for SRB to decide when DARAW should transfer requests to data disks. For example, if the SRB value of a data disk exceeds $SRB_{th}$, then DARAW needs to wake up the data disk, to which buffered data should be transferred from buffer disks.

**Algorithm 2** The dynamic request allocation algorithm for writes or DARAW

```
if targeting data disk is not sleeping then
  if targeting data disk is not sleeping  then
    write the request into targeting data disk
  else
    if buffer disk i having same targeting requests then
      write the request in buffer disk i
      Corresponding SRB ++
    else
      write the lowest load buffer disk
    end if
  end if
end if
if more than 3 working buffer disks are blank then
  turn off 1
end if
while Each Disk j in PSS do
  if SRB_j >= SRB_th  then
    turn on data disk j
    write all requests targeting at j into disk j
    turn off data disk i
    clear SRBj
  end if
end while
```

Recall that each data disk has a corresponding SRB value to track how many data sets have been buffered. This parameter plays a vital role in minimizing energy consumption of parallel disk systems. It is evident that energy overhead incurred by power-state transitions may diminish energy conserved by placing disks into the low-power state. Keeping track of

the number of buffered write requests, DARAW aims to substantially reduce the number of unnecessary power transitions in data disks.

Once a request is written into a buffer disk, SRB of the target data disk will be increased by 1. If there are enough number of buffered requests for measured by the SRB value of a data disk, DARAW writes all the buffered requests into the data disk at one time after turning the data disk into the active state. To judge whether the SRB value is large enough, I need to compare SRB of each data disk with a threshold value SRBth. The larger the SRBth is set, the greater the reduction in the number of power-state transitions, which ultimately lead to lower energy consumption. Let $SRB_i$ denote the SRB value of data disk $i$; let $SRB_{ij}$ be the number of requests targeting on data disk $i$ while being buffered in buffer disk $j$. $SRB_j$ can be derived from $SRB_{ij}$. In other words, $SRB_j$ is the sum of $SRB_{ij}$ of all the buffer disks. Thus, we have

$$SRB_j = \sum_{j=1}^{n} SRB_i^j \tag{4.1}$$

where $n$ is the number of buffer disks. Note that each SRB value in a parallel I/O system is updated continuously. In the following sections, $i$ represents the ith buffer disk number; $j$ represents the $jth$ data disk number; $n$ is number of buffer disks; $m$ is the number of data disk.

### 4.2.3 Energy Consumption Analysis

In a process of building an energy consumption model, I consider two types of energy dissipation in parallel I/O systems with buffer disks. The first one is energy consumption of disks when they are either in the active or sleep state. Another is energy dissipation induced by power-state transitions, including disk spinning down and spinning up. When a disk transitions between power states, the energy consumption penalty and time penalty may become a dominating factor that negatively affect energy savings. This is because I need to perform mechanical operations, e.g. speed up the disk from 0 RPM (Round per Minutes) to its standard working RPM and vice versa.

Concretely, I analyze energy consumption of parallel disk systems with DARAW by developing an energy consumption model as follows. Total energy consumption in a parallel disk system is comprised of energy consumption for serving all requests in data disks and buffer disks, idle energy consumption of all data disks and buffer disks, and penalty energy consumption of spinning up and down operations in all disks. In the case of small writes, both seek times and rotational delays are very small. Therefore, I model seek time and rotational delay using their average values. In previous studies, similar approaches were used to model seek times and rotational delays for small disk requests [113]

Let $e_{total}$ be the total energy dissipation in a parallel disk system. $e_{total}$ can be expressed by Eq. 4.2 below:

$$e_{total} = e_A^B + e_I^B + e_S^B + e_A^D + e_I^D + e_S^D + e_O^D + e_O^D \tag{4.2}$$

where $e_{S,B}$ and $e_{I,B}$ are energy consumption of buffer disks when they are in the active and low-power state; $e_{S,D}$ and $e_{I,D}$ are energy consumption of data disks when they are in the active and low-power state; $e_{P,B}$ and $e_{P,D}$ are the energy overhead experienced by buffer and data disks.

The energy consumption of buffer and data disks when they are active can be written as:

$$\begin{aligned}
& e_{S,B} + e_{S,D} \\
&= \sum_{i=1}^{m} \sum_{k=1}^{l} x_{k,B,i} P_{A,B,i} T_{A,B,i} + \sum_{j=1}^{n} \sum_{k=1}^{l} x_{k,D,j} P_{A,D,j} T_{A,D,j} \\
&= \sum_{i=1}^{m} \sum_{k=1}^{l} \left( x_{k,B,i} \left( P_{A,B,i} \left( t_{SK,k,B,i} + t_{RT,k,B,i} + \frac{s_k}{B_{B,i}} \right) \right) \right) \\
&\quad + \sum_{j=1}^{n} \sum_{k=1}^{l} \left( x_{k,D,j} \left( P_{A,D,j} \left( t_{SK,k,D,j} + t_{RT,k,D,j} + \frac{s_k}{B_{D,j}} \right) \right) \right)
\end{aligned} \tag{4.3}$$

where element $x_{k,B,i}$ is "1" if request $k$ is responded by the $i$th buffer disk and is "0", otherwise. Similarly, $x_{k,D,j}$ is "1" if request $k$ is responded by the $j$th data disk and is "0", otherwise. $T_{A,B,i}$ and $T_{A,D,j}$ are the service times of requests $i$ and $j$. $T_{A,B,i}$ is the summation of $t_{SK,k,B,i}$, $t_{RT,k,B,i}$, and $s_k/B_{B,i}$, which are the seek time and rotational latency of the request, and the data transfer time depending on the data size $s_{ij}$ and the transfer rate $B_i$ of the disk. Similarly, $T_{A,D,j}$ is the summation of $t_{SK,k,D,j}$, $t_{RT,k,D,j}$, and $s_k/B_{D,j}$, which are the seek time and rotational latency of the request, and the data transfer time.

In idle energy consumption largely depends on idle time periods that can be derived from the serving time and the last request's finishing time. Eqs. 4.4- 4.6 describe a way of quantifying energy consumption when disks are idle. The function max() returns the finishing time of the last request written on a data disk (not a buffer disk). Since all disks working in parallel, the total working time for all disks would be the value returned by max() multiple the number of disks. Moreover, the spinning up and spinning down operations penalty time needs to be subtract from max().

$$e_{I,B} + e_{I,D} = \sum_{i=1}^{m} P_{I,B,i} T_{I,B,i} + \sum_{j=1}^{n} P_{A,D,j} T_{A,D,j} \tag{4.4}$$

$$\begin{aligned} T_{I,B,i} = & \max_{k=1}^{l}(x_{k,D,j} f_k) \\ & - \sum_{k=1}^{l} x_{k,B,i} \left( t_{SK,k,B,i} + t_{RT,k,B,i} + \tfrac{s_k}{B_{B,i}} \right) \\ & - \sum_{k=1}^{l} (y_{k,B,U,i} \times t_{P,U,B,i} + y_{k,B,D,i} \times t_{P,D,B,i}) \end{aligned} \tag{4.5}$$

$$\begin{aligned} T_{I,D,j} = & \max_{k=1}^{l}(x_{k,D,j} f_k) \\ & - \sum_{k=1}^{l} x_{k,D,j} \left( t_{SK,k,D,j} + t_{RT,k,D,j} + \tfrac{s_k}{B_{D,j}} \right) \\ & - \sum_{k=1}^{l} (z_{k,B,U,i} \times t_{P,U,B,i} + z_{k,B,D,i} \times t_{P,D,B,i}) \end{aligned}$$

(4.6)

The energy overhead from power-state transitions can be calculated by Eqs. 4.7 and 4.8, where $y_{k,B,U,i}$, $y_{k,B,D,i}$, $z_{k,D,U,j}$, and $z_{k,D,D,j}$ are the number of spinning up and spinning down times. $E_{k,B,U,i}$, $E_{k,B,D,i}$, $E_{k,B,U,j}$, and $E_{k,D,D,j}$ are the energy consumed by each power-state transition.

$$e_{P,B} = \sum_{i=1}^{m} \sum_{k=1}^{l} y_{k,B,U,i} \times E_{P,B,U,i} + \sum_{i=1}^{m} \sum_{k=1}^{l} y_{k,B,D,i} \times E_{P,B,D,i} \tag{4.7}$$

$$e_{P,D} = \sum_{j=1}^{n} \sum_{k=1}^{l} z_{k,D,U,j} \times E_{P,D,U,j} + \sum_{j=1}^{n} \sum_{k=1}^{l} z_{k,D,D,j} \times E_{P,D,D,j} \tag{4.8}$$

### 4.3 Performance Evaluation

To evaluate the performance of DARAW scheme, I conducted extensive experiments using various disk I/O traces representing real-world workload conditions. Please note that the experimental results hereafter largely rely on the small writes. The trace file used in our simulation contains several important parameters such as arrival time, data size, cylinder number, targeting data disk, and arrival time. The interval time decides the workload of one trace, which is calculated by Eq.4.9. Here R is a random number between 0 and 1, $\lambda$ is larger than 0, e is the base of natural logarithm and parameter $\lambda$ is used to control arrival rate. The larger $\lambda$ is, the heavier the work load the trace provides.

$$Interval\ Time = -\log_e(R/\lambda) \tag{4.9}$$

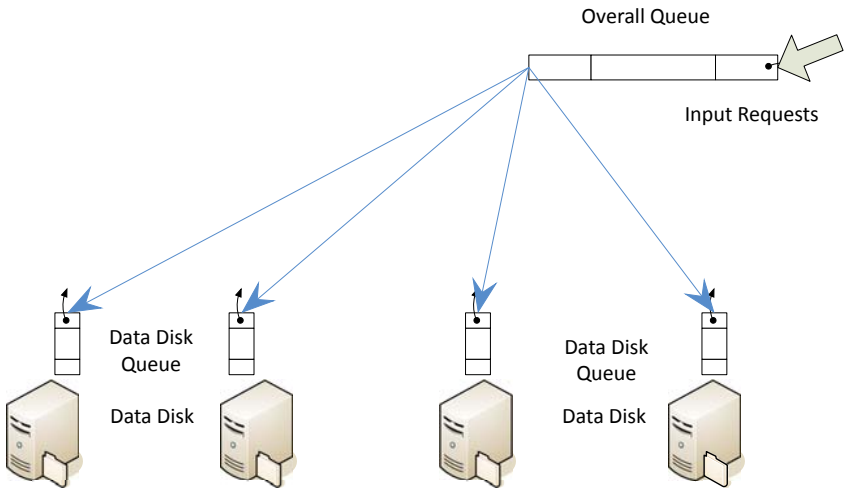


Figure 4.6: Traditional Storage System Structure

The function AT(Ri), which is shown in Eq. 4.10, is defined to generate the arrival time of request i.

$$AT(R_i) = \begin{cases} 0, & i = 0 \\ AT(R_{i-1}) + Interval\ Time, & i \geq 1 \end{cases} \tag{4.10}$$

The architecture of baseline algorithm used for comparison is illustrated in Fig.4, which is essentially the traditional storage system architecture without buffer disks layer. In this algorithm, I will spin up the targeting disks once a request comes. Similarly, when the request is completed, the disk will be immediately spun down unless there are more requests waiting in the queue. Table 4.2 and Table 4.3 summarize the important parameters of two real world disks (IBM 36z15 Ultrastar and IBM 40GNX Travelstar) used in our experiments.

This group of experimental results plots the trend for baseline algorithms in terms of average response time, disk transition times and energy consumption. From Fig. 4.7, I can see that the workload of traces increases significantly as$\lambda$grows. For example, 1000 requests access the storage system in 100,000 milliseconds when $\lambda$=0.01 and 50,000 milliseconds when $\lambda$=0.02. The workload when $\lambda$=0.01 is around twice of the workload of $\lambda$=0.02. An interesting observation from Fig. 4.8 is that the average responding time of higher performance

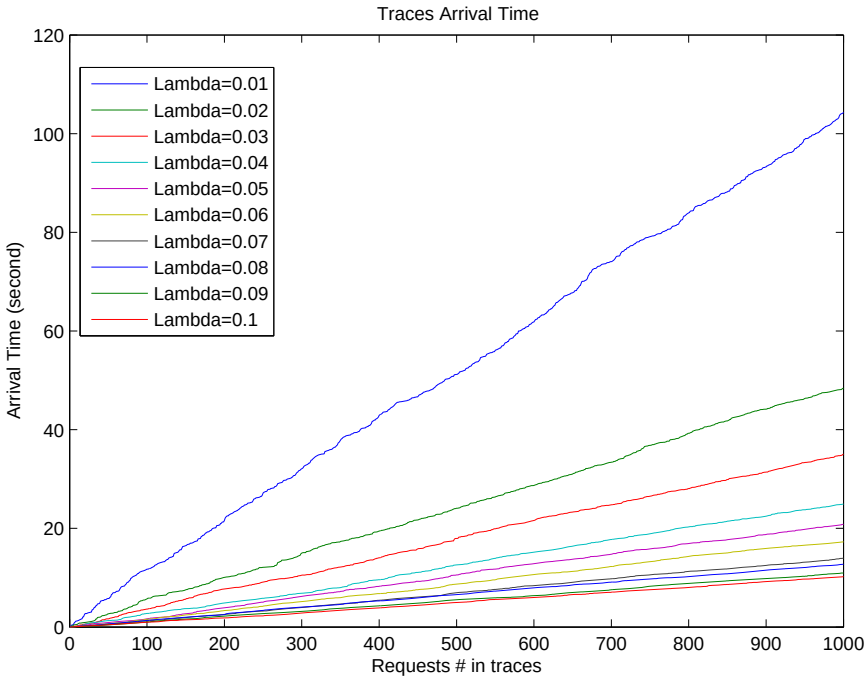


Figure 4.7: Trace Arrival Time

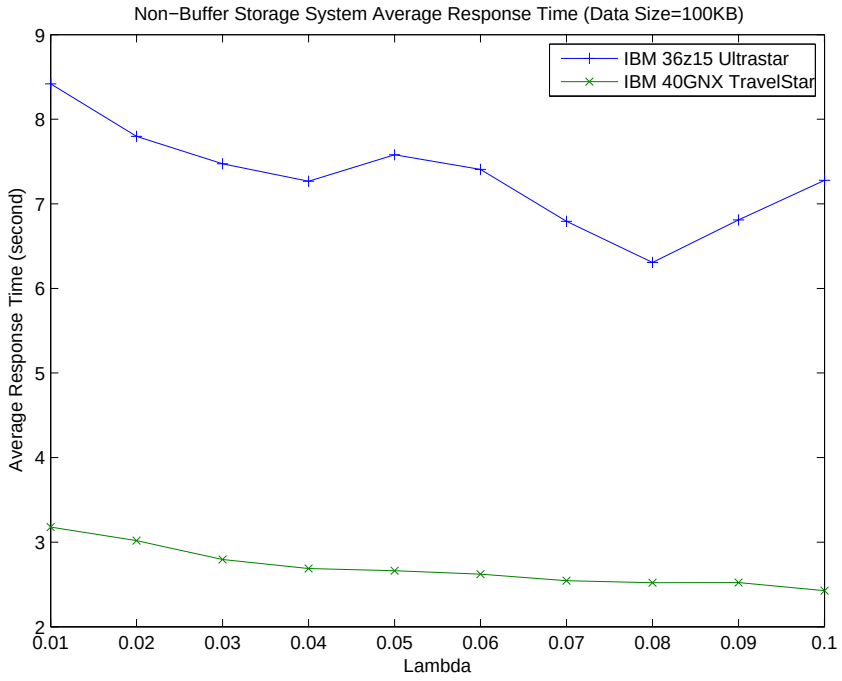


Figure 4.8: Non-BUD Average Response Time

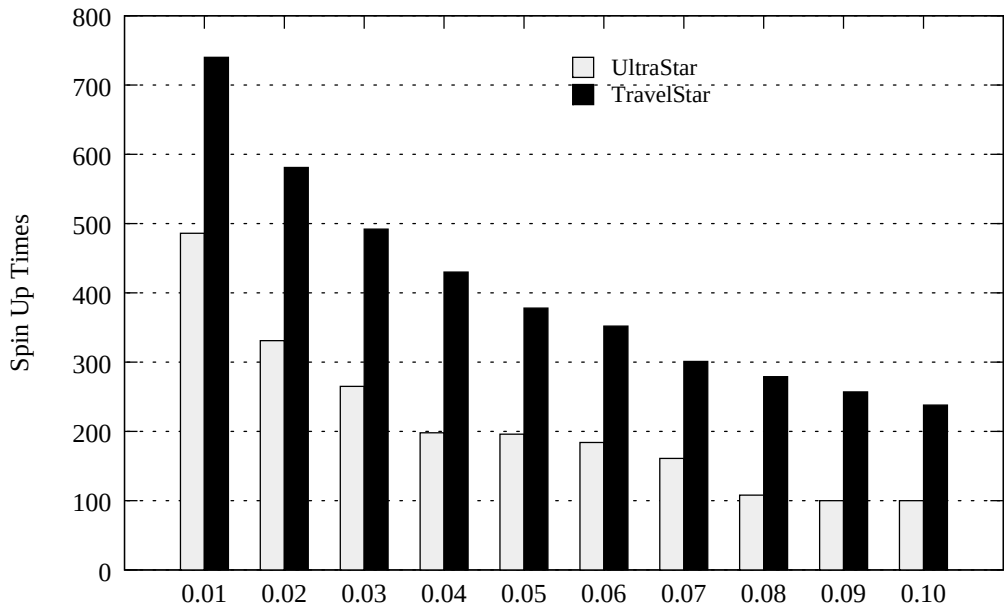


Figure 4.9: Non-BUD Spin Times

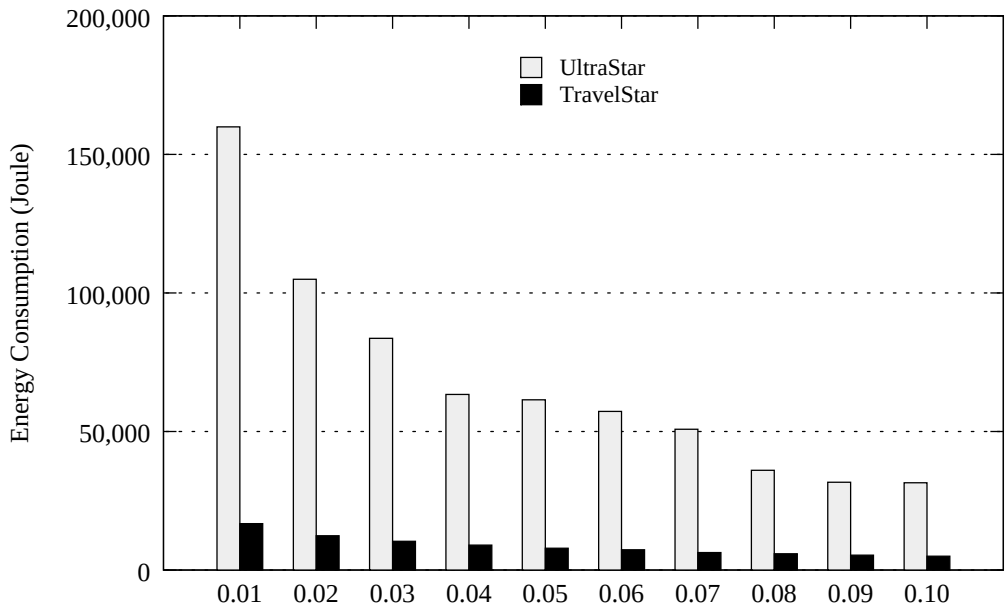


Figure 4.10: Non-BUD Energy Consumption

disk (IBM 36z16 Ultrastar) is even worse than IBM 40GNX Travelstar, which is the low performance disk. The rationale behind is that the spin up and spin down time of Ultrastar is much higher than Travelstar's. Since I generate small writes in our experimental results, the overhead of spin up and spin down are much higher than the time for transmitting data. In other words, the overhead of spin up and spin down dominates the average response time of the storage system. As you can see from Fig. 4.9, the total spin up times of Ultrastar disk are much lower than the spin up times of Travelstar. This is because the spinning up delay of Ultrastar is much longer than the delay of Travelstar. Therefore, a Ultrastar disk has more opportunities to receive another request between the time period of starting to spin up and starting to spinning down. As$\lambda$increases, the average inter-arrival time between each pair of two continuous requests decreases, which will lead to the decrease of spinning up times of both Ultrastar and Travelstar. Fig. 4.10 depicts the energy consumption trend for these two types of disks. Here I have two important observations. First, I find that the overall energy consumption of Ultrastar is higher than that of Travelstar. Second, as$\lambda$increases (i.e. heavy workload), the energy consumption is reduced for both type of disks.

This group of results is to compare our proposed DARAW with the baseline algorithms in terms of average response time, disk transition times and energy consumption. These results are generated based upon the parameters shown in Table 4.5. In this simulation, I use the low performance disks, IBM 40GNX Travelstar, as both buffer disks and data disks. As you can see from Fig. 4.11 and Fig. 4.14, when$\lambda$is small, DARAW algorithm and architecture could save significant energy. The energy conservation rate is as high as 50% to 60%. However, when$\lambda$is getting larger (i.e. workload is getting heavier), there is not much space for energy conservation. This is simply because I do not have enough opportunities to switch disks to low power states when the workload is really heavy.

Fig. 4.13 clearly shows that DARAW can greatly reduce the spinning up and spinning down times. The entire storage system could benefit from reduced transition times because frequent transitions will not only consume huge energy but also degrade the performance.

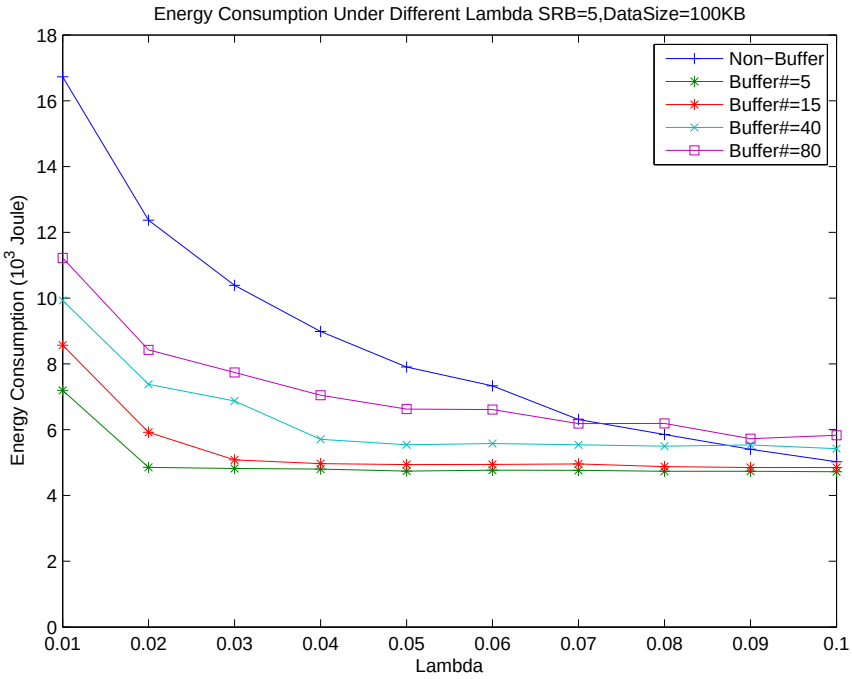


Figure 4.11: BUD Travelstar Energy Consumption

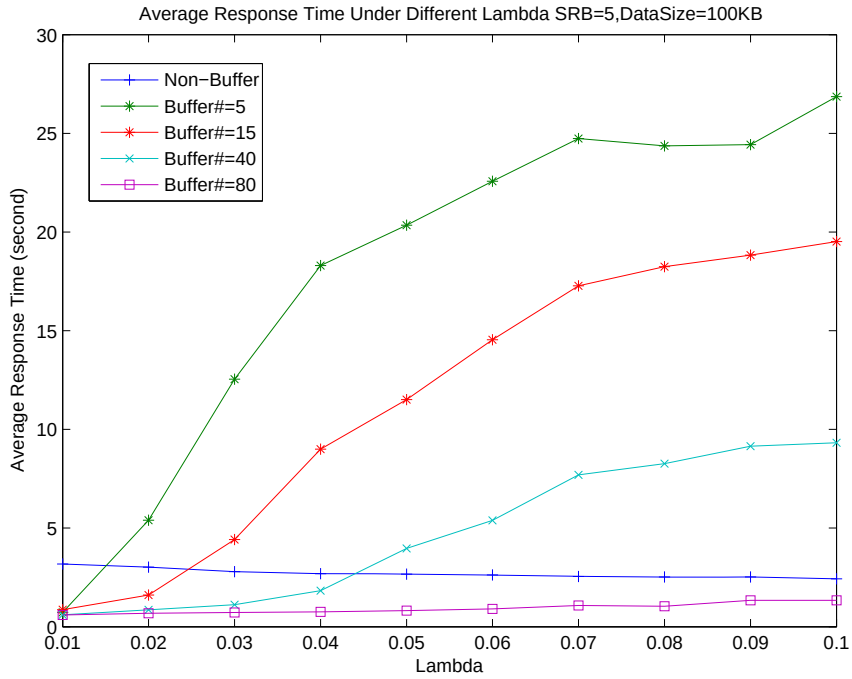


Figure 4.12: BUD Travelstar Average Response Time

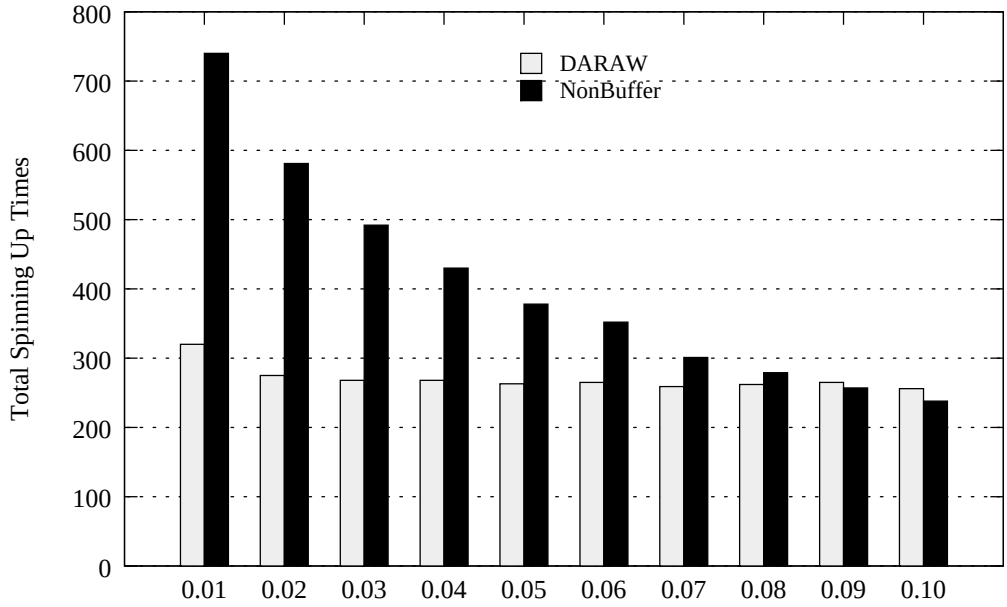


Figure 4.13: BUD Travelstar Spin Times

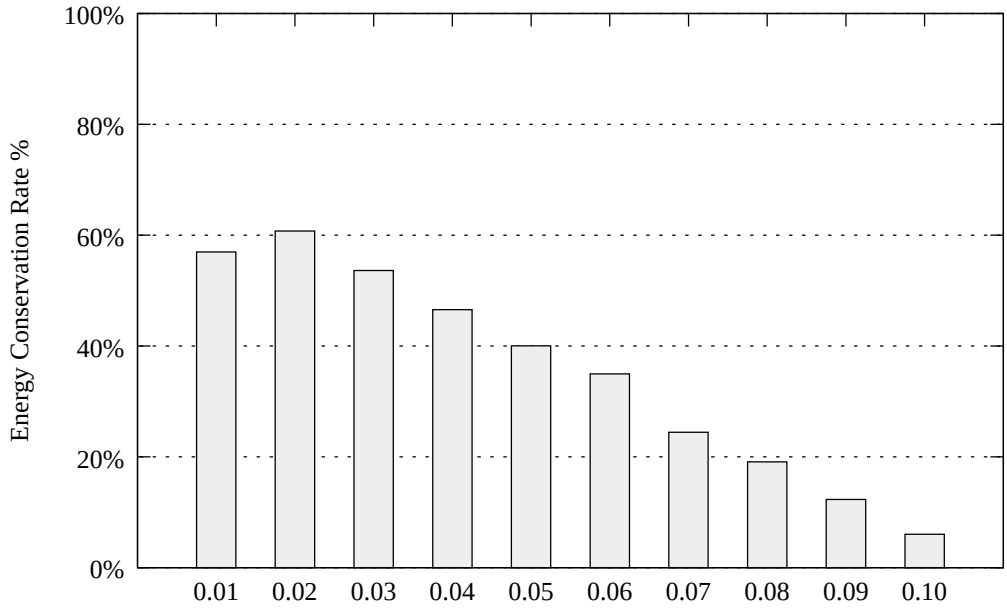


Figure 4.14: BUD Travelstar Energy Conservation Rate

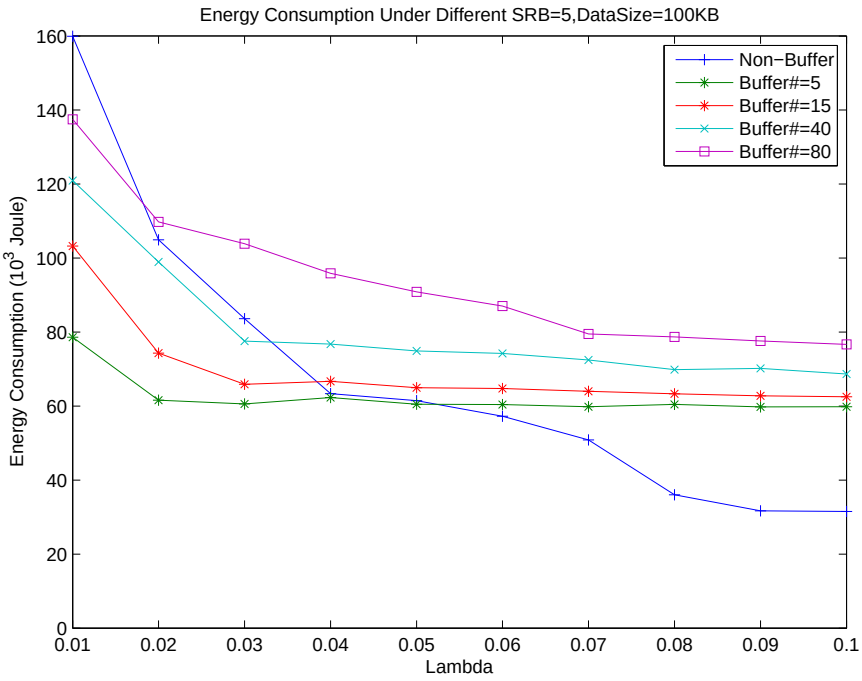


Figure 4.15: BUD Ultrastar Energy Consumption

Fig. 4.11 and Fig. 4.12 show the impact of buffer disk number. In Fig. 4.12, I conclude that the more buffer disks I use, the more bandwidth I can achieve. Consequently, the average response time is decreased. However, Fig. 4.11 also shows that the energy consumption will increase due to large number of buffer disks. There is always a tradeoff between performance and energy conservation. A small number of buffer disks may limit the bandwidth and degrade the performance and a large number of buffer disks will consume more energy. In the worst case, the request waiting queue may overflow when requests' coming speed is faster than the storage systems' writing speed. When the workload is light ($\lambda$=0.01), DARAW can significantly improve both energy conservation rate and average response time compared with the traditional storage systems. However, the overall performance will be degraded when workload is very high for small number of buffer disks. To tackle this problem, we can add more buffer disk thereby increasing the bandwidth and improve the performance. Correspondingly, the energy consumption will be increased as well.

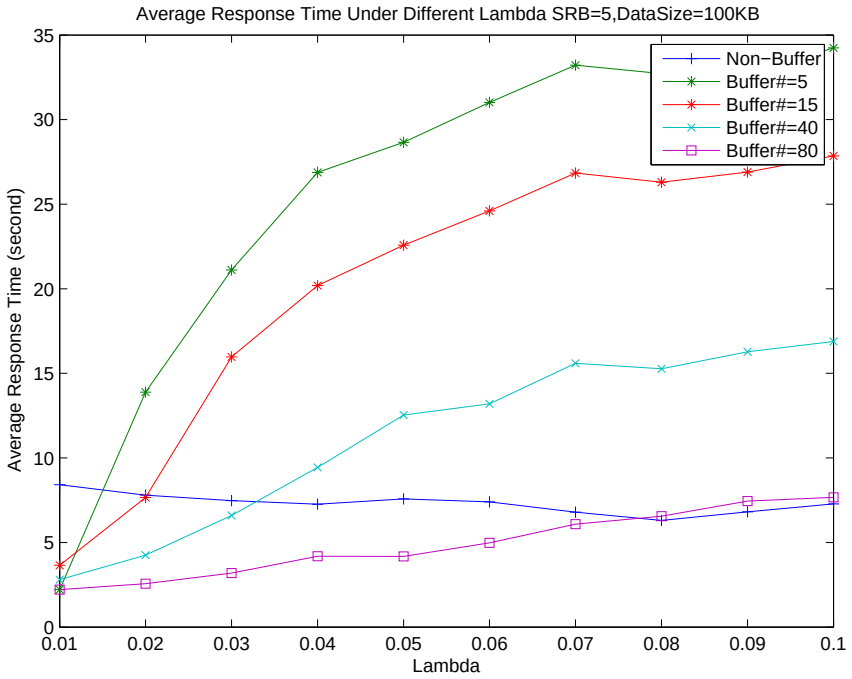


Figure 4.16: BUD Ultrastar Average Response Time

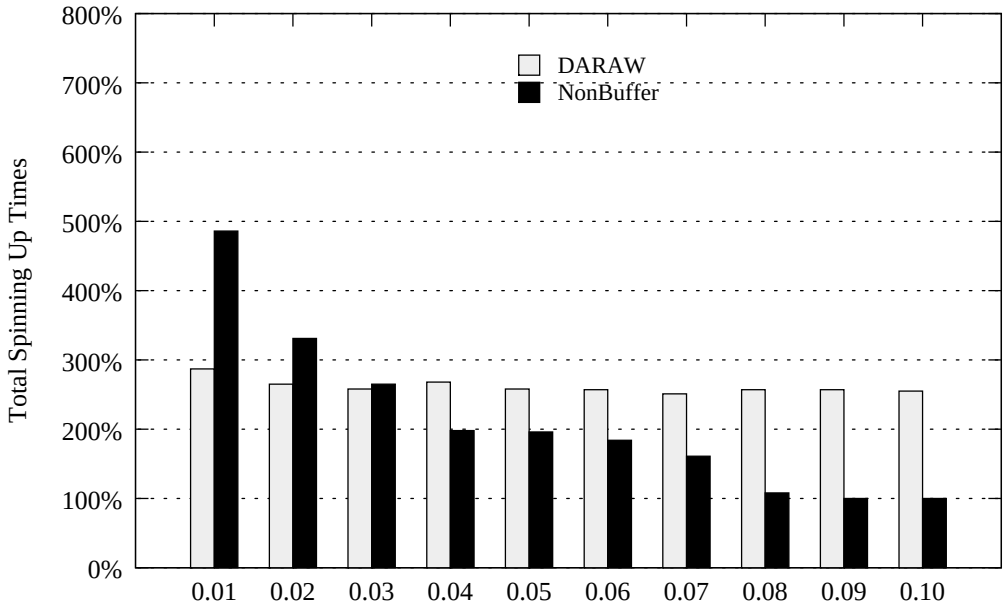


Figure 4.17: BUD Ultrastar Spin Times

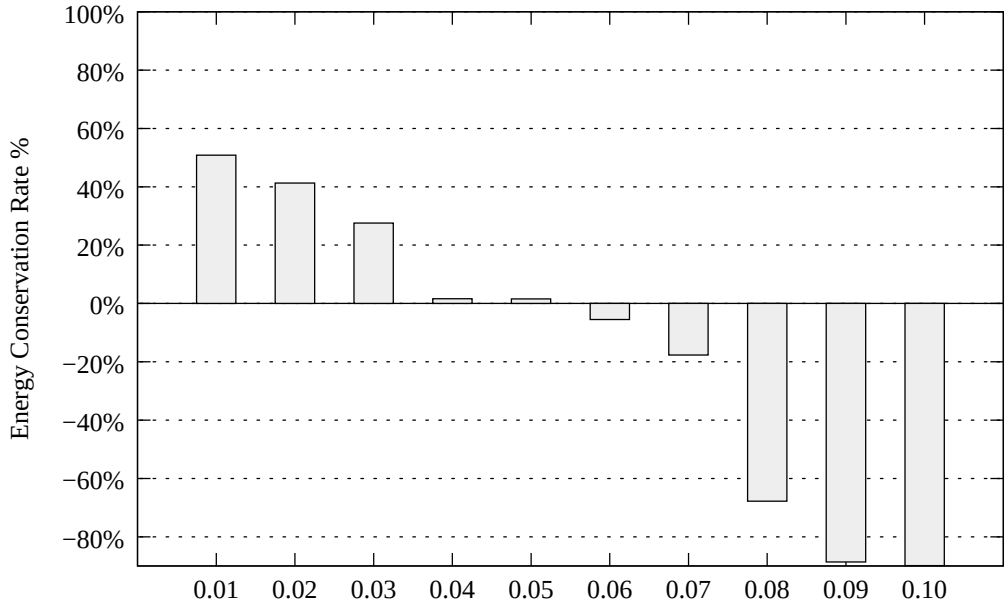


Figure 4.18: BUD Ultrastar Energy Conservation Rate

This group of experimental results are very similar with the previous results expect that we use high performance disks, IBM 36z15 Ultrastar, as both buffer disks and data disks. Fig. 4.15 shows the similar energy consumption trends to Fig. 4.15. The energy consumption trend goes up even faster in Fig. 4.15 because the spinning up and spinning down overhead of IBM 36z15 Ultrastar is even higher than the IBM 40GNX Travelstar's overhead. When$\lambda$is really high, DARAW even consumes more energy than traditional strategy. However, for low work load ($\lambda$=0.01), DARAW is able to conserve roughly about 50% energy compared with the traditional strategy. In addition, the average response time and spinning up times are also reduced. Fig. 4.16 tells us that it is not wise to have large number of buffer disks in DARAW since more buffer disks consume more energy and increase the budget. Actually, when$\lambda$is around 0.01, 1 to 5 buffer disks could serve the requests with impressing performance. That indicates that DARAW will likely be accepted by the market because it will not cost lots of money to purchase many buffer disks.

In this simulation results, I use the high performance disks, IBM 36z15 Ultrastar as buffer disks and low performance disks, IBM 40GNX Travelstar as the data disks. Since I found that in DARAW storage system architecture, buffer disk is the data transmitting

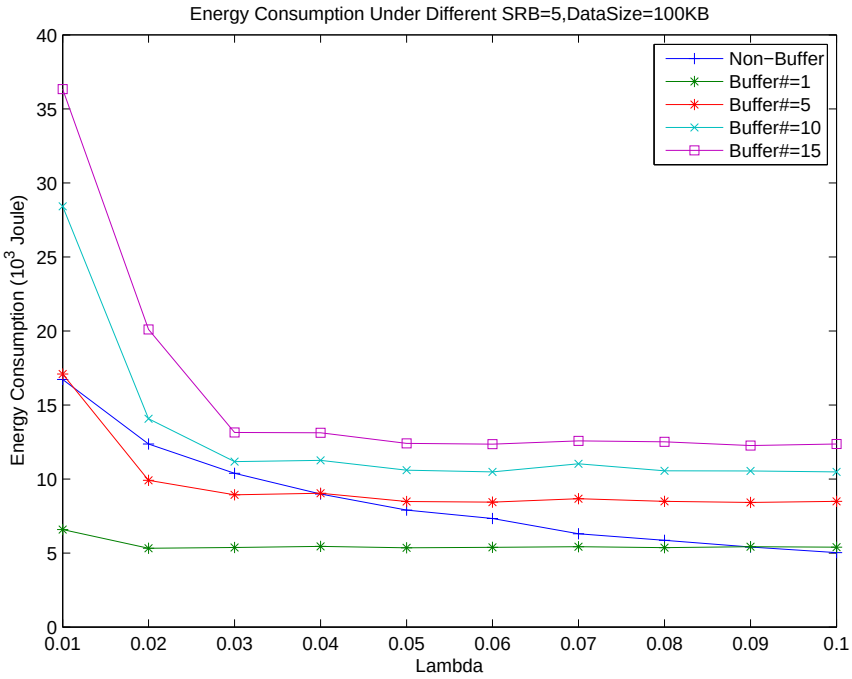


Figure 4.19: BUD Ultrastar Data Disk Travelstar Buffer Disk, Average Response Time

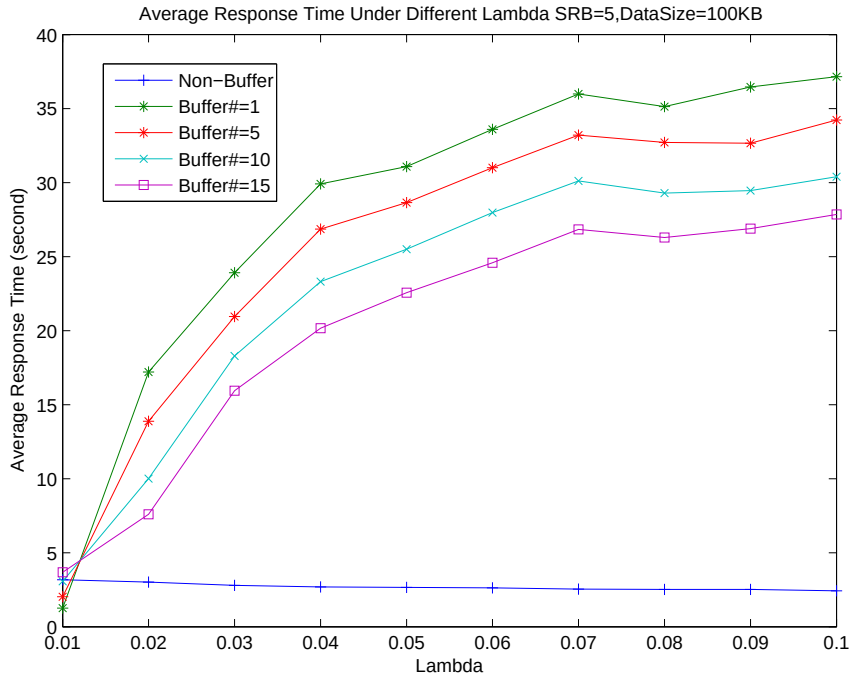


Figure 4.20: BUD Ultrastar Data Disk Travelstar Buffer Disk, Spin Times

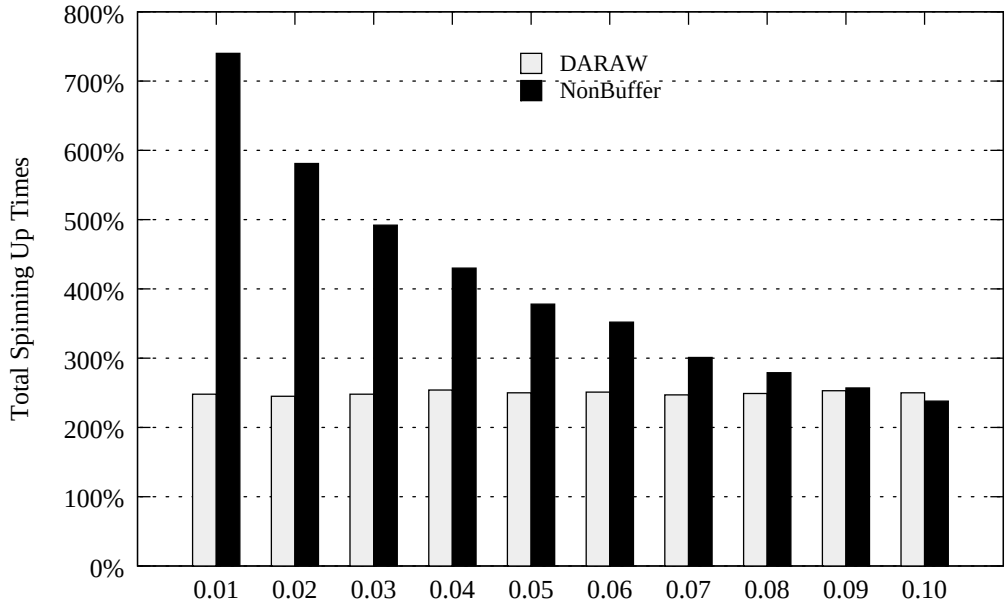


Figure 4.21: Ultrastar Data Disk Travelstar Buffer Disk, Spin Times

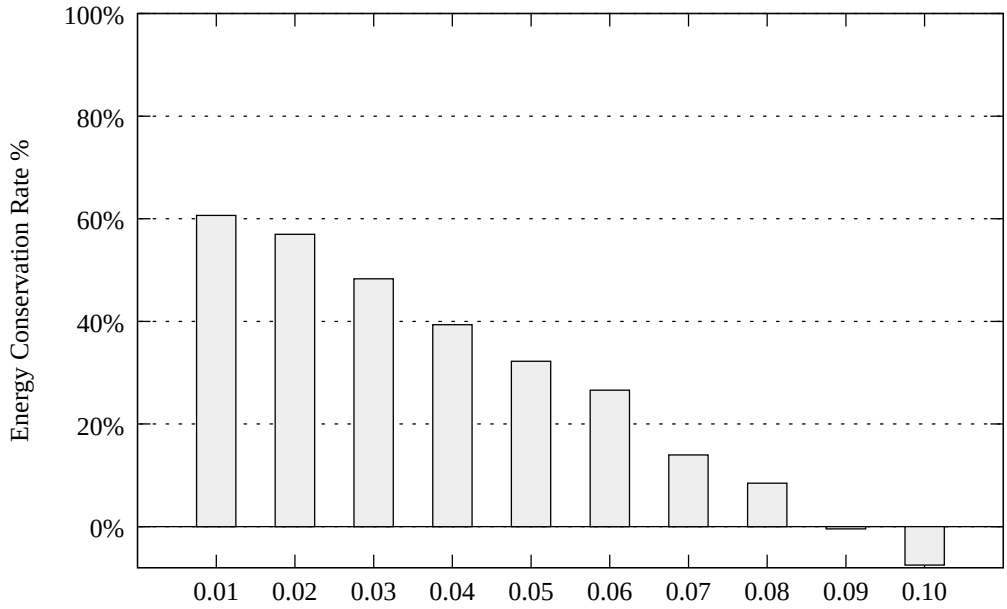


Figure 4.22: Ultrastar Data Disk Travelstar Buffer Disk, Energy Conservation Rate

bottleneck, so I come up with an idea that instead of adding more disks as the same as data disks into buffer layer, I add few higher performance disks into buffer disks. This strategy needs to be applied very carefully because higher performance also means higher energy consumption rate and higher overhead. Hence, I just add very small number high performance buffer disks in the storage system to avoid obvious energy consumption increasing cause by high performance disks. From Fig. 4.19 4.20, and 4.21, I can see that the average response time is reduced by more than 60% if I add one high performance buffer disk and spinning up times are reduced by more than 66%. The energy conservation rate is almost 60% as well.

Figs. 4.11 to 4.22 show that DARAW works well for parallel I/O systems with both high performance disks and mobile disks. DARAW achieves promising results when the arrival rate is low. When the request arrival rate rises, I can either use high-performance hard drives or add more buffer disks to boost I/O performance. If the arrival rate is high, all data disks are busy serving requests, leaving no opportunity to save energy. As the SRB parameter grows, DARAW is given a greater window of opportunity to conserve energy. However, if the SRB is too large, it may cause a "traffic jam" inside the parallel I/O system with buffer disks. In the following part, I am going to present the result I collected from our simulation with different SRC value.

In the part of simulation, I am going to present how the system performance will be affect under different SRB. Figs. 4.23, 4.24, 4.25, 4.26, show the energy consumption and average response time of a parallel disk system with DARAW and the same disk system without DARAW. Because non-buffer disk storage system has no buffer disks and all requests will go to data disks directly, neither response time nor energy consumption will be affect by changing SRB. The results plotted in Fig. 4.23 and Fig. 4.25 indicate that when I increase SRB, more energy can be saved. However, since the simulation I did in Fig. 4.25 has more buffer disks, DARAW needs a larger SRB than in Fig. 4.23 to make its energy consumption less than traditional way. The results were expected since when the SRB grows, the system can write more requests into data disks with reduced number of power state transitions.

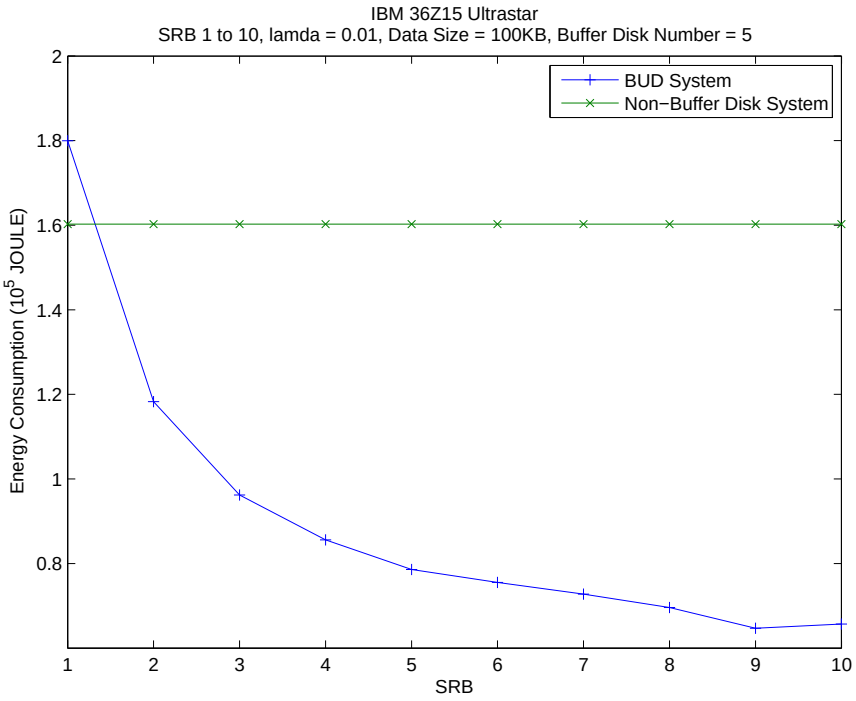


Figure 4.23: IBM 36Z15 Ultrastar. Energy consumption

However, I also observe that when the SRB equals to 1, the energy consumption is even greater than the disk system without DARAW. This interesting tend can be explained as follows. Our parallel disk system has a buffer-disk layer that also consumes energy. If there is insufficient number of requests written into a data disk when a power-state transition occurs, energy conserved cannot offset energy overhead introduced by the buffer disk. When I did the experiment with a trace generated by increasing values of $\lambda$, I observe that energy consumptions in both the non-DARAW parallel disk system and the system with DARAW decrease. Note that all the traces have the same number of disk requests.

This implies the fact that when$\lambda$is high, all requests are arriving at the system within a shorter period of time, making all the disks stay in the active state for a shortened time interval. This is the reason behind the result that energy consumption of the system with DARAW when$\lambda$is set to 0.02 is slightly smaller than that of the system when$\lambda$is 0.01. However, the power consumption of the non-DARAW disk system is significantly smaller when$\lambda$is 0.01 as compared to$\lambda$= 0.02. Once the arrival rate goes up, each data disk in the non-DARAW

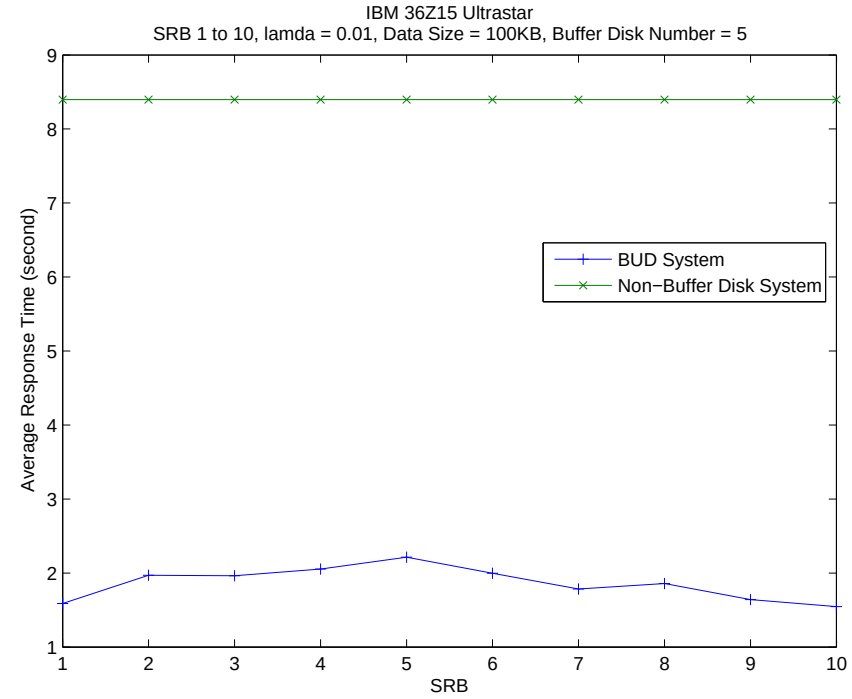


Figure 4.24: IBM 36Z15 Ultrastar. Average response time

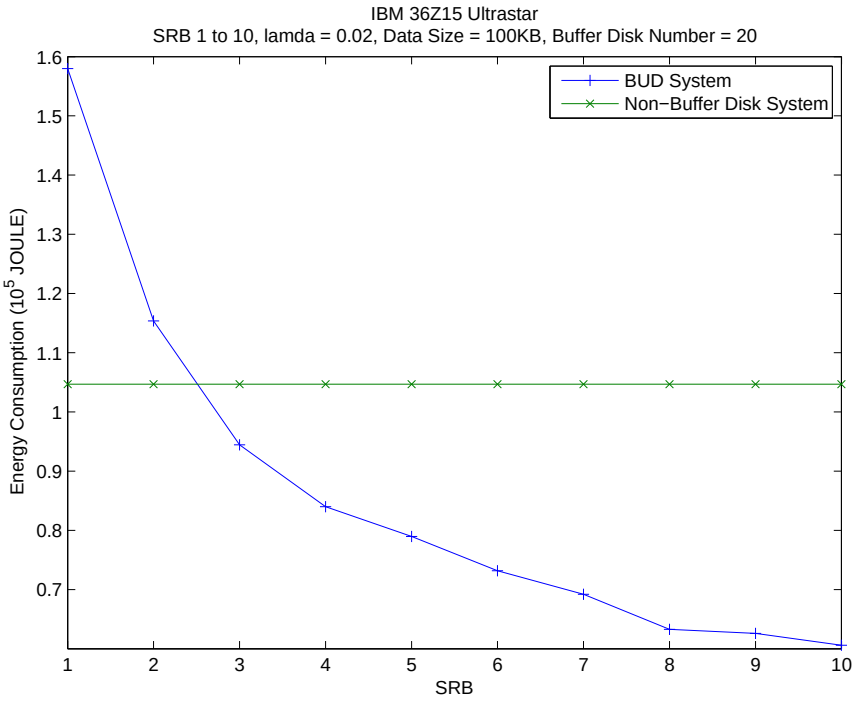


Figure 4.25: IBM 36Z15 Ultrastar. Energy consumption

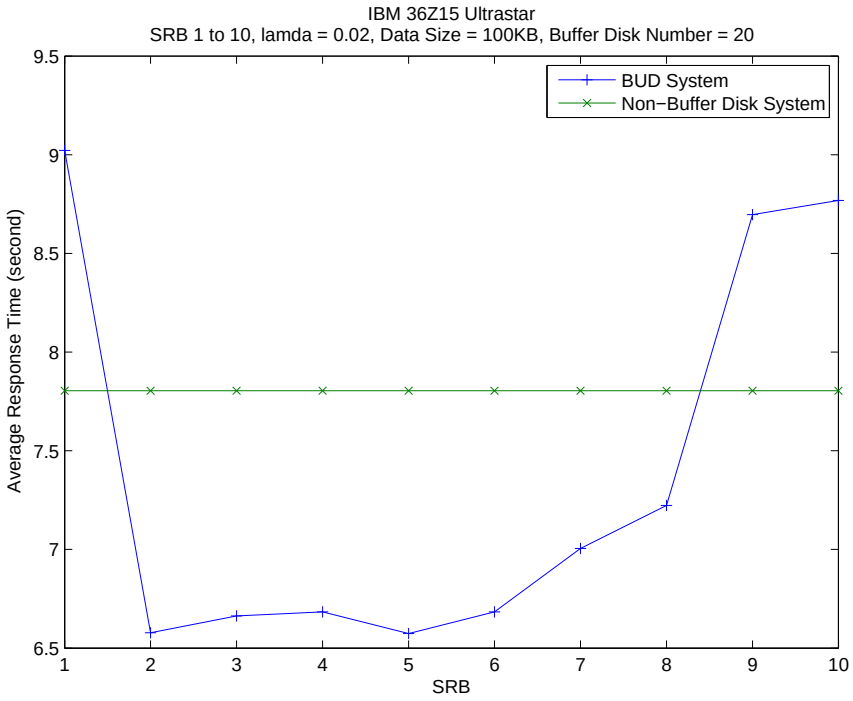


Figure 4.26: IBM 36Z15 Ultrastar. Average response time

system has greater probability to receive a request when it is working. Thus, the number of power-state transitions can be noticeably reduced. When$\lambda$is set to 0.02, there is less of an opportunity to simultaneously save energy and satisfy response times. When I increase the number of buffer disks from 5 to 20, DARAW can conserve energy while guaranteeing reasonably short response times.

An appealing result shown in Fig. 4.23 to Fig. 4.26 is that compared with the parallel I/O system without DARAW, our approach not only achieves significant energy savings, but also reduces response times. This is because when$\lambda$is small (e.g., 0.01), data disks in the non-DARAW parallel I/O system frequently spin up to serve coming requests then immediately spin down, thereby introducing an increased power-transition overhead that leads to longer response times. In DARAW, the response time is the time when a request is written in to a data or buffer disk. Since buffer disks can serve coming requests when data disks are sleeping, the response time can be noticeably shortened.

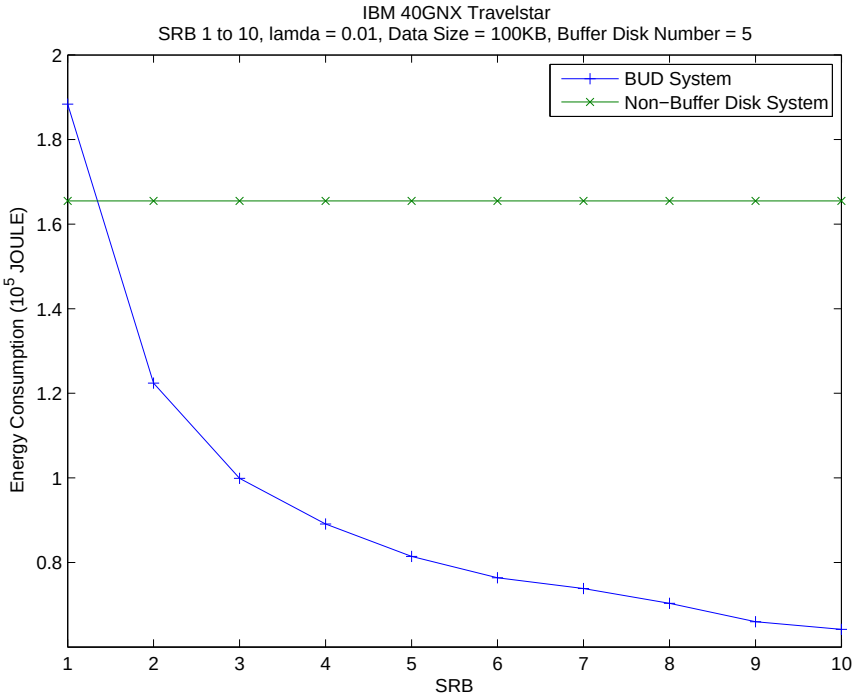


Figure 4.27: IBM 40GNX Travelstar. Energy Consumption

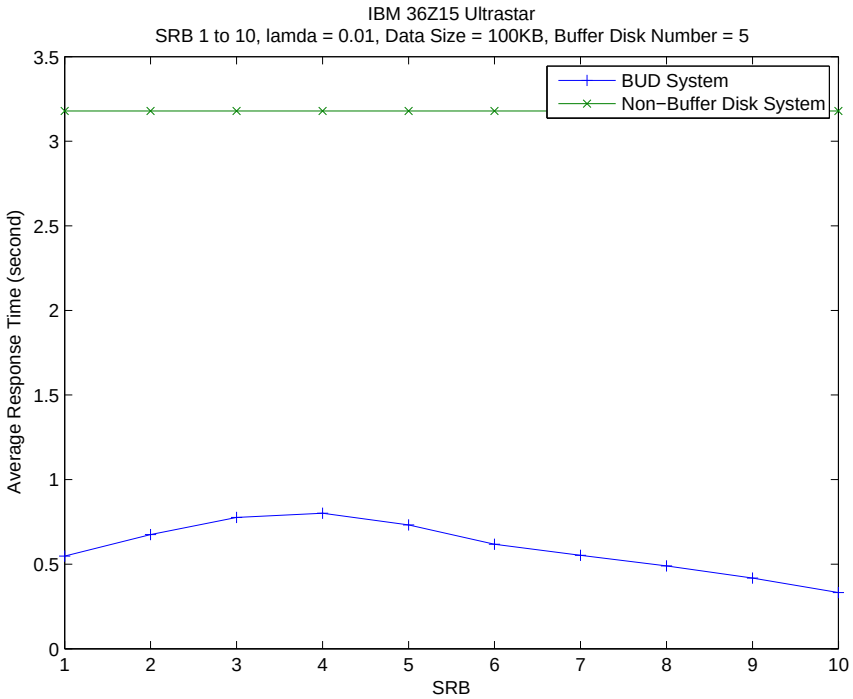


Figure 4.28: IBM 40GNX Travelstar. Average Response Time

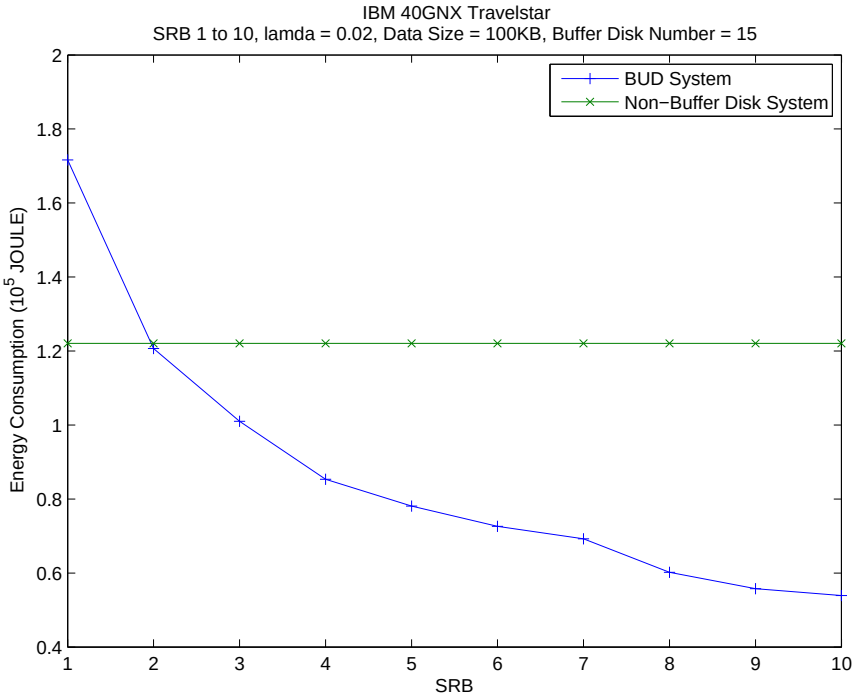


Figure 4.29: IBM 40GNX Travelstar. Energy Consumption

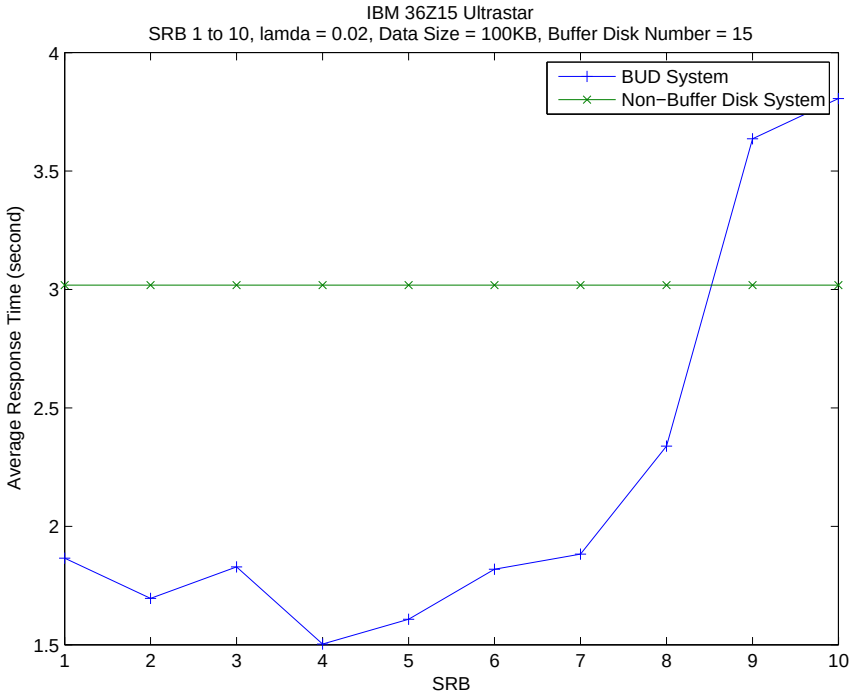


Figure 4.30: IBM 40GNX Travelstar. Average Response Time

There is a very interesting observation concerning the average response time when$\lambda$is fixed to 0.02 and the number of buffer disks is 20. If I use a small SRB, some buffer disks will transition many times to serve the large amount of requests in buffer disks. When the SRB value is increased, buffer disks tend to serve more write requests while keeping data buffered until the next appropriate requests without spinning down. Hence, when SRB is larger, DARAW can save more energy. When SRB is too high, however, an increasing number of requests with the same target data disk are more likely to be served by one buffer disk. This means the requests will be jammed in buffer disk because of the high value of SRB.

IBM 40GNX is a hard disk whose performance is not as high as IBM 36Z1, but its spinning penalty is much smaller. I conducted extensive experiments using IBM 40GNX to evaluate the impacts of DARAW on mobile disks like IBM 40GNX. Experimental results shown in Fig. 4.27 to Fig. 4.30illustrate that DARAW can significantly reduce energy dissipation of parallel I/O systems with mobile disks while providing reasonably short response times.

The trends in energy consumption comparisons in Fig. 4.27 to Fig. 4.30 are similar to those in Fig. 4.23 to Fig. 4.26. Interestingly, when it comes to mobile disks such as IBM 40GNX, significant energy savings can be achieved with a small number of buffer disks. For example, I only need 15 buffer disks, as compared to 20 buffer disks when the IBM 36Z15 disks are used, to save energy and maintain small response times.

It is interesting to observe that since IBM 40GNX does not perform as fast as IBM 36Z15, buffer disks do not experience many power-state transitions under light workload conditions. Therefore, the average response time is short when SRB is small. When SRB is in an ideal range, average response time is acceptable. Once SRB becomes too large, the response time will increase as incoming requests are stuck in buffer queues for longer time intervals.

Fig. 4.23 to Fig. 4.30 show that DARAW works well for parallel I/O systems with both high performance disks and mobile disks. DARAW achieves promising results when the

arrival rate is low. When the request arrival rate rises, I can either use high-performance hard drives or add more buffer disks to boost I/O performance. If the arrival rate is high, all data disks are busy serving requests, leaving no opportunity to save energy. As the SRB parameter grows, DARAW is given a greater window of opportunity to conserve energy. However, if the SRB is too large, it may cause a "traffic jam" inside the parallel I/O system with buffer disks.

## 4.4 Summary

In this research, I first presented the design of parallel I/O systems with buffer disks. To conserve energy in parallel I/O systems serving write requests, I developed an algorithm - dynamic request allocation algorithm for writes or DARAW - to energy efficiently allocate and schedule disk requests. This goal is achieved by making use of buffer disks in parallel I/O systems to accumulate small writes to form a log, which can be transferred to data disks in a batch way. DARAW is able to improve parallel I/O energy efficiency by the virtue of employing a small number of buffer disks to serve a majority of write requests, thereby keeping a large number of data disks in low-power state for longer period times. For each data disk in a parallel I/O system, DARAW keeps track of an important parameter called Sum of Requests in Buffer or SRB, which is the number of buffered requests targeting the data disk. The concept of SRB makes it possible to determine how many buffered write requests should DARAW transfer into the corresponding data disk at one time. When SRB is increased, energy savings and response times may both increase. When response times increase due to high workload, I can either use high-performance hard drives or add more buffer disks to boost I/O performance. To quantify the energy efficiency and performance of DARAW, I carried out experiments using parallel I/O systems with buffer disks. In order to analyze how DARAW works under different workload and different types of hard disks, I did extensive experiments to prove that DARAW could significantly conserve energy and shorten average response time when the workload is low. Experimental results show

that DARAW is conducive to reducing energy dissipation in parallel disk systems while maintaining reasonably low response times. Compared to parallel I/O systems with high-performance disks, parallel I/O systems with mobile disks can achieve better energy efficiency by the virtue of DARAW.

In this research, I focused on parallel I/O systems with homogeneous disks. Currently, I am developing write-buffer schemes to improve energy efficiency of parallel I/O systems with heterogeneous disks.

If workload is so high that it is not necessary to turn off any disk to conserve energy, no strategy is better than keeping all disks on. If workload is low, it is not worthwhile neither spinning up and spinning down for each request, nor keep all disks on to wait coming requests.

Table 4.1: Definitions of Notation

| Notation | Definition |
|---|---|
| $SRB_j$ | Sum of Request targeting at the jth data disk in all buffer disk |
| $SRB_i^j$ | Sum of Requests targeting at the jth data disk in ith Buffer disk |
| $e_{total}$ | total energy consumption of the whole storage system |
| $e_{S,B}$ | energy consumption of buffer disk in active state(serving) |
| $e_{S,D}$ | energy consumption of data disk in active state(serving) |
| $e_{I,B}$ | energy consumption of buffer disk in lower-power state |
| $e_{I,D}$ | energy consumption of data disk in lower-power state |
| $e_{P,B}$ | energy overhead or penalty experienced by buffer disk |
| $e_{P,D}$ | energy overhead or penalty experienced by data disk |
| $x_{k,B,i}$ | 1 if request k is responsed by the ith buffer disk, 0 otherwise |
| $x_{k,D,j}$ | 1 if request k is responsed by the jth data disk, 0 otherwise |
| $P_{A,B,i}$ | active power of the ith buffer disk |
| $P_{A,D,j}$ | active power of the jth data disk |
| $T_{A,B,i}$ | total service time of the ith buffer disk |
| $T_{A,D,j}$ | total service time of the jth data disk |
| $t_{SK,k,B,i}$ | seek time of request k on the ith buffer disk |
| $t_{SK,k,D,j}$ | seek time of request k on the jth data disk |
| $t_{RK,k,B,i}$ | rotational latency of request k on the ith buffer disk |
| $t_{RK,k,D,j}$ | rotational latency of request k on the jth data disk |
| $S_k$ | the data size of request k |
| $B_{B,i}$ | data transfer bandwidth of the ith buffer disk |
| $B_{D,j}$ | data transfer bandwidth of the jth data disk |
| $y_{k,B,U,i}$ | spin up times of the ith buffer disk to finish all k requests |
| $y_{k,B,D,i}$ | spin down times of the ith buffer disk to finish all k requests |
| $z_{k,D,U,j}$ | spin up times of the jth data disk to finish all k requests |
| $z_{k,D,D,j}$ | spin down times of the jth data disk to finish all k requests |
| $E_{k,B,U,i}$ | energy consumption of spinning up the ith buffer disk |
| $E_{k,B,D,i}$ | energy consumption of spinning down the ith buffer disk |
| $E_{k,D,U,j}$ | energy consumption of spinning up the jth data disk |
| $E_{k,D,U,j}$ | energy consumption of spinning down the jth data disk |
| $e$ | the base of natural logarithm |
| $R$ | a random number |
| $\lambda$ | rate parameter in exponential distribution |

Table 4.2: IBM 36z15 Ultrastar

| System Parameter | Values |
|---|---|
| Rotations Per Minute | 10000 RPM |
| Working Power | 13.5W |
| Standby Power | 2.5W |
| Spin up Energy | 135 Joule |
| Spin down Energy | 13 Joule |
| Spin up Time | 10.9 sec |
| Spin down Time | 1.5 sec |
| Transfer Rate | 52.8 MB/s |

Table 4.3: IBM 40GNX Travelstar

| System Parameter | Values |
|---|---|
| Rotations Per Minute | 5400 RPM |
| Working Power | 3 W |
| Standby Power | 0.25 W |
| Spin up Energy | 8.7 Joule |
| Spin down Energy | 0.4 Joule |
| Spin up Time | 3.5 sec |
| Spin Down Time | 0.5 sec |
| Transfer Rate | 25 MB/s |

Table 4.4: Experimental Values for Baseline Experiment

| System Parameter | Values |
|---|---|
| Disk Type | IBM 36Z15 and IBM 40GNX |
| $\lambda$ | 0.010.10 |
| Data Size/request | 100KB |
| SRB | 5 |
| Buffer Disk Amount | 0 (NonBuffer Disk System) |
| Data Disk Amount | 100 |
| Trace Size | 1000 requests |

Table 4.5: Experimental Values for Both Buffer disks and Data Disks are low performance disks

| System Parameter | Values |
|---|---|
| Disk Type | IBM 40GNX Travelstar |
| $\lambda$ | 0.010.10 |
| Data Size/request | 100KB |
| SRB | 5 |
| Buffer Disk Amount | 0, 5, 15, 40,80 |
| Data Disk Amount | 100 |
| Trace Size | 1000 requests |

Table 4.6: Experimental Values for Both Buffer disks and Data Disks are high performance disks

| System Parameter | Values |
|---|---|
| Disk Type | IBM 36Z15 Ultrastar |
| $\lambda$ | 0.010.10 |
| Data Size/request | 100KB |
| SRB | 5 |
| Buffer Disk Amount | 0 , 5, 15, 40, 80 |
| Data Disk Amount | 100 |
| Trace Size | 1000 requests |

Table 4.7: Experimental Values for High Buffer disks And Low Data Disks

| System Parameter | Values |
|---|---|
| Disk Type | IBM 36Z15 Ultrastar IBM 40GNX Travelstar |
| $\lambda$ | 0.010.10 |
| Data Size/request | 100KB |
| SRB | 5 |
| Buffer Disk Amount | 0, 1,5, 10, 15 |
| Data Disk Amount | 100 |
| Trace Size | 1000 requests |

Chapter 5

An Energy-Efficient Cluster Storage System

## 5.1 Introduction

Cluster storage systems - essential building blocks in many high-performance computers - have been widely adopted to support data-intensive applications running on high-performance computing platforms. Optimizing energy consumption in cluster storage systems has strong impacts on the cost of backup power-generation and cooling equipment in cost-effective cluster computing infrastructures. I was motivated to address the energy saving issues in cluster storage systems, because a significant fraction of the operation cost of data centers is due to energy consumption in storage systems. For example, the average power consumption of TOP 10 supercomputing systems is 1.32 Mwatt, in which a large portion is contributed by storage systems [5]. Dell Texas Data Center reported that 37 percent of the energy consumed by supercomputers is cost by storage systems [4]. In addition to emerging high-performance disk drives with high power needs, increasing storage requirements imposed by data-intensive applications make it desirable to design energy-efficient cluster storage systems. Several novel techniques proposed to conserve energy in storage systems include dynamic power management schemes [27] [66], power-aware cache management strategies [118], power-aware prefetching schemes [105], software-directed power management techniques [107], redundancy techniques [87], and multi-speed settings [43] [46] '[60]. A few innovative techniques have been developed to substantially reduce energy dissipation in traditional server clusters [86] [55] [18] [13] [31] [32]. However, the research on the improvement of energy efficiency in cluster storage systems is still in its infancy. It is imperative to develop new cluster storage systems that can exhibit high energy efficiency and I/O performance for high-end data-intensive computing.

In this Chapter, I detail an approach to implementing an energy-efficient cluster storage system called ECOS. To achieve high aggregate I/O bandwidth under heavey workloads, I design a clsuter storage system where each I/O node embraces multiple disks - one buffer disk and several data disks. The basic idea behind ECOS is to redirect disk requests from data disks to buffer disks within I/O nodes. Redirecting requests to buffer disks is a driving force of energy saving, because I/O load is skewed toward buffer disks so that data disks can be placed into standby in a long time period to conserve energy. Spinning down/up disks inevitably introduce extra energy overhead. As such, adding a buffer disk in each I/O node aims to reduce the number of disk spin downs/ups. To balance I/O load among I/O nodes, ECOS attempts to redirect disk requests from a heavily loaded I/O node into other I/O nodes with light load.

The ECOS storage system treats read and write requests differently. Let us first outline the approach to processing reads in ECOS. Then, I summarize how ECOS handles writes in an energy efficient way. Given an I/O node with one buffer disk and multiple data disks, ECOS dynamically fetches popular blocks from data disks into the buffer disk. The prefetching strategy in ECOS was developed in the recognition of data access history and power modes of disks. Since read requests can be serviced from both buffer disks or data disks, a consistency mechanism was created in ECOS to ensure that the data cached in the buffer disks is consistent with the original data stored in the data disks. Each buffer disk performs as a cache of popular data and an LRU (Least-Recently-Used) policy was implemented as a replacement scheme. If a replace data is clean, it can be simply discarded. Otherwise, the data must be written back to its home data disk.

One of our recent studies was focused on an algorithm - DARAW - handling writes in parallel storage systems with buffer disk [97]. Having been extended to deal with writes in the context of cluster storage systems, the DARAW algorithm was implemented as a core component in the ECOS system. When it comes to large write requests (e.g., larger than 500MB), data should be be issued directly to data disks. In contrast, small write requests

have to be sent to an active buffer disk. Once the data of a write request is transferred to buffer or data disks, an acknowledgement is returned to an application that issued the request.

When a buffer disk in an I/O node is overloaded, then the corresponding data disks within the I/O node need to be spinned up to balance I/O accesses. The challenging issue in this component of the research is to determine the optimal number of standby data disks to be activated in respond to high I/O traffic. The goal is to spin up as few data disks as possible, keeping the utilization of each disk below 100 percent.

MAID [21], PDC [85], [110], and BUD [97] [74] - four existing energy-efficient parallel disk systems - are conducive to achieving high energy efficiency with a small fraction of I/O delays. Our ECOS system is fundamentally different from these parallel storage systems, because ECOS is a cluster storage system with loosely-coupled parallel disks cross multiple I/O nodes whereas the other four systems contain tightly-coupled parallel disks (e.g., disk arrays).

Compared with other disk energy conservation techniques, the ECOS cluster storage sytsem has the following three unique features.

• First of all, it has no need to modify data-intensive applications when they are ported from traditional cluster storage systems to ECOS.

• Second, there is no necessity to add extra hardware such as flash drives into cluster storage systems.

• Third, ECOS maintains an acceptable level of I/O performance by the virtue of parallel buffer disks across multiple I/O nodes.

A prototype of ECOS was implemented in a Linux cluster, where each I/O node contains one buffer disk and two data disks. The power manager in ECOS relies on a system call in the Linux kernel to spin down and spin up disk drives. Experimental results show that ECOS improves the energy efficiency of traditional cluster storage systems without using buffer disks. Adding one extra buffer disk into each I/O node seemingly has negative impact

on energy saving. Interestingly, our results indicate that ECOS equipped with extra buffer disks is more energy efficient than the same cluster storage system without the buffer disks. The implication of the experiments is that using existing data disks in I/O nodes to perform as buffer disks can achieve even higher energy efficiency.

In summary, the main contributions of this study are:

• I designed an energy-efficient disk architecture to reduce energy dissipation in cluster storage disk systems;

• I developed a disk power model for cluster storage systems; and

• I implemented an energy-efficient cluster storage system that consists of modules like disk request processing, data movement, data replacement, and power management for I/O nodes.

The remainder of this Chapter is organized as follows. After the presentation of an energy-efficient architecture for cluster storage systems, Section 5.2 details our evaluation methodology and a testbed used to implement ECOS. Section 5.3 presents experimental results. Finally, Section 5.4 concludes this research with future research directions.

## 5.2 Design and Implimentation of ECOS

A cluster storage system is comprised of an array of I/O nodes connected by a high-speed network. In recent years, most research efforts on reducing energy consumption in parallel disk systems. However, the issue of using buffer-disk architectures to reduce energy consumption in cluster storage systems is not well investigated. Our long-term goal is to develop fundamental techniques to save energy of large-scale cluster storage systems. The objective of this study, which is paving a way towards that goal, is to design and implement the ECOS system - an energy-efficient cluster storage system in which disk request processing, data movement/placement strategies, power management, and prefetching schemes are holistically integrated to save energy. The rationale for this study is that the development of

ECOS will promote more energy-efficient resource management techniques for storage systems in general and cluster storage systems in particular. In this section, I first detail design issues including the system architecture and hardware configuration of the ECOS storage system. Then, I describe various implementation issues in ECOS.

### 5.2.1 Detailed Design

The architecture of ECOS (see Fig. 5.1) is an extension of the architecture of traditional cluster storage systems, where each I/O node manages one local disk. Like the traditional cluster storage systems, ECOS has large files striped across a number of I/O nodes connected through a high-speed network. Since each I/O node in ECOS contains a buffer disk and multiple data disks, files might be distributed across a number of disks within one I/O node. All the compute nodes in the system can directly access I/O nodes through the nextwork.

**Hardware Configurations.** Disk I/O parallelisms can be provided in forms of inter-request and intra-request parallelism. Inter-request parallelism allows multiple independent requests to be served simultaneously by multiple I/O nodes in ECOS, whereas intra-request parallelism enables a single disk request to be processed by multiple I/O nodes in parallel. A parallelism degree of a data request is the number of I/O nodes to which the requested data is striped. In the design of ECOS, I consider two different configurations to deal with inter-request and intra-request I/O parallelisms, respectively.

The first ECOS configuration - aiming to support inter-request parallelisms - consists of four major components: a RAM buffer residing in compute nodes, $m$ buffer disks, $n$ data disks, and an energy-aware buffer-disk controller. The RAM buffer with a size ranging from several megabytes to gigabytes is residing in the main memory. The buffer-disk controller carefully coordinates disk request processing, data movement/placement strategies, data striping, power management, and prefetching schemes. Please refer to the next subsection for details of how the controller is developed. It is to be noted that in most cases, the number

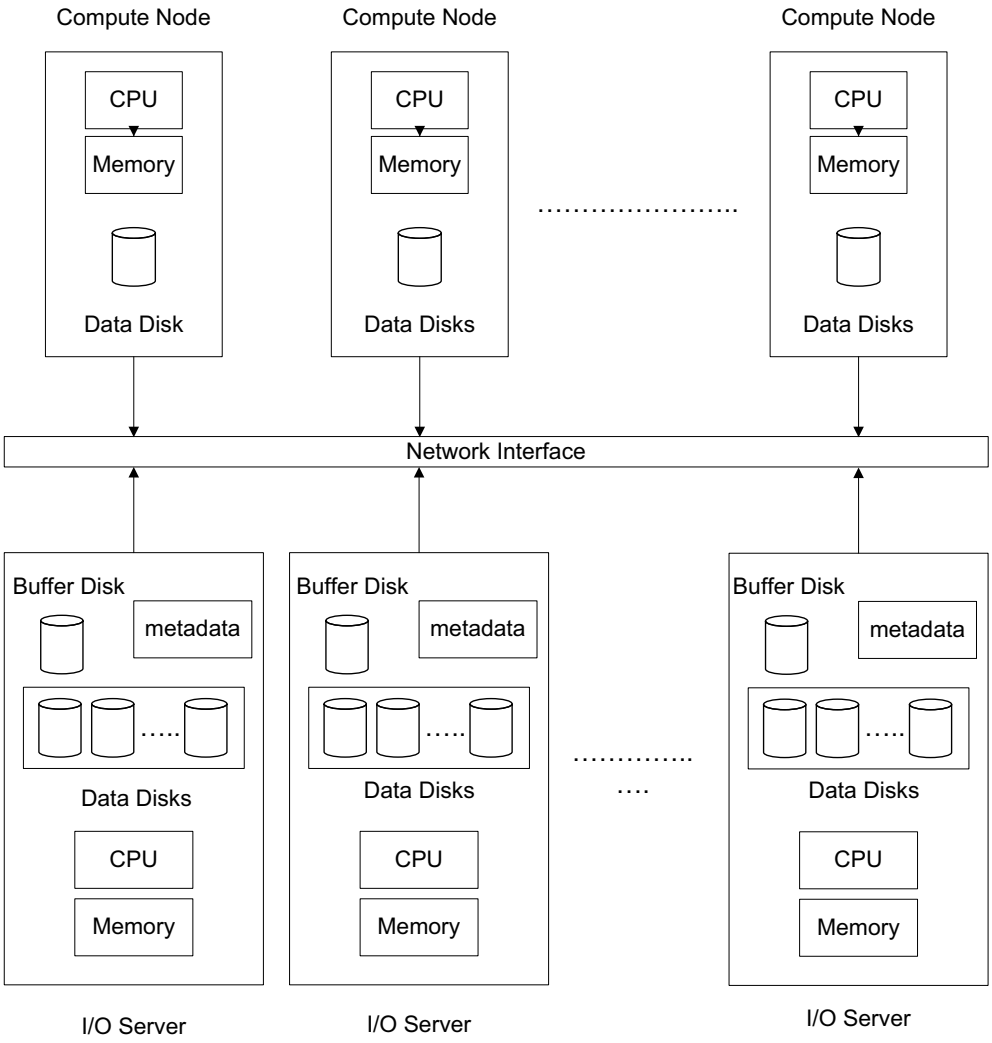


Figure 5.1: The architecture of ECOS - an energy efficient cluster storage system. Each I/O node in ECOS contains a buffer disk and multiple data disks. Large files are striped across a number of I/O nodes connected through a high-speed network. Alternatively, large files might be distributed across a number of data disks within one I/O node.

of buffer disks $m$ is smaller than the number of data disks $n$, and values of $m$ and $n$ are independent of one another for workloads with inter-request parallelisms.

The second ECOS configuration is designed for disk workloads with intra-request parallelisms. This configuration is similar to the previous one except that all the data disks in ECOS are conceptually partitioned into $k$ groups of parallel disks each of which has $m$ disk drives. Given $m$ buffer disks, the second configuration is capable of serving disk requests with parallelism degrees as high as up to $m$.

There are two general ways of placing buffer disks. In the first approach, each I/O node only contains a single buffer disk serving all the other data disks within the I/O node. Alternatively, all the buffer disks can be grouped and placed into one or more I/O nodes. I/O nodes containing only buffer disks are called buffer I/O nodes; I/O nodes equipped with data disks are referred to as data I/O nodes. Comparing these two approaches, I advocate for the first one and the reason is two-fold. First, large popular files can be striped across multiple buffer disks residing in multiple I/O nodes. In doing so, any I/O node is most unlikely to become a performance bottleneck. Second, a buffer disk within an I/O node can dedicate to data disks within the same I/O node, eliminating unnecessary communications between the buffer disk and data disks in other I/O nodes. Enforcing buffer disks to serve data disks within the same I/O node can reduce data transfers among I/O nodes through the network. As a result, evenly placing buffer disks across all the I/O node not only can achieve high aggregate I/O bandwidth, but also can improve network performance by reducing network traffic.

**Buffer Disk Controller.** The buffer disk controller, a centerpiece in the ECOS storage system, critically affects the overall performance and energy efficiency of I/O nodes. Therefore, I address several challenging issues related to the design of an energy-aware buffer disk controller. The buffer disk controller be designed and implemented to achieve the following specific goals. First, the buffer disk controller aims to minimize the number of active buffer

disks while maintaining reasonably quick response times for disk requests. Second, the controller has to energy-efficiently deal with read and write requests issued to I/O nodes. Third, the controller must move data from buffer disks to home data disks in an energy-efficient way. Fourth, the controller is intended to incorporate an energy-aware prefetching strategy to dynamically fetch the most popular data into buffer disks, thereby allowing most data disks to be in the sleep mode to save energy. Design issues of the energy-aware buffer disk controller are discussed as follows.

**Disk Request Processing.** Recall that ECOS treats large and small writes in different ways. Large write requests are issued directly to data disks, whereas small write requests may be redirect to the corresponding active buffer disk if the home data disk is placed in standby. Ideally, the buffer disk handling the small write requests and their home data disks should reside in the same I/O node. If requested data of a read operation is not in the RAM buffer in a compute node, the request must be processed by I/O nodes in ECOS. In this case, the result data of the request must be returned from one or more buffer disks if the data is residing in the buffer disks. Otherwise, the request will have to be passed on to the home data disks. If the home data disks are in the active mode, then the read request can be quickly responded. In cases where the corresponding data disks are in the standby state, it is not an energy-efficient way to immediately wake up the standby data disks to handle a single read request. Immediately spinning up data disks upon the arrival of a single read request ultimately result in a large number of power state transitions while shortening idle times.

To provide energy savings, ECOS aims to judiciously change power states of data disks to handle read requests. To make the best tradeoffs between energy conservation and quick response times, the power management strategy in ECOS make an effort to enable data disks to stay in the standby state for longer periods of time by clustering read requests together and providing long disk idle times. This goal is achieved by delaying the responses

of incoming read requests if their home data disks are standby. Specifically, I consider the following two scenarios when . First, if read requests arrive to data disks at the time when the data disks are being accessed by other disk requests, the newly arrived read requests will be inserted to the waiting queues of data disks. Second, when read requests arrive to data disks that are placed in the standby mode, the processes of the read requests are delayed for a certain amount of time as long as the requestsquality of service can be guaranteed.

**Data Movement Strategies.** There are two advantages of buffering data first in buffer disks and moving the data to data disks later on. First, only buffer disks need to stay active to efficiently process a large number of write requests during bursty periods. Active buffer disks make it possible for data disks to be kept in the standby state for long period of time to conserve energy. Second, when buffer disks are sitting idle or less busy, buffered data can be moved from buffer disks to home data disks. Thus, busty access patterns are created for data disks in a way many disk requests are clustered together. The data disks can effectively handle many data requests issued from the buffer disks during the active periods.

Data movement strategies determine conditions under which data movement processes are initiated by spinning up home data disks. Data movement strategies play an important role in the ECOS storage system; therefore, I designed a data movement mechanism to energy-efficiently move data from buffer disks back to home disks. Our initial design is straightforward. Thus, buffered data sets will be moved back to their home data disks when buffer disks are sitting idle. To determine whether a buffer disk is sitting idle, I simply detect if there is any request waiting in the processing queue of the buffer disk. This data movement strategy is suitable for disk traffic where there is a period of long idle time between two subsequent request bursts.

**Data Placement.** Data placement, allocation of all popular files into buffer disks, can

significantly affect energy efficiency and performance of ECOS. To fully exploit the capacity of parallel I/O, researchers have extensively investigated data placement algorithms for parallel disk systems. Conventional wisdom in the design of data placement mechanisms is to minimize a cost function while allocating data onto an array of independent disks. Most cost functions were focused on performance metrics (e.g., mean response time), ignoring the issue of energy saving. It is appealing to design data placement strategies to achieve high energy efficiency and quick response times in the context of cluster storage systems.

Son *et al.* investigated disk layout algorithms to reduce energy consumption in disks [104]. Their algoirthms decide the most appropriate set of disks to store a given disk-resident array, allowing other disks to run in a low speed. Son's disk layout approach is capable of reducing disk systems supporting array-based scientific applications. To apply *et al.*'s algorithms, one need to modify the source code of applications to make the applications be aware of disk layout information. Our data placement scheme are orthogonal to the existing strategy in that combining ours with Son *et al.*'s disk layout approach can further reduce energy consumption caused by array-based scientific applications.

Let us consider the problem of moving $q$ popular files $f_1,\ f_2,\ \ f_q$ from data disks into $m$ buffer disks of in ECOS. The data placement mechanism relies on an online access rate monitor, which estimates the mean access rate $\lambda_i$ of file $f_i$ $(1 \leq i \leq q)$. Compared with large files, small files usually have higher access rates. The data placement issue in ECOS can be represented as a partition problem of a set $F = 1, \cdots, m$ designated as $\{F_1, F_2, \cdots, F_m\}$, where $F_j$ is a set of files placed in the buffer disk of the $j$th I/O node. The data placement process aims to solve the partitioning problem while enhancing energy efficiency and reducing I/O response time. The data placement mechanism helps in improving I/O perofrmance by fetching files with similar access rates into the same buffer disk. More importantly, this mechanism is energy efficient because buffer disks holding files with low access rates might be placed into the standby mode to conserve energy.

The data placement algorithm performs. First, all the popular files in $F$ are sorted in descending order of their access rates $\lambda_i$. Next, the algorithm assigns a contiguous files of $F$ to the buffer disk in an I/O node until its utilization reaches a threshold. This file assignment process is repeatedly executed until all files in $F$ are placed into buffer disks. The data placement mechanism can be periodically accutated to place the most popular data in buffer disks. Choosing the most appropriate value for the period largely depends on dynamically changing I/O workloads.

### 5.2.2 Implementation Issues

The software modules of ECOS were built in Ubuntu 8.04, where the Linux kernel version is 2.6.24. I chose to use Linux as the runtime environment, because Linux provides system calls to spin down/up disks.

**Data Transfer.** The communication module that is in charge of data transfer among I/O nodes and compute nodes in the tested cluster was implemented using the TCP/IP protocol. In each I/O node, there is a daemon process that performs the following functions. First, upon the arrival of an incoming request sent from a compute node, the daemon process creates a process receiving data from the compute node while waiting for other incoming requests. Second, the daemon checks the power status of all the disks within the I/O node. Third, the daemon process coordinates with the data movement module to move buffered data back to home data disks.

**Log Disks.** Evidence from previous studies suggested that seek times of small disk request dominates. To alleviate this situation, I choose to use sequential access disks (a.k.a., log disks) as buffer disks, thereby making the seek time of most write requests to be zero. Data can be written onto the log disks in a sequential manner to improve performance of the buffer disks in I/O nodes. Disk head of a log disk is, in most cases, positioned on an

empty track that is available for incoming write requests. The seek times of write requests handled by buffer disks are zero unless the buffer disks are in a process of moving data to data disks or responding read requests.

**Power Management.** A dynamic power management policy is designed for ECOS to minimize the number of active buffer disks while maintaining reasonably quick response times. An approach to conserving energy of I/O nodes in cluster storage systems is to aggressively transit more buffer disks to the standby mode during periods of low disk load. More standby buffer disk can be spinned up to serve if demand for buffer disk bandwidth is high. Previous studies showed that disk access patterns are bursty in nature, indicating that waking up some standby buffer disks can quickly absorb a large number of disk requests during bursty periods.

It is important to quantify disk workload activities, because the appropriate number of active buffer disk largely depends on disk workloads. To simply the implementation of the dynamic power mangement policy, I enforce the timeout policy that places disks into the standby state if they are sitting idle for a certain amount of time (e.g., 10 seconds). A standby buffer disk needs to be spinned up to serve if the utilization of one of the active buffer disks exceeds the maximum threshold (e.g., 90%).

**Energy Consumption Model.** An energy cosumption model was implemented in ECOS to calculate energy dissipation in I/O nodes. I chose to use a model rather than an instrument to measure energy consumed by I/O nodes because the model allows us to evaluate impacts of a wide variety of disk drives on energy efficiency of ECOS.

For comparison purpose, I also implement an energy consumption model for a cluster storage system without employing buffer disks in I/O nodes. Before presenting the energy consumption models of the ECOS and non-ECOS systems, I first summarize the notation in Table 5.1.

Table 5.1: Notation for Modeling Energy Consumption in the ECOS and non-ECOS systems

| Notation | Definition |
|---|---|
| $n$ | Number of data disks |
| $m$ | Number of buffer disks |
| $E_{ECOS}$ | Total energy consumption of ECOS |
| $E_i^D$ | Energy consumption of data disk $i$ |
| $E_j^B$ | Energy consumption of buffer disk $j$ |
| $\alpha_{D,i}$ | Energy penalty of spinning up data disk $i$ |
| $\beta_{D,i}$ | Energy penalty of spinning down data disk $i$ |
| $\alpha_{B,i}$ | Energy penalty of spinning up data disk $i$ |
| $\beta_{B,i}$ | Energy penalty of spinning down data disk $i$ |
| $E_{non-ECOS}$ | Total energy consumption of non-ECOS |
| $P_{D,i}^A$ | Active power of data disk $i$ |
| $P_{D,i}^S$ | Standby power of data disk $i$ |
| $P_{B,j}^A$ | Active power of buffer disk $j$ |
| $P_{B,i}^S$ | Standby power of buffer disk $i$ |
| $T_{D,i}^A$ | Active time of data disk $i$ |
| $T_{D,i}^S$ | Standby time of data disk $i$ |
| $T_{B,j}^A$ | Active time of buffer disk $j$ |
| $T_{B,j}^S$ | Standby time of buffer disk $j$ |
| $N_{D,i}^{up}$ | Number of spin-ups of data disk $i$ |
| $N_{D,i}^{down}$ | Number of spin-downs of data disk $i$ |
| $N_{B,i}^{up}$ | Number of spin-ups of buffer disk $i$ |
| $N_{B,i}^{down}$ | Number of spin-downs buffer disk $i$ |
| $R$ | energy conservation Rate |

Let $E_i^D$ and $E_j^B$ be the energy dissipation in the $i$th data disk and $j$th buffer disk, respectively. The total energy consumption $E_{ECOS}$ of the ECOS system is the sum of energy dissipation in $n$ data disks and $m$ buffer disks. Thus, the energy consumption in ECOS can be expressed by Eq. (5.1).

$$\mathrm{E}_{ECOS} = \sum_{i=1}^{n} \mathrm{E}_i^D + \sum_{j=1}^{m} \mathrm{E}_j^B \tag{5.1}$$

The energy consumption $E_i^D$ of data disk $i$ in the ECOS system is the summation of the energy incurred by the data disk when it is in the active, idle, standby, and transition states. Thus, $E_i^D$ can be calculated by Eq. (5.2)).

$$\mathrm{E}_i^D = \mathrm{P}_{D,i}^A \mathrm{T}_{D,i}^A + \mathrm{P}_{D,i}^I \mathrm{T}_{D,i}^I + \mathrm{P}_{D,i}^S \mathrm{T}_{D,i}^S + \\ + \mathrm{N}_{D,i}^{up} \alpha_{D,i} + \mathrm{N}_{D,i}^{down} \beta_{D,i} \quad (5.2)$$

where $P_{D,i}^A$, $P_{D,i}^I$, and $P_{D,i}^S$ are the power of data disk $i$ when the disk is in the active, idle, and standby mode; $T_{D,i}^A$, $T_{D,i}^I$, and $T_{D,i}^S$ are time intervals when the disk is in the three power states, $N^u p_{D,i}$ and $N^d own_{D,i}$ are the numbers of spin-ups and spin-downs; and $\alpha_{D,i}$ and $\beta_{D,i}$ are the energy penalty of spin-ups/downs. I observed that active power and idle power of many hard drives are very close and; therefore, I can simply the above equation by assuming that active power and idle power are identical (i.e., $P_{D,i}^A = P_{D,i}^I$. Thus, Eq. (5.2)) can be simplied as Eq. (5.3):

$$\mathrm{E}_i^D = \mathrm{P}_{D,i}^A \mathrm{T}_{D,i}^A + \mathrm{P}_{D,i}^S \mathrm{T}_{D,i}^S + \mathrm{N}_{D,i}^{up} \alpha_{D,i} + \mathrm{N}_{D,i}^{down} \beta_{D,i} \quad (5.3)$$

Energy dissipation $E_j^B$ of buffer disk $j$ in ECOS can be derived in the same means as that of data disks. Hence, $E_j^B$ in Eq. (5.1) can be written as:

$$\mathrm{E}_i^B = \mathrm{P}_{B,i}^A \mathrm{T}_{B,i}^A + \mathrm{P}_{B,i}^I \mathrm{T}_{B,i}^I + \mathrm{P}_{B,i}^S \mathrm{T}_{B,i}^S + \\ + \mathrm{N}_{B,i}^{up} \alpha_{B,i} + \mathrm{N}_{B,i}^{down} \beta_{B,i} \quad (5.4)$$

Under relatively high I/O workloads, it is unlikely to spin down any buffer disks. As a results, buffer disks are not placed into standby; all the buffer disks are kept in the active mode. The numbers of buffer disk spin-ups/downs are zero; there is no power penalty of

spin-ups/downs. Consequently, the Eq. 5.5 can be simply as:

$$\mathrm{E}_{B,j} = \mathrm{P}^{A}_{B,j}\mathrm{T}^{A}_{B,j} \tag{5.5}$$

Now I am positioned to consider the energy consumption of a non-ECOS system. To make fair and conservation comparisons, I model a non-ECOS systems with $n$ data disks. In other words, I remove $m$ buffer disks from the ECOS system in order to turn the cluster storage system into a non-ECOS system. Let $E_i$ denote the energy consumption of the $i$th disk in non-ECOS. Then, $E_i$ can be derived from the energy incurred by the disk when it is in the active, idle, standby, and transition states. In case where the active power and idle power are very close, I can express the total energy consumption of the non-ECOS system using Eq. (5.6)

$$\begin{aligned}
\mathrm{E}_{non-ECOS} &= \sum_{i=1}^{n} \mathrm{E}_i \\
&= \sum_{i=1}^{n} (\mathrm{P}^{A}_{i}\mathrm{T}^{A}_{i} + \mathrm{P}^{I}_{i}\mathrm{T}^{I}_{i} + \mathrm{P}^{S}_{i}\mathrm{T}^{S}_{i} + \mathrm{N}^{up}_{i}\alpha_i + \mathrm{N}^{down}_{i}\beta_i) \\
&\approx \sum_{i=1}^{n} (\mathrm{P}^{A}_{i}\mathrm{T}^{A}_{i} + \mathrm{P}^{S}_{i}\mathrm{T}^{S}_{i} + \mathrm{N}^{up}_{i}\alpha_i + \mathrm{N}^{down}_{i}\beta_i)
\end{aligned} \tag{5.6}$$

where $P^{A}_{i}$, $P^{I}_{i}$, and $P^{S}_{i}$ are the power of disk $i$ when the disk is in the active, idle, and standby mode; $T^{A}_{i}$, $T^{I}_{i}$, and $T^{S}_{i}$ are time intervals when the disk is in the three power states, $N^{u}p_i$ and $N^{d}own_i$ are the numbers of spin-ups and spin-downs; and $\alpha_i$ and $\beta_i$ are the energy penalty of spin-ups/downs.

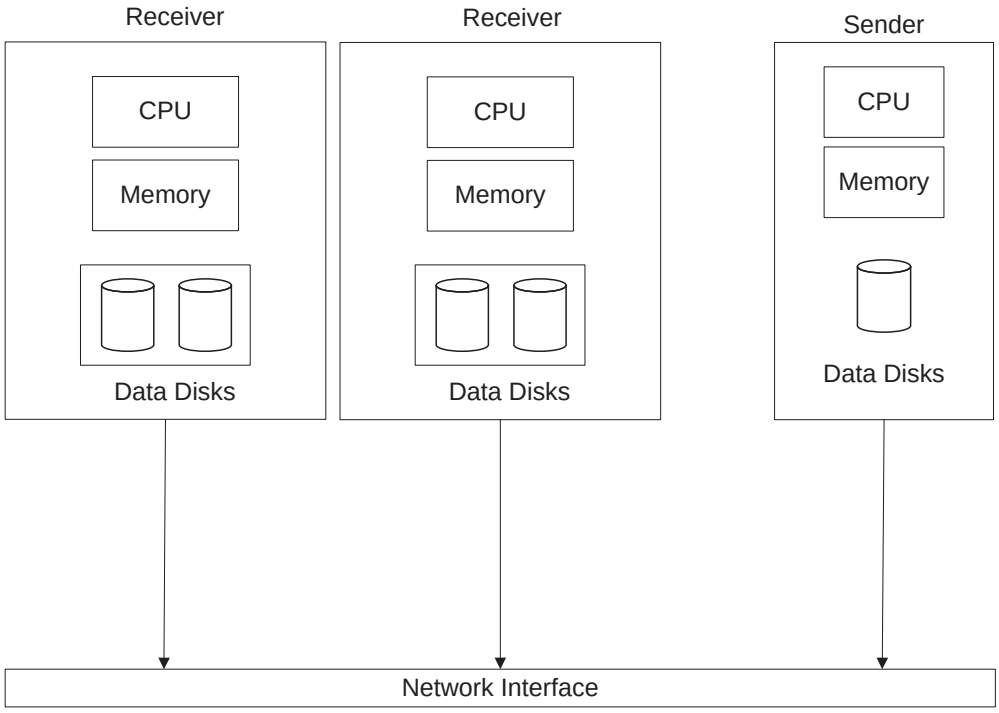


Figure 5.2: Test Platform without Buffer Disks

## 5.3 Experimental Results

In this section, I first describe a way of setting up a cluster as a testbed. Second, Last, experimental results are comprehensively analyzed.

### 5.3.1 Experiment Details

In the implemention of ECOS (Fig. 5.2), each I/O node consists of three local disks including one buffer disk and two data disks. Buffer disks under relatively high I/O workloads are never spinned down; as a result, idle buffer disks can serve any incoming disk request immediately without paying spin-up panelty. To reduce network traffic incurred by communications among I/O nodes, I implemented the request processing module to ensure that a buffer disk within an I/O node gives high priorty to data disks within the same I/O node. By default, disk requests are redirected from a data disk to a buffer disk within the same

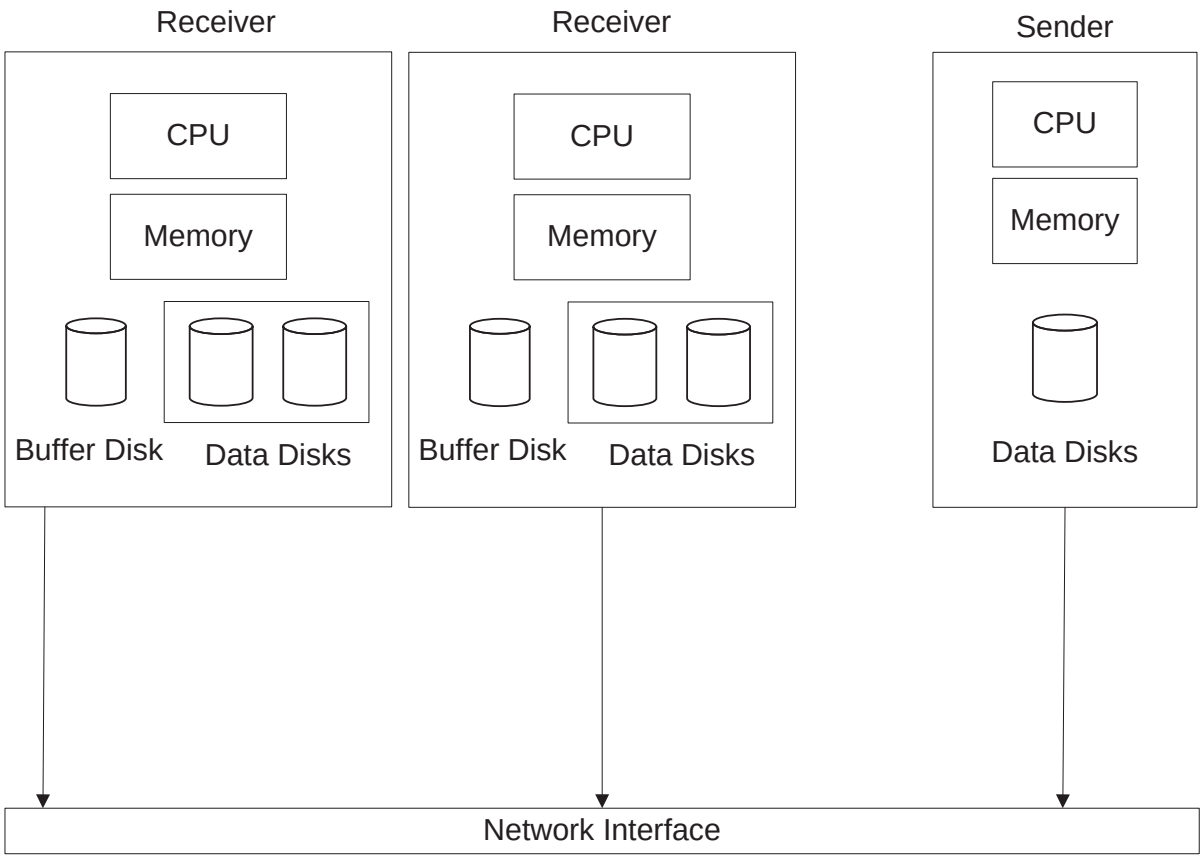


Figure 5.3: Test Platform with Buffer Disks

I/O node. If one I/O node is heavily loaded while another one has hight I/O load, requests might be redirected from the data disks in the heavily loaded node to the buffer disk in the lightly loaded one. Fig. 5.3 presents the implementation of the traditional design which has no buffer disks.

I developed a micro-benchmark running on a compute node. The microbench randomly issues disk requests based on the Poisson process. To reflect real-world I/O access patterns, I intentionally inserted idle periods between two consective request groups sent to I/O nodes in ECOS. In doing so, the micro-benchmark can issue a large number of disk requests representing I/O burstiness. It is observed that idle periods significantly affect energy efficiency of ECOS.

There is an important parameter referred to as Sum of Requests in Buffer or SRB, which affects the energy efficiency of ECOS. Hence, in our experiments, I focus on the impacts of SRB on the energy efficiency of the ECOS systems.

In addition to energy efficiency, energy conservation rate defined by Eq. (5.7) is used as a metric to quantitatively compare ECOS with non-ECOS.

$$\mathrm{R} = (1 - \frac{\mathrm{E}_{ECOS}}{\mathrm{E}_{nonECOS}}) \times 100\% \tag{5.7}$$

Table 5.2 summarizes the disk configuration of the tested cluster.

Table 5.2: Disks Configuration

| Disk Category | I/O Node 1 |
|---|---|
| Buffer Disk | Maxtor DiamondMax Plus 9 80GB |
| Data Disk 1 | WesternDigital 400 20GB |
| Data Disk 2 | WeaternDigital 400 20GB |
| **Disk Category** | **I/O Node 2** |
| Buffer Disk | Seagate Barracuda 7200.7 80GB |
| Data Disk 1 | WesternDigital 400 20GB |
| Data Disk 2 | Maxtor D740X-6L 20GB |

### 5.3.2 Performance Evaluation

In this subsection, let us present energy dissipation and energy conservation rate of ECOS. I first evalute the impacts of the SRB value and idle gap on energy conservation rate. In this experiment, SRB is decrease from 400 down to 25; the idle gap is varied from 50 to 300 Sec. Results plotted in Fig. 5.4 reveals that when I/O workloads are low, ECOS can significantly improve energy efficiency over non-ECOS. For example, ECOS conserves energy by up to more than 20% when the idle gap between two consective request groups is 300 or 200 Sec.

Recall that each I/O node in ECOS contains one buffer disk and two data disks. To make fair comparisons between ECOS and non-ECOS, I implemented a timeout policy in non-ECOS to place a disk into the standby mode if the disk has been sitting idle for a period of time (e.g., 20 Seconds). Compared with I/O nodes in non-ECOS, each I/O node in ECOS contains an extra buffer disk. Although adding an buffer disk in each I/O node seemingly imposes negative impact on energy efficiency, the results show that extra buffer disks can save a significant amount of energy. I contribute this trend to the fact that energy saved by the buffer disks is larger than the energy overhead introduced by the extra buffer disks. In constrast to light I/O workloads, an extremely high I/O load can prevent ECOS from producing energy savings. High I/O workloads eliminate long idle periods in both buffer and data disks, thereby reducing the number of opportunities for data disks to be placed in the standby mode.

It is intriguing to observe from Fig. 5.4 that when the idle gap is as short as 50 Sec., the energy conservation rate becomes even negative, meaning that ECOS consumes more energy than non-ECOS. Recall that if the number of bufferred requests targeting at the same data disk equal to SRB, then the corresponding data disk must spin up so that the bufferred data can be moved back to this data disk. The data disk is spinned down to the standby mode under the following two conditions. (1) No read request is retriving data from the data disk; and (2) no write requests are currently being handled by the disk. The larger the SRB value,

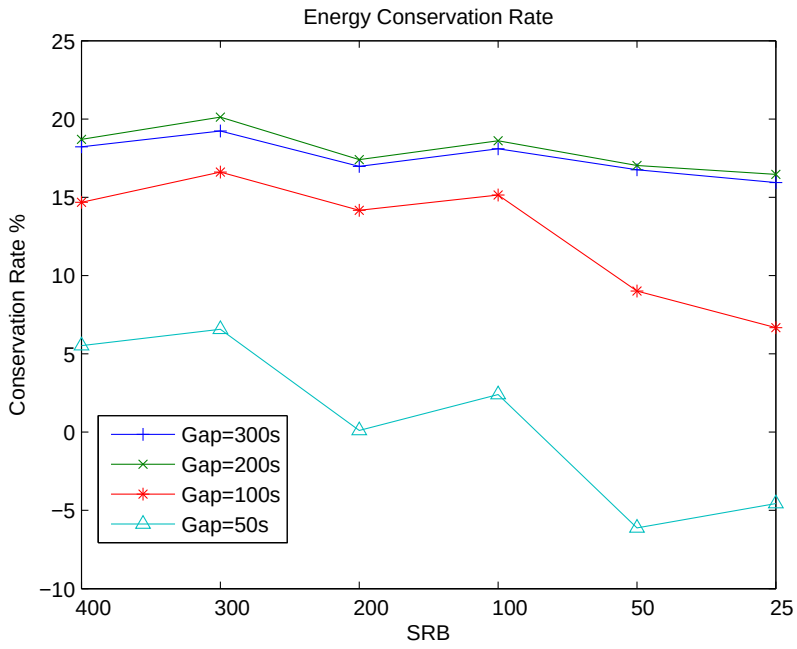


Figure 5.4: Energy Conservation Rate

the more energy can be conserved in ECOS. However, I/O access patterns can greatly affect the energy conservation rate. The best time to transfer data between a buffer disk and data disks within an I/O node is at the time when the node is idle. ECOS forces buffer disks to copy data back to data disks when the SRB requirements are satisfied. Fig. 5.4 confirms that high frequency of data movement during I/O burstiness can reduce conservation rate.

It is worth noting that I implemented ECOS on a heterogenous cluster storage system, where the hard drives in the tested I/O nodes are not identical. As such, the goal of the second experiment is two-fold. First, I intend to show our approach can achieve high energy efficient for both homogeneous and heterogeneous cluster storage systems. Second, I plan to observe how overall system energy saving is affected by energy conservation provided in each I/O node. In what follows, I plot eight figures showing the energy consumption and energy conservation rate of the two individual I/O nodes in ECOS. Please note that the hard drives in I/O node 2 are more power consuming and faster than those in I/O node 1. Results depicted in Figs. 5.5-5.12 show although energy efficiency of the two I/O nodes are different, the energy-efficiency trends of the two nodes are quite similar.

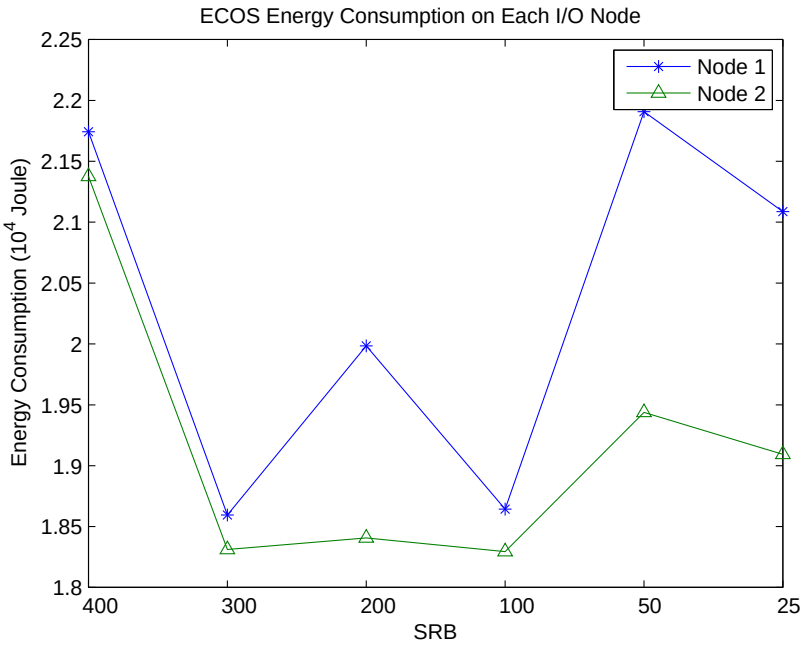


Figure 5.5: Energy Consumption in I/O Nodes, idle time gap is 50s

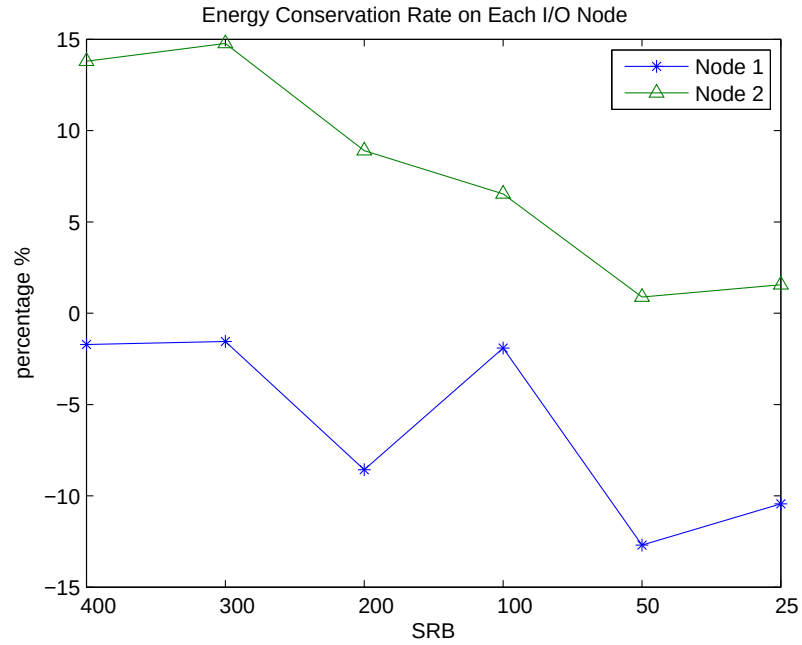


Figure 5.6: Energy Conservation Rate in I/O Nodes, idle time gap is 50s

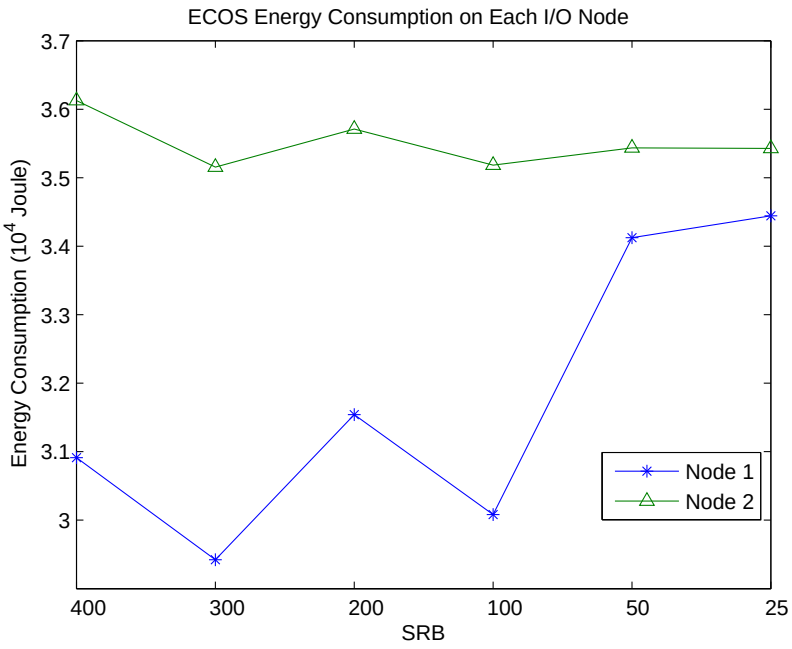


Figure 5.7: Energy Consumption in I/O Nodes, idle time gap is 100s

Fig. 5.5 shows that the energy dissipation in I/O node 1 is larger than that of I/O node 2. The buffer disk in node 2 saves more energy than the buffer disk in node 1, because of the following four reasons. First, the workload is high due to a small value of idle gap (i.e., 50 Seconds). Second, the performance of the buffer disk in node 2 is higher than that of the buffer disk in node 1. Third, compared with node 1, node 2 can quickly move data back to the home data disks. This trend is more pronounced under high I/O workloads with enormous I/O bustiness. Last, data disks in node 1 are more likely to stay in the active state due to the slow process of moving data back to the home disks. Fig. 5.6 shows the energy consumption rate of I/O node 2 is higher than that of I/O node 1. This is mainly because time spent in moving data from a buffer disk to data disks in node 2 is shorter than that spent in moving data in node 1. The implication behind this result is that fast data movement between a buffer disk and data disks can help in achieving high energy efficiency.

Figs. 5.7 and 5.8 plot energy consumption and energy conservation rate of the two I/O nodes. I observe from Figs. 5.7 and 5.8 that for a medium workload (e.g., idle gap is set to 100 Seconds), there is a low propability of transferring data between buffer and data disks

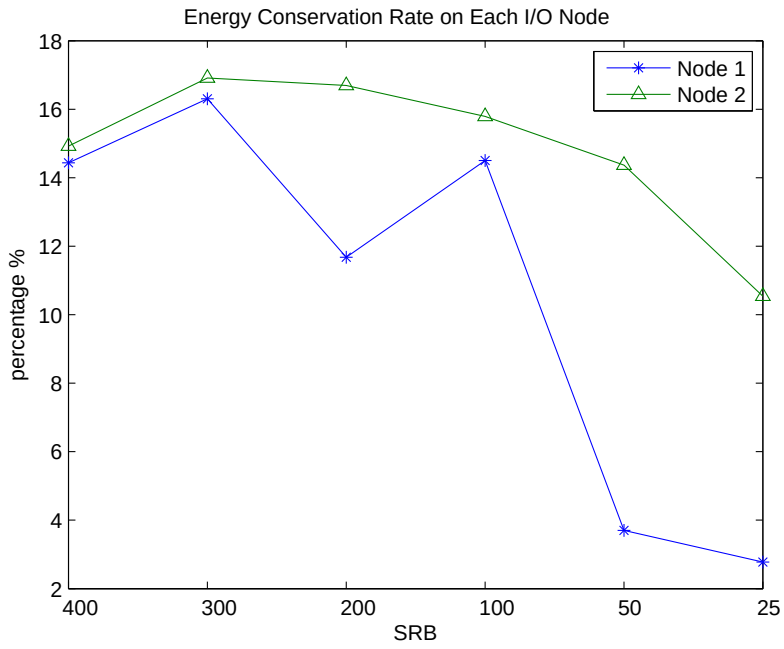


Figure 5.8: Energy Conservation Rate in I/O Nodes, idle time gap is 100s

during I/O burstiness. Therefore, the discrepancy between the data movement times of the two I/O nodes starts diminishing with decreasing I/O workload. Unlike high I/O load that makes node 2 more energy efficient than node 1, medium I/O allows node 1 exhibit more energy-efficient than node 2 (see Fig. 5.7). Furthermore, Fig. 5.7 indicates that node 1 is more sensitive to SRB than node 2; the sensitivity can be analyzed as follows. First, decreasing the SRB value increases the frequency of data movement between buffer disks and their corresponding data disks. Second, the high data movement frequency leads to a high probability of transferring data back and forth between a buffer disk and data didks during I/O bustiness.

When the idle gap increases to 200 Seconds, a small SRB does not significantly affect the energy consumption in both I/O nodes. Fig. 5.9 illustrates that energy consumption slowly increases when SRB decreases. Fig. 5.10 shows that under condition that SRB is 400 or 200, the data movement operations tend to occur within an idle gap between two consecutive request groups. This result indicates that to deliever the high performance, the data movement mechaism must be actuated at the time between two consecutive I/O

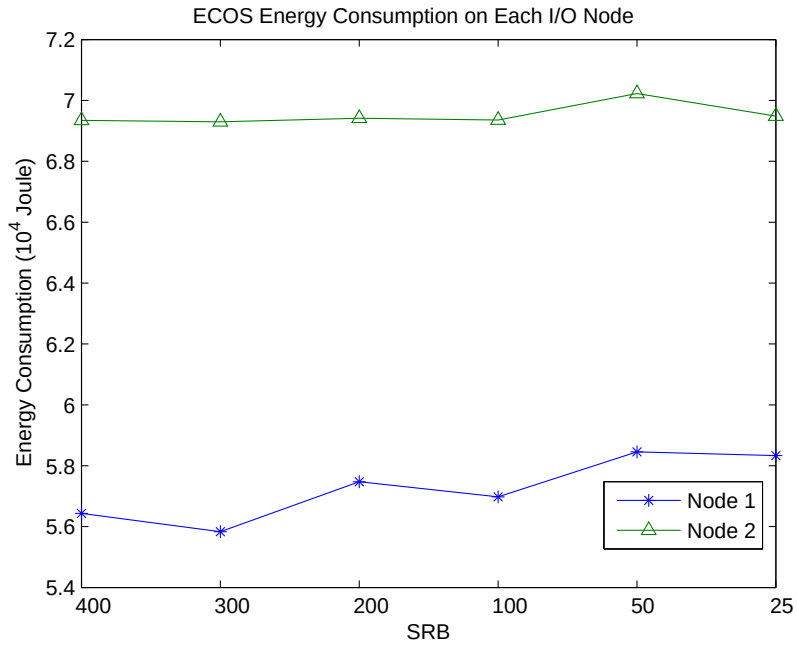


Figure 5.9: Energy Consumption in I/O Nodes, idle time gap is 200s

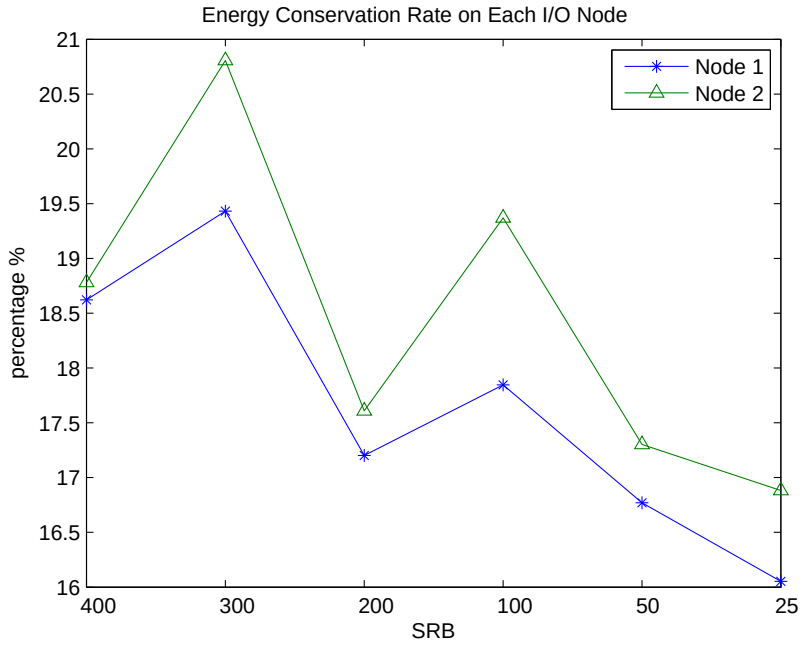


Figure 5.10: Energy Conservation Rate in I/O Nodes, idle time gap is 200s

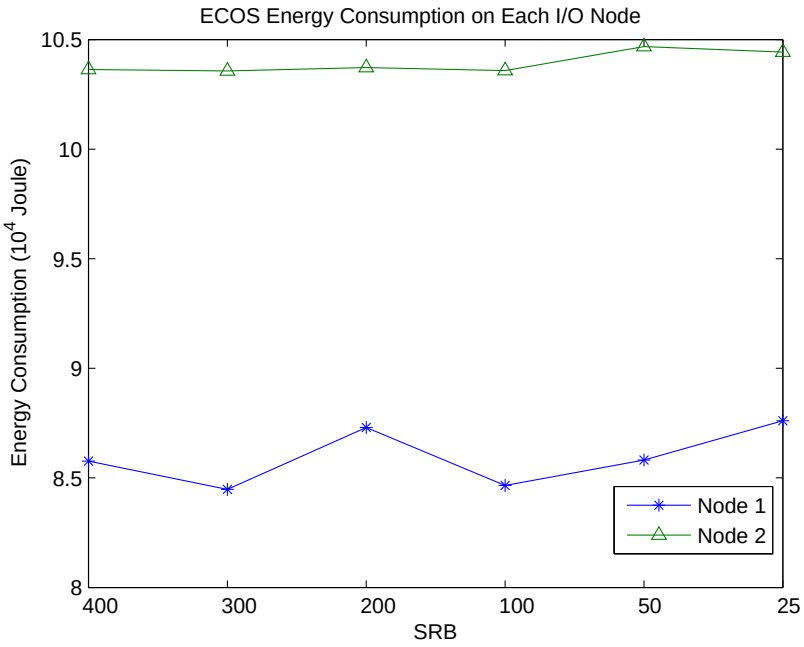


Figure 5.11: Energy Consumption in I/O Nodes, idle time gap is 300s

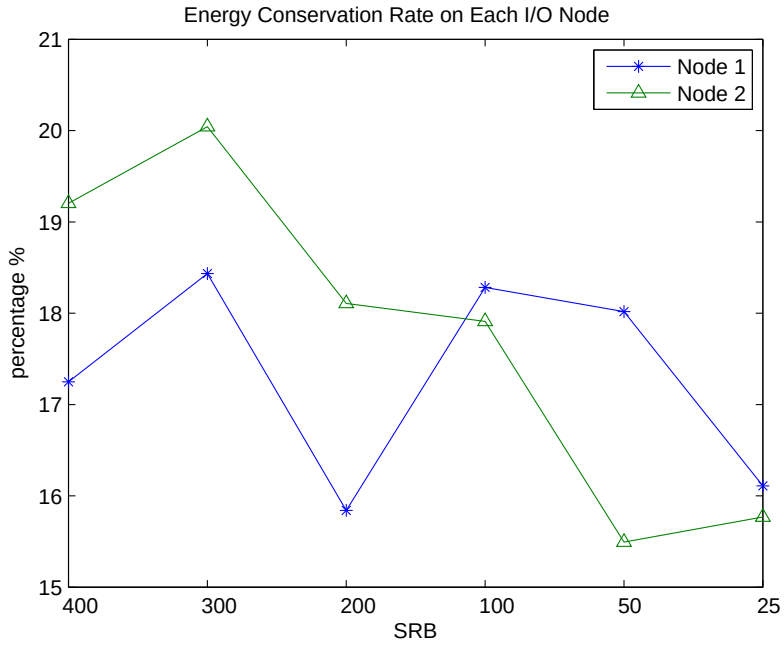


Figure 5.12: Energy Conservation Rate in I/O Nodes, idle time gap is 300s

bustiness. Fig. 5.12 shows that energy conservation rate when idle gap is set to 300 Seconds. The curves plotted in Fig. 5.12 are consistent to those depicted in Figs. 5.10 and 5.6.

## 5.4 Summary

Cluster storage systems are cost-effective building blocks for many high-end computing infrastructures. Optimizing energy efficiency of cluster storage systems remains an open issue. In this reserach, I designed and implemented an energy-efficient cluster storage system called ECOS. Each I/O node in ECOS controls multiple disks - one buffer disk and several data disks. The key idea behind ECOS is to redirect disk requests from data disks to the buffer disks. To improve I/O performance of buffer disks, ECOS attempts to balance I/O load among all I/O nodes in the cluster storage system. Redirecting requests is a driving force of energy saving and the reason is two-fold. First, ECOS makes an effort to keep buffer disks active while placing data disks into standby in a long time period to conserve energy. Second, ECOS reduces the number of disk spin downs/ups in I/O nodes.

The ECOS system was implemented in a Linux cluster, where each I/O node contains one buffer disk and two data disks. Results show that ECOS improves energy efficiency of traditional cluster storage systems without using buffer disks. Interestingly, our results indicate that ECOS equipped with extra buffer disks is more energy efficient than the same cluster storage system without the buffer disks. Using existing data disks in I/O nodes to perform as buffer disks can achieve even higher energy efficiency.

# Chapter 6

# Can We Improve Energy Efficiency of Secure Disk Systems without Modifying Security Mechanisms?

## 6.1 Introduction

In the past decade, energy efficiency has become an ever increasing priority in computer science research [88]. Computers have traditionally been designed with performance metrics being the main focus of the design. Since our sources of energy to power computers are not limitless, it is imperative to design energy efficient computer architectures. Disk systems tend to be large consumers of energy consumption [70] and; therefore, I have designed an energy efficient parallel disk framework (see [119] for details of our disk framework). Apart from high energy efficiency, security mechanisms are equally important for disk systems to support a wide range of data-intensive applications that are security sensitive. Although previous research has focused on producing a relationship between energy efficiency and security strength for mobile devices (see, for example, [17]), it is challenging to make good tradeoffs between high security and energy efficiency for storage systems in general and for disk systems in particular.

The long-term goal of this research is to develop energy-efficient security mechanisms for disk systems without significantly degrading disk performance. Similar design goals can be found in the literature (see, for example, [70] and [75]). This study started off with having a goal of developing a matrix that would outline the tradeoffs between energy efficiency and security in the context of large-scale disk systems.

Generally speaking, there are two approaches to implementing energy-efficient security-aware disk systems. The first one is to improve the energy efficiency of security mechanisms in

disk systems (see, for example, [17]); the second approach advocates for integrating conventional security services with energy-efficient disk architectures. The first approach makes an effort to implement energy-efficient security mechanisms in traditional disk systems, whereas the second one is focused on energy-efficient disk systems without modifying existing security mechanisms. In this study, I focus on the second general approach. Thus, I attempt to answer an intriguing question of whether it is possible to seamlessly integrate security services with energy-efficient disk systems without modifying the source code of security services.

To determine if it is possible to conserve energy consumption of existing security services using new energy-efficient disk systems, I will have to investigate I/O access patterns of encryption and integrity checking algorithms. Studying I/O access patterns of disk requests issued by the security services requires knowledge of encryption algorithms and hash functions. In this research, I first investigate the I/O characteristics of encryption algorithms and hash functions. Next, I apply these I/O characteristics within the energy-efficient buffer disk architecture or BUD (see [98] for detailed information concerning BUD) to investigate the possibility of leveraging BUD to reduce energy dissipation caused by the existing encryption algorithms and hash functions. Very recently, I had designed and implemented software modules to separately handle read [73] and write [98] requests within the BUD architecture, which will be briefly described in the next Section. In addition, I had made some generalizations about the BUD architecture. One of our previous studies showed that the BUD architecture is extremely sensitive to hard disks' Break Even Time, which is defined as the size of an idle time required for a disk to energy efficiently transition from the active state to the standby state. In hard disks, for example, the break even time often exceeds 10 seconds. Such a large disk break even time indicates that the BUD architecture is maximally energy efficient in applications that are not disk intensive. Any software module that leaves an opportunity for these break even times to be met allows the BUD architecture to save energy. Hence, security modules that can take full advantage of BUD to conserve energy should not bottlenecked at disk I/O operations. In other words, disk requests issued by the

security modules must be sufficiently sparse to produce noticeable energy savings. To answer the fundamental question of whether I can improve energy efficiency of secure parallel disk systems without modifying security mechanisms in the disk systems, I choose the BUD disk architecture as target energy-efficient disk systems. A vital part of this study is to profile encryption algorithms and hash functions in the context of disk systems. I intend to figure out if I/O access patterns of the encryption algorithms and hash functions would allow the energy-efficient BUD disk architecture to conserve energy. The key I/O features I focused on were arrival patterns of disk request operations issued by the encryption algorithms and hash functions. I aimed to determine if the disk operations can yield sufficient idle periods for the BUD disk architecture to reduce disk energy consumption using the energy-efficient data management strategies I have previously studied.

The rest of the Chapter is organized as follows. Section 6.2 provides an overview of an energy-efficient disk architecture. Section 6.3 describes our test bed setup. Section 6.4 presents the experimental results and explanations of the trends in our results. Finally I end with a summary and some future work possibilities.

## 6.2 Overview of the BUD Disk Systems

Although a significant amount of energy can be saved if idle disks are turned to the standby mode, short idle periods (i.e., smaller than the disk break even time) prevents idle disks to be switched to standby to conserve energy. This problem can be solved by aggregating smaller idle periods into idle times that are larger than the disk break even time. I implemented an idle time aggregation process in the buffer-disk architecture or BUD (see Fig. 6.1 below) using buffer disks to temporally buffer disk requests while keeping data disks to standby as long as possible.

The buffer disk controller - a center piece in the BUD architecture - is responsible for the dynamic power management in both buffer and data disks. The two areas where our previous studies have focused is the buffer disk controller [119] and energy-efficient prefetching [73].

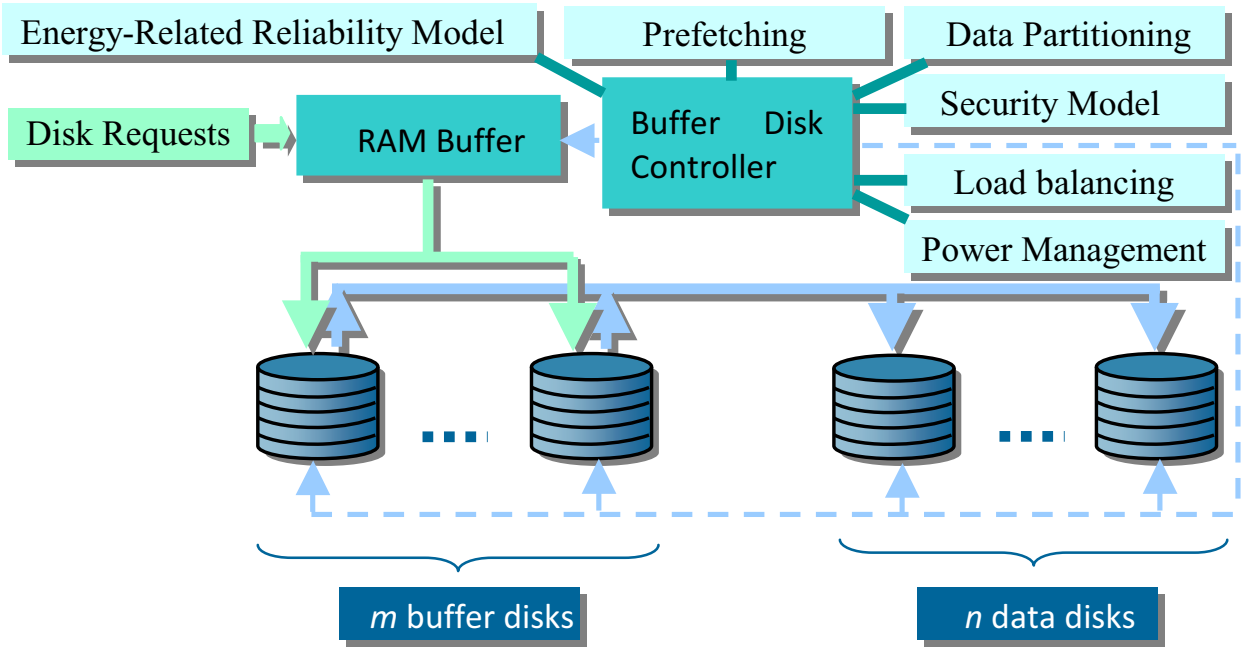


Figure 6.1: The buffer-disk architecture or BUD for parallel disk systems

Moreover, I have extensively investigated a disk write buffer strategy to improve parallel I/O energy efficiency [98]. There has been work on load balancing the buffer disks and controlling writes to the data disks [78]. For example, buffering write requests in the buffer disks and writing out data when certain criteria are met [98][78]. To further improve energy efficiency of parallel disk systems, I have developed energy-efficient data partitioning schemes, data placement strategies, and data movement algorithms [80][68]. It is worth noting that the BUD architecture embraces a security component, which not only provides security services but also measures security overhead imposed by the integrated security mechanisms. The detailed information concerning the BUD architecture along with a set of energy-efficient data management strategies can be found in [119][98][73].

## 6.3 Experimental Setup

In the BUD architecture prefetching and data buffering are closely related to total capacity of buffer disks as well as the arrival rate of disk requests. The buffer disks can,

literally speaking, prefetch and buffer data aggressively until the disk capacity is reached. When the buffer disks become full, either a portion of buffered data must be moved to the standby data disks or part of prefetched data has to be evicted from the buffer disks. The time taken for the buffer disks to reach their capacity largely depends on request arrival rates, data size, and storage capacity. Since the total buffer disk storage space is managed by with the BUD controller, this study was not meant to address the issue of buffer disk capacity. Thus, let us focus on access patterns (e.g., disk request arrival rates and I/O processing time) to explore the possibilities of achieving high energy efficiency in secure parallel disk systems without modifying security mechanisms.

To capture access patterns of disk requests issued by confidentiality and integrity services, I have to profile encryption algorithms and hash functions. I chose to use a Linux computer because of the open source nature of Linux and availability of free software like GkrellM, Conky and XySSL. The first software monitor used in our experiments is GkrellM. Although GkrellM is capable of providing disk, memory, and CPU usage statistics, I were unable to find a quick method to produce the output in a text file. Hence, I moved to using Conky, which is a lightweight system monitor that is highly configurable and supports text output. After making use of the XySSL libraries to evaluate an array of security mechanisms, I generated disk trace files for further analysis.

Our created a testbed using one Linux PC. Table 6.1 outlines the important properties of the testbed. The CPU speed of the testbed is high enough that the CPU will probably

Table 6.1: System Parameters of the Testbed

| | |
|---|---|
| CPU Speed | Pentium 4 2.4 GHZ |
| Memory | 512 MB |
| Operating System | Ubuntu 7.10 |
| USB 1.1 | 2 Mb/s |
| HD Bus | IDE |

not be the performance bottleneck for the evaluated security mechanisms. The bandwidth of the memory component in our testbed is relatively low for a Pentium 4 computer. I chose

to connect a flash drive to the testbed by a USB port, because the USB interface was used to emulate a network interface card in our experiments. A hard drive is connected to a SCSI Bus with the bandwidth of at least 5MB/s.

The Linux operating system was used to perform our experiments, because it is probably a bit easier to find the required software needed to profile encryption algorithms and hash functions. The Linux-based software tools used in our experiments allow us to make any changes to the software to monitor and study access patterns of security services.

Once I have our OS chosen, I am in a position to identify software that allow us to test our encryption algorithms and hash functions. An ideal software tool should include the implementation of a wide range of popular encryption algorithms and hash functions. Apart from a security software tool, I have to choose a software tool enabling us to monitor the CPU, memory, and read/write performance of the hard drive and flash drive connected to the USB port. After I found software to fulfill the above requirements, I was ready to profile encryption algorithms and hash functions.

The first software tool I chose to use is XySSL, which implements a set of well-known encryption algorithms accompanied by testing programs. In this study, I pay particular attention to two encryption algorithms - 3DES [79] and AES [2]. 3DES - slow in software - was developed in response to the weakness of DES. 3DES is a strong encryption algorithm, but it is typically slower than AES. 3DES uses a 64 bit block size to encrypt data [83]; it uses a 192 bit key that is split into three different keys. In the 3DES algorithm, these three 64 bit keys are required to encrypt and decrypt data using DES. Each of these 64 bit keys uses 8 bits for parity checking leaving 3DES with an effective key strength of 168 bits.

AES is the current standard for data encryption widely adopted by the US Government [83]. AES is typically employed with a 192-bit encryption key. Block size in AES is 128-bit in length, meaning that AES encrypts twice as much data as 3DES at each call of the function implementing the encryption. AES is not only fast in software but also requiring a small amount of memory. In addition to encryption algorithms, several hash functions were

implemented in XySSL. To this end, I decided to evaluate MD5 [96] and SHA (1-2) [3]. MD5 is an algorithm that produces a hash output value of 128 bits. SHA-1 generates 160-bit hash values, whereas SHA-2 can provide a hash bit varying in size between 224 and 512 bits. I also evaluated RSA verification implemented in XySSL. Please note that all of the aforementioned encryption algorithms and hash functions, except for 3DES, are coupled with testing programs in XySSL. Thus, I had to implement a testing program for 3DES based on the AES testing program.

To monitor our testbed computer, I take full advantage of GKrellM - a software monitor that is capable of monitoring the CPU, memory, and disk I/O subsystems. GKrellM seems to be a promising software tool. The only problem with GKrellM, however, is the lack of well defined text output. There is no straightforward way to produce text output. Hence, I would have to dig into the source code of GKrellM and implement a module to deal with text output. Alternatively, I would have to find another system monitor that has some kinds of text output. I chose the second approach due to time constraints and began our search for a system monitor with text output.

Fortunately, Conky - a lightweight system monitor that is highly configurable - is able to yield a text output file. There is a listing of all the variables that Conky keeps track of. If I need to output of these variables to a text file, I simply specify the variable in a configuration file. The configuration file allows us to format output files, making it possible for us to put our desired variables in the csv format readable by Microsoft Excel. In our experiments, the variables that are kept track of include events of CPU, memory, and disk read/writes in the testbed. Since I substituted a USB-based flash drive for a networked disk system, all data reads are initiated from the USB drive. I was able to control the testing programs in XySSL to guarantee all disk writes are physically issued to the hard disk in our testbed. In doing so, I was capable of separately monitoring I/O access patterns of the flash drive and hard drive where the encryption algorithms and hash functions are evaluated.

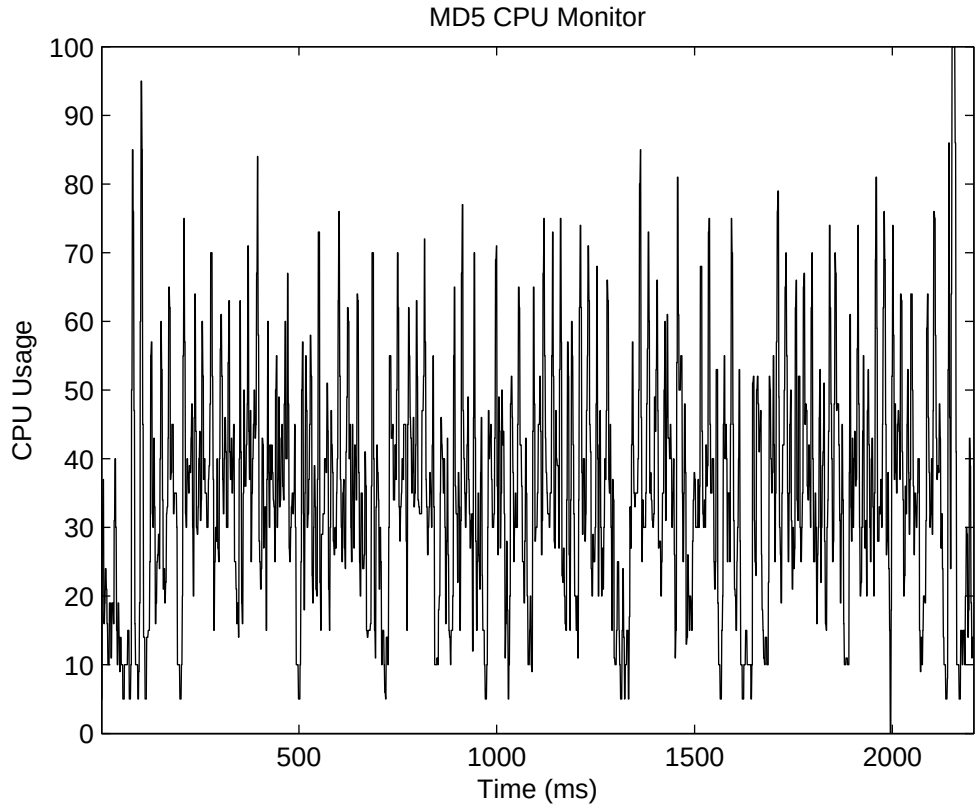


Figure 6.2: CPU usage (measured in percentage) of the testbed when MD5 is evaluated

## 6.4 Experimental Results and Analysis

Now I am positioned to present experiment results and analysis in the this sections. i evaluated three different types of system events, namely, CPU usage, flash drive read bandwidth, and hard disk write bandwidth. Recall that I extensively investigated three hash-function algorithms, one RSA signature verification algorithm, and two block and stream encryption algorithms in our experiment.

In our experiment, I use a flash drive to emulate source data retrieved from a remote disk system connected to the testbed through a network. The size of the source data set is 900Mbyte; the data was organized and stored in 78 files residing in the USB-based flash drive.

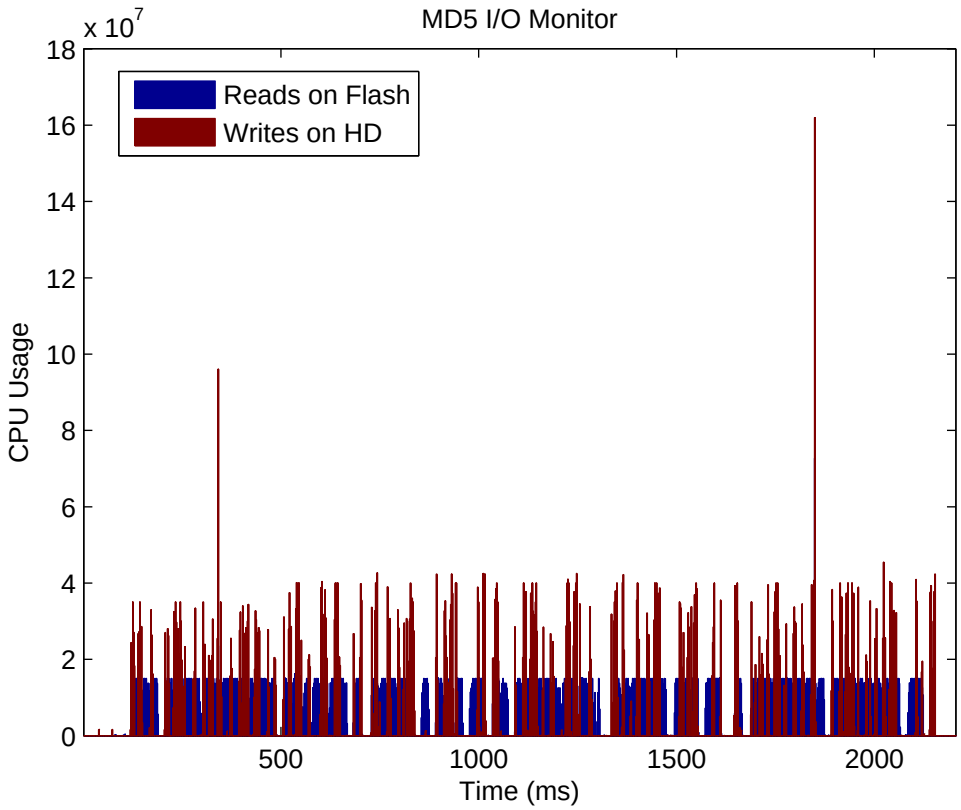


Figure 6.3: Read/write bandwidth of the testbed when MD5 is evaluated

MD5 used to be one of the most popular hash functions for data integrity checking services. Although MD5 is no longer considered as the most secure algorithm for integrity checking [15], it is still a good representative hash function to be considered in this research. Hence, I started our experiment by exhibiting the workload of CPU and I/O of the testbed when MD5 is running to ensure that a file has not been tampered with. Fig. 6.2 shows the CPU usage when the testbed is using MD5 for integrity checking. Since the MD5 hash function is not CPU-intensive, I observed from Fig. 6.2 that the CPU usage is almost always lower than 60% and very rarely exceeds 80%. Hence, I concluded that CPU is not the performance bottleneck of MD5.

Fig. 6.3 shows the read/write bandwidth of the disk subsystem in the testbed. It is observed from Fig. 6.3 that 15MB/Sec. is the maximum I/O read bandwidth achieved by the testbed MD5 is evaluated. Such a maximum read bandwidth is apparently the upper bound of the bandwidth exhibited by the USB-based flash drive. A second observation drawn from Fig. 6.3 is that the average write bandwidth of the disk subsystem is 32MB/Sec., which rarely reach the maximum write bandwidth. More interestingly, the flash drive experiences a limited number of idle periods during the execution of MD5. Compared with read access patterns, write access patterns contain more and longer idle times. Fig. 6.3 reveals that data read bandwidth becomes the performance bottleneck of the disk system where MD5 is applied for data integrity checking. More importantly, Fig. 6.3 suggests that without modifying MD5, I can apply the BUD architecture to reduce energy consumption of disk systems integrated with MD5 for data integrity checking. BUD can conserve energy for MD5, because many small idle periods in disks can be grouped to form large idle periods, which in turn allow disks to be switched to the standby mode to save energy.

SHA-1 is a second hash function evaluated in our testbed. The access patterns of SHA-1 are very similar to those of MD5. SHA-1 is more secure than MD5 (see [111] for detailed comparisons), because SHA-1 is more complicated in implementation than MD5. Fig. 6.4

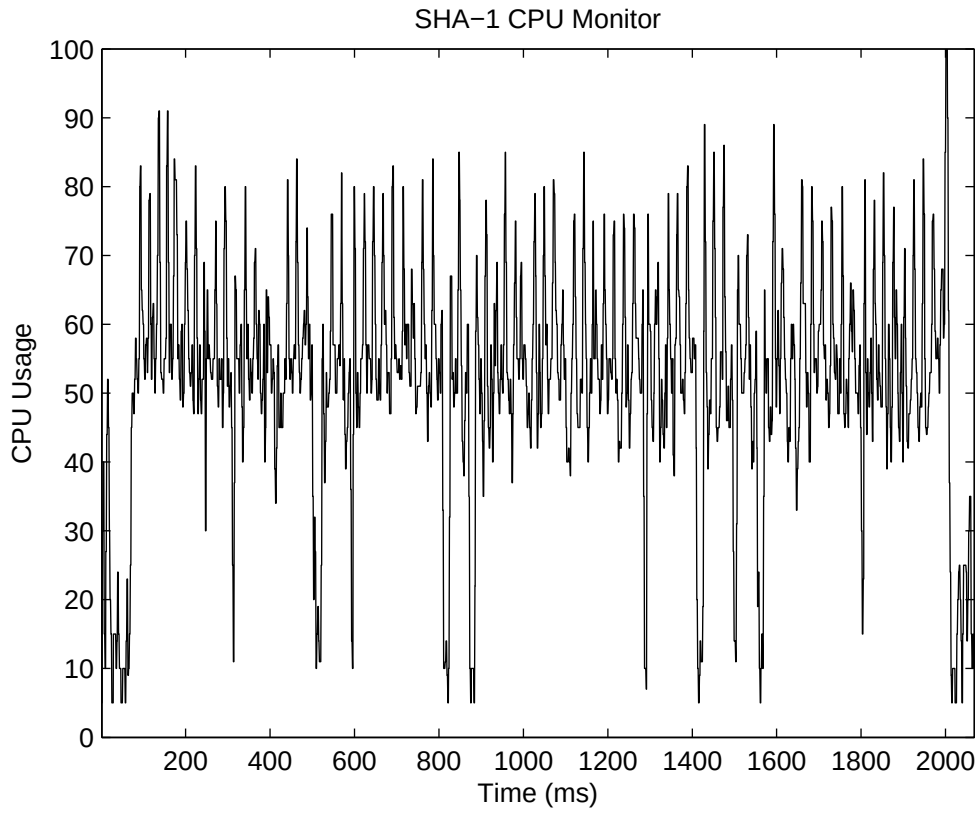


Figure 6.4: CPU usage (measured in terms of percentage) of the testbed when SHA-1 is evaluated

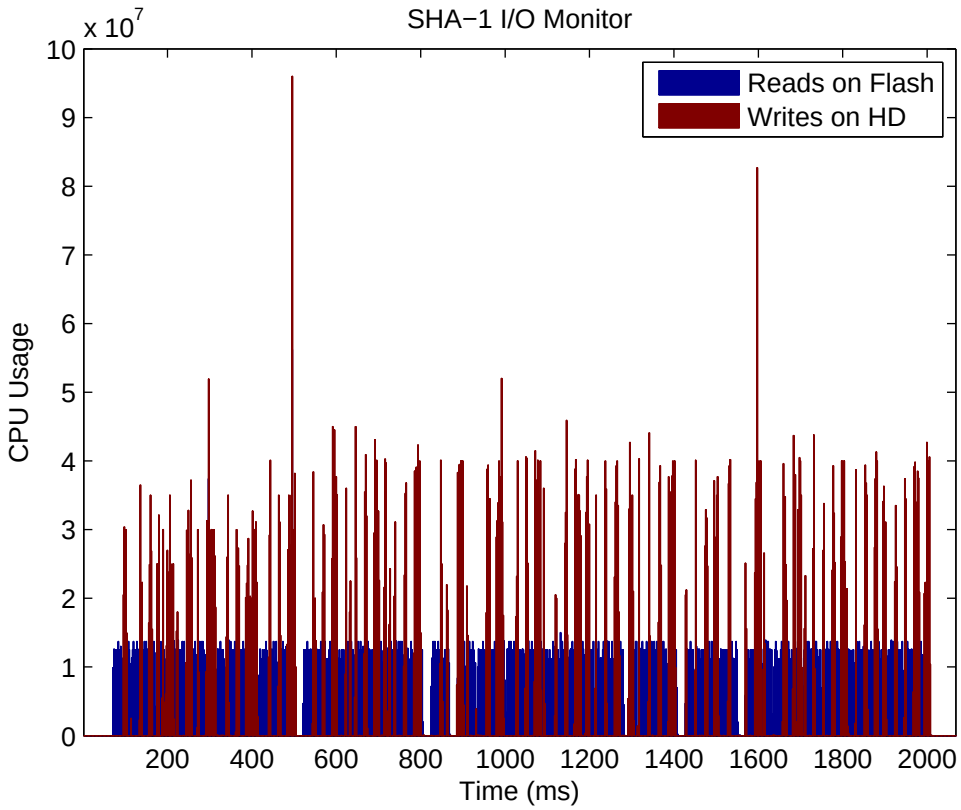


Figure 6.5: Read/write bandwidth of the testbed when SHA-1 is evaluated

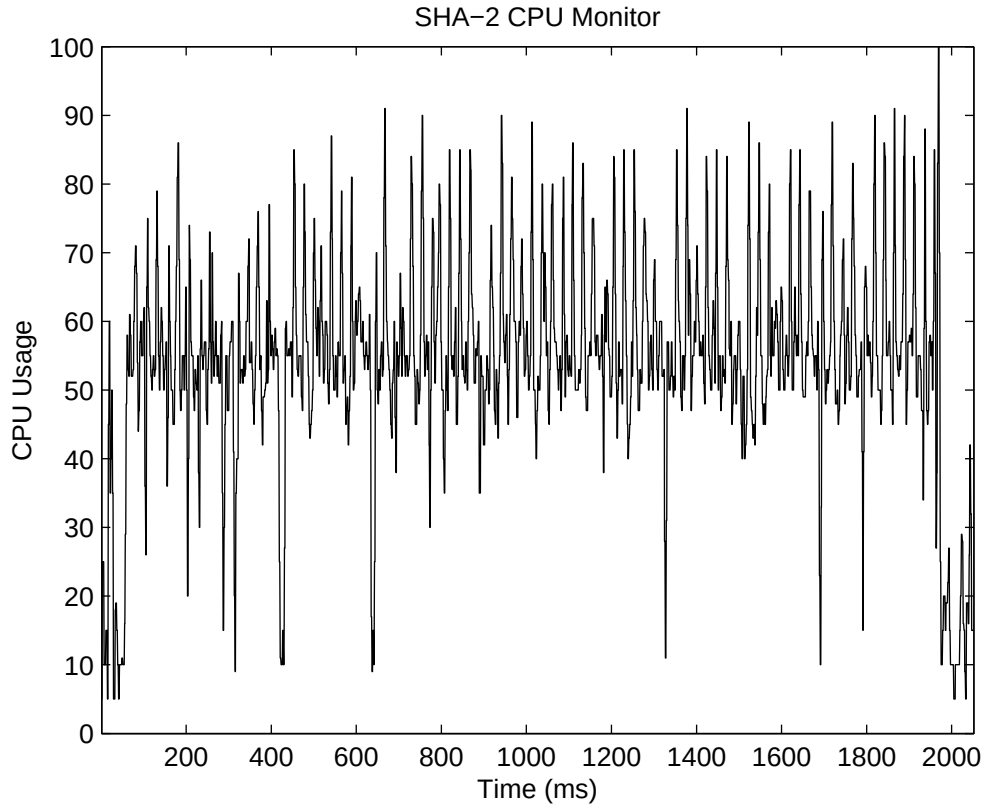


Figure 6.6: CPU usage (measured in terms of percentage) of the testbed when SHA-2 is evaluated

indicates that the average CPU usage of the testbed when SHA-1 is evaluated is higher than that of the same testbed when MD5 is running.

Fig. 6.5 shows that in almost all the cases, the testbed keeps reading data from the flash drive at the highest bandwidth. There is only very small number of gaps among reading events, meaning that idle periods rarely occur in SHA-1. Unlike read requests, disk write operations (see Fig. 6.5) in SHA-1 demonstrate more idle periods. This is mainly because with respect to SHA-1, reading data from the flash drive is the performance bottleneck of the testbed system. Like Fig. 6.3, Fig. 6.5 indicates that the SHA-1 can be seamlessly integrated with the BUD architecture to achieve both high energy efficiency and data integrity. Such integration can be straightforwardly realized by employing a software module of the existing SHA-1 service on top of BUD without even changing the source code of SHA-1.

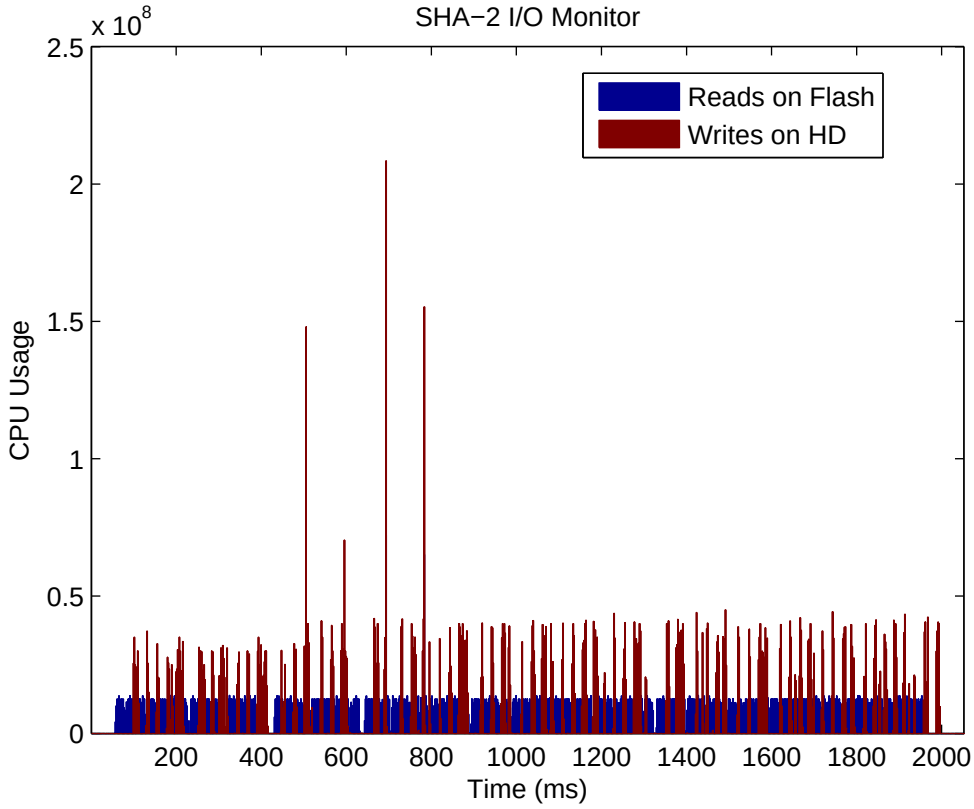


Figure 6.7: Read/write bandwidth of the testbed when SHA-2 is evaluated

SHA-2 is comprised of a group of hash functions. Without loss of generality, in this experiment let us evaluate SHA-256, which can be considered as one of the most secure hash functions in this study. Fig. 6.6 shows that when SHA-2 is evaluated, the CPU usage of the testbed system is slightly higher than that of the same testbed with SHA-1. Fig. 6.7 indicates that disk I/O characteristics of SHA-2 are very close to those of SHA-1. In other words, data read bandwidth of the flash drive is the performance bottleneck of SHA-2. An implication of this result is that without changing the source code of SHA-2, it is possible to leverage the BUD architecture to improve energy efficiency of disk systems where SHA-2 is employed.

RSA - widely adopted as a public-key encryption scheme. - has signature generation and verification functions. In this experiment, I evaluate RSA's signature and verification functions in our testbed. Before I started the test, I generated signature files for each file

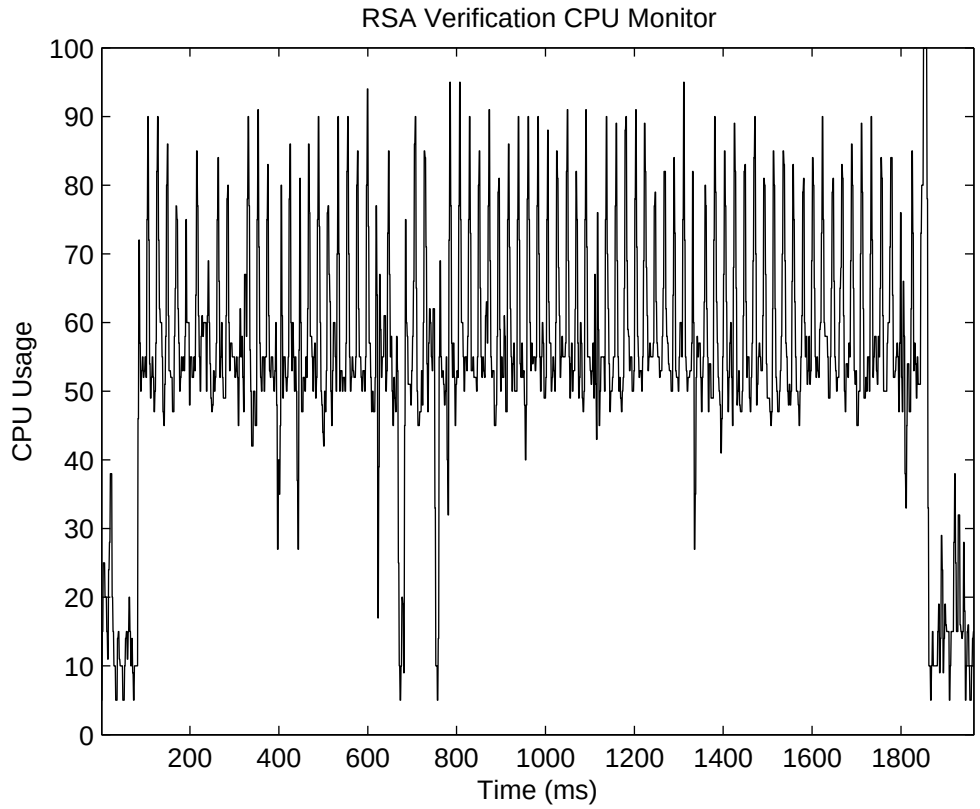


Figure 6.8: CPU usage (measured in terms of percentage) of the testbed when RSA Verification is evaluated

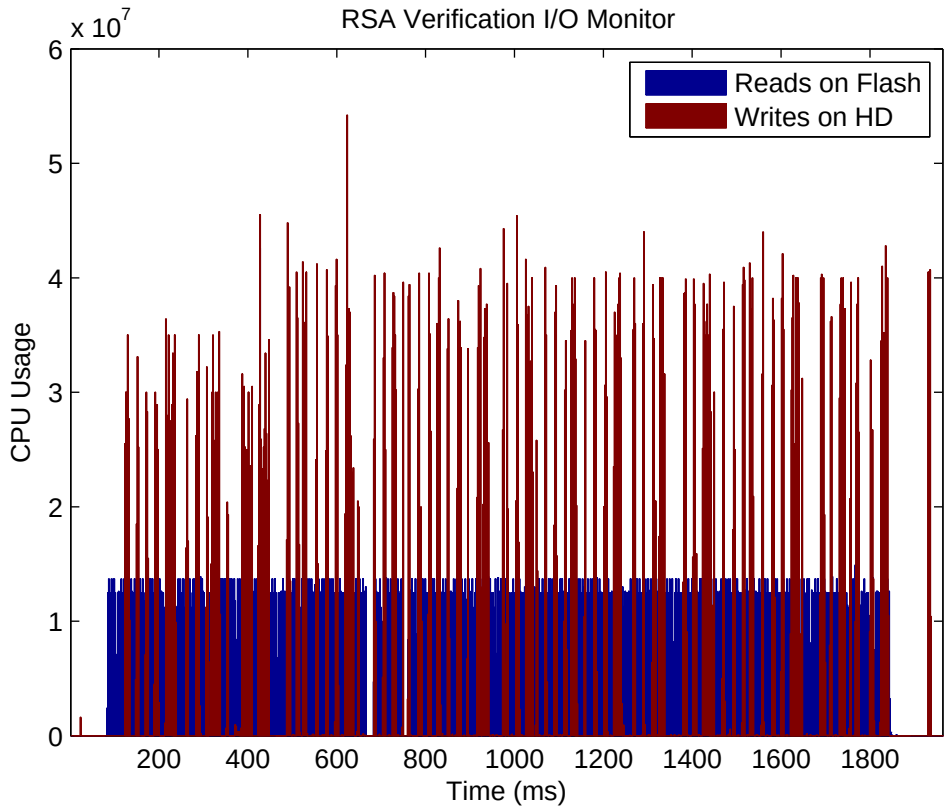


Figure 6.9: Read/write bandwidth of the testbed when RSA Verification is evaluated

residing in the USB-based flash drive. In this case, each input data file is coupled with a corresponding signature file.

I observed from Figs. 8 and 9 that RSA processes the input data set faster than the other three hash functions, suggesting that the CPU and I/O load imposed by RSA verification function is lower than those of the above studied hash functions.

Since the CPU load in RSA is reduced compared with the other three hash functions, Fig. 6.9 confirms that the read bandwidth of the flash drive is the performance bottleneck of the testbed. Due to the fact that very few idle periods exist in the flash drive, it is unlikely to leverage the BUD architecture to reduce energy dissipation of reads in RSA. Fortunately, the results obtained from Fig. 6.9 indicate that executing RSA in the BUD disk architecture can provide energy savings for writes issued to the hard drive. The energy savings become possible for writes in RSA, because there are a large number of small idle periods that can be aggregated by BUD to form larger idle periods.

Next, let us study the access patterns of AES and 3DES - two block/stream encryption algorithms. The AES (Advanced Encryption Standard) is a block cipher standard published by the US government in November 2001. After input data is retrieved from the flash drive in the testbed, AES encrypts the input data with 256-bit keys and stores cipher text on local hard drive in the testbed. Results plotted in Figs. 10 and 11 show that the CPU and I/O load caused by AES are very well balanced. For example, the CPU usage, read/write bandwidth of the testbed running AES tend to reach their upper bounds. When it comes to reads, there is almost no idle period found during the course AES's execution. Although AES does not issue writes as intensively as reads, idle periods in the hard drive are smaller than those of cases for MD5, SHA-1/SHA-2, and RSA.

3DES, CPU-intensive in nature, is very slow in software. For example, it took approximately 190 Sec. for AES to encrypt all the 78 files in the flash drive; 3DES spent more than 25,000 Sec. in processing the same set of files. Fig. 6.12 clearly shows that the CPU load is extremely high and CPU becomes the performance bottleneck in our testbed running 3DES.

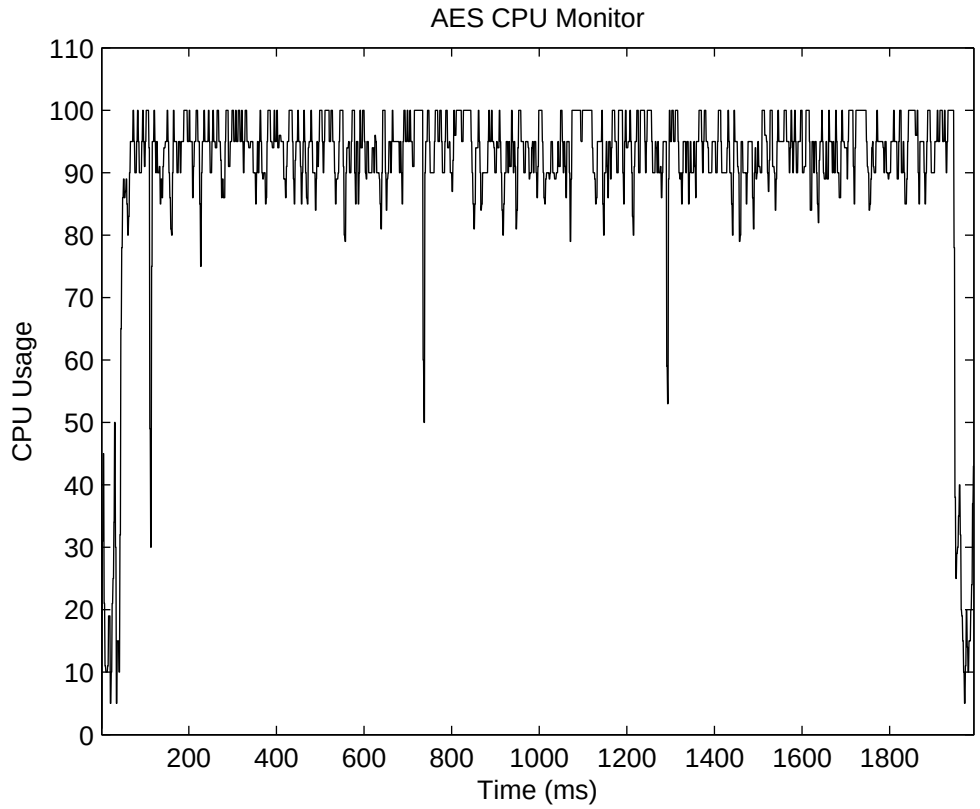


Figure 6.10: CPU usage (measured in terms of percentage) of the testbed when AES is evaluated

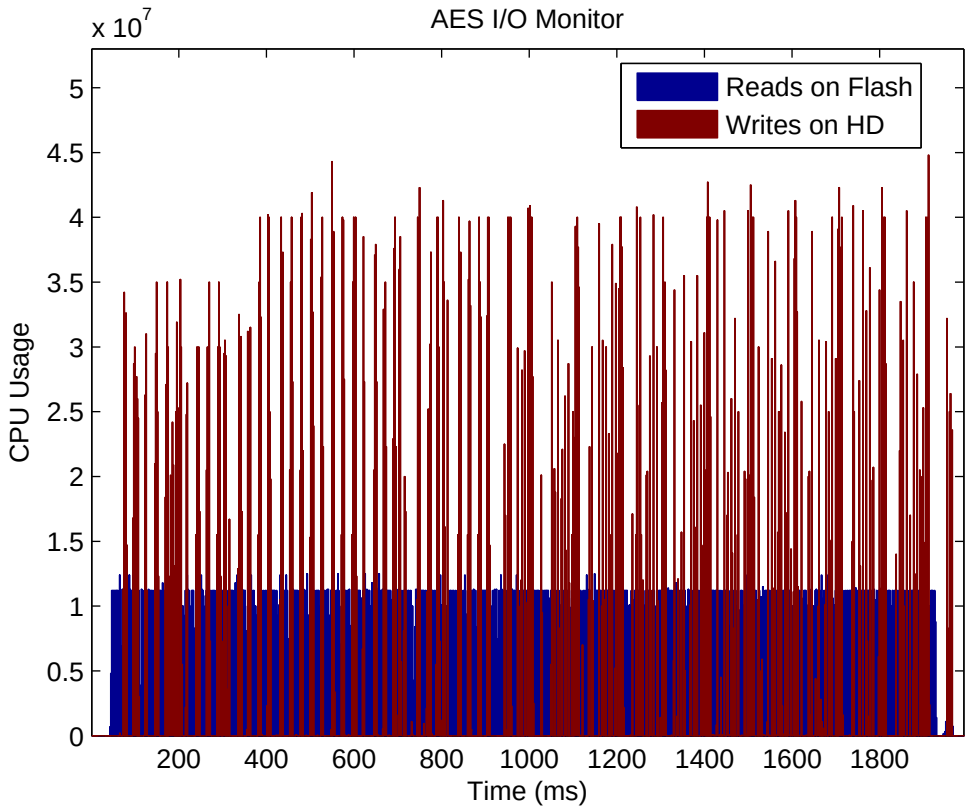


Figure 6.11: Read/write bandwidth of the testbed when AES is evaluated

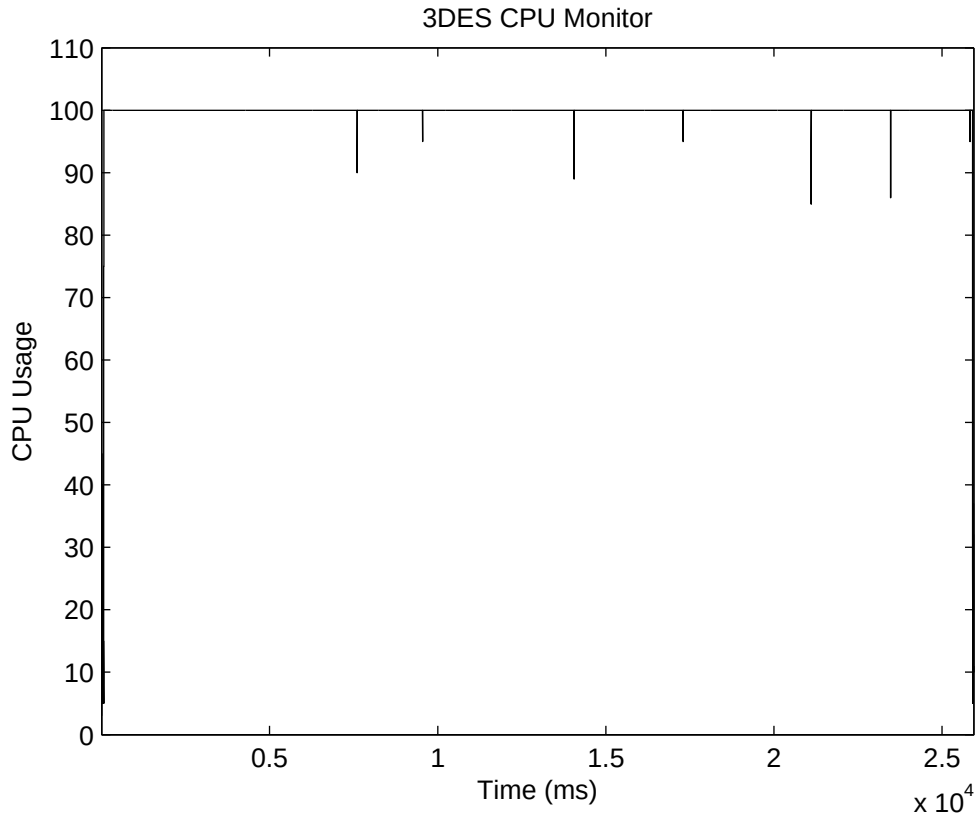


Figure 6.12: CPU usage (measured in terms of percentage) of the testbed when 3DES is evaluated

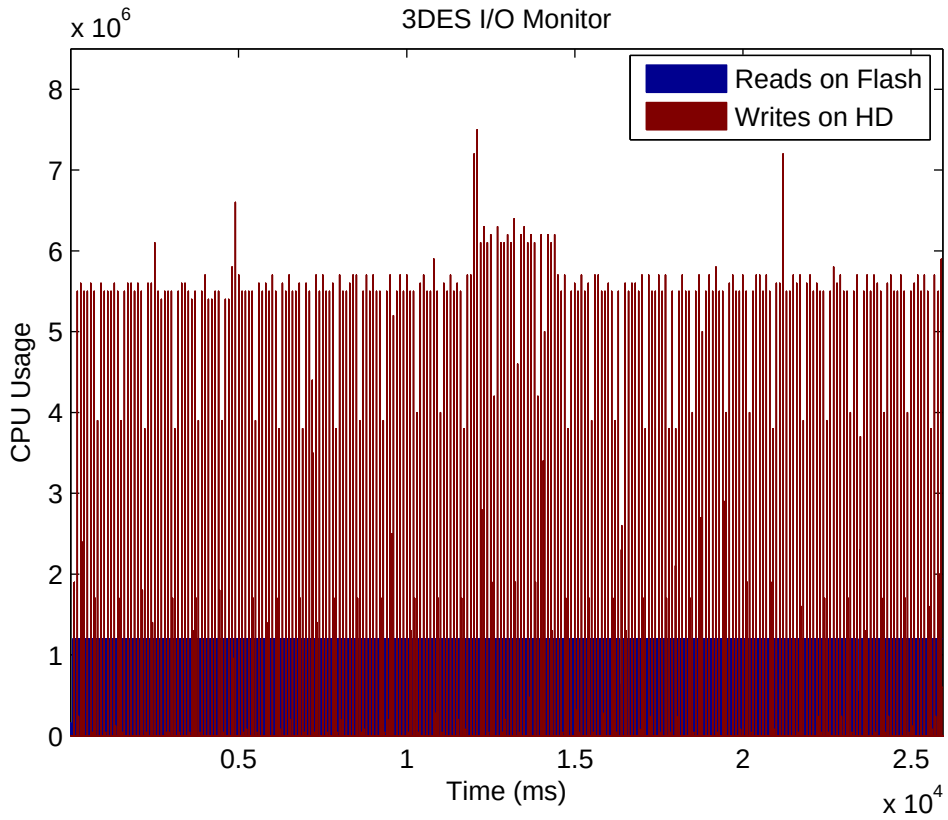


Figure 6.13: Read/write bandwidth of the testbed when 3DES is evaluated

Results depicted in Fig. 6.13 suggest that the BUD disk architecture can be seamlessly integrated with 3DES to cluster small disk idle periods together in large idle time frames, which enable disks to be operated in the standby mode for long time intervals to save energy.

## 6.5 Summary and Future Work

Achieving both high energy efficiency and security in disk systems is challenging, because energy efficiency and data security are often two conflicting goals. There are two general approaches to improving security and energy-efficiency in disk systems. The first approach is to modify existing security mechanisms to enhance energy efficiency of security services in disk systems. In contrast, the second approach is to seamlessly integrate security services with energy-efficient disk systems without modifying security mechanisms. In this research, I took the first step toward the second approach by answering an intriguing question of whether I can improve energy efficiency of security mechanisms in disk systems without changing the source code of security services.

Target energy-efficient disk systems considered in this study is the buffer disk architecture or BUD (see [98] for detailed information regarding BUD). The BUD disk architecture aims at aggregating many small idle periods in disks into a few large idle intervals so that disks can be turned into the standby mode and kept in standby as long as possible to conserve energy. In this research, I first built a testbed containing a disk subsystem, USB-based flash drive, Linux operating systems, and six encryption modules and hash functions. Next, I captured CPU usage and I/O access patterns of a disk system where the encryption modules and hash functions were tested and evaluated. Finally, I analyzed the possibility of leveraging the BUD disk architecture to reduce energy consumption incurred by the existing encryption algorithms and hash functions in the context of disk systems.

Table 6.2 summarizes the CPU utilization and read/write bandwidth of the testbed running the six security services. Table 6.2 shows that except for 3DES, the CPU is not a performance bottleneck for the other five security services. Among the six evaluated security

Table 6.2: Summary of the CPU usage, read/write load of the testbed running the six encryption algorithms and hash functions. The two rightmost columns show possibilities of employing the BUD architecture to save energy for secure disk systems without modifying the security mechanisms. (L: Low, M: Medium, H: High, VH: Very High EH: Extremely High)

| | CPU Load | Read Load | Write Load | Save Energy(Reads)? | Save Energy(Writes)? |
|---|---|---|---|---|---|
| MD5 | M | H | M | Unlikely | Yes |
| SHA1 | M | VH | M | Unlikely | Yes |
| SHA2 | M | VH | M | Unlikely | Yes |
| RSA | M | VH | M | No | Yes |
| AES | VH | VH | M | No | Yes |
| 3DES | EH | M | L | Yes | Yes |

services, 3DES is the only one that can make use of BUD to improve energy efficiency for disk reads and writes. This is mainly because 3DES is very slow in software, leaving many small idle periods in the hard disks and flash drive. Although I was unable to conclude that 3DES is the most energy efficient security service on BUD, it is certain that 3DES is a good representative security mechanism that can benefit a whole lot from BUD. Furthermore, BUD can be employed to reduce energy consumption of writes issued by the MD5, SHA-1, SHA-2, RSA, AES modules (see the rightmost column in Table 6.2). It is unlikely to minimize energy consumption of reads for MD5, SHA-1, SHA-2 using BUD. Even worse, it is almost impossible for BUD to reduce energy consumption of reads for RSA and AES. Now I am positioned to conclude that the BUD disk architecture can provide an ideal energy-efficient data storage platform for security mechanisms, which have high CPU usage and issue sparse disk requests.

As part of future work, I will take full advantage from the generated I/O traces to drive our BUD disk simulator, which allows us to quantitatively study how BUD can produce energy savings for security mechanisms. Currently, i am in a process of building a cluster storage system based on the BUD architecture to investigate if BUD can be applied to clusters to conserve energy. For a large cluster storage system with sufficiently large CPU and disk capacities, it is intriguing to evaluate the energy efficiency of a variety of hash

functions and encryption algorithms on the BUD-based cluster storage system. For the I/O-intensive security services, I will implement a dynamic resource manager in BUD to keep parallel data disks active to achieve high aggregated read bandwidth while allocating a limited number of CPUs to perform integrity checking. In doing so, BUD will be capable of conserving energy dissipation in processors in the cluster storage system. When it comes to CPU-intensive security services, the dynamic resource manager will save energy by keeping all CPUs active to carry out encryption while force a small number of buffer disks to buffer data.

Chapter 7

ES-MPICH2: A Message Passing Interface with Enhanced Security

## 7.1 Introduction

Large cluster computing systems have been widely deployed and utilized by national laboratories, corporations, and government research centers. Some clusters are usually built upon local area networks, which are physically isolated from public networks like the Internet; therefore, the security issue of information exchanging among computing nodes in a locally operated cluster is not a major concern. Without any security mechanism to preserve confidentiality, messages transferred among the computing nodes are just plain-texts that can be easily manipulated. Due to the fast development of the Internet, however, an increasing number of universities and companies are connecting their cluster computing systems to public networks to provide high accessibility. Those networks connecting to the Internet can be accessed by anyone from anywhere. This opens a possibility for security leakages if there is no information assurance protection on classified or confidential data transmitted to and from cluster computing nodes.

Recently, a geographically distributed cluster system proposed by Sun Microsystems has attracted many interests from both academia and industry communities. In largely distributed clusters, computing nodes are geographically deployed in various computing sites. Information processed in a distributed cluster is shared among a group of distributed processes or users by the virtue of message passing protocols (e.g. message passing interface - MPI) running on a public network like the Internet.

Public networks have increasingly become a potential threat to distributed cluster computing environments. Security of clusters connected by a public network must address four

aspects, namely, confidentiality, integrity, availability, and authentication. Rather than addressing all the security aspects, I focus on preserving confidentiality of messages passed among computing nodes in a unsecured cluster. Nevertheless, an integrity checking service can be readily incorporated into our security framework by applying a public-key cryptography scheme. In an MPI framework equipped with the public-key scheme, sending nodes can encode messages using their private keys. In the message receiving procedure, any nodes can use public keys corresponding to the private keys to decode messages. Please refer to Section 7.5.6 for details of how to add integrity checking services in our MPI framework called ES-MPICH2.

Since there is an open accessible nature of the open networks, providing confidentiality services for these large-scale distributed clusters becomes a non-trivial and challenging problem. To address this issue, I enhanced the security of the MPI (Message Passing Interface) protocol by encrypting and decrypting messages sent and received among computing nodes.

In this study I focus on MPI rather than other protocols, because MPI is one of the most popular communication protocols for cluster computing environments. Numerous scientific and commercial applications running on clusters were developed using the MPI protocol. Among a variety of MPI implementations, I picked MPICH2 developed by the Argonne National Laboratory. The design goal of MPICH2 - a widely used MPI implementation - is to combine portability with high performance [42]. I integrated encryption algorithms into the MPICH2 library Thus, data confidentiality of MPI applications can be readily preserved without a need to change the source codes of the MPI applications. Data communications of a conventional MPI program can be secured without converting the program into the corresponding secure version, since I provide a security enhanced MPI-library with the standard MPI interface. In what follows, I summarize the four major contributions of this study.

• I implemented a standard MPI mechanism called ES-MPICH2 to offer data confidentiality for secure network communications in message passing environments. Our proposed

security technique incorporated in the MPICH2 library can be very useful for protecting data transmitted in open networks like the Internet.

• The ES-MPICH2 mechanism allows application programmers to easily write secure MPI applications without additional code for data-confidentiality protection. I seek an channel-level solution in which encryption and decryption functions are included into the MPICH2 library. Our ES-MPICH2 maintains a standard MPI interface to exchange messages while preserving data confidentiality.

• The implemented ES-MPICH2 framework provides a secured configuration file that enables application programmers to selectively choose any cryptographic algorithm and symmetric-key in ES-MPICH2. This feature makes it possible for programmers to easily and fully control the security services incorporated in the MPICHI2 library. To demonstrate this feature, I implemented the AES and Triple DES algorithms in ES-MPICH2. I also show in this research how to add other cryptographic algorithms in to the ES-MPICH2 framework.

• I have used ES-MPICH2 to perform a detailed case study using the Sandia Micro Benchmarks and the Intel MPI benchmarks. I focus on runtime performance overhead introduced by the cryptographic algorithms.

This Chapter is organized as follows: Section 7.2 demonstrates the vulnerabilities of existing MPI implementations by describing a security threat model for clusters connected by public networks. Section 7.3 not only provides a reason for focusing on the confidentiality issue of MPICH2 rather than other MPI implementations, but also gives an overview of the MPICH2 implementation. Section 7.4 presents the motivation of this work by showing why secured MPI is an important issue and also outlines the design of ES-MPICH2 - the message passing interface with enhanced security. Section 7.5 describes the corresponding implementation details of ES-MPICH2. Section 7.6 discusses some experimental results and compares the performance of ES-MPICH2 with that of MPICH2. Finally, Section 7.7 states the summary and future work of this study.

## 7.2 Threat Model

A geographically distributed cluster system is one in which computing components at local cluster computing platforms communicate and coordinate their actions by passing messages through public networks like the Internet. To improve the security of clusters connected to the public networks, one may build a private network to connect an array of local clusters to form a large scale cluster. Building a private network, however, is not a cost-effective way to secure distributed clusters. The Internet - a very large distributed system - can be used to support large-scale cluster computing. Being a public network, the Internet becomes a potential threat to distributed cluster computing environments.

Confidentiality, integrity, availability, and authentication are four important security issues to be addressed in clusters connected by an unsecured public network. In this study, I pay particular attention to confidentiality services for messages passed among computing nodes in a unsecured cluster. Although preserving confidentiality is our primary concern, an integrity checking service can be seamlessly incorporated into the ES-MPICH2 framework by applying a public-key cryptography scheme (see Section 7.5.6 for an approach to incorporating integrity services in ES-MPICH2). For example, sending nodes encode messages using private keys. If one alters the messages, the ciphertext can not be deciphered correctly using public keys corresponding to the private keys. Thus, the receiving nodes can perform message integrity check without the secure exchange of secret keys.

I first describe the confidentiality aspect of security in clusters followed by three specific attack instances. I believe new attacks are likely to emerge, but the confidentiality aspect will remain unchanged. Confidentiality attacks attempts to expose messages being transmitted among a set of collaborating processes in a cluster. For example, if attackers gain network administrator privilege, they can intercept messages and export the messages to a database file for further analysis. Even without legitimate privilege, an attacker still can sniff and intercept all messages in a cluster on the public network. Such attacks result in

the information leakage of messages passed among computing nodes in geographically distributed clusters. Cryptography and access control are widely applied to computer systems to safeguard against confidentiality attacks.

I identify the following three confidentiality attacks on MPI programs running on distributed clusters:

• *Sniffing Message Traffic*: Message traffic of an MPI program can be sniffed. For example, when MPCH2 is deployed in a cluster connected by a Gigabit Ethernet network, attackers can sniff plaintext messages transmitted through the TCP socket. Message sniffing can reveal security-sensitive data, metadata, and information.

• *Snooping on Message Buffer*: In an MPI program, buffers are employed to send and receive messages. Regardless of specific MPI implementations, message buffers are created before the send and receive primitives are invoked. Attackers who snoop into the message buffers in memory can access data and information without being given specific access privileges.

• *Message Traffic Profiling*: Message traffic profiling attacks seek to use message type, timestamps, message size, and other metadata to analyze message exchange patterns and types of protocols being used in message transmissions. For example, an attacker can monitor the network connection of a cluster running an MPI program. If a message has been regularly transmitted, the attacker can speculate the importance of the message and intercept the content of the message.

Confidentiality services can effectively counter the aforementioned threats in MPI applications running on clusters connected by a public network. In this research, I encode messages using the Advanced Encryption Standard (AES) and the Triple Data Encryption Standard (Triple-DES or 3DES). In the case that attackers intercept messages in an MPI program, they are unable to transform the ciphertext into the original plaintext due to the lack of data encipherment keys.

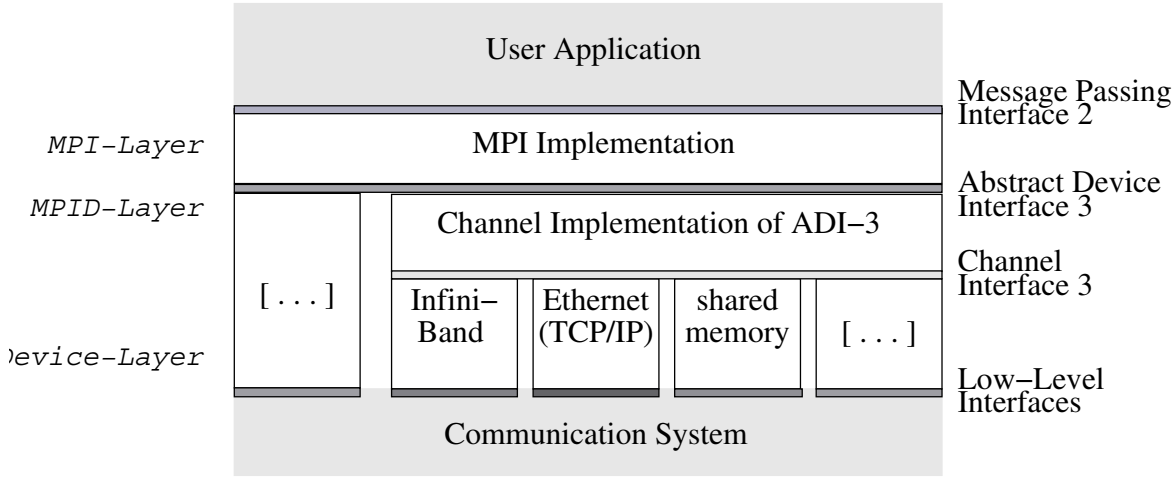


Figure 7.1: Hierarchical Structure of MPICH2 [40]

### 7.3 MPICH2 Overview

MPICH - one of the most popular MPI implementations - were developed at the Argonne National Laboratory [42]. The early MPICH version supports the MPI-1 standard. MPICH2 - a successor of MPICH - not only provides support for the MPI-1 standard, but also facilitates the new MPI-2 standard, which specifies functionalities like one-sided communication, dynamic process management, and MPI I/O [40]. Compared with the implementation of MPICH, MPICH2 was completely redesigned and developed to achieve high performance, maximum flexibility, and good portability.

Fig. 7.1 shows the hierarchical structure of the MPICH2 implementation, where there are four distinct layers of interfaces to make the MPICH2 design portable and flexible. The four layers, from top to bottom, are the message passing interface 2 (MPI-2), the abstract device interface (ADI3), the channel interface (CH3), and the low-level interface. ADI3 - the third generation of the abstract device interface - in the hierarchical structure (see Fig. 7.1) allows MPICH2 to be easily ported from one platform to another. Since it is non-trivial to implement ADI3 as a full-featured abstract device interface with many functions, the CH3 layer simply implements a dozen functions in ADI3 [69].

As shown in Fig. 7.1, the TCP socket Channel, the shared memory access (SHMEM) channel, and the remote direct memory access (RDMA) channel are all implemented in the layer of CH3 to facilitate the ease of porting MPICH2 on various platforms. Note that each one of the aforementioned channels implements the CH3 interface for a corresponding communication architecture like TCP sockets, SHMEM, and RDMA. Unlike an ADI3 device, a channel is easy to implement since one only has to implement a dozen functions relevant for with the channel interface.

To address the issues of message snooping in the message passing environments on clusters, I seek to implement a standard MPI mechanism with confidentiality services to counter snooping threats in MPI programs running on a cluster connected an unsecured network. More specifically, I aim to implement cryptographic algorithms in the TCP socket channel in the CH3 layer of MPICH2 (see Fig. 7.2 and section 7.5 for details of how to construct a cryptosystem in the channel layer).

## 7.4 Description of ES-MPICH2

### 7.4.1 Motivation

Computing nodes in a distributed cluster are geographically deployed in several computing sites on an open network. Processes within a communication group need to transmit security-sensitive messages through unsecured communication channels. Preserving data confidentiality in a message passing environment over an untrusted network is critical for a wide spectrum of security-aware MPI applications, because unauthorized access to the security-sensitive messages by untrusted processes can lead to serious security breaches. Hence, it is imperative to protect confidentiality of messages exchanged among a group of trusted processes.

There are three common approaches to improving security of MPI applications. In first approach, application programmers can add source code to address the issue of message confidentiality. Such an application-level security approach not only makes the MPI applications

error-prone, but also reduce the portability and flexibility of the MPI applications. In the second approach, the MPI interface can be extended in the way that new security-aware APIs are designed and implemented (see, for example, MPISec I/O [90]). This MPI-interface-level solution enables programmers to write secure MPI applications with minimal changes to the interface. Although the second approach is better than the first one, this MPI-interface-level solution typically requires extra code to deal with data confidentiality. The third approach - a channel-level solution - is proposed in this study to address the drawbacks of the above two approaches. The channel-level solution aims at providing message confidentiality in a communication channel that implements the Channel Interface 3 in MPICH2 (see Fig. 7.1).

### 7.4.2 The Design of ES-MPICH2

The goal of the development of the ES-MPICH2 mechanism is to enable application programmers to easily implement secure enhanced MPI applications without additional code for data-confidentiality protection. In our programming model, processes belonging to a communication group trust each other. Any process that is outside of this communication group is not trusted by the member processes in the group. In other words, trusted processes within a communication group (i.e., a group of processes spawned in an MPI application) can efficiently encrypt and decrypt messages delivered among the trusted processes. The communication group as a whole will have to control the encryption and decryption procedures. With ES-MPICH2 in place, secure MPI application programmers are able to flexibly choose a cryptographic algorithm, key size, and data block size for each MPI application that needs data confidentiality protection.

ES-MPICH2 offers message confidentiality in an MPI programming environment by incorporating MPICH2 with encryption and decryption algorithms. In the process of designing ES-MPICH2, I integrated the AES and 3DES algorithms into the MPICH2 library. Unlike the application-level and MPI-interface-level security solutions, ES-MPICH2 encrypts and

decrypts messages at the transmission layer without modifying the MPI interface or application source code. ES-MPICH2 maintains a standard MPI interface to exchange messages while preserving data confidentiality. Thus, this feature of ES-MPICH2 allows application programmers to easily write secure MPI applications with no requirement of additional code to deal with message confidentiality protection.

The ES-MPICH2 implementation has the following four design goals:

• **Message Confidentiality:** ES-MPICH2 aims to preserve message confidentiality from unauthorized accesses by untrusted processes. I leverage the AES to protect the confidentiality of messages, because AES is an encryption standard adopted by the U.S. government. For comparison purpose, I also consider 3DES in the design of ES-MPICH2. AES with 128-bit keys can provide adequate protection for classified messages up to the SECRET level. The implementation of AES in products intended to protect national security systems and/or information must be reviewed and certified by NSA prior to their acquisition and use [6]. In the next section (i.e., Section 7.5), I will explain why I paid particular attention on block ciphers like AES. In this study, I integrated data confidentiality services with MPICH2 by implementing the cryptographic algorithms in a CH3 channel. I will also discuss in section 7.5 the integration of security services with the TCP socket channel in the CH3 layer of MPICH2.

• **Complete Transparency:** ES-MPICH2 is intended to improve the security of any conventional MPI program without adding extra code to perform message protection. Thus, preserving message confidentiality in MPICH2 is entirely transparent to application programmers. Such confidentiality transparency is feasible and the reason is two-fold. First, the encryption and decryption processes can be built in the MPICH2 library at the channel transmission layer. Second, I maintain the same interface as the APIs of the MPICH2 implementation. Therefore, I can enhance the security of conventional MPI applications by running the applications in the ES-MPICH2 environment without modifying the source code.

• **Compatibility and Portability:** Ideally, ES-MPICH2 needs to be easily ported from one platform to another with no addition to the application source code. ES-MPICH2 is an extension of MPICH2 and; thus, ES-MPICH2 should have the same level of portability as MPICH2. However, it is challenging to achieve high portability in ES-MPICH2, because I have to implement a cryptographic subsystem in each channel in the CH3 layer in MPICH2. In the current version of ES-MPICH2, only a secured TCP socket channel has been implemented and tested. Thus, ES-MPICH2 should work without modifications on any parallel and distributed computing system that supports TCP communications and the MPICH2 library. The SHMEM and RDMA channels have not been completely supported in ES-MPICH2 and; therefore, the focus of this research is the TCP socket channel in MPICH2. Nevertheless, one can take a similar approach described in Section 7.5 to integrating data confidentiality services with the SHMEM and RDMA channels in the CH3 layer of MPICH2.

• **Extensibility:** ES-MPICH2 must allow application programmers to selectively choose any cipher techniques and keys incorporated in MPICH2. This design goal makes it possible for programmers to flexibly select any cryptographic algorithm implemented in ES-MPICHI2. Although I implemented AES and 3DES in the channel layer of MPICH2, I will show in the next section how to add other cryptographic algorithms (e.g., Elliptic Curve Cryptography, [12]) to the ES-MPICH2 environment.

## 7.5 Implementation Details

During the implementation of ES-MPICH2, I addressed the following five development questions: (1) Among the multiple layers in the hierarchical structure of MPICH2, in which layer should I implement cryptographic algorithms? (2) Which cryptosystem should I choose to implement? (3) How to implement secure key management? (4) How to use the implemented ES-MPICH2? (5) How to add integrity checking services to ES-MPICH2?

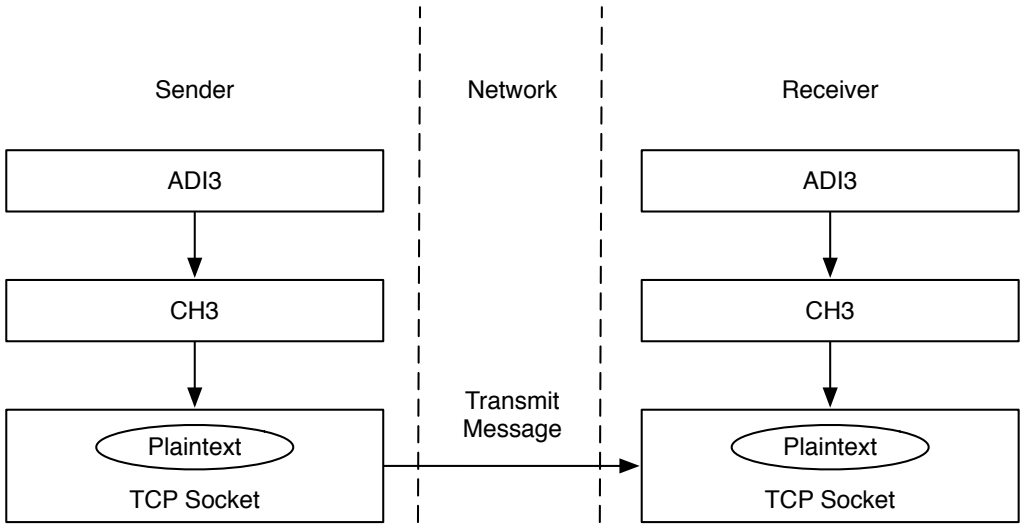


Figure 7.2: Message passing implementation structure in MPICH2.

### 7.5.1 Ciphers in the Channel Layer

Fig. 7.2 outlines the message passing implementation structure in the original version of MPICH2. In such a hierarchical structure of MPICH2, messages are passed from a sending process to a receiving process through the abstract device interface (ADI3), the channel interface (CH3), and the TCP socket channel. Cryptographic subsystems may be implemented in one of the three layers (i.e., ADI3, CH3, or the TCP socket channel). To achieve the design goal of complete transparency, I chose to implement cryptographic algorithms in the TCP socket channel. Compared with ADI3 and CH3, the TCP socket channel is the lowest layer of the MPICH2 hierarchy. Implementing cryptosystems in the lowest layer can preserve message confidentiality in any conventional MPI program without adding extra code to protect messages. Fig. 7.3 depicts the implementation structure of ES-MPICH2, where a cryptosystem is implemented in the TCP socket layer. Thus, messages are encrypted and decrypted in the TCP socket channel rather than the ADI3 and CH3 layers.

Fig. 7.4 shows that the encryption and decryption functions in ES-MPICH2 interact with the TCP socket to provide message confidentiality protection in the TCP socket layer. Before a message is delivered through the TCP socket channel, data contained in the message

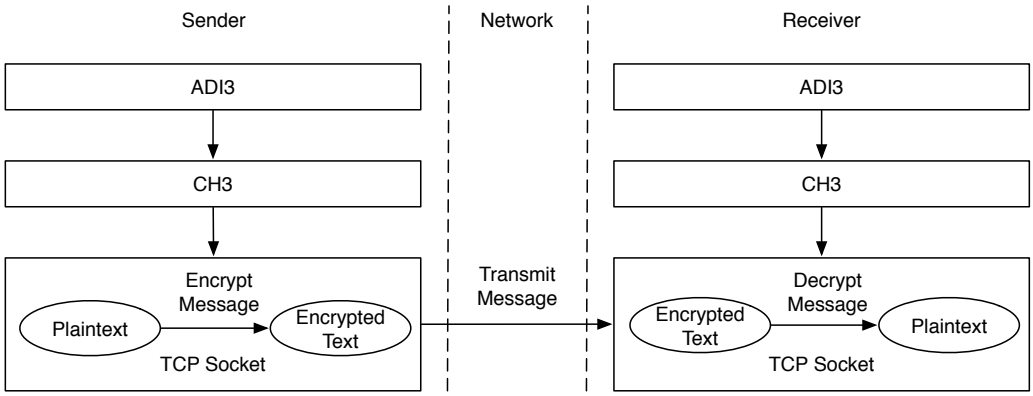


Figure 7.3: Message passing implementation structure in ES-MPICH2 with encryption and decryption processes. A cryptosystem is implemented in the TCP socket layer to achieve the design goal of complete transparency.

is encrypted by a certain cryptographic algorithm like AES and 3DES. Upon the arrival of an encrypted message in a receiving node, the node invokes the corresponding decryption function to decrypt the message. Fig. 7.4 demonstrates that ES-MPICH2 maintains the same application programming interface or API as that of MPICH2 by implementing the encryption and decryption algorithms in the TCP socket level. The confidentiality services of ES-MPICH2 were implemented in the MPICH2 libraries, thereby being totally transparent to MPI application programmers.

### 7.5.2 Block Ciphers

In the ES-MPICH2 framework, I implemented the AES and 3DES cryptographic algorithms in MPICH2 version 1.0.7. I focus on block ciphers in the implementation of ES-MPICH2, because a block cipher transforms a fixed-length block of plaintext into a block of ciphertext of the same length. If the case where ciphertext and plaintext are different in length, MPI applications have to be aware of such a difference in order to correctly decode ciphers. Keeping in mind that securely passing messages should be transparent to MPI application programmers, I advocate the use of block ciphers rather than non-block-ciphers that force programmers to be aware of the lengths of plaintext and ciphertext.

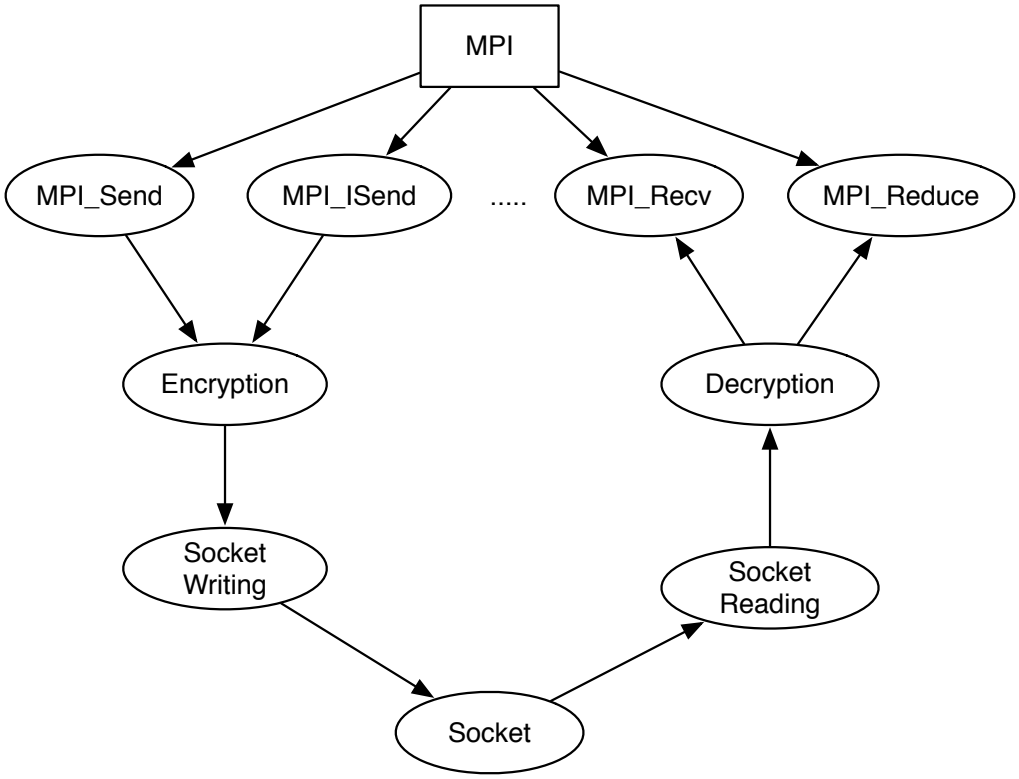


Figure 7.4: The interface between the encryption/decryption processes and the TCP socket. ES-MPICH2 maintains the same API as that of MPICH2.

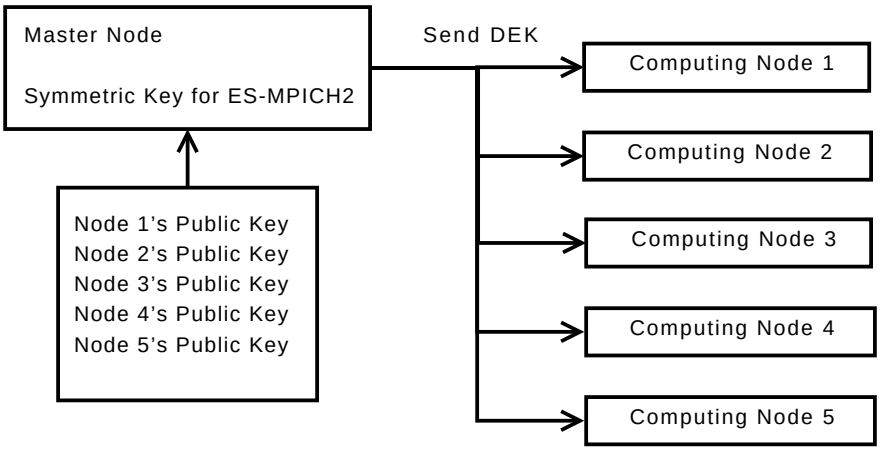


Figure 7.5: Key management in ES-MPICH2. Public key cryptography employed in ES-MPICH2 relies on interchange keys (i.e., public and private keys) to exchange data encipherment keys (DEK) in a secure way.

### 7.5.3 Key Management

The goal of key management is to dynamically establish secure message-passing channels by distributing cryptographic keys. ES-MPICH2 maintains two types of keys - data encipherment keys (a.k.a., session keys) and interchange keys. A session key is a randomly generated cryptographic key associated with message passing among computing nodes. A interchange key is a cryptographic key bound to a particular computing node. In the MPI initialization phase of each MPI application, a data encipherment key is created and shared among all the communicating processes.

Fig. 7.5 presents the key management infrastructure in ES-MPICH2. Public key cryptography employed in ES-MPICH2 relies on interchange keys (i.e., public and private keys) to securely exchange session keys in an unsecured network. More specifically, when a master node attempts to share new session keys with other slave nodes, the master node uses the slave nodes' public keys to encrypt the session keys. The slave nodes make use of their private keys to decipher messages containing the session keys. Then, the master node and slave nodes can securely communicate using the MPI framework.

Please note that the key management infrastructure in ES-MPICH2 is different from that in MPISec I/O because I put the exchange keys into configuration files in each node. In

this infrastructure, I use operating system access control mechanism to protect and encrypt files storing keys. Enciphering the configuration files containing the interchange keys may not work in a cluster connected by an unsecured network because an attacker may trick system administrators into downloading a malware program that captures keystrokes as well as the configuration files and delivers them to the attacker. A more secure solution is to store interchange keys on physical devices like smart card and ROM. Please refer to [25][29][76] for details of how to store and protect interchange keys.

### 7.5.4 Socket Programming

In socket programming, there is a buffer containing data sent and received through the TCP socket channel. Fig. 7.6 demonstrates the encryption and decryption process in ES-MPICH2. Originally, MPICH2 fills the socket buffer with data in plaintext. The data is written to the socket of a sending node and; then, a receiving node fills its receiving buffer with the plaintext data read from socket. In ES-MPICH2, plaintext data is initially stored in the sending buffer. Next, ES-MPICH2 encrypts the plaintext data in the sending buffer. Thus, the plaintext is replaced by the ciphertext. Finally, the data written in the socket of the sending node is encrypted using a specific encryption algorithm. The socket buffer in the receiving node is filled with the encrypted data directly received from the socket. Then, MPICH2 decrypts the ciphertext, which is replaced by the plaintext in the receiving buffer. Because the plaintext and ciphertext are identical in length in block cipher algorithms, the sizes of the buffers in both the sending and receiving nodes remain unchanged after the encryption and decryption processes.

### 7.5.5 Usage

The security features of ES-MPICH2 can be configured without modifying any MPI application source code. To securely pass messages using ES-MPCH2, the following configurations must be set before MPI initialization. First, a security option should be enabled

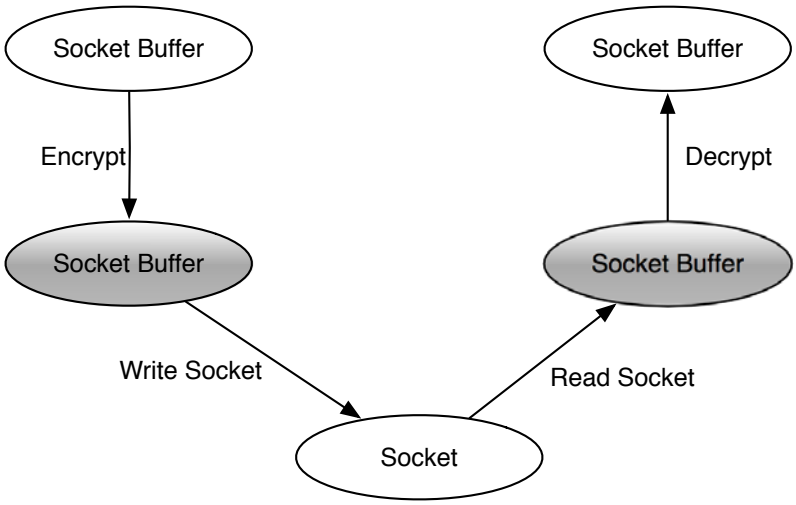


Figure 7.6: ES-MPICH2 Socket Details

or disabled. If the security feature disabled, ES-MPICH2 gracefully degrades to the original version of MPICH2 (v1.0.7). Second, one has to select a specific cryptographic algorithm implemented in ES-MPICH2. In the current version of ES-MPICH2, MPI application users can choose one out of two cryptographic schemes - AES and 3DES. The extensibility feature of ES-MPICH2 allows programmers to choose other cryptography techniques other than AES and 3DES by implementing and adding a wide range of block cipher algorithms in ES-MPICH2. Third, exchange keys must be securely stored in a configuration file or a physical device in each node (see Section 7.5.3 for details on the key management issue.) After a particular security feature is configured, users can run their MPI programs in the same way as they should run the programs in MPICH2. Thus, if an MPI program can be executed in MPICH2, one can also run the MPI program in ES-MPICH2 without modifying the source code of the program.

### 7.5.6 Incorporating Integrity Services in ES-MPICH2

In addition to confidentiality services, integrity checking services can be seamlessly incorporated into the ES-MPICH2 framework. In what follows, I address the implementation issue of how to integrate integrity checking services in ES-MPICH2.

**Spreading feature of block encryption algorithms:** Block encryption algorithms have a spreading feature in which means if even 1 bit is changed in ciphertext, the decrypted text will be completely different from the original plaintext. Altered messages causing fatal errors can not be interpreted. Although using the spreading feature is not a reliable solution, the spreading feature does provide an integrity checking method. Since both AES and 3DES are block encryption algorithms, ES-MPICH2 may rely on the spreading feature to perform integrity checking.

**Public Key:** An integrity service can be added into ES-MPICH2 using a public-key encryption scheme, in which sending nodes encode messages using private keys whereas receiving nodes use the corresponding public keys to decipher the ciphertext. Before a node delivers an encrypted message in ES-MPICH2, the node encrypts the message using the private key of the sending node. To check the integrity of the cipher message, a receiving node simply needs to decode the cipher message by applying the public key of the sending node.

**Hash Functions:** Hash functions are widely used in integrity checking and digital signatures. MD5, SHA-1, and SHA-2 can be implemented to check the integrity of encrypted messages in ES-MPICH2. The hash is a cryptographic checksum or message integrity code that sending and receiving nodes must compute to verify messages. For example, a sending node can use a hash function and a shared key to compute the checksum for a message. A receiving node needs to perform the same hash function on the received message using the shared key. A hash function is applied after a message is encrypted in a sending node. The same hash function is used before the ciphertext is decoded in a receiving node. In doing so, the integrity of encrypted messages can be checked.

## 7.6 Experimental Evaluation

To evaluate the features and performance of ES-MPICH2, I implemented ES-MPICH2 and deployed it on two clusters with different configurations. The first cluster has six nodes

of 2.2 GHz Intel Celeron processors with 2 GB memory. The second cluster contains ten nodes. The master node has a 3.0 GHz Intel Pentium Core 2 Duo processor with 1 GB memory, whereas the nine slave nodes have 333 MHz Intel Pentium II processors with 64 MB memory. The six nodes in the first cluster are connected by a 1 Gbps Ethernet LAN. The 10 nodes in the second cluster are connected by a 100 Mbps Ethernet LAN. Apparently, the overall performance of the first cluster is higher than that of the second cluster.

Table 7.1: The Configuration of A 6-NodeCluster of Intel Celeron Processors

| | Node ×6 |
|---|---|
| CPU | Intel Celeron 450 2.2GHz |
| Memory | 2GB |
| OS | Ubuntu 9.04 Jaunty Jackalope |
| Kernel version | 2.6.28-15-generic |
| Network | 1000Mbps |

### 7.6.1 A 6-node Cluster of Intel Celeron Processors

**Experimental Testbed**

Let us first evaluate the performance of both MPICH2 and ES-MPICH2 on a 6-node cluster. Table 7.1 reports the configuration of the first cluster with six identical computing nodes of Intel Celeron processors. The operating system used in the six nodes is Ubuntu 9.04 Jaunty Jackalope. The computing nodes are connected by a 1 Gbps network. All the slave nodes share a disk on the master node through the network file system (NFS) [102]. The MPI library used in the 6-node cluster is MPICH2 version 1.0.7. I run the Sandia Micro Benchmarks and the Intel MPI Benchmarks to evaluate and compare the performance of MPICH2 and ES-MPICH2. When I test ES-MPICH2 in each experiment, I set the cryptographic service to AES and 3DES, respectively. The length of data encipherment keys generated and distributed in ES-MPICH2 is 192-bit.

Table 7.2: Performance Metrics used in the Sandia Micro Benchmark Suite (SMB)

| Metric | Explanation |
|---|---|
| iter_t | total amount of time for the loop to complete |
| work_t | for each iteration of the post-work-wait loop the amount of work performed |
| overhead_t | the length of time that a processor is engaged in the transmission or reception of each message |
| base_t | message transfer time calculation threshold |

**SMB: Sandia Micro Benchmark**

The Sandia National Laboratory developed the Sandia Micro Benchmark Suite (a.k.a., SMB) to evaluate and test high-performance network interfaces and protocols. Table 7.2 described the four performance metrics used in the SMB benchmark suite. These metrics include total execution time (i.e., iter_t), CPU execution time for iterations (i.e., work_t), message passing overhead (i.e, overhead_t), and message transfer time calculation threshold (i.e, threshold or base_t). The detailed information on these metrics can be found at http://www.cs.sandia.gov/smb. Please note that the message passing overhead can be derived by subtracting the CPU execution time from the total execution time. Each benchmark has 1000 iterations.

Fig. 7.7 shows the total execution time of the SMB benchmark running on the original MPI implementation (i.e., MPICH2) as well as AES-based ES-MIPCH2 and 3DES-based ES-MPICH2. I observe from this figure that when the message size is small (e.g., 1 KB), the performance of ES-MPICH2 is very close to that of MPICH2. These results indicate that ES-MPICH2 can preserve confidentiality of small messages with negligible overhead.

Fig. 7.8, Fig. 7.9, Fig. 7.10, and Fig. 7.11 show the total execution time, CPU time, overhead, and threshold of MPICH2 and ES-MPICH2 when the message size is set to 2 KB, 16 KB, 128 KB, 512 KB, and 1024 KB, respectively. The results plotted in Fig. 7.8 show that AES-based ES-MPICH2 and MPICH2 have similar performance in the case of small messages. However, when 3DES is employed in ES-MPICH2, the security overhead of ES-MPICH2 becomes noticeable even for small messages. Figs. 7.9- 7.11 illustrate that both

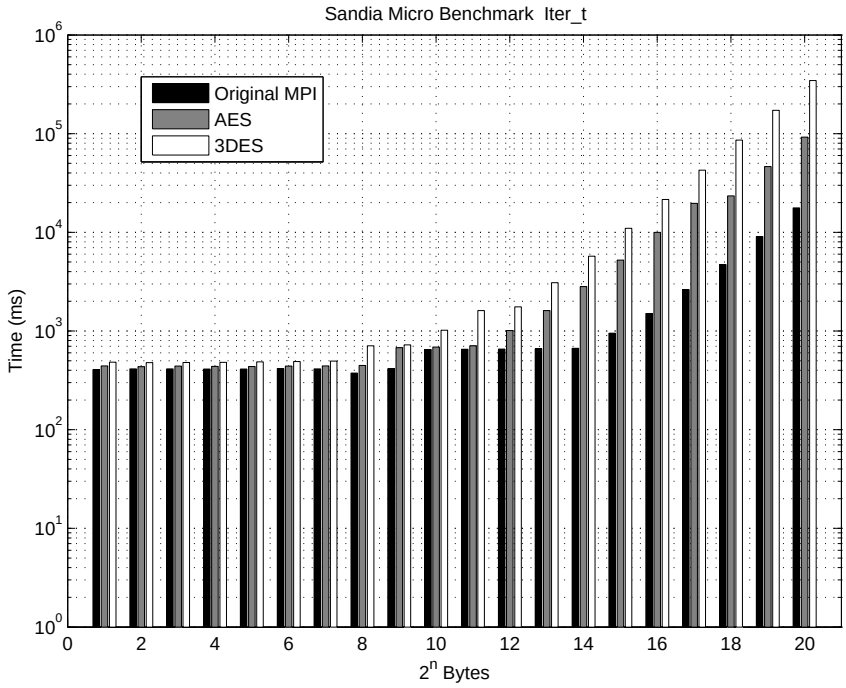


Figure 7.7: Sandia Micro Benchmark iter_time

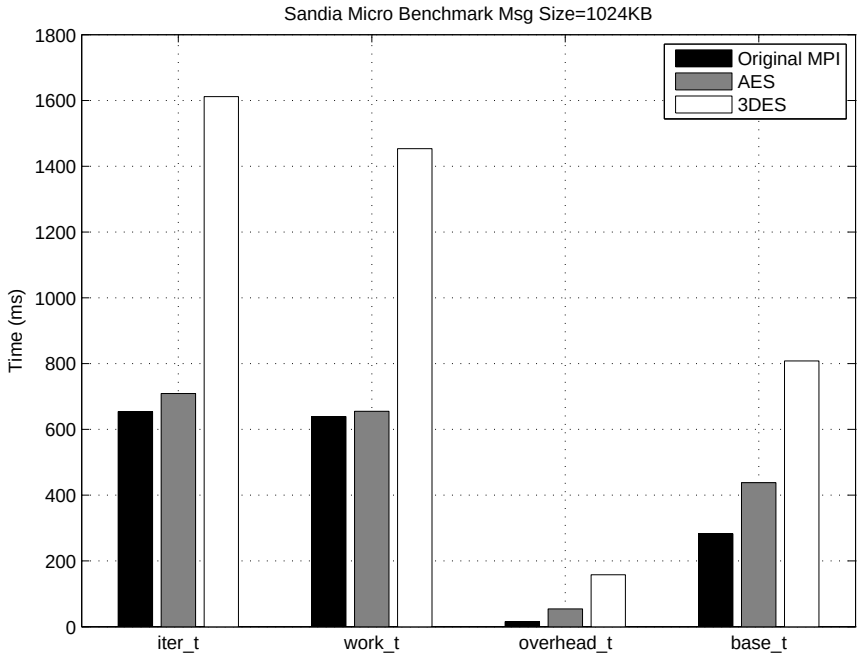


Figure 7.8: Sandia Micro Benchmark Message Size is 2KB

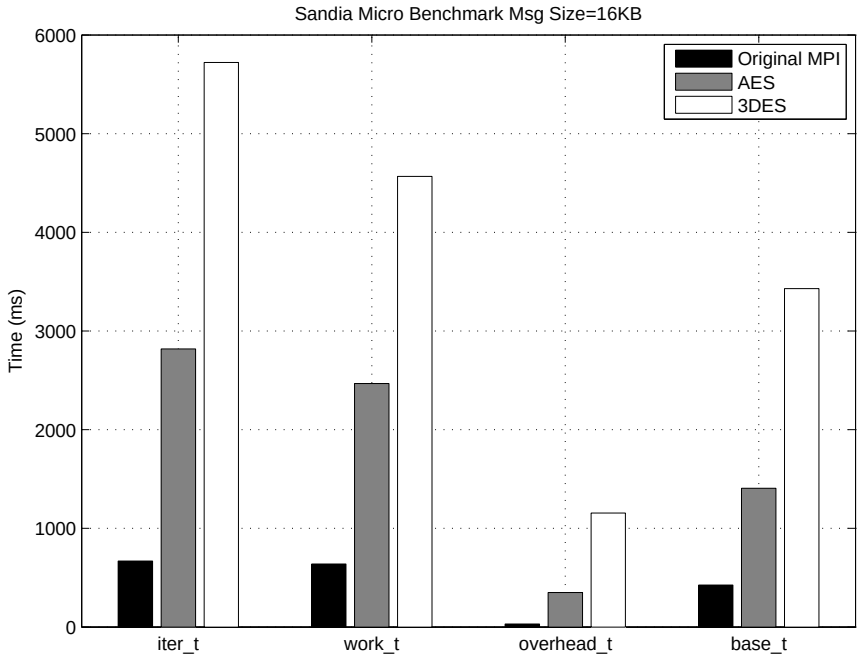


Figure 7.9: Sandia Micro Benchmark Message Size is 16KB

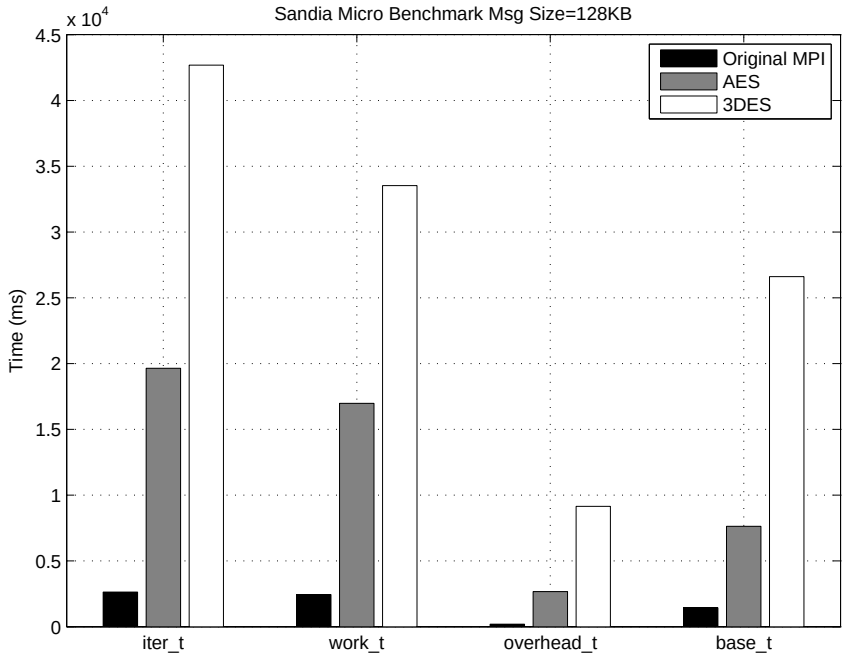


Figure 7.10: Sandia Micro Benchmark Message Size is 128KB

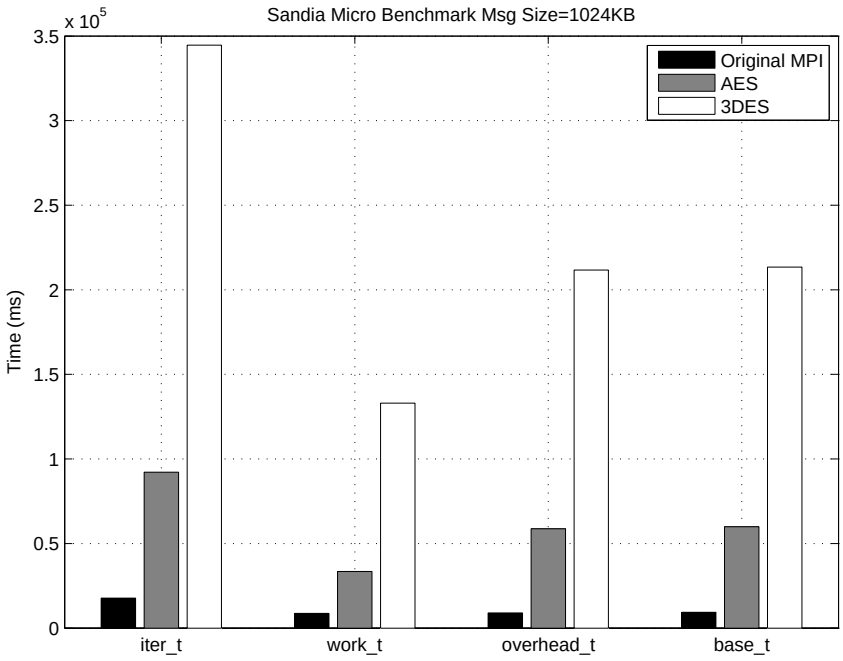


Figure 7.11: Sandia Micro Benchmark Message Size is 1024KB

AES and 3DES in ES-MPICH2 introduce much overhead that makes ES-MPICH2 performs worse than MPICH2 - the original MPI implementation. Security overhead in ES-MPICH2 becomes more pronounced with increasing message size. Since AES has better performance than 3DES, AES-based ES-MPICH2 is superior to 3DES-based ES-MPICH2. I recommend the following two approaches to lowering overhead caused by encryption and decryption modules in ES-MPICH2. First, one can reduce the security overhead in ES-MPICH2 by enhancing the performance of block cipher algorithms. Second, multicore processors can boost efficiency of the encryption and decryption modules, thereby benefiting the performance of ES-MPICH2.

### IMB: Intel MPI Benchmarks

The Intel MPI benchmark suite or IMB was developed for testing and evaluating implementations of both MPI-1 [26] and MPI-2 [37] standards. IMB contains approximately 10,000

Table 7.3: Intel MPI Benchmarks

| Benchmarks | Classification | Semantics |
|---|---|---|
| PingPong | Single Transfer | MPI-1 |
| PingPing | Single Transfer | MPI-1 |
| Sendrecv | Parallel Transfer | MPI-1 |
| Exchange | Parallel Transfer | MPI-1 |
| Bcast | Collective | MPI-1 |
| Allgather | Collective | MPI-1 |
| Allgatherv | Collective | MPI-1 |
| Scatter | Collective | MPI-1 |
| Scatterv | Collective | MPI-1 |
| Gather | Collective | MPI-1 |
| Gatherv | Collective | MPI-1 |
| Alltoall | Collective | MPI-1 |
| Alltoallv | Collective | MPI-1 |
| Reduce | Collective | MPI-1 |
| Reduce_Scatter | Collective | MPI-1 |
| Allreduce | Collective | MPI-1 |
| Window | Other | MPI-2 |

lines of code to measure the performance of important MPI functions [23][99]. I have evaluated the performance of ES-MPICH2 and the original MPICH2 by running the benchmarks on the 6-node cluster. Table 7.3 lists all the Intel benchmarks used to measure the performance of ES-MPI2 and MPICH2. The benchmarks in IMB-MPI1 can be categorized in three groups: single transfer, parallel transfer, and collective benchmarks. Single transfer benchmarks are focusing on a single message transferred between two communicating processes. Unlike single transfer benchmarks, parallel transfer benchmarks aim at testing patterns and activities in a group of communicating processes with concurrent actions. Collective benchmarks are implemented to test higher level collective functions, which involve processors within a defined communicator group. Please refer to http://software.intel.com/en-us/articles/intel-mpi-benchmarks for more information concerning IMB.

Figs. 7.12 and 7.13 show the performance of PingPong and PingPing - two single transfer benchmarks in IMB. Since single transfer benchmarks are used to test a pair of two active processes, I run PingPong and PingPing on two nodes of the 6-node cluster. The total

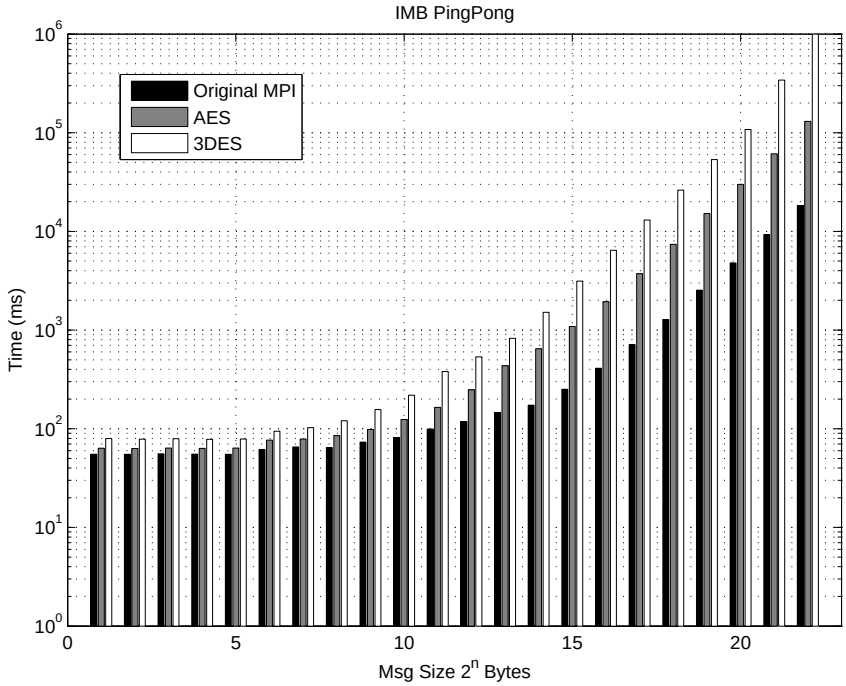


Figure 7.12: Intel MPI Single Transfer Benchmark PingPong

execution times of PingPong and PingPing go up when the message size increases because larger messages give rise to higher encryption and decryption overheads. Compared with MPICH2, the execution times of AES-based and 3DES-based ES-MPICH2 are more sensitive to message size.

Now I analyze the performance of Sendrecv and Exchange - two parallel transfer benchmarks in IMB - running on ES-MPICH2 and MPICH2 on the 6-node cluster. Sendrecv, in which the main purpose is to test the MPI_Sendrecv function, consists of processes forming a periodic communication chain. Similarly, Exchange is a benchmark focusing on the evaluation of the MPI_ISend, MPI_Waitall, and MPI_Recv functions. Unlike the aforementioned single transfer benchmarks, message passing operations in these two parallel benchmarks are performed in parallel.

Fig. 7.14 plots the performance results of the SendRecv benchmark on the cluster, where each node receives data from its left neighbor and then sends data to its right neighbor. The total execution time of the SendRecv benchmark does not noticeably change when I vary the

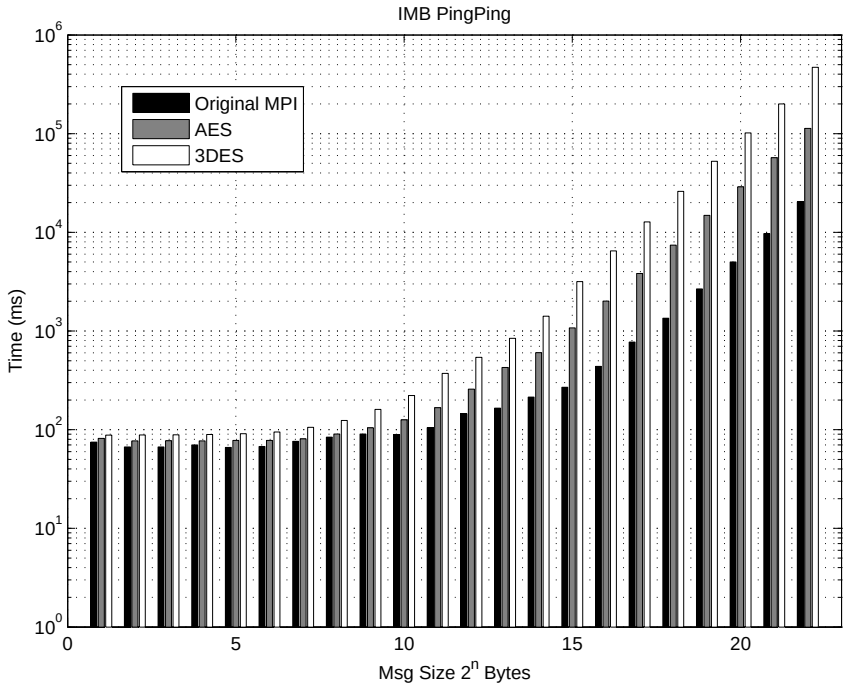


Figure 7.13: Intel MPI Single Transfer Benchmarks PingPing

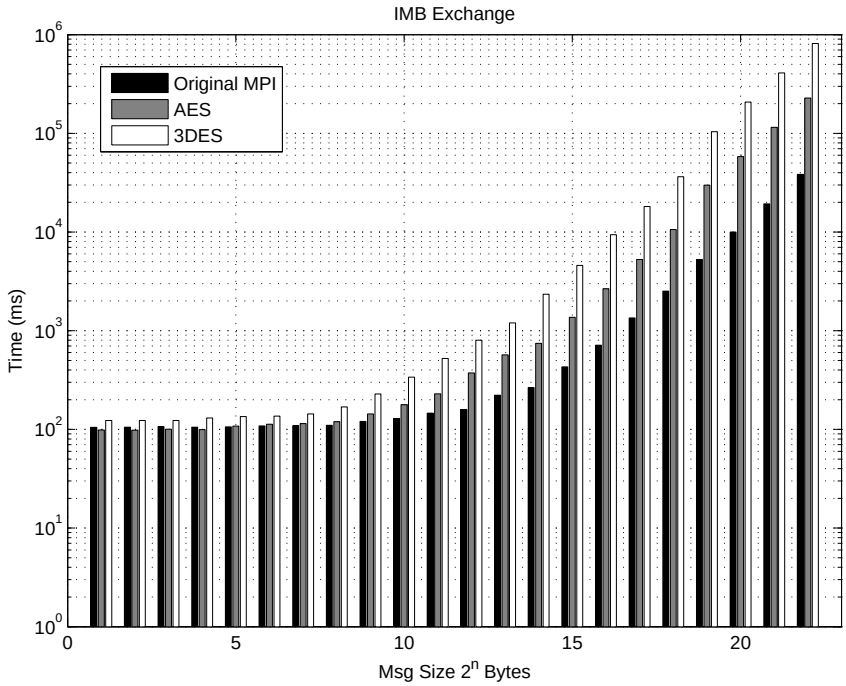


Figure 7.14: Intel MPI Parallel Benchmarks PingPong

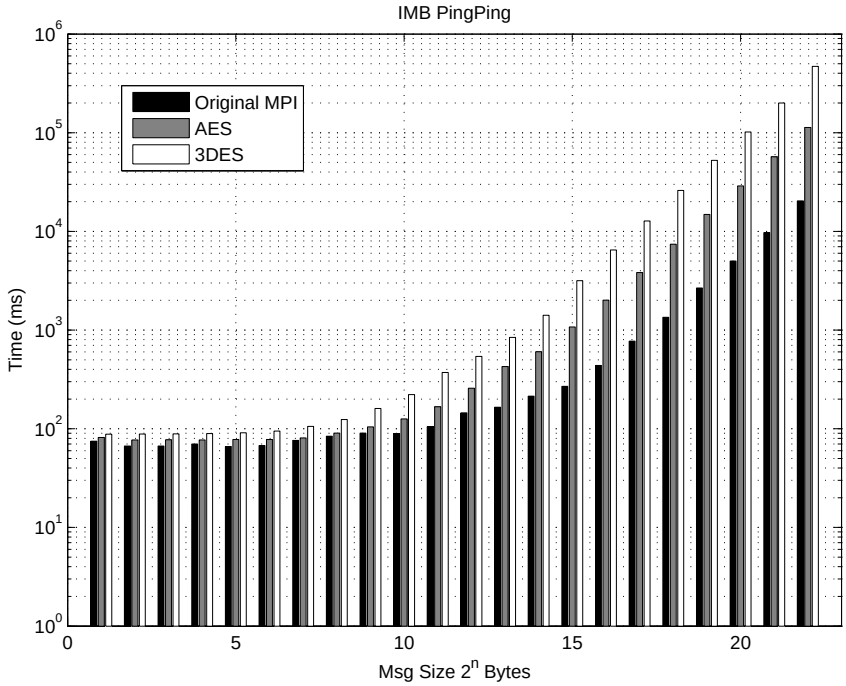


Figure 7.15: Intel MPI Parallel Benchmarks PingPing

number of computing nodes in the cluster. I attribute this trend to the factor that message passing in multiple nodes are carried out in parallel rather than serially. Thus, increasing the number of nodes does not affect SendRecv's total execution time. With respect to parallel transfers, the performance of AES-based and 3DES-based MPICH2 is close to that of the original version of MIPCH2 when message size is relatively small. When it comes to large messages, AES-based ES-MPICH2 has better parallel transfer performance than 3DES-based MPICH2.

Fig. 7.15 depicts the total execution time of the Exchange benchmark. Comparing Fig. 7.15 with Fig. 7.14, I realize that regardless of the MPI implementations, the execution time of the Exchange benchmark is longer than that of the SendRecv benchmark under the condition of same message size. This is mainly because in Exchange each node transfer data to both left and right neighbours in the communication chain. Thus, communication time in Exchange is larger than that in SendRecv. As a result, the total execution time of Exchange is approximately two times higher than that of Sendrecv when message size is large.

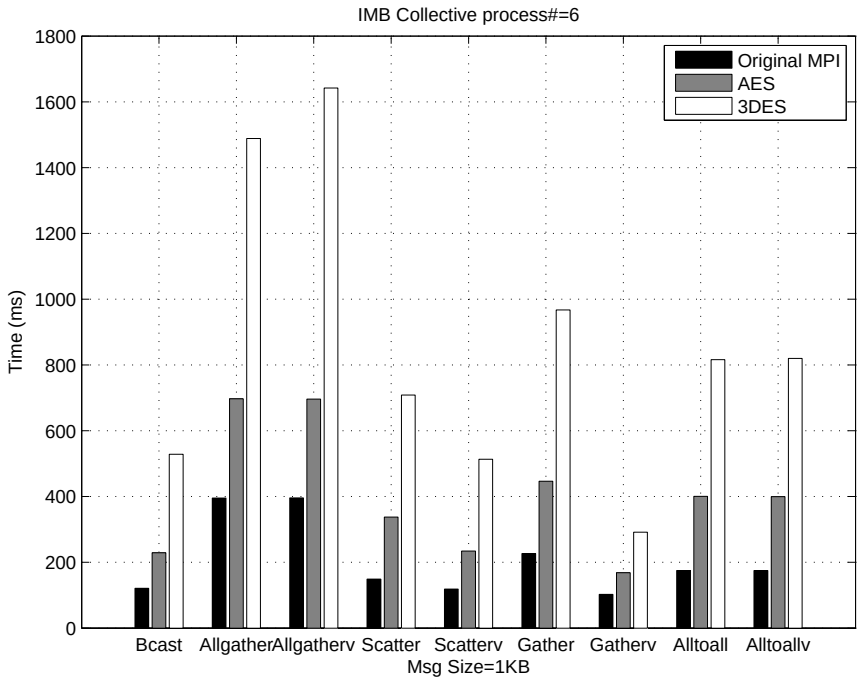


Figure 7.16: Intel MPI Benchmarks Collective Group A, Message Size is 1KB, nodes amount is 6

Let us vary message size and evaluate the performance of collective benchmarks. I run the benchmark 10 times on each MPI implementation and report the average execution times. Figs. 7.16-7.17 show the performance of the first group of nine collective benchmarks. I observe from these figures that the total execution time of each collective benchmark continually increases with increasing message size. MPICH2 has better performance than AES-based and 3DES-based ES-MPICH2 across all the collective benchmarks, because the confidentiality is preserved at the cost of message passing performance. Figs. 7.19-7.20 plot the execution times of the second group of three benchmarks. The performance results of the second benchmark group are consistent with those of the first benchmark group reported in Figs. 7.16-7.17.

Fig. 7.22 shows the results of the Window benchmark, which aims to test MPI-2 functions like MPI_Win_create, MPI_Win_fence, and MPI_Win_free. In this benchmark, a window size message is transferred to each node, which in turn creates a window using a specified

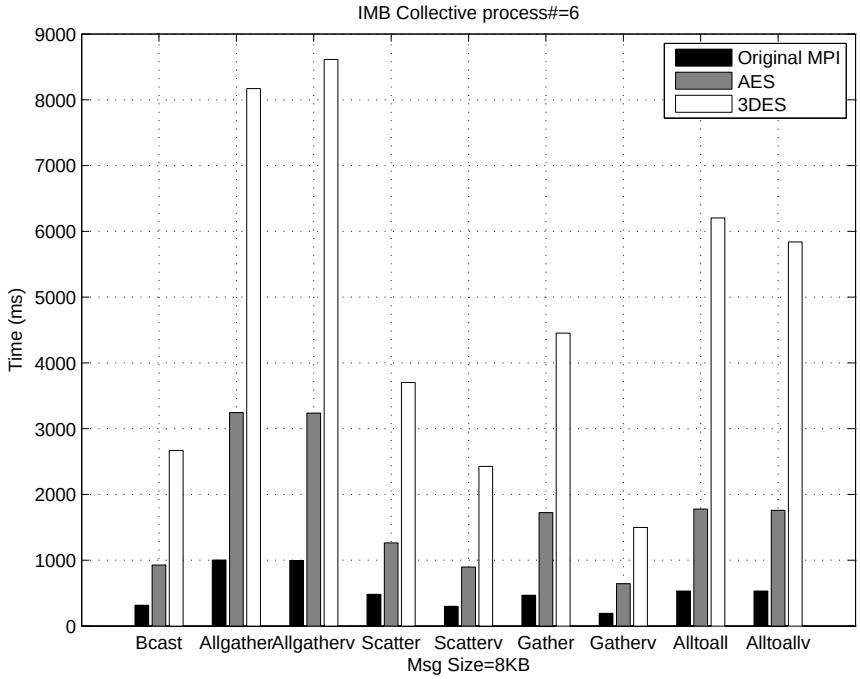


Figure 7.17: Intel MPI Benchmarks Collective Group A, Message Size is 8KB, nodes amount is 6

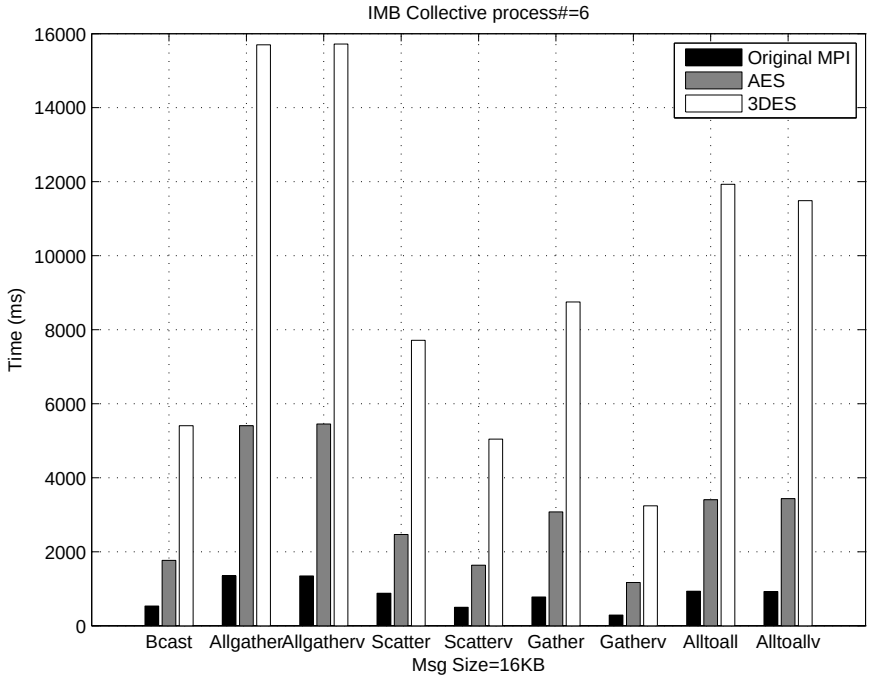


Figure 7.18: Intel MPI Benchmarks Collective Group A, Message Size is 16KB, nodes is 6

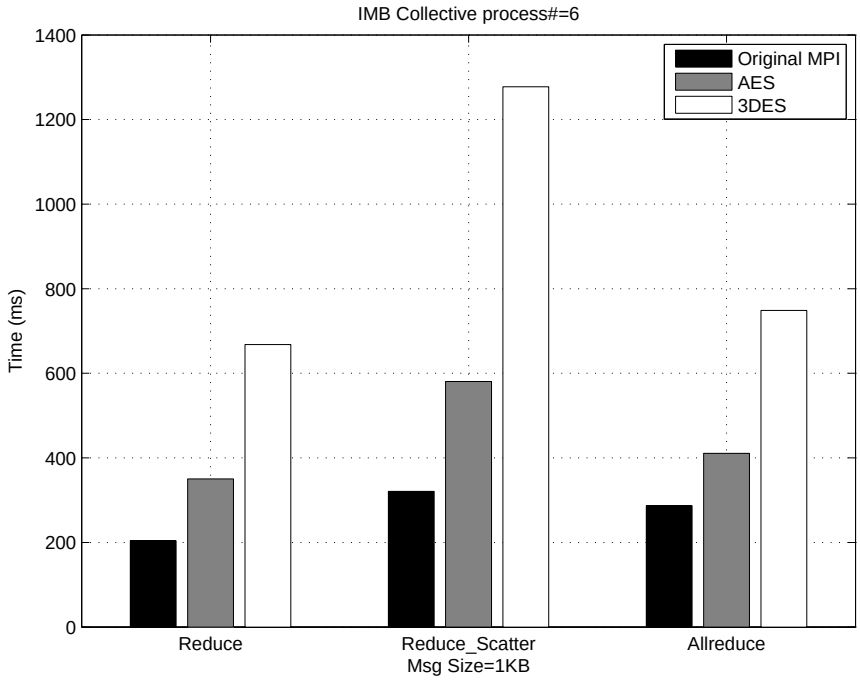


Figure 7.19: Intel MPI Benchmarks Collective Group B, Message Size is 1KB, nodes amount is 6

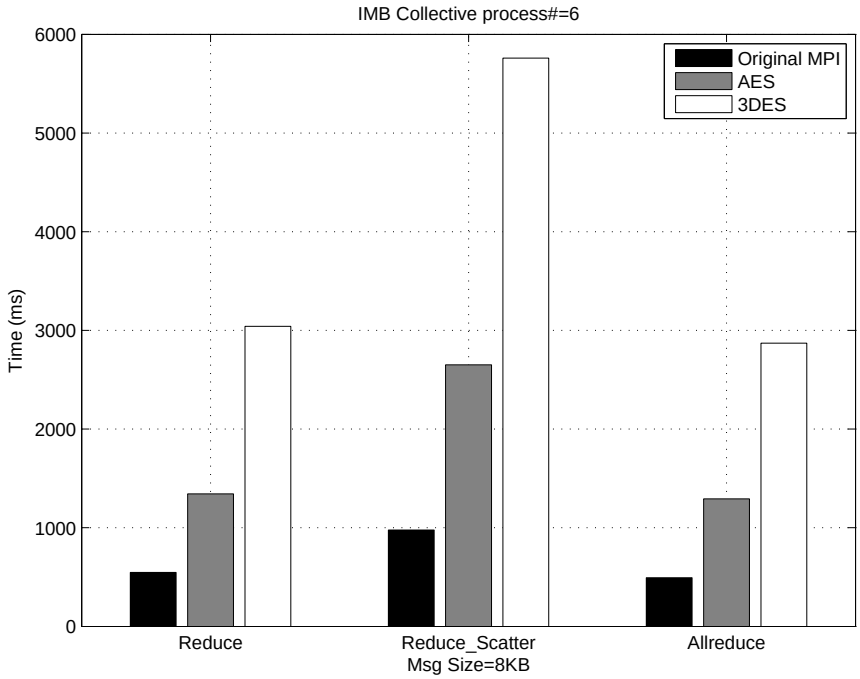


Figure 7.20: Intel MPI Benchmark Collective Group B, Message Size is 8KB, nodes amount is 6

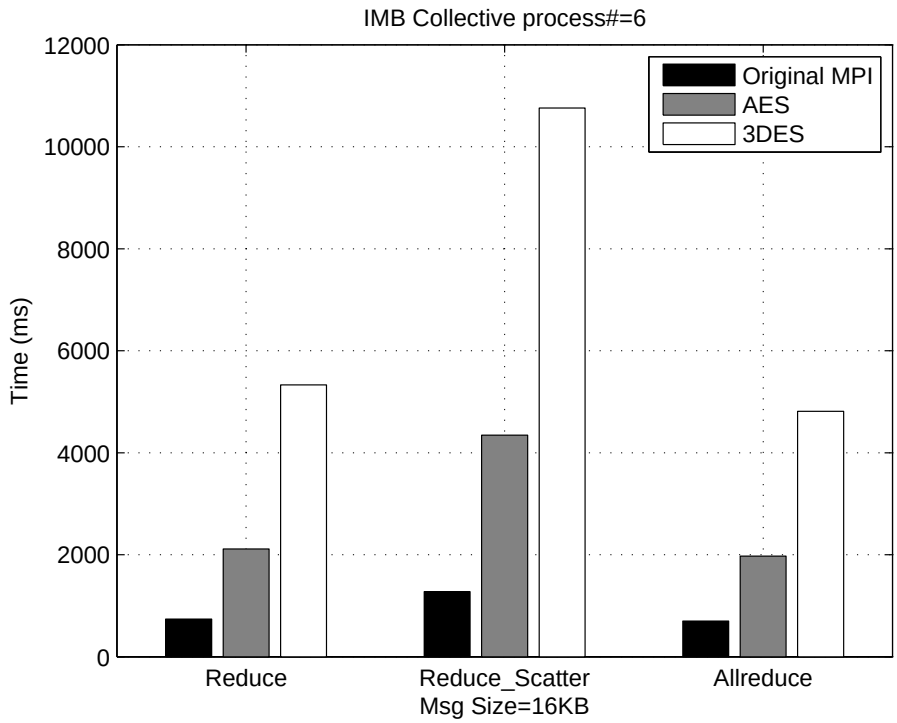


Figure 7.21: Intel MPI Benchmark Collective Group B, Message Size is 16KB, nodes amount is 6

size. Fig. 7.22 indicates that the execution time of the benchmark is not sensitive to message size. The results confirm that AES-based ES-MPICH2 improves the security of the Window benchmark on MPICH2 with marginal overhead.

### 7.6.2 A 10-node Cluster of Intel Pentium II Processors

#### Experimental Testbed

Now I evaluate the performance of MPICH2 and ES-MPICH2 on a 10-node cluster of Intel Pentium II processors. The cluster configuration is summarized in Table 7.4. The operating system running on this cluster is Fedora Core release 4 (Stentz). Although the processors of the nine slave nodes are 333 MHz Intel Pentium II, the master node contains a 3.0 GHz Intel Pentium Core 2 Duo processor, which is almost ten times faster than the processors in the slave nodes. Each slave node has only 64 MB memory, whereas the master node has 1 GB memory. All the ten nodes are connected by a 100 Mbps Ethernet network.

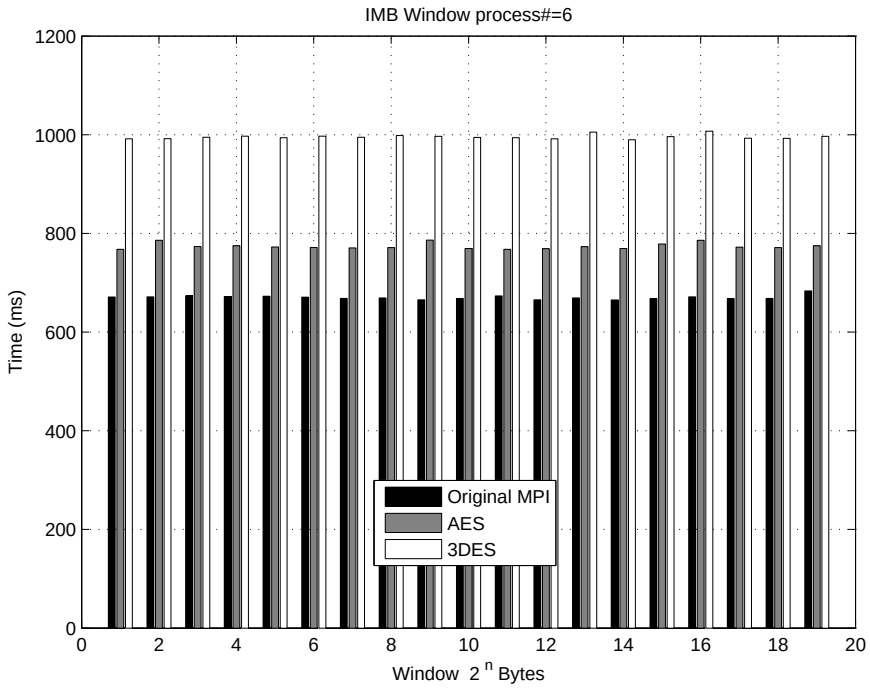


Figure 7.22: Intel Micro Benchmark Window 6 nodes

Like the first cluster, all nodes in the 10-node cluster share disk space on the master node through the network file system (NFS).

Table 7.4: The Configuration of A 10-Node Clusterof Intel Pentium II Processors

| | Master ×1 | Slaves ×9 |
|---|---|---|
| CPU | Pentium Core 2 Duo 3.00GHz | Pentium II 333MHz |
| Memory | 1GB | 64MB |
| OS | Fedora Core release 4 | Fedora Core release 4 |
| Kernel | 2.6.12 − 1.1456 FC4smp | 2.6.17 − 1.2142 FC4 |
| Network Adapter | 100Mbps | 100Mbps |

**SMB: Sandia Micro Benchmark**

Figs. 7.23-7.25 reveal the total execution time, CPU time, overhead, and threshold of MPICH2 and ES-MPICH2 when the message size is set to 1 KB, 16 KB, and 32 KB,

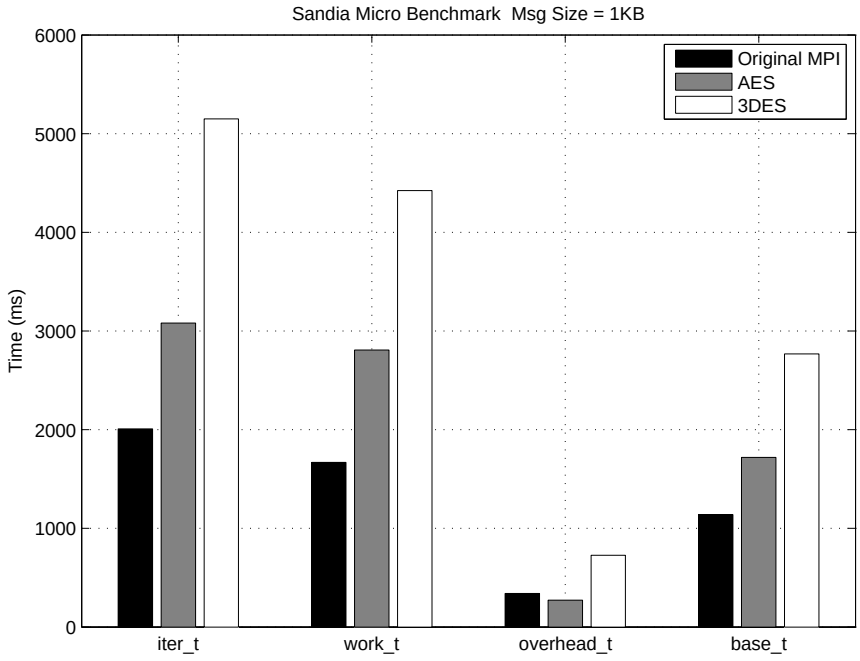


Figure 7.23: Sandia Micro Benchmark time Message Size=1KB

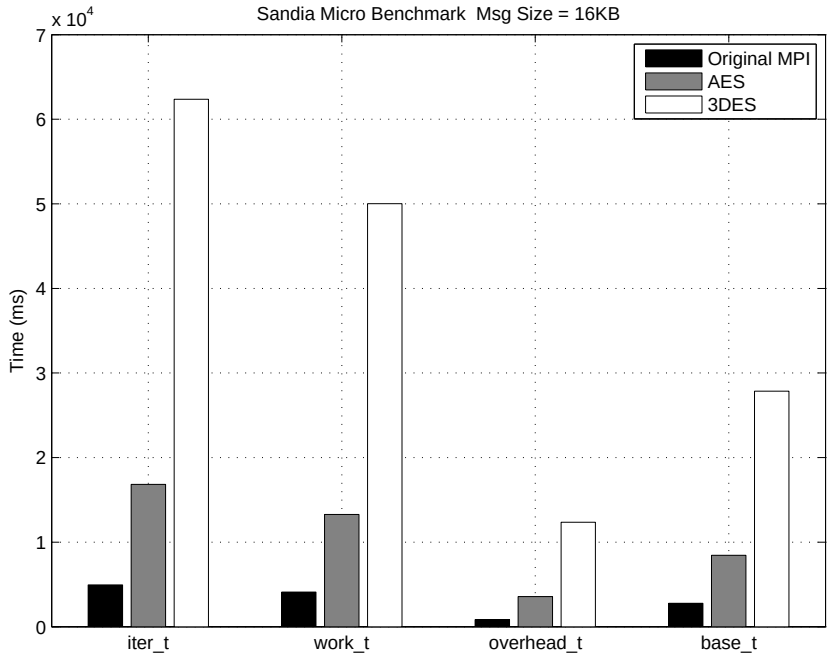


Figure 7.24: Sandia Micro Benchmark time Message Size=16KB

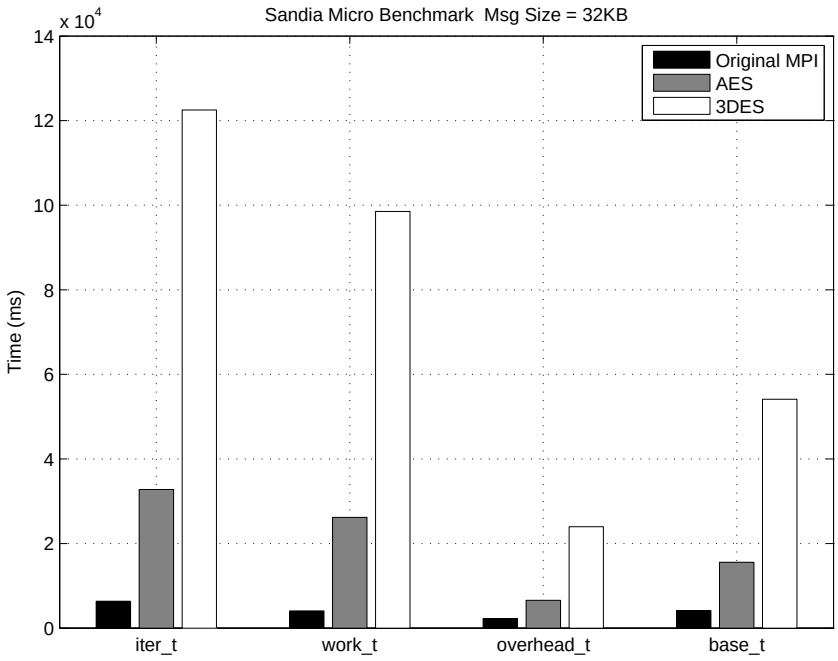


Figure 7.25: Sandia Micro Benchmark time Message Size=32KB

respectively. The results show that the performance of AES-based and 3DES-based ES-MPICH2 is noticeably worse than that of MPICH2, because encryption and decryption modules in ES-MPICH2 introduce security overhead. This trend is true even when messages are small (e.g., 1 KB). Comparing Fig. 7.23 and Fig. 7.8, I observed that the first cluster is significantly faster than the second one. As a result, the performance of AES-based MPICH2 on the first cluster is very close to that of MPICH2 when message size is smaller than 2 KB. Thus, I conclude that improving processor speed in a cluster can substantially reduce the security overhead in ES-MPICH2.

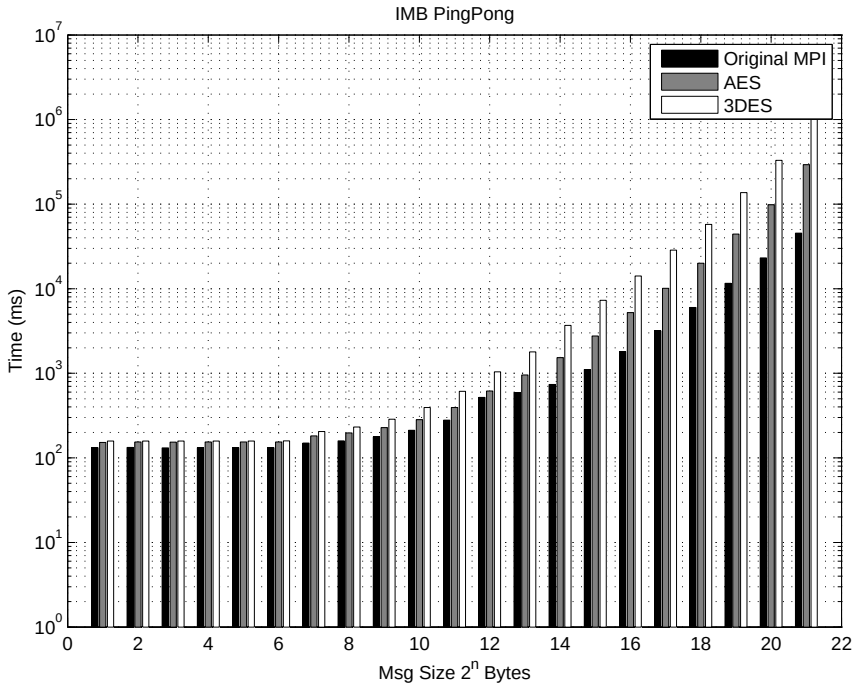


Figure 7.26: Intel MPI Benchmark PingPong

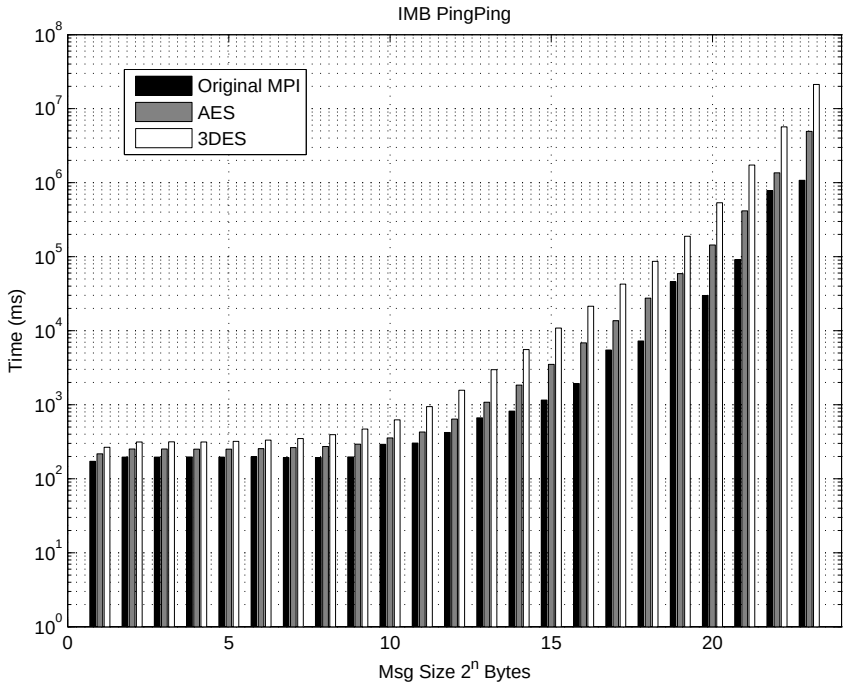


Figure 7.27: Intel MPI Benchmark PingPing

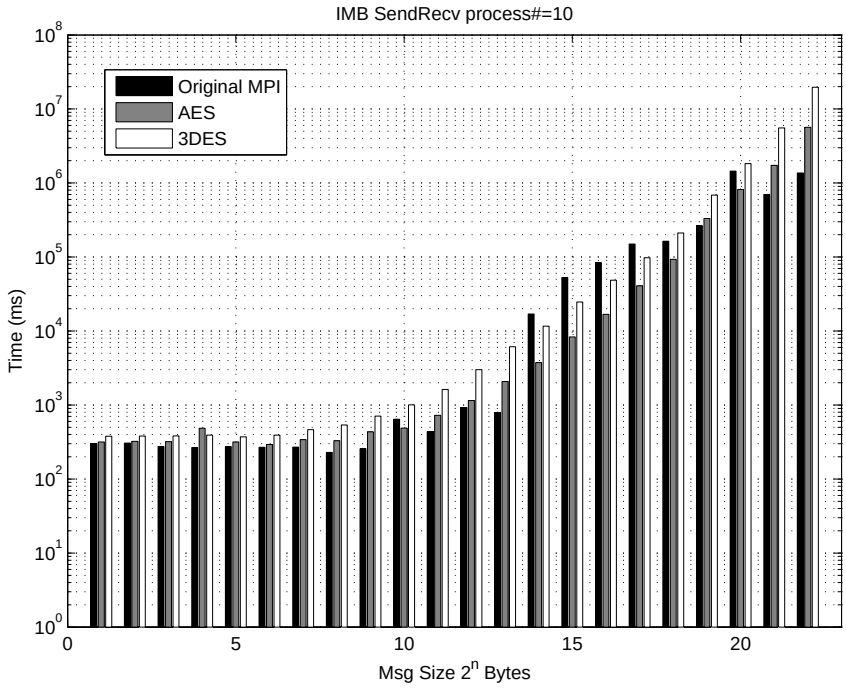


Figure 7.28: Intel MPI Benchmark SendRecv 10 nodes

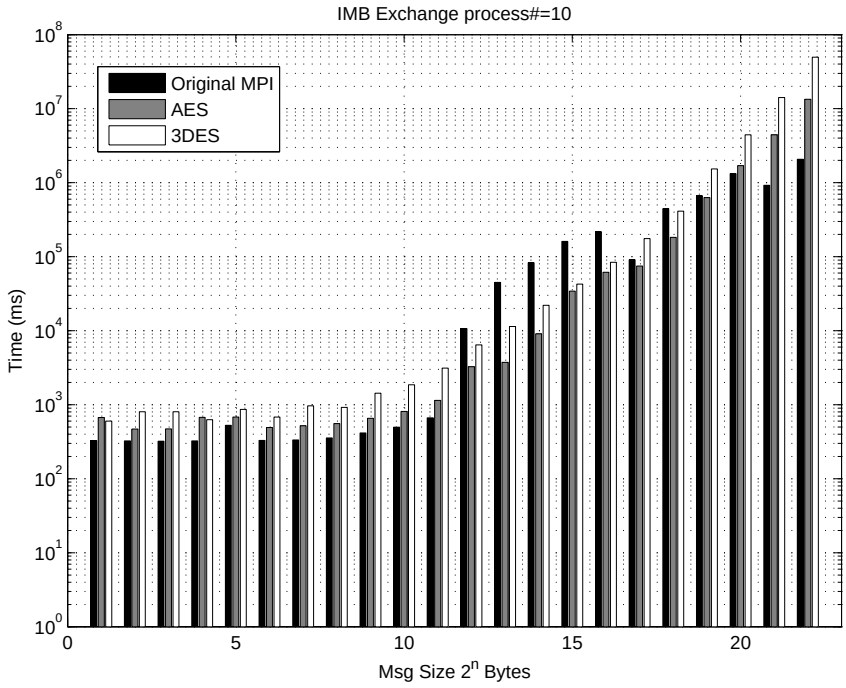


Figure 7.29: Intel MPI Benchmark Exchange 10 nodes

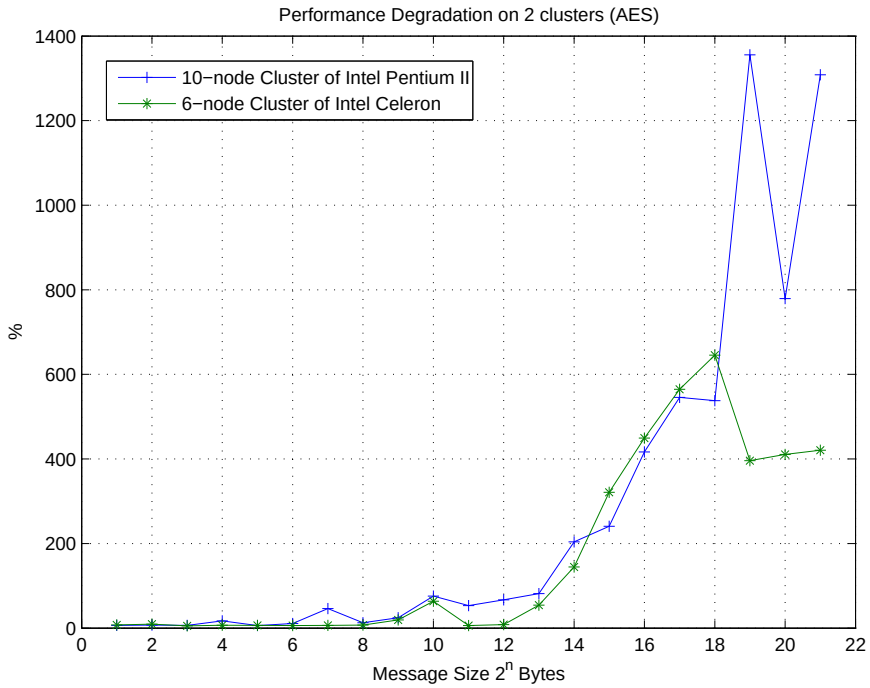


Figure 7.30: Performance Degradation of 2 clusters (AES). Benchmark is SMB

### IMB: Intel MPI Benchmarks

Figs. 7.26-Fig. 7.29 depict the performance of the PingPong, PingPing, SendRecv, and Exchange benchmarks in IMB. The total execution times of the four IMB benchmarks increases with increasing message size. Compared with MPICH2, the execution time of ES-MPICH2 is more sensitive to message size. More importantly, Figs. 7.26-Fig. 7.29 demonstrate that when ES-MPICH2 is deployed on a slow cluster, ES-MPICH2 preserves message confidentiality by substantially degrading the performance of the original MPICH2. By comparing the Intel benchmark performance on both the 6-node cluster (see Figs. 7.12-Fig. 7.15and 10-node clusters (see Figs. 7.26-Fig. 7.29), I observe that the performance gap between MPICH2 and ES-MPICH2 on the fast cluster is much smaller than the performance gap on the slow cluster. An implication of this observation is that security overhead in ES-MPICH2 can be significantly reduced by deploying ES-MPICH2 in a high-end cluster.

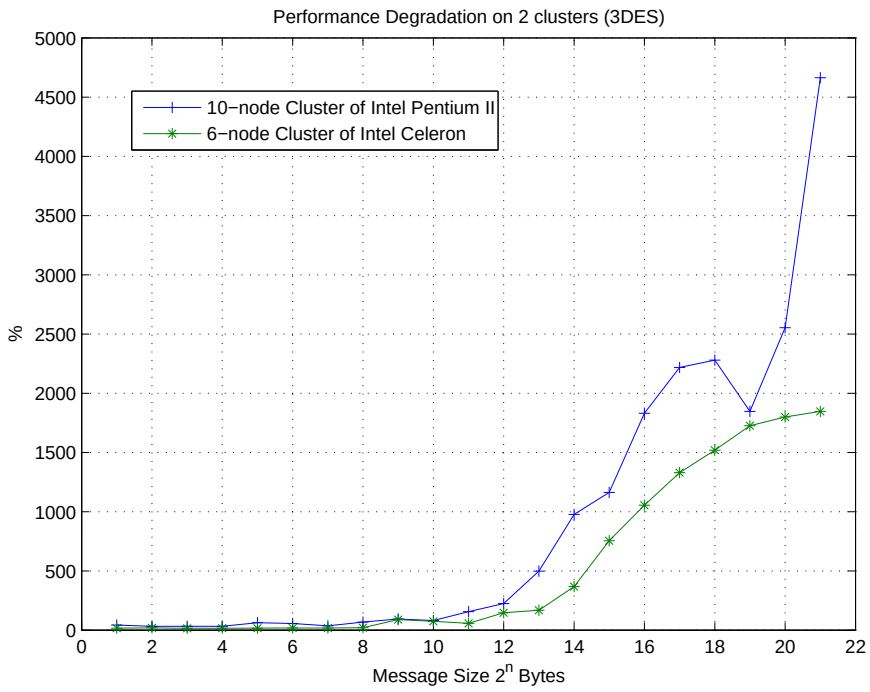


Figure 7.31: Performance Degradation of 2 clusters (3DES). Benchmark is SMB

To intuitively evaluate the impact of cluster computing capacity on ES-MPICH2, I compared the two clusters in terms of the performance degradation due to additional overhead associated with the confidentiality services. The performance degradation is calculated as the performance differences between ES-MPICH2 and MPICH2 divided by the performance of MPICH2.

Figs. 7.30 and 7.31 graphically depict the performance degradation of AES-based ES-MPICH2 and 3DES-based ES-MPICH2 when the SMB benchmark suite is executed on the two Linux clusters. The results plotted in Figs. 7.30 and 7.31 reveal that the performance degradation of ES-MPICH2 is marginal when the message size is small (e.g., $<$ 256 Bytes). The performance degradation becomes more severe with larger messages. Comparing the performance degradation of ES-MPICH2 on two clusters, I confirmed that using a high-end cluster can substantially minimize the performance degradation caused by security overhead in ES-MPICH2. For example, when the message size is larger than 256 KB, the performance

degradation of AES-based ES-MPICH2 is reduced by more than half when the 6-node cluster of Intel Celeron is used instead of the 10-node cluster of Intel Pentium II (see Figs. 7.30).

## 7.7 Summary and Future Work

To address the issue of providing confidentiality services for large-scale clusters connected by an open unsecured network, I aim at improving the security of the message passing interface (MPI) protocol by encrypting and decrypting messages communicated among computing nodes. In this study, I implemented the ES-MPICH2 framework, which is based on MPICH2. ES-MPICH2 is a secure, compatible, and portable implementation of the message passing interface standard. Compared with the original version of MPICH2, ES-MPICH2 preserves message confidentiality in MPI applications by integrating encryption techniques like AES and 3DES into the MPICH2 library.

In light of ES-MPICH2, programmers can easily write secure MPI applications without additional source code for data-confidentiality protection in open public networks. The security feature of ES-MPICH2 is entirely transparent to MPI programmers because encryption and decryption functions are implemented at the channel-level in the MPICH2 library. MPI-application programmers can fully configure any confidentiality services in MPICHI2, because a secured configuration file in ES-MPICH2 offers the programmers flexibility in choosing any cryptographic schemes and keys seamlessly incorporated in ES-MPICH2. Besides the implementation of AES and 3DES in ES-MPICH2, other cryptographic algorithms can be readily integrated in the ES-MPICH2 framework. I used the Sandia Micro Benchmarks and the Intel MPI benchmarks to evaluate and analyze the performance of MPICH2.

Confidentiality services in ES-MPICH2 do introduce additional overhead because of security operations. In the case of small messages, the overhead incurred by the security services is marginal. The security overhead caused by AES and 3DES becomes more pronounced in ES-MPICH2 with larger messages (e.g., the message size is larger than 256 KB).

Our experimental results show that the security overhead in ES-MPICH2 can be significantly reduced by high-performance clusters. For example, the overhead added by AES in ES-MPICH2 is reduced by more than half when the 6-node cluster of Intel Celeron is used instead of the 10-node cluster of Intel Pentium II. In addition to high-end clusters, the following two solutions can be applied to further reduced overhead caused by confidentiality services in ES-MPICH2. First, AES/3DES hardware implementations can lower security overhead in ES-MPICH2. Second, security co-processors can hide the overhead by allowing the encryption and decryption processes to be executed in parallel with the message passing processes.

I am currently investigating varies means of reducing security overhead in ES-MPICH2. For example, I plan to study if multicore processors can substantially lower the overhead of confidentiality services in ES-MPICH2.

Another interesting direction for future work is to consider several strong and efficient cryptographic algorithms like the Elliptic Cureve Cryptography (ECC) in ES-MPICH2. Since ECC is an efficient and fast cryptographic solution, both the performance and the security of ES-MPICH2 are likely to be improved by incorporating ECC.

A third promising direction for further work is to integrate encryption and decryption algorithms in other communication channels like SHMEM and InfiniBand in MPICH2 because an increasing number of commodity clusters are built using standalone and advanced networks such as Infiniband and Myrinet.

# Chapter 8

# Conclusion and Future Work

In this dissertation, I proposed a group of designs, algorithms, and implementations to deal with energy efficiency issues and security issues in cluster computing systems. This chapter concludes the dissertation by summarizing the contributions and describing future directions. The chapter is organized as follows: Section 8.1 highlights the main contributions of the dissertation. In section 8.2, we concentrate on some future directions, which are extensions of our past and current research on green computing for high-performance computing platforms.

## 8.1 Main Contributions

Cluster computing is a large research area including high-performance computing, real-time scheduling, parallel processing, high-performance network, large scale storage systems, and data confidentiality. Electricity bills are getting hugh because clustered computers cost a significant amount of energy by both processors and disks. Data confidentiality is another big issue in data centers. In a security-aware storage system, performance is lower due to the cryptographic algorithms cost. To make it easier to secure data in a cluster computing system, we developed Enhanced-Security MPICH2 based on MPICH2 developed by Argonne National Laboratory.

### 8.1.1 Time-Aware Dynamic Voltage Scaling

In this chapter, I proposed a time-aware dynamic voltage scaling algorithm to conserve energy consumption for parallel applications in parallel computing systems. It makes use of the dynamic voltage scaling technique (DVS) to provide significant energy savings for

both high-performance and mobile clusters. The novel feature of TADVS is that it employs DVS only to parallel tasks that are followed by idle processor times. Experimental results show that TADVS is capable of conserving energy without adversely affecting performance of high-performance clusters and mobile clusters.

### 8.1.2 Energy-Efficient Storage Systems

In this chapter, I first presented the design of parallel I/O systems with buffer disks. To conserve energy in parallel I/O systems serving write requests, I developed an algorithm - dynamic request allocation algorithm for writes or DARAW - to energy efficiently allocate and schedule disk requests. This goal is achieved by making use of buffer disks in parallel I/O systems to accumulate small writes to form a log, which can be transferred to data disks in a batch way. DARAW is able to improve parallel I/O energy efficiency by the virtue of employing a small number of buffer disks to serve a majority of write requests, thereby keeping a large number of data disks in low-power state for longer period times. In this research, I focused on parallel I/O systems with homogeneous disks.

### 8.1.3 Energy-Efficient Cluster Storage Systems

In this chapter, I designed and implemented an energy-efficient cluster storage system called ECOS. Each I/O node in ECOS controls multiple disks - one buffer disk and several data disks. The key idea behind ECOS is to redirect disk requests from data disks to the buffer disks. To improve I/O performance of buffer disks, ECOS attempts to balance I/O load among all I/O nodes in the cluster storage system. The ECOS system was implemented in a Linux cluster, where each I/O node contains one buffer disk and two data disks. Results show that ECOS improves energy efficiency of traditional cluster storage systems without using buffer disks.

### 8.1.4 Security-Aware Storage Systems

Achieving both high energy efficiency and security in disk systems is challenging, because energy efficiency and data security are often two conflicting goals. In this chapter, I took the first step toward the second approach by answering an intriguing question of whether I can improve energy efficiency of security mechanisms in disk systems without changing the source code of security services.

### 8.1.5 Enhanced-Security MPICH2

To address the issue of providing confidentiality services for large-scale clusters connected by an open unsecured network, I aim at improving the security of the message passing interface (MPI) protocol by encrypting and decrypting messages communicated among computing nodes. In this chapter, I implemented the ES-MPICH2 framework, which is based on MPICH2. ES-MPICH2 is a secure, compatible, and portable implementation of the message passing interface standard. Compared with the original version of MPICH2, ES-MPICH2 preserves message confidentiality in MPI applications by integrating encryption techniques like AES and 3DES into the MPICH2 library.

## 8.2 Future Work

we have found several interesting issues which have not been well dressed. This section overviews some of these open issues that need further investigation. These open issues present opportunities for my future research.

### 8.2.1 Solid State Drives - Internal Parallelism and Reliability

Most research of Solid State Drives architectures rely on Flash Translation Layer (FTL) algorithms and wear-leveling. However, internal parallelism in Solid State Drives has not been well explored.

Reliability is a critical issue for all flash based storage. In this study, we showed that could update data in buffer. Updating data in the write buffer of an SSD can reduce the number of erasures, thereby improving the reliability of the SSD. We plan to quantitatively evaluate the reliability impact of our write buffers on SSDs

### 8.2.2 Hybrid Storage Systems

The storage system combined with different storage media is a future direction. The collaboration of different storage devices will decide performance. Hybrid storage systems could reduce cost, improve performance and reliability. We plan to explore the research on hybrid storage systems

Bibliography

[1] Striping and buffer caching for software raid file systems in workstation clusters. In *ICDCS '99: Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, page 544, Washington, DC, USA, 1999. IEEE Computer Society.

[2] Specification for the advanced encryption standard (aes). Federal Information Processing Standards Publication 197, 2001.

[3] *Secure hash standard*. National Institute of Standards and Technology, Washington, 2002. Note: Federal Information Processing Standard 180-1.

[4] Where does power go. `http://www.greendataproject.org/index.php?option=com_content&task=view&id=44&Itemid=60`, 2007.

[5] Power consumption of supercomputers. `http://www.top500.org/lists/2008/06/highlights/power`, June 2008.

[6] National Security Agency. National policy on the use of the advanced encryption standard (aes) to protect national security systems and national security information cnss policy no. 15 fact sheet no. 1, June 2003.

[7] Tarek A. AlEnawy and Hakan Aydin. On energy-constrained real-time scheduling. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 165–174, Washington, DC, USA, 2004. IEEE Computer Society.

[8] Hakan Aydi, Pedro Mejía-Alvarez, Daniel Mossé, and Rami Melhem. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, RTSS '01, pages 95–, Washington, DC, USA, 2001. IEEE Computer Society.

[9] Sung Hoon Baek and Kyu Ho Park. Matrix-stripe-cache-based contiguity transform for fragmented writes in raid-5. *IEEE Trans. Comput.*, 56(8):1040–1054, 2007.

[10] S. Bansal, P. Kumar, and K. Singh. An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems. *Parallel and Distributed Systems, IEEE Transactions on*, 14(6):533 – 544, 2003.

[11] Rakesh Barve, Mahesh Kallahalla, Peter J. Varman, and Jeffrey Scott Vitter. Competitive parallel disk prefetching and buffer management. In *IOPADS '97: Proceedings of the fifth workshop on I/O in parallel and distributed systems*, pages 47–56, New York, NY, USA, 1997. ACM.

[12] Ian F. Blake, G. Seroussi, and N. P. Smart. *Elliptic curves in cryptography*. Cambridge University Press, New York, NY, USA, 1999.

[13] P. Bohrer, E. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony. *The Case for Power Management in Web Servers*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[14] Ron Brightwell, David S. Greenberg, Brian J. Matt, and George I. Davida. Barriers to creating a secure mpi, 1997.

[15] T. Burd, T. Pering, A. Stratakos, and R. Brodersen. A dynamic voltage scaled microprocessor system. In *Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC. 2000 IEEE International*, 2000.

[16] Juan-Carlos Cano, Dongkyun Kim, and Pietro Manzoni. Cera: Cluster-based energy saving algorithm to coordinate routing in short-range wireless networks. In *ICOIN*, pages 306–315, 2003.

[17] R. Chandramouli, S. Bapatla, K. P. Subbalakshmi, and R. N. Uma. Battery power-aware encryption. *ACM Trans. Inf. Syst. Secur.*, 9:162–180, May 2006.

[18] J. Chase, D. Anderson, P. Thackar, A. Vahdat, and R. Boyle. Managing energy and server resources in hosting centers. *Proc. 18th Symp. on Operating Systems Principles*, October 2001.

[19] Chunhong Chen and Majid Sarrafzadeh. Provably good algorithm for low power consumption with dual supply voltages. In *ICCAD99: IEEE/ACM International Conference on Computer-Aided Design*, pages 76–79, 1999.

[20] John Chung-I Chuang and Marvin A. Sirbu. Distributed network storage service with quality-of-service guarantees. *Journal of Network and Computer Applications*, 23(3):163 – 185, 2000.

[21] D. Colarelli and D. Grunwald. Massive arrays of idle disks for storage archives. *Proc. ACM/IEEE Conf. on Supercomputing*, pages 1–11, 2002.

[22] D. Coppersmith, D. B. Johnson, S. M. Matyas, T. J. Watson, Don B. Johnson, and Stephen M. Matyas. Triple des cipher block chaining with output feedback masking, 1996.

[23] Intel Corporation. Intel mpi benchmarks user guide and methodology description, 2008.

[24] Joan Daemen and Vincent Rijmen. The design of rijndael, 2002.

[25] Dorothy E. Denning. Secure personal computing in an insecure network. *Commun. ACM*, 22(8):476–482, 1979.

[26] Jack J. Dongarra, Steve W. Otto, Marc Snir, and David Walker. An introduction to the mpi standard. Technical report, Knoxville, TN, USA, 1995.

[27] Fred Douglis, P. Krishnan, and Brian Marsh. Thwarting the power-hungry disk. In *WTEC'94: Proceedings of the USENIX Winter 1994 Technical Conference on USENIX Winter 1994 Technical Conference*, pages 23–23, Berkeley, CA, USA, 1994. USENIX Association.

[28] E. Pinheiro E. V. Carrera and R. Bianchini. Conserving disk energy in network servers. *Proc. Int'l Conf. on Supercomputing*, 2003.

[29] W. Ehrsam, S. Matyas, C. Meyer, and W. Tuchman. A cryptographic key management scheme for implementing the data encryption standard. *IBM Systems Journal*, 17(2):106–125, 1978.

[30] Adam J. Elbirt, W. Yip, B. Chetwynd, and C. Paar. An fpga-based performance evaluation of the aes block cipher candidate algorithm finalists. *IEEE Trans. Very Large Scale Integr. Syst.*, 9(4):545–557, 2001.

[31] E. N. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. *Proc. 2nd Workshop on Power-Aware Computing Systems*, February 2002.

[32] M. Elnozahy, M. Kistler, and R. Rajamony. Energy conservation policies for web servers. *Proc. 4th USENIX Symp. on Internet Technologies and Systems*, March 2003.

[33] M. J. Field, Paul A. Bash, and Martin Karplus. A combined quantum mechanical and molecular mechanical potential for molecular dynamics simulations. *J. Comput. Chem.*, 11:700–733, May 1990.

[34] B. Forney, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. Storage-aware caching: Revisiting caching for heterogeneous storage systems, 2002.

[35] V.W. Freeh, Feng Pan, N. Kappiah, D.K. Lowenthal, and R. Springer. Exploring the energy-time tradeoff in mpi programs on a power-scalable cluster. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, page 4a, 2005.

[36] R. Ge, Xizhou Feng, and K.W. Cameron. Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, page 34, 2005.

[37] Al Geist, William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing L. Lusk, William Saphir, Tony Skjellum, and Marc Snir. Mpi-2: Extending the message-passing interface. In *Euro-Par '96: Proceedings of the Second International Euro-Par Conference on Parallel Processing*, pages 128–135, London, UK, 1996. Springer-Verlag.

[38] Binny S. Gill and Dharmendra S. Modha. Wow: wise ordering for writes - combining spatial and temporal locality in non-volatile caches. In *FAST'05: Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies*, pages 10–10, Berkeley, CA, USA, 2005. USENIX Association.

[39] Pawan Goyal, Divyesh Jadav, Dharmendra S. Modha, and Renu Tewari. Cachecow: Qos for storage system caches. In *Eleventh International Workshop on Quality of Service (IWQoS 03), Monterey, CA*, 2003.

[40] R. Grabner, F. Mietke, and W. Rehm. Implementing an mpich-2 channel device over vapi on infiniband. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pages 184–, April 2004.

[41] Ren Grabner, Frank Mietke, and Wolfgang Rehm. Implementing an mpich-2 channel device over vapi on infiniband. *Parallel and Distributed Processing Symposium, International*, 9:184a, 2004.

[42] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel Comput.*, 22(6):789–828, 1996.

[43] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. Drpm: dynamic speed control for power management in server class disks. pages 169–179, June 2003.

[44] J. M. Haile. *Molecular dynamics simulation: elementary methods*. Wiley-Interscience, New York, NY, 1992.

[45] P. Hamalainen, M. Hannikainen, T. Hamalainen, and J. Saarinen. Configurable hardware implementation of triple-des encryption algorithm for wireless local area network. In *ICASSP '01: Proceedings of the Acoustics, Speech, and Signal Processing, 200. on IEEE International Conference*, pages 1221–1224, Washington, DC, USA, 2001. IEEE Computer Society.

[46] David P. Helmbold, Darrell D. E. Long, Tracey L. Sconyers, and Bruce Sherrod. Adaptive disk spin—down for mobile computers. *Mob. Netw. Appl.*, 5(4):285–297, 2000.

[47] Dean Hildebrand, Lee Ward, and Peter Honeyman. Large files, small writes, and pnfs. In *ICS '06: Proceedings of the 20th annual international conference on Supercomputing*, pages 116–124, New York, NY, USA, 2006. ACM Press.

[48] Inki Hong, Gang Qu, M. Potkonjak, and M.B. Srivastavas. Synthesis techniques for low-power hard real-time systems on variable voltage processors. In *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, pages 178 –187, December 1998.

[49] L. Hong, Z. Liang, A. Viswambharant, A. Kaufman, and M. Wax. Reconstruction and visualization of 3d models of colonic surface. *Nuclear Science, IEEE Transactions on*, 44(3):1297 –1302, June 1997.

[50] Y. Hotta, M. Sato, H. Kimura, S. Matsuoka, T. Boku, and D. Takahashi. Profile-based optimization of power performance by using dynamic voltage scaling on a pc cluster. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 8 pp., 2006.

[51] J. W. Hsieh, T. W. Kuo, P. L. Wu, and Y. C. Huang. Energy-efficient and performance-enhanced disk using flash-memory cache. *Proc. Int'l Symp. on Low Power Electronics and Design*, pages 334–339, 2007.

[52] Yennun Huang. Developing reliable applications on cluster systems. In *Proceedings of the 15th Symposium on Reliable Distributed Systems*, SRDS '96, pages 165–, Washington, DC, USA, 1996. IEEE Computer Society.

[53] AMD Inc. *AMD PowerNow Technology.* 2000.

[54] Intel Inc. *The Intel XScale Microarchitecture Technical Summary.* 2000.

[55] S. Jin and A. Bestavros. Gismo: A generator of internet streaming media objects and workloads. *ACM SIGMETRICS Performance Evaluation Review*, November 2001.

[56] N. Joukov and J. Sipek. Greenfs: Making enterprise computers greener by protecting them better. *Proc. ACM SIGOPS Operating Systems Review*, pages 69–80, 2008.

[57] Mahesh Kallahalla and Peter J. Varman. Improving parallel-disk buffer management using randomized writeback. In *Proc. Int'l Conf. Parallel Processing*, pages 270–277, 1998.

[58] E.J. Kim, G.M. Link, K.H. Yum, N. Vijaykrishnan, M. Kandemir, M.J. Irwin, and C.R. Das. A holistic approach to designing energy-efficient cluster interconnects. *Computers, IEEE Transactions on*, 54(6):660 – 671, June 2005.

[59] Gregory A. Koenig, Xin Meng, Adam J. Lee, Michael Treaster, Nadir Kiyanclar, and William Yurcik. Cluster security with nvisioncc: Process monitoring by leveraging emergent properties. In *In IEEE Cluster Computing and Grid (CCGrid*, 2005.

[60] P Krishnan, M P Long, and Scott J Vitter. Adaptive disk spindown via optimal rent-to-buy in probabilistic environments. Technical report, Durham, NC, USA, 1995.

[61] Yu-Kwong Kwok and I. Ahmad. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. *Parallel and Distributed Systems, IEEE Transactions on*, 7(5):506 –521, May 1996.

[62] Subramanian Lakshmanan, Mustaque Ahamad, and H. Venkateswaran. Responsive security for stored data. *IEEE Transactions on Parallel and Distributed Systems*, 14(9):818–828, 2003.

[63] Manhee Lee and Eun Jung Kim. A comprehensive framework for enhancing security in infiniband architecture. *IEEE Trans. Parallel Distrib. Syst.*, 18(10):1393–1406, 2007.

[64] SangKeun Lee and Chong-Sun Hwang. Efficient, energy conserving transaction processing in wireless data broadcast. *IEEE Trans. on Knowl. and Data Eng.*, 18(9):1225–1238, 2006. Member-Kitsuregawa,, Masaru.

[65] Yann-Hang Lee, Yoonmee Doh, and C. M. Krishna. Edf scheduling using two-mode voltage-clock-scaling for hard real-time systems. In *Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems*, CASES '01, pages 221–228, New York, NY, USA, 2001. ACM.

[66] Kester Li, Roger Kumpf, Paul Horton, and Thomas Anderson. A quantitative analysis of disk drive power management in portable computers. In *WTEC'94: Proceedings of the USENIX Winter 1994 Technical Conference on USENIX Winter 1994 Technical Conference*, pages 22–22, Berkeley, CA, USA, 1994. USENIX Association.

[67] Chun-Hsien Liu, Chia-Feng Li, Kuan-Chou Lai, and Chao-Chin Wu. A dynamic critical path duplication task scheduling algorithm for distributed heterogeneous computing systems. In *Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on*, 0 2006.

[68] Cong Liu, Xiao Qin, S. Kulkarni, Chengjun Wang, Shuang Li, A. Manzanares, and S. Baskiyar. Distributed energy-efficient scheduling for data-intensive applications with deadline constraints on data grids. In *Performance, Computing and Communications Conference, 2008. IPCCC 2008. IEEE International*, pages 26 –33, 2008.

[69] J. Liu, W. Jiang, P. Wyckoff, D.K. Panda, D. Ashton, D. Buntinas, W. Gropp, and B. Toonen. Design and implementation of mpich2 over infiniband with rdma support. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pages 16–, April 2004.

[70] Lanyue Lu, P. Varman, and Jun Wang. Diskgroup: Energy efficient disk layout for raid1 systems. In *Networking, Architecture, and Storage, 2007. NAS 2007. International Conference on*, pages 233 –242, 2007.

[71] M. I. Lutwyche, M. Despont, U. Drechsler, U. Dürig, W. Häberle, H. Rothuizen, R. Stutz, R. Widmer, G. K. Binnig, and P. Vettiger. Highly parallel data storage system based on scanning probe arrays. *Applied Physics Letters*, 77:3299–+, November 2000.

[72] Stefan Mangard, Manfred Aigner, and Sandra Dominikus. A highly regular and scalable aes hardware architecture. *IEEE Trans. Comput.*, 52(4):483–491, 2003.

[73] A. Manzanares, K. Bellam, and Xiao Qin. A prefetching scheme for energy conservation in parallel disk systems. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1 –5, 2008.

[74] Adam Manzanares, Xiaojun Ruan, Shu Yin, and Mais Nijim. Energy-aware prefetching for parallel disk systems: Algorithms, models, and evaluation. *IEEE Int'l Symp. on Network Computing and Applications*, 2009.

[75] Bo Mao, Dan Feng, Hong Jiang, Suzhen Wu, Jianxi Chen, and Lingfang Zeng. Graid: A green raid storage architecture with improved energy efficiency and reliability. In *Modeling, Analysis and Simulation of Computers and Telecommunication Systems, 2008. MASCOTS 2008. IEEE International Symposium on*, pages 1 –8, 2008.

[76] S. Matyas and C. Meyer. Generation, distribution, and installation of cryptographic keys. *IBM Systems Journal*, 17(2):126–137, 1978.

[77] Inc. Micron Technology. Wearing-leveling techniques in nand flash devices. *Micron Technology, Inc. Specification*, 2008.

[78] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. Write off-loading: Practical power management for enterprise storage. *Trans. Storage*, 4:10:1–10:23, November 2008.

[79] National Institute of Standards and Technology. *FIPS PUB 46-3: Data Encryption Standard (DES)*. October 1999. supersedes FIPS 46-2.

[80] M. Nijim, A. Manzanares, and Xiao Qin. An adaptive energy-conserving strategy for parallel disk systems. In *Distributed Simulation and Real-Time Applications, 2008. DS-RT 2008. 12th IEEE/ACM International Symposium on*, pages 75 –82, 2008.

[81] Mais Nijim, Xiao Qin, and Tao Xie. Modeling and improving security of a local disk system for write-intensive workloads. *Trans. Storage*, 2(4):400–423, 2006.

[82] S. Pande, D.P. Agrawal, and J. Mauney. A scalable scheduling scheme for functional parallelism on distributed memory multiprocessor systems. *Parallel and Distributed Systems, IEEE Transactions on*, 6(4):388 –399, April 1995.

[83] C. Parikh and P. Patel. Performance evaluation of aes algorithm on various development platforms. In *Consumer Electronics, 2007. ISCE 2007. IEEE International Symposium on*, pages 1 –6, 2007.

[84] Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP '01, pages 89–102, New York, NY, USA, 2001. ACM.

[85] E. Pinheiro and R. Bianchini. Energy conservation techniques for disk array-based servers. *Int'l Conf. on Supercomputing*, pages 68–78, 2004.

[86] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. *Proc. Workshop on Compilers and Operating Systems for Low Power*, September 2001.

[87] Eduardo Pinheiro, Ricardo Bianchini, and Cezary Dubnicki. Exploiting redundancy to conserve energy in storage systems. *SIGMETRICS Perform. Eval. Rev.*, 34(1):15–26, 2006.

[88] Nachiketh R. Potlapally, Srivaths Ravi, Anand Raghunathan, and Niraj K. Jha. A study of the energy consumption characteristics of cryptographic algorithms and security protocols. *IEEE Transactions on Mobile Computing*, 5:128–143, February 2006.

[89] M. Pourzandi, D. Gordon, W. Yurcik, and G.A. Koenig. Clusters and security: distributed security for distributed systems. In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, volume 1, pages 96–104 Vol. 1, May 2005.

[90] Ramya Prabhakar, Christina Patrick, and Mahmut Kandemir. Mpisec i/o: Providing data confidentiality in mpi-i/o. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:388–395, 2009.

[91] Xiao Qin. Performance comparisons of load balancing algorithms for i/o-intensive workloads on clusters. *J. Netw. Comput. Appl.*, 31(1):32–46, 2008.

[92] Xiao Qin and Hong Jiang. A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters. *J. Parallel Distrib. Comput.*, 65:885–900, August 2005.

[93] V. Ramsurrun and K.M.S. Soyjaudah. A highly available transparent linux cluster security model. In *Performance, Computing and Communications Conference, 2008. IPCCC 2008. IEEE International*, pages 69–76, Dec. 2008.

[94] S. Ranaweera and D.P. Agrawal. A task duplication based scheduling algorithm for heterogeneous systems. In *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*, 2000.

[95] Raju Rangaswami, Zoran Dimitrijevic, Edward Chang, and Klaus E. Schauser. Mems-based disk buffer for streaming media servers. *Data Engineering, International Conference on*, 0:619, 2003.

[96] R. Rivest. *The MD5 Message-Digest Algorithm*. RFC Editor, United States, 1992.

[97] X. J. Ruan, A. Manzanares, K. Bellam, Z. L. Zong, and X. Qin. Daraw: A new write buffer to improve parallel i/o energy-efficiency. *Proc. ACM Symp. on Applied Computing*, 2009.

[98] Xiaojun Ruan, Adam Manzanares, Kiranmai Bellam, Xiao Qin, and Ziliang Zong. Daraw: a new write buffer to improve parallel i/o energy-efficiency. In *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, pages 299–304, New York, NY, USA, 2009. ACM.

[99] Subhash Saini, Robert Ciotti, Brian T. N. Gunney, Thomas E. Spelce, Alice Koniges, Don Dossa, Panagiotis Adamidis, Rolf Rabenseifner, Sunil R. Tiyyagura, and Matthias Mueller. Performance evaluation of supercomputers using hpcc and imb benchmarks. *J. Comput. Syst. Sci.*, 74(6):965–982, 2008.

[100] D. A. Schecter, D. H. E. Dubin, K. S. Fine, and C. F. Driscoll. Vortex crystals from 2d euler flow: Experiment and simulation. *Physics of Fluids*, 11:905–914, April 1999.

[101] M. Schmitz, B. Al-Hashimi, and P. Eles. Energy-efficient mapping and scheduling for dvs enabled distributed embedded systems. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '02, pages 514–, Washington, DC, USA, 2002. IEEE Computer Society.

[102] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. Network file system (nfs) version 4 protocol. 2003.

[103] Willy Sisilo, Fangguo Zhang, and Yi Mu. Privacy-enhanced internet storage. In *AINA '05: Proceedings of the 19th International Conference on Advanced Information Networking and Applications*, pages 603–608, Washington, DC, USA, 2005. IEEE Computer Society.

[104] S. W. Son, G. Chen, and M. Kandemir. Disk layout optimization for reducing energy consumption. In *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*, pages 274–283, New York, NY, USA, 2005. ACM.

[105] Seung Woo Son and Mahmut Kandemir. Energy-aware data prefetching for multi-speed disks. In *CF '06: Proceedings of the 3rd conference on Computing frontiers*, pages 105–114, New York, NY, USA, 2006. ACM.

[106] S.W. Son, M. Kandemir, and A. Choudhary. Software-directed disk power management for scientific applications. pages 4b–4b, April 2005.

[107] S.W. Son, M. Kandemir, and A. Choudhary. Software-directed disk power management for scientific applications. pages 4b–4b, April 2005.

[108] Daniel Stodolsky, Mark Holland, William V. Courtright, II, and Garth A. Gibson. Parity logging disk arrays. *ACM Trans. Comput. Syst.*, 12(3):206–235, 1994.

[109] Peter J. Varman and Rakesh M. Verma. Tight bounds for prefetching and buffer management algorithms for parallel i/o systems. *IEEE Trans. Parallel Distrib. Syst.*, 10(12):1262–1275, 1999.

[110] Jun Wang, Huijun Zhu, and Dong Li. eraid: Conserving energy in conventional disk-based raid system. *IEEE Transactions on Computers*, 57(3):359–374, 2008.

[111] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full sha-1. In *In Proceedings of Crypto*, pages 17–36. Springer, 2005.

[112] Andreas Weissel, Björn Beutel, and Frank Bellosa. Cooperative i/o: a novel i/o semantics for energy-aware applications. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation Due to copyright restrictions we are not able to make the PDFs for this conference available for downloading*, pages 117–129, New York, NY, USA, 2002. ACM.

[113] Ravi Wijayaratne and A. L. Narasimha Reddy. Integrated qos management for disk i/o. In *ICMCS '99: Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, page 9487, Washington, DC, USA, 1999. IEEE Computer Society.

[114] Qing Yang and Yiming Hu. Dcd — disk caching disk: A new approach for boosting i/o performance. pages 169–169, May 1996.

[115] Lin Yuan and Gang Qu. Analysis of energy reduction on dynamic voltage scaling-enabled systems. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(12):1827 – 1837, 2005.

[116] Hu Zhang, Weiguo Wu, Xiaoshe Dong, Depei Qian, and Luogeng Dai. A study on data placement of extensible parallel storage system. pages 610–615, July 2007.

[117] Yumin Zhang, Xiaobo Hu, and D.Z. Chen. Task scheduling and voltage selection for energy minimization. In *Design Automation Conference, 2002. Proceedings. 39th*, 2002.

[118] Qingbo Zhu, Francis M. David, Christo F. Devaraj, Zhenmin Li, Yuanyuan Zhou, and Pei Cao. Reducing energy consumption of disk storage using power-aware cache management. In *HPCA '04: Proceedings of the 10th International Symposium on High Performance Computer Architecture*, page 118, Washington, DC, USA, 2004. IEEE Computer Society.

[119] Z. Zong, M. Briggs, N. O'Connor, and X. Qin. An energy-efficient framework for large-scale parallel storage systems. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1 –7, 2007.

[120] Ziliang Zong, Mais Nijim, Adam Manzanares, and Xiao Qin. Energy efficient scheduling for parallel applications on mobile clusters. *Cluster Computing*, 11:91–113, March 2008.

[121] Ziliang Zong, Mais Nijim, Adam Manzanares, and Xiao Qin. Energy efficient scheduling for parallel applications on mobile clusters. *Cluster Computing*, 11:91–113, March 2008.

[122] Ziliang Zong, Xiao Qin, Xiaojun Ruan, Kiranmai Bellam, Yiming Yang, and Adam Manzanares. A simulation framework for energy efficient data grids. In *Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come*, WSC '07, pages 1417–1423, Piscataway, NJ, USA, 2007. IEEE Press.