Dynamic I/O-Aware Load Balancing and Resource Management for Clusters

by

XIAO QIN

A DISSERTATION

Presented to the Faculty of
The Graduate College at the University of Nebraska
In Partial Fulfillment of Requirements
For the Degree of Doctor of Philosophy in Computer Science

Under the Supervision of Professor Hong Jiang

Lincoln, Nebraska

July, 2004

To my parents
and Ying Peng

Acknowledgements

First of all, I would like to express my heartfelt thanks to my advisor, Dr. Hong Jiang, for his guidance, support, and encouragement during the course of my graduate study at the University of Nebraska-Lincoln. I am especially indebted to him for teaching me both research and writing skills, which have been proven beneficial for my current research and future career. Without his endless efforts, knowledge, patience, and answers to my numerous questions, this dissertation research would have never been possible. The experimental methods and results presented in this dissertation have been influenced by him in one way or the other. It has been a great honor and pleasure for me to do research under Dr. Jiang's supervision.

I would also like to thank Dr. David R. Swanson for discussing ideas and reviewing the original versions of my dissertation. The discussions in addition to his valuable comments have tremendously helped in improving the quality of each chapter in the dissertation.

I am grateful to Dr. Steve Goddard not only for reviewing my dissertation, but more importantly for teaching me the right way to present the motivation of my dissertation research. His insightful feedback helped me improve the presentation of the dissertation in many ways.

I am indebted to Dr. Byrav Ramamurthy for enlightening me on the issue of communication-aware load balancing presented in Chapter 7 of the dissertation. My work related to network I/O has been made possible by his helpful suggestions.

I would like to express my gratitude to Dr. Xiao Cheng Zeng for participating in my presentation of the dissertation proposal. I am equally grateful to him for reading my dissertation.

I am particularly grateful to Yifeng Zhu and Feng Xian for their input and discussions on my research. I also wish to thank all members of our research group who have contributed to the SDI project sponsored by NSF. Some of these people include Xili Liu, Yijun Lu, Jameela Al-Jaroodi, Nader Mohamed, Xi Yang, Jun Gao and all other friends who helped me during my study in Lincoln. I am grateful to Sumanth J.V., Makoto Furukawa, Siva-Mohan Sunkavalli and Cory Lueninghoener for maintaining the Research Computing Facility and my computing environment where my experiments were conducted.

I owe a debt of gratitude to Donna McCarthy for offering me good suggestions on a variety of issues such as social life and cooking.

I am very much indebted to Liping Pang and Zongfen Han, my advisors when I was a master student at Huazhong University of Science and Technology, for giving me a solid foundation in computer science.

I thank my parents and family for their unlimited support and strength. Without their dedication and dependability, I could not have pursued my Ph.D. degree at the University of Nebraska.

Most of all, I must thank my wife Ying Peng, whose endless love and encouragement have been my source of inspiration. During the past year, Ying has provided me with sufficient time needed to do research and write this dissertation. I would have never succeeded without her.

Table of Contents

Chapter

List of Tables

List of Figures

Chapter 1

Introduction

With the advent of powerful processors, fast interconnects, and low-cost storage systems, clusters of commodity workstations or PCs have emerged as a primary and cost-effective infrastructure for large-scale scientific and web-based applications. A large fraction of these applications are parallel and data-intensive, since these applications deal with a large amount of data transferred either between memory and storage systems or among nodes of a cluster via interconnection networks. Unfortunately, there is growing evidence that disk- and network-I/O tend to become severe performance bottlenecks for data-intensive applications executing on clusters [Rosario and Choudhary, 1994][Feng et al., 2003].

We believe that an efficient way to alleviate the I/O bottleneck problems is to design resource management mechanisms for clusters to accommodate the characteristics of these classes of data-intensive applications. In particular, dynamic load-balancing techniques can improve the performance of clusters by attempting to assign work, at run time, to machines with idle or under-utilized resources. The objective of this dissertation work is to investigate dynamic load-balancing support for parallel data-intensive applications running on clusters.

This chapter first presents the problem statement in Section 1.1. In Section 1.2, we describe the scope of this research. Section 1.3 highlights the main contributions of this dissertation, and Section 1.4 outlines the dissertation organization.

1.1 Problem Statement

In this section, we start with an overview of new trends in cluster computing. Section 1.1.2 itemizes several challenging cluster-specific requirements on resource management imposed by these technological trends. Finally, Section 1.1.3 presents the initial motivation for the dissertation research by illustrating the limitations of existing solutions.

1.1.1 New Trends

A cluster is a parallel and distributed processing system comprising a group of interconnected commodity computers, which work together as an integrated computing platform [Sterling et al., 1999]. Over the last ten years, clusters of commodity workstations or PCs have become increasingly popular as a powerful and cost-effective platform for parallel programming. It is believed that clusters are becoming an alternative to traditional supercomputers in many application domains. This trend has been made possible in part by the rapid development in processor, networking, and storage technologies that lead to small, inexpensive, high-performance computers.

Previous studies have shown that in cluster environments, allocating and managing resources in an efficient way can deliver high performance as well as high throughput [Subramani et al., 2002]. In particular, there exists a large body of excellent research related to load-balancing techniques, showing that load balancing support can achieve high performance and parallel speed up in cluster systems [Figueira and Berman, 2001][Lavi and Barak, 2001][Xiao et al., 2002].

For general-purpose workloads that may comprise CPU-, memory-, and data-intensive applications, scheduling and load balancing based on the notion of resource time-sharing are envisioned as more suitable than those strategies based on that of resource space-sharing [Anglano, 2000][Arpaci-Dusseau and Culler, 1997]. This is because time-sharing approaches can better fit the needs of workloads with interactive and I/O-intensive applications. To achieve a high utilization of a variety of resources such as CPU, memory,

disks, and networks, these resources are often shared among competing applications, which may be subject to resource fluctuations during the course of execution [Figueira and Berman, 2001][Weiss-man, 2002]. In time-sharing clusters, effective load balancing schemes can be applied to continually sustain high performance.

Due to the high scalability and low cost of clusters, they have been increasingly applied to a wide variety of application domains, including a data-intensive computing domain where applications frequently explore, query, and analyze large datasets. Typical examples of data-intensive applications include long-running simulations, satellite data processing, remote-sensing database systems, medical image databases, high-volume visualization, biological sequence analysis, and data and web mining, to name just a few. Data-intensive applications may take full advantages of the use of cluster systems, since clusters can provide huge yet cost-effective computing power in such a way that the data-intensive applications are able to process large-scale datasets in a reasonable amount of time.

In addition to achieving high performance, clusters have a good potential to provide large-scale storage resources for data-intensive applications. This is because each node of a cluster has one or more local disks, and the cluster may allow applications to access local disks to fully utilize the storage capacities of the whole system.

1.1.2 Cluster-Specific Requirements

Although clusters are becoming popular as a high-performance computing platform for data-intensive applications, increasing evidence has shown that performances of data-intensive applications are much more severely limited by a combination of a persistent lack of high disk- and network-I/O bandwidth and a significant increase in I/O activities in such applications. In other words, performance bottlenecks for data-intensive applications running in cluster environments are caused by disk- and network-I/O rather than CPU or memory performance [Rosario and Choudhary, 1994][Feng et al., 2003]. There are multiple reasons for this phenomenon. First, the performance gap between processors and

I/O subsystems in clusters is rapidly widening. For example, processor performance has seen an annual increase of 60% or so for the last two decades, while the overall performance improvement of disks has been hovering around an annual growth rate of 7% during the same period of time [Hennessy and Patterson, 2002]. Second, many commonly used interconnects, such as FDDI, Fast Ethernet, and ATM, are much slower than those used in massively parallel processors (MPP) or symmetric shared-memory processors (SMP). Third, the heterogeneity of various resources in clusters makes the I/O bottleneck problem even more pronounced.

Executing data-intensive applications in cluster environments imposes several challenging requirements on resource management design. In particular, five new requirements on load-balancing design for clusters are summarized below.

1. *Load-balancing mechanisms must be highly scalable and reliable.* The use of centralized load-balancing schemes is a common approach to resource management. Centralized schemes typically require a head node that is responsible for handling load distribution in a cluster. As the cluster size scales up, the head node quickly becomes a bottleneck, causing significant performance degradation. In addition, a centralized scheme has the potential problem of poor reliability because permanent failures of the central load balancer can render the entire load balancing mechanism dysfunctional. Therefore, the focus of this dissertation is on designing scalable and reliable load-balancing techniques by avoiding a centralized head node to handle load distribution.
2. *The next-generation of clusters provide single-execution environments that continuously operate, while users submit a variety of applications with a wide range of resource requirements for executions.* For this reason, a single load-balancing scheme has to deliver high performance under a wide spectrum of workload conditions where data-intensive jobs share resources with many memory- and CPU-intensive jobs.
3. *It becomes imperative for load-balancing schemes to consider heterogeneity in various resources.* The reason for this is two-fold. First, clusters are likely to be

heterogeneous in nature, since new nodes with divergent properties may be added to the systems to improve performance. Second, heterogeneity in disks inevitably induces more significant performance degradation when coupled with imbalanced load of memory and I/O resources. For this reason, load balancing approaches must strive to hide the heterogeneity of resources in such a way that there is no need to change any source code of data-intensive applications due to the resource heterogeneity.

4. *Clusters must provide sustained high-performance for both memory- and data-intensive applications if they are to be widely used in numerous application domains.* Given this goal, clusters require resource managers to minimize the number of page faults for memory-intensive jobs while increasing the utilization of disk I/O buffers for data-intensive jobs. A common approach has been to statically configure I/O buffer sizes, and this approach performs well for circumstances under which workloads can be modeled and predicted. However, real-world workloads in cluster environments are dynamic, imposing a requirement for resource managers to improve performance of clusters with dynamic workload conditions.
5. *Load balancers must improve the effective utilization of network resources.* This requirement is made necessary by two facts. First, the vast majority of clusters use commodity interconnects such as FDDI, Fast Ethernet, ATM, and Myrinet, and some of these connections exhibit low bandwidth as well as high latency. Second, moving large amounts of data from one component to another through commodity interconnects is likely to result in a major performance bottleneck.

This dissertation addresses these five cluster-specific requirements by presenting a series of I/O-aware load-balancing schemes and a disk-I/O buffer controller, which aim at reducing disk I/O latencies as well as communication overhead.

1.1.3 Limitations of Existing Approaches

Any efficient approach to enhancing the performance of existing applications running on clusters without changing any source code of the applications will likely make use of load-balancing techniques to improve the usage of resources. Recall that the first cluster-specific requirement is high scalability and reliability and, hence, the focus of this dissertation is on distributed load balancing support for clusters rather than centralized ones. Several distributed load-balancing schemes have been presented in the literature, primarily considering CPU [Czajkowski et al., 1998][Hui and Chanson, 1999], memory [Acharya and Setia, 1999] [Voelker, 1997], and a combination of CPU and memory resources [Xiao et al., 2002]. Although these policies achieve high system performance by increasing the utilization of the targeted resources, they are less effective when a workload comprises a large number of data-intensive jobs and I/O resources exhibit imbalanced load.

A large body of work can be found in the literature that addresses the issue of balancing the load of disk I/O. For example, Lee et al. proposed two file assignment algorithms that balance the load across all disks [Lee et al., 2000]. The I/O load balancing policies in these studies have been proven to be effective in improving overall system performance by fully utilizing available hard drives. Zhang et al. proposed three I/O-aware scheduling schemes that are aware of a job's spatial preferences [Zhang et al., 2003]. However, because the above techniques are developed to balance explicit disk I/O load, these approaches become less effective for complex workloads under which data-intensive tasks share resources with many memory- and CPU-intensive tasks. Unlike the existing I/O-aware load balancing schemes, our technique tackles the problem by considering both explicit disk I/O invoked by application programs and implicit disk I/O induced by page faults.

For data-intensive applications, parallel file systems (such as PVFS [Carns et al., 2000]) supporting clusters can deliver high performance and scalable storage by connecting independent storage devices together. Additionally, collective I/O where all processes work together to handle large-scale I/O operations has been proposed to improve disk I/O performance in many parallel applications. There exists a large base of research related to

efficient support for parallel I/O in scientific applications running in cluster environments [Cho, 1999]. These techniques can greatly improve disk I/O performance on a large variety of cluster configurations.

Many researchers have shown that disk I/O cache and buffer are useful mechanisms to optimize storage systems. Ma et al. have implemented active buffering to alleviate the I/O burden by using local idle memory and overlapping I/O with computation [Ma et al., 2002]. Forney et al. have investigated storage-aware caching algorithms in heterogeneous clusters [Forney et al., 2001]. While the above approaches tackle the I/O bottleneck problem by considering storage systems, our technique address the issue of high-performance I/O at the application level by fully utilizing various resources including hard drives.

Recently, some efforts have been made on storing data for data-intensive applications in a huge disk array or storage area network. However, in this dissertation we choose to utilize commodity IDE disks that already exist as an integral part of each cluster node. The idea is motivated in part by the fact that the average price of commodity IDE disks, which stood at US$0.5/GB by August 2003, has been steadily decreasing, and it makes great economic sense to use IDE for its cost-effectiveness. Our approach is a desirable way to develop cost-effective clusters in the sense that the approach provides high performance storage services without requiring any additional hardware. Furthermore, our approach can potentially fulfill the high bandwidth requirements of data-intensive applications, thereby making clusters more scalable. In contrast, cluster-attached disk arrays or storage area networks rely on single node servers, where all the disks are connected with one node. In single-node servers with disk arrays or storage area networks, I/O processing nodes tend to become a performance bottleneck for a large-scale cluster. Although such a bottleneck can be alleviated by a variety of techniques such as caching [Forney et al., 2001] and collective I/O [Cho, 1999], these techniques are still more expensive and beyond the scope of this dissertation.

1.2 Scope of the Research

The dissertation research focuses on distributed load-balancing techniques for data-intensive applications executing on clusters as well as a dynamic buffer optimization approach.

We have proposed an I/O-aware load-balancing scheme to meet the needs of a cluster system with a variety of workload conditions. This approach is able to balance implicit disk I/O load as well as that of explicit disk I/O. Importantly; we have extended the preliminary I/O-aware load-balancing strategy from five different angles. First, we have incorporated preemptive job migration to further improve the system performance over non-preemptive schemes. Second, we have developed an approach for hiding the heterogeneity of resources, especially that of I/O resources. Third, we have devised two effective load-balancing schemes for parallel I/O-intensive jobs. Fourth, we have developed a feedback control mechanism to adaptively manipulate disk I/O buffer sizes. Fifth, we have proposed a communication-aware load-balancing approach, which is effective in improving network utilization under workload situations with high communication intensity.

1.3 Contributions

Due to the rapidly widening performance gap between processors and I/O subsystems in clusters, storage and network subsystems tend to become performance bottlenecks for data-intensive applications executing on clusters. To remedy the I/O-bottleneck problem, our research investigates dynamic distributed load-balancing schemes that are capable of achieving high utilization for disks and networks. In what follows, we list the key contributions of the dissertation.

- **A New Disk-I/O-Aware Load-Balancing Scheme:** We have developed a new disk-I/O-aware load-balancing scheme that significantly improves the overall performance of clusters under a broad range of workload conditions. The new scheme relies on a

technique that allows a job's I/O operations to be conducted by a node that is different from the one in which the job's computation is assigned, thereby permitting the job to access remote I/O.

- **Preemptive Job Migrations:** We have proposed a second load-balancing scheme where a running job is eligible to be migrated only if it is expected to improve the overall performance. We have shown that preemptive job migrations can improve the performance of a cluster over non-preemptive schemes, and the proposed approach achieves the improvement in mean slowdown by up to a factor of 10.
- **Consideration of the Heterogeneity of Resources:** There is a strong likelihood for upgraded clusters or networked clusters to become heterogeneous in terms of resources such as CPU, memory, and disk storage. Since heterogeneity in disks inevitably leads to significant performance degradation when coupled with an imbalanced load of I/O and memory resources, we have devised an approach for hiding the heterogeneity of resources, especially that of disk I/O resources, by balancing disk I/O workloads across all the nodes in a cluster.
- **Support for Parallel Applications:** We have extended our work in load balancing for sequential jobs by developing two simple yet effective disk-I/O-aware load-balancing schemes for parallel data-intensive jobs running on clusters. Using a set of real data-intensive parallel applications and synthetic parallel jobs with various I/O characteristics, we show that these two schemes consistently improve the performance over existing load-balancing schemes that are not disk-I/O-aware.
- **Improve Buffer Utilization:** In addition to our research in issues related to load balancing, we have developed a feedback control mechanism to improve the performance of a cluster by adaptively manipulating I/O buffer sizes. Experimental results from simulations show that this mechanism is effective in enhancing the performance of a number of existing load-balancing schemes.

- **Communication-Aware Load Balancing:** We have proposed a communication-aware load-balancing scheme that is effective in increasing network utilization of clusters. To facilitate the proposed load-balancing scheme, we have introduced a behavior model for parallel applications with large requirements of CPU, memory, network, and disk I/O resources. The proposed load-balancing scheme can make full use of this model to quickly and accurately determine load induced by a variety of parallel applications.

1.4 Dissertation Organization

This dissertation is organized as follows. In Chapter 2, related work in the literature is briefly reviewed.

In Chapter 3, we develop two disk-I/O-aware load-balancing schemes, referred to as IOCM and WAL-PM, to improve the overall performance of a cluster system with a general and practical workload including intensive I/O activities.

To make the load-balancing policies presented in Chapter 3 more practical, we explore two disk-I/O-aware load-balancing schemes for parallel jobs in Chapter 4.

In Chapter 5, we study the issue of dynamic load balancing for heterogeneous clusters under I/O- and memory-intensive workload conditions.

In Chapter 6, a new feedback control algorithm for managing I/O buffers is presented and its performance is evaluated.

In Chapter 7, we propose a communication-aware load balancing support technique that is capable of improving the performance of applications with high communication demands.

Finally, Chapter 8 summarizes the main contributions of this dissertation and comments on future directions for this research.

Chapter 2

Related Work

Clusters are becoming popular as high-performance computing platforms for numerous application domains, and a variety of techniques have been proposed to improve performances of clusters. This chapter briefly presents previous approaches found in the literature that are most relevant to our research from two perspectives: load balancing and high performance storage systems.

2.1 Load Balancing

In general, load balancing provides parallel and distributed systems with an ability to avoid the situation where some resources of the systems are overloaded while others remain idle or underloaded. It is well understood that excessively overloading a portion of resources can substantially reduce the overall performance of the systems. This section presents a summary of work related to load balancing techniques. Specifically, we describe the important features in a wide range of load balancing approaches. These features include static and dynamic load balancing (See Section 2.1.1), considerations for system heterogeneity (See Section 2.1.2), and migration strategies (See Section 2.1.3).

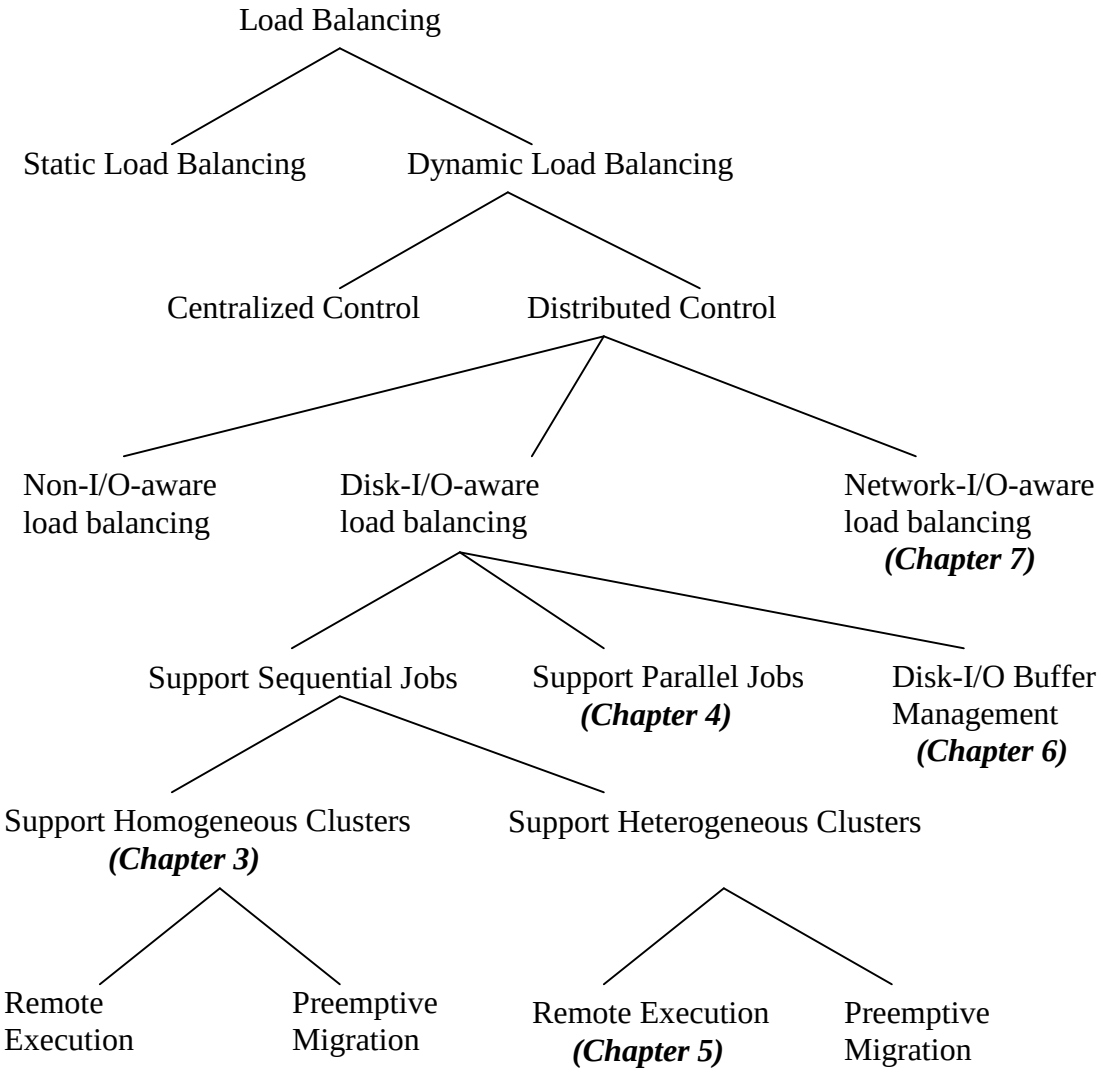


Figure 2.1 A simplified taxonomy of the approaches to the load balancing problem

2.1.1 Static vs. Dynamic Load Balancing: A Simplified Taxonomy

We can classify existing load balancing approaches into two broad categories: static and dynamic (See Figure 2.1). In each category, there can be further classifications to associate certain specific attributes to a group of schemes. These attributes include, but are not

limited to, control philosophy (centralized vs. decentralized), targeted resource for optimization, type of jobs (sequential vs. parallel, real-time vs. non-real-time, etc.), nature of the underlying system (homogeneous vs. heterogeneous), migration strategies (remote execution vs. preemptive migration), etc. Figure 2.1 shows a simplified (subset of) a taxonomy of load balancing schemes most relevant to this dissertation research.

A large body of work has been done for static load-balancing schemes that do not rely on the current state of nodes [Li and Kameda, 1998]. Tantawi and Towsley have exploited optimal static load balancing for a single job class in star and bus network configurations [Tantawi and Towsley, 1985]. Based on this study, Kim and Kameda have proposed two static load-balancing algorithms, which are not only more effective but also able to further improve system performance [Kim and Kameda, 1992]. Chronopoulos et al. have developed a static load-balancing scheme for Computational Fluid Dynamics simulations on a network of heterogeneous workstations [Chronopoulos et al., 2001].

In contrast, dynamic load balancing approaches provide an ability to improve the quality of load distribution at run-time at a reasonable cost of communication and processing overhead. McCann et al. have studied a dynamic scheduling strategy, which is aware of resource needs of submitted tasks [McCann et al., 1993]. Their design is based on a centralized manager that handles scheduling on each processor, and the approach is not very scalable.

Condor has been developed to harvest idle cycles of a cluster of computers by distributing batch jobs to idle workstations [Douglis and Ousterhout, 1991]. One of the ultimate goals of Condor is to guarantee that workstations immediately become available for their owners when the owners are about to access the machines. This goal is approached by detecting an owner's activity at an idle machine, and migrating background jobs to other idle machines when the owner begins accessing his or her machine. For reliability purpose, Condor periodically makes checkpoints for jobs, thereby making it possible to restore and resume jobs in the presence of software and hardware failures. In addition, Condor offers a

flexible mechanism where each machine's owner can specify conditions under which the machine is considered idle.

Watts and Taylor have examined a cohesive and practical dynamic load-balancing framework, which is scalable, portable and easy to use [Watts and Taylor, 1998]. Their approach aims at improving existing strategies by incorporating a new diffusion algorithm to offer a good tradeoff between total work transfer and run time. In addition, the load balancing approach takes advantage of a task selection mechanism, allowing task size and communication cost to guide task movement.

Harchol-Balter and Downey have proposed a CPU-based dynamic load balancing policy that is effective in improving CPU utilization [Harchol-Balter and Downey, 1997]. Zhang et al. focused on dynamic load sharing policies that consider both CPU and memory services among the nodes. The experimental results show that their policies not only improve performance of memory-intensive jobs, but also maintain the same load sharing quality of CPU-based policies for CPU-intensive jobs [Zhang et al. 2000].

This dissertation only addresses the dynamic load-balancing problem. Hereafter, we refer to the dynamic load-balancing problem simply as load balancing.

2.1.2 Homogeneous vs. Heterogeneous Load Balancing

At another level, load balancing approaches can also be broadly broken down into homogenous and heterogeneous ones (See Figure 2.1). The focus of homogeneous load balancing schemes is to improve the performance of homogeneous parallel and distributed systems. For example, Colajanni et al. have addressed the issue related to dynamic load balancing in the context of distributed homogeneous Web servers [Colajanni et al., 1997]. The results show that their strategies are robust and effective in balancing load among Web servers.

On the other hand, heterogeneous load balancing approaches attempt to boost the performance of heterogeneous clusters, which comprise a variety of nodes with different performance characteristics in computing power, memory capacity, and disk speed. Some

static and dynamic scheduling policies for heterogeneous parallel systems can be found in [Menasce et al., 1995]. A static load balancing scheme for Computational Fluid Dynamics simulations on a network of heterogeneous workstations has been studied by Chronopoulos et al. Their load-balancing algorithm takes both the CPU speed and memory capacity of workstations into account [Chronopoulos et al., 2001]. To dynamically balance computational loads in a heterogeneous cluster environment, Cap and Strumpen have explored heterogeneous task partitioning and load balancing [Cap and Strumpen, 1993]. Xiao et al. have investigated an approach that considers both system heterogeneity and effective usage of memory resources, thereby minimizing both CPU idle time and the number of page faults in heterogeneous systems [Xiao et al., 2002].

The dynamic CPU- and memory-based load balancing techniques for heterogeneous systems have been extensively studied. However, disk-I/O-aware load balancing in the context of heterogeneous systems is an interesting area yet to be explored. Therefore, in Chapter 5 we examine the problem of load balancing for data-intensive applications executing on heterogeneous clusters.

2.1.3 Remote Execution vs. Job Migrations

Some existing load balancing techniques rely on remote execution, while others depend on process migration that requires operating systems or middleware to package the state of migrated processes (See Figure 2.1).

Distributed Queuing system (DQS) is a cluster computing system that supports most of existing operating systems [Duke et al., 1994]. Similar to Condor, DQS suspends running background jobs if keyboard or mouse activities of a private machine are detected. By doing so, DQS can guarantee the local autonomy of private machines. Interestingly, jobs in DQS may be suspended and resumed rather than being migrated from one machine to another, since the focus of DQS is to distribute jobs among a group of shared machines.

The Butler system allows users to utilize idle resources by providing transparent remote execution on idle workstations [Dannenberg and Hibbard, 1985]. However, Butler

does not support job migrations. When machine owners begin accessing their workstations, Butler delivers messages to inform remote users and kills the processes submitted by the remote users. Consequently, the work of remote jobs may be lost upon the returns of machine owners.

Load Sharing Facility (LSF) has been developed to queue and distribute jobs among a large group of heterogeneous clusters consisting of thousands of workstations [Zhou et al., 1993]. Unlike Condor systems, local autonomy is not a main concern in LSF, and it is assumed in LSF that every machine is sharable by multiple jobs submitted from different machines. The goal of LSF is to evenly distribute load among cluster machines to achieve short turn-around time and high system utilization. LSF does not have the ability to migrate jobs from one node to another, meaning that a job can only run on the node to which it is originally submitted.

Harchol-Balter and Downey have proposed a CPU-based preemptive migration policy. The results indicate that their preemptive migration policy is more effective in achieving high performance than non-preemptive migration policies in cases where workload conditions are CPU-intensive [Harchol-Balter and Downey, 1997]. Zhang et al. have developed a number of preemptive load sharing policies for both CPU- and memory-intensive workloads. In contrast, their results show that the performance of the policies with preemptive migrations is worse than that of the non-preemptive migration ones under memory-intensive workload situations [Zhang et al., 2000]. In Chapters 3 and 4, we address the intriguing issue of whether preemptive job migration can achieve performance improvement over non-preemptive schemes for both sequential and parallel data-intensive applications running on homogeneous clusters.

2.2 High Performance Storage Systems

There is a large body of research related to high performance storage systems in clusters. In this section, we survey previous techniques supporting high performance storage systems.

In particular, Section 2.2.1 describes disk-striping techniques. Section 2.2.2 summarizes previous work related to parallel file systems. Finally, disk I/O caching and buffering techniques are briefly overviewed in Section 2.2.3.

2.2.1 Disk Striping

Striping data across multiple storage devices and accessing the striped data in parallel have been envisioned as an efficient way of reducing the performance gap between processors and storage systems [Salem and Garcia-Molina, 1986]. The goal of disk striping techniques is to combine several storage disks into a large logical disk. To do so, the storage systems are configured in a way that can optimize striping layout parameters for various workload characteristics.

The PASSION input/output library has been developed to optimizing disk I/O accesses to out-of-core arrays in parallel programs [R. Bordawekar et al., 1994]. More specifically, the compiler in PASSION optimizes data distribution by generating explicit parallel disk I/O calls from the specifications in high-level languages. Since the data distribution optimization is done at compile-time in PASSION, it is imperative that programmers modify application source code in order to inform the compiler to generate I/O calls with optimized data distribution. Moreover, PASSION is not able to meet the needs of systems where their states change during the course of execution.

The disk striping research pioneered by Scheuermann et al. is close to our work presented in Chapters 3, 4 and 5, because they have studied a heuristic algorithm for load balancing in disk striping systems [Scheuermann et al., 1998]. The idea is motivated by the concept of disk cooling, attempting to move some parts of striped files from overloaded disks to disks with low utilization. Although their approach can balance disk I/O load among disks, it is less effective in optimizing the utilization of other resources such as memory and networks. We believe that in cases where each node of a cluster consists of several disk devices, it would be appealing to combine our I/O-aware load balancing schemes with their disk cooling method to achieve additional performance improvement. In

other words, while disk cooling is applied to balance the load of disks within a node, our schemes can be deployed to solve the load imbalance problem among the nodes.

2.2.2 Parallel File Systems

Parallel file systems such as PVFS [Ligon and Ross, 1996] and GFS [Preslan et al., 1999] are important components that support high performance storage systems for clusters. Although commodity network interconnections in clusters may have relatively low bandwidth and high latency, parallel file systems can increase the effective bandwidth by striping data over several disks residing on different cluster nodes.

We have designed and analyzed a cost-effective and fault-tolerant parallel virtual file system (CEFT-PVFS) that has been implemented on the PrairieFire supercomputer at UNL [Zhu et al., Cluster03][Zhu et al., SNAPI03]. The CEFT-PVFS system has been proposed to extend the existing PVFS system [Ligon and Ross, 1996] by mirroring while at the same time trying to maintain a reasonably high level of write performance. Experimental study suggests that it is feasible to build a considerably reliable storage system with multi-terabyte capacity without any additional hardware cost. To optimize the performance of CEFT-PVFS, dynamic load-balancing algorithms for read and write operations have been proposed and developed [Zhu et al., LCI03][Zhu et al., CCGrid03].

SPSort is an application that only takes 17 minutes to sort a tera bytes of data on an RS/6000 SP with 488 nodes, where 336 RAID arrays are attached to 56 of the SP nodes [Wyllie, 1999]. Note that the input and output data for SPSort is stored in the GPFS parallel file system, which stripes files across the RAID arrays. Unfortunately, due to the large scale of the system and non-commodity parts of some hardware/software, the cost of the system running SPSort is very high.

The Compaq Computer Corporation and the DOE Sandia laboratories run the TeraByteSort benchmark on a cost-effective cluster with 72 nodes of commodity parts, where each node comprises 8 local disks and 400 MHz two-way SMP Pentium II processors. The nodes are connected by ServerNet-I with Virtual Interface Architecture

(VIA) [Compaq et al, 1997]. The result shows that the TerabyteSort benchmark takes 46.9 minutes. The processor utilization is only 23 percent, while more than 60% time is spent merging a tera bytes of data. This result implies that the benchmark performance is bottlenecked by both disk- and network-I/O performance.

Arpaci-Dusseau et al. have proposed a new data-flow programming paradigm along with I/O subsystem for cluster environments, where distributed queues and data redundancy are deployed [Arpaci-Dusseau et al., 1999]. To improve read performance, adaptive load balancing has been incorporated into a declustering strategy with duplicated data on disks. Due to the new data-flow environment, the source code of existing parallel applications has to be modified in order to make use of the new I/O subsystem. However, this is not the case for our disk-I/O-aware load balancing schemes proposed in Chapters 3-5, because our load balancing approaches can enhance the performance of existing applications without changing any source code of the applications. Arpaci-Dusseau et al. have conduced extensive experiments and, based on a cluster with one or more slower disks, they conclude that the system performance degrades gracefully with the increasing number of slower disks. Our results reported in Chapter 5 are consistent with this conclusion.

PPFS is a parallel file system implemented as a user-level library on top of Unix file systems, offering users a POSIX-like interface [Huber et al., 1995]. PPFS provides various APIs that allow applications to define access patterns as well as caching and prefetching policies.

PIOUS is a parallel file system with a Unix-style file interface designed for workstation clusters [Moyer and Sunderam, 1995]. Natarajan and Iyer have investigated the performance of PIOUS on a DEC Alpha Supercluster where disk bandwidth is 5 MB/Sec. and interconnection is switch-based FDDI [Natarajan and Iyer, 1996]. Cho et al. have confirmed that message passing across an FDDI interconnection is a performance bottleneck for PIOUS on a HP workstation cluster [Cho et al., 1997][Natarajan and Iyer, 1996]. They have simulated a small-scale cluster with 8 compute and I/O nodes connected

by FDDI to compare a local disk architecture against a remote disk architecture. Their results indicate that the performance of the local disk architecture is better than that of the remote disk architecture. The communication-aware load balancing approach presented in Chapter 7 can be applied to PIOUS to alleviate the interconnection bottleneck problem.

2.2.3 I/O Caching and Buffering

Although commodity disks in clusters have low bandwidth compared with that of main memory, many researchers have shown that disk I/O caching and buffering are useful mechanisms to achieve reasonably high performance of commodity disks.

Ma et al. have implemented active buffering to alleviate the I/O burden imposed by I/O-intensive applications by using locally idle memory capacity and overlapping I/O with computation [Ma et al., 2002]. Forney et al. have investigated storage-aware caching algorithms that partition the cache and explicitly account for the differences in performance across devices in heterogeneous clusters [Forney et al., 2001]. To sustain high performance for dynamic disk I/O patterns, PPFS can adaptively configure caching, prefeching, and striping policies to fulfill the requirements of dynamic systems with a wide range of I/O access patterns [Huber et al., 1995].

Although we focus solely on balancing disk I/O load in Chapters 3-5, the approaches proposed here are capable of improving the buffer utilization of each node, which in turn increases the buffer hit rate and reduces disk I/O access frequency. In Chapter 6, we develop a feedback control mechanism to improve the performance of a cluster by manipulating disk I/O buffer sizes. The load-balancing schemes presented in this dissertation are orthogonal but complementary to the existing caching/buffering techniques and, thus, integrating our proposed schemes into the caching/buffering techniques can provide additional performance improvement.

2.3 Summary

The objective of this dissertation is to present I/O-aware load balancing approaches, which is based on previous research efforts in load balancing and high performance storage systems. Therefore, this chapter has overviewed a variety of existing techniques related to load balancing and high performance storage systems for cluster environments.

Note that excessively overloading a portion of resources can substantially reduce overall system performance. Consequently, load balancing attempts to avoid an imbalanced resource utilization in clusters where some resources of the systems are overloaded while others remain idle or underloaded.

In the first part of this chapter, we have presented a simplified taxonomy of load balancing approaches. In addition, we have compared existing approaches from three aspects: static and dynamic load balancing, considerations for system heterogeneity, and migration strategies.

In the second part of this chapter, we have surveyed existing techniques used in high performance storage systems. These techniques include disk-striping techniques, parallel file systems, and disk I/O caching/buffering techniques.

Chapter 3

Disk-I/O-Aware Load Balancing Policies for Sequential Jobs

In this Chapter, we consider the problem of distributed dynamic load balancing among a cluster of homogeneous nodes (the terms node and workstation are used interchangeably throughout this dissertation) connected by a high-speed network, where tasks arrive at each node dynamically and independently, and share resources available there.

Specifically, two disk-I/O-aware load-balancing schemes, referred to as IOCM and WAL-PM, are presented in this chapter to improve the overall performance of a cluster system with a general and practical workload including I/O activities. The proposed schemes dynamically detect I/O load imbalance on nodes of a cluster, and determine whether to migrate some I/O load or running jobs from overloaded nodes to other less- or under-loaded nodes. Besides balancing I/O load, the schemes judiciously take into account both CPU and memory load sharing in the system, thereby maintaining the same level of performance as existing schemes when I/O load is low or well balanced.

The rest of the chapter is organized as follows. In the section that follows, the design and system assumptions are outlined. Section 3.2 presents the IOCM and WAL-PM schemes, and Section 3.3 describes the simulator. The performance evaluation of these

schemes is presented in Section 3.4. Finally, Section 3.5 concludes this chapter by summarizing the main contributions of the chapter.

3.1 Assumptions and Notations

In a cluster system, a centralized node (or "head node") could apply a broadcast mechanism (e.g., *gather* and *scatter* like operations) to handle load distribution in a dedicated computing cluster. As the head node becomes increasingly overloaded with the growth of cluster size, the system will inevitably suffer a significant performance drop when the head node becomes a severe bottleneck. A distributed load balancer can be utilized to effectively alleviate the potential burden of the head node, thereby distributing some of its workload among other nodes.

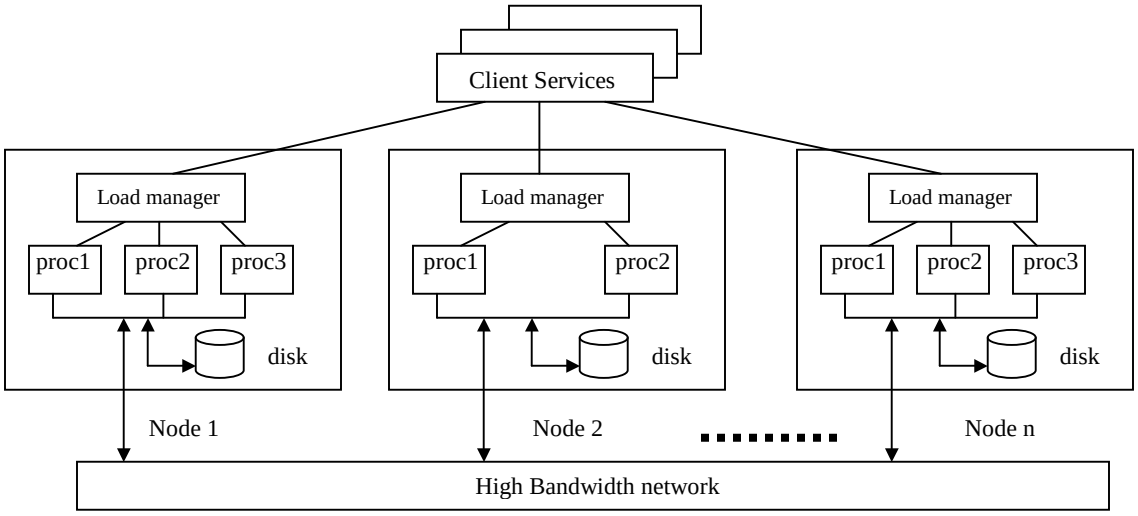


Figure 3.1 A block diagram of the system architecture. Each node maintains a load manager. Users use client services to interact with the environment

As depicted in Figure 3.1, each node has a combination of multiple types of resources, such as CPU, memory, network connectivity, and disks. Moreover, each node maintains a load manager, which is responsible for controlling, monitoring, and load balancing the available resources, in addition to handling the execution of processes generated from the local node.

We assume that in a realistic cluster structure, every job has a "home" node that it prefers for execution [Lavi and Barak, 2001]. The rationale behind this home model is two-fold: (1) the input data of a job has been stored in the home node and, (2) the job was created on its home node. An implication of this assumption is that data initially retrieved by a task is available on the task's home node. Throughout this chapter, we refer to the data initially stored in its home node as *initial data,* which is required to be shipped along with a migrated task. This assumption is conservative in the sense that it makes our approach less effective, because migration overhead imposed by initial data can be potentially avoided by replicating it across all the nodes in a cluster.

We further assume that the notion of home node is a logical one since the "tenant" of this "home", namely the job, may move from one physical node to another due to migrations, taking its "home" (i.e., all the data or the "state" of the job execution associated with the job) with it. This assumption avoids the issue of data consistency among the possible multiple datasets of the same job if its home does not follow the job migration.

Since resource demands of applications may not be known in advance, Section 7.2 introduces an application model that captures the typical characteristics of CPU, memory, network, and disk I/O activities within an application. The load-balancing schemes presented in Chapters 3-5 also can apply this model to accurately determine disk I/O, CPU, and memory requirements of various applications. Note that the resource requirements of an application can be either obtained by repeatedly executing the application off-line or generated by an on-line profiling monitor.

We also assume that the network in our model provides full connectivity in the sense that any two nodes are connected through either a physical link or a virtual link. This

assumption is arguably reasonable for modern interconnection networks (e.g. Myrinet [Boden et al., 1995] and InfiniBand [Wu et al., 2004]) that are widely used in high-performance clusters. Both Myrinet and Infiniband networks provide pseudo-full connectivity, allowing simultaneous transfers between any pair of nodes. Note that this simplification of the network is commonly used in many load-balancing models [Harchol-Balter and Downey, 1997][Hui and Chanson, 1999][Surdeanu et al., 2002][Zhang et al., 2000].

The memory burden placed by migrating data is not considered in our model, because data movement can be handled by storage and network interface controllers without local CPU's intervention or buffer in the main memory. This technique, referred to as Off-Processor I/O, is well studied in [Geoffray, 2002]. Hence, it is reasonable to assume that data migration from one node to another involves no CPU overhead.

It is assumed in many load-balancing schemes that no two or more jobs arrive at different nodes at the same time [Harchol-Balter and Downey, 1997][Zhang et al., 2000]. When a job is submitted through the client service (which may be managed and housed by the head node) to its home node, the load manager assigns the job to a node with the least load.

Each load manager periodically receives reasonably up-to-date global load information from resource monitors [Hsu and Liu, 1986][Stankovic, 1984], which monitors resource utilization of the cluster and periodically broadcasts global load information to other nodes of the cluster. When the number of communicating nodes in the system becomes large or when the system load is heavy, the communication overhead tends to be excessive. To reduce this undesirable overhead, Shin and Chang proposed the notion of buddy sets and preferred lists [Shin and Chang, 1989].

Disk I/O operations generally fall into two categories: blocking and non-blocking I/O. For simplicity, we assume that all I/O operations issued by tasks are blocking, meaning that, for each task, its computation and I/O operations are not overlapped in time. This simplification is conservative in the sense that it underestimates the performance benefits

from the proposed scheme because this assumption causes a number of undesired migrations with negative impact.

Note that the above assumptions are applied to the load balancing schemes presented in Chapters 3, 4, 5, and 7.

3.2 Two Disk-I/O-Aware Load Balancing Policies

In this section, we discuss the issue of dynamic load balancing among a cluster of nodes, $M = \{M_1, M_2, \ldots, M_n\}$, connected by a high-speed network. In particular, two disk-I/O-aware load-balancing policies are presented to achieve performance improvements of a cluster system with a general and practical workload. Each node in the cluster is capable of migrating a newly arrived job *(referred to as remote execution)* or a currently running job preemptively to another node *(referred to as job migration)* if the load manager detects that the local node is heavily loaded.

Note that some existing systems support either remote execution or job migration. For example, Mosix, an operating system for Linux clusters, provides a process migration mechanism on the kernel level [Barak et al., 1999]. RHODOS is a microkernel-based distributed operating system that implements both remote execution and process migration [Paoli et al., 1996]. In RHODOS the load balancing strategies, which makes migration decisions, are separated from the process transferring mechanisms by an appropriate interface.

3.2.1 I/O-CPU-Memory (IOCM) Based Load-Balancing Policy

In the first part of this study, we present IOCM, which is a dynamic I/O-aware load-balancing scheme that also takes into account the CPU and memory demands of jobs. Each job is described by its requirements for CPU, memory, and I/O, which are measured by the number of jobs running in the nodes, Mbytes, and number of disk accesses per million instructions (No./MI for short), respectively.

Algorithm: I/O-CPU-Memory based load balancing (IOCM)
Input: Job *j*, node *i*, **Output:** $M_{IO}(j)$, $M_{CM}(j)$
Step1: **if** I/O load on node *i* is not overloaded **then** $M_{IO}(j) \leftarrow$ local node *i*;
else $M_{IO}(j) \leftarrow$ node with the minimal I/O load;
Step2: **if** memory in node *i* is not overloaded **then** /* Balance CPU/memory load */
if CPU is not overloaded **then** $M_{CM}(j) \leftarrow$ local node *i*;
else $M_{CM}(j) \leftarrow$ node with the minimal CPU load;
else $M_{CM}(j) \leftarrow$ node with the minimal memory load;
Step3: **if** $M_{IO}(j) \neq i$ or $M_{CM}(j) \neq i$ **then**
if $M_{IO}(j) = M_{CM}(j)$ **then** Calculate the migration cost;
else Calculate both the migration cost and the remote I/O access cost;
Determine if the migration can improve the expected performance;
Step 4: **if** I/O migration is worthwhile **then**
if $M_{IO}(j) \neq i$ **and** initial data is stored in node *i* **then**
migrate initial data from node *i* to node $M_{IO}(j)$;

Figure 3.2 Pseudo code of I/O-CPU-Memory based load balancing

For a job *j*, arriving in a local node *i*, the IOCM scheme attempts to balance three different resources simultaneously following five main steps. First, if node *i*'s I/O load is overloaded, a candidate node, $M_{IO}(j)$, that processes the I/O operations issued by the job, is chosen in accordance with the I/O load status. Node *i* is I/O-overloaded, if: (1) its I/O load is the highest; and (2) the ratio between its I/O load and the average I/O load across the system is greater than a threshold, which is set to 1.25 in our experiments. This optimal value, which is consistent with the result reported in [Wu et al., 2001], is obtained from an experiment where the threshold is varied from 1.0 to 2.0. Second, when the memory of node *i* is overloaded, IOCM judiciously determines another candidate node, $M_{CM}(j)$, the one with the lightest memory load, to execute the job. When the node has sufficient memory space, a CPU-base policy is applied to make the load sharing decision. Third, compute migration cost and remote I/O access cost, and finalize the load balancing decision. Fourth, data migration from node *i* to $M_{IO}(j)$ is invoked if the migration is helpful in boosting the performance and the data accessed by job *j* is not initially available in node $M_{IO}(j)$.

Likewise, the job is migrated to node $M_{CM}(j)$ if such migration improves the expected performance. Fifth and finally, the network load and the load status in nodes $M_{IO}(j)$ and $M_{CM}(j)$ are updated. A detailed pseudo code of the IOCM scheme is presented in Figure 3.2.

Table 3.1 Some Notation Symbols and Their Definitions

Symbol	Definition
t_j	Computation time of job j
a_j	Age of job j
$load_{CPU}(i)$	CPU load index of node i
$load_{MEM}(i)$	Memory load index of node i
$l_{page}(i, j)$	Implicit I/O load of job j assigned to node i
$l_{IO}(i, j)$	Explicit I/O load of job j assigned to node i
$r_{MEM}(j)$	Memory space requested by job j
$n_{MEM}(i)$	Memory space in bytes that is available to the job on node i
μ_i	Page fault rate
λ_j	I/O access rate of job j assigned to node i
$H(i, j)$	I/O buffer hit rate of job j assigned to node i
$d_{buf}(i, j)$	Buffer size allocated to job j
$d_{data}(j)$	Amount of data job j retrieves from or stores to the disk
d_j^{RW}	Average data size of I/O accesses of job j
$R(i, k, j)$	Response time of job j on node i remotely accessing data on node k
$r_{PM}(i , j)$	Expected response time of a candidate migrant j on node i
c_j	Migration cost of job j
m_j	Memory size of the migrant job j
d_j^W	Amount of disk (I/O) data generated at the runtime by job j
w_j	Percentage of I/O operations that store data to the local disk

3.2.2 Implicit and Explicit I/O Load

In this load-balancing policy three load indices for CPU, memory and I/O resources are described below (see Table 3.1 for a summary of notation used in this chapter):

(1) The CPU load index of node i is characterized by the number of jobs running on the node [Zhang et al., 2000][Xiao et al., 2002], denoted as $load_{CPU}(i)$.

(2) The memory load index of node i, denoted $load_{MEM}(i)$, is the sum of the memory space allocated to those jobs with their computational tasks assigned to node i. Thus,

$$load_{MEM}(i) = \sum_{j \in M_i} l_{MEM}(j), \tag{3.1}$$

where $l_{MEM}(j)$ represents the memory requirement of job j.

(3) The I/O load index measures two types of I/O accesses: the implicit I/O requests induced by page faults and the explicit I/O requests resulting from the I/O tasks. Let $l_{page}(i, j)$ and $l_{IO}(i, j)$ denote the implicit and explicit I/O load of job j assigned to node i, respectively, then, the I/O load index of node i can be defined as:

$$load_{IO}(i) = \sum_{j \in M_i} l_{page}(i, j) + \sum_{j \in M_i} l_{IO}(i, j). \tag{3.2}$$

The calculation of I/O load is more complicated because of the need to determine both implicit and explicit I/O load. When the node's available memory space is larger than or equal to the memory demand, there is no implicit I/O load imposed on the disk. Conversely, when the memory space of a node is unable to meet the memory requirements of the running jobs, the node tends to encounter a large number of page faults. The load of implicit I/O largely depends on programs' behaviors and buffer replacement policies. The implicit I/O load can be measured by monitoring inter-page fault intervals or an analytical model. For simplicity, we choose to apply the following model, which has been successfully used by other researchers [Zhang et al., 2000], to approximately determine the implicit I/O load of a job. Note that implicit I/O load is inversely proportional to available user memory space and proportional to the page fault rates and memory space requirements of running jobs. Thus, $l_{page}(i, j)$ is defined as follows,

$$l_{page}(i, j) = \begin{cases} 0 & \text{if } load_{MEM}(i) \le n_{MEM}(i), \\ \mu_i \sum_{k \in M_i} r_{MEM}(k) \Big/ n_{MEM}(i) & \text{otherwise,} \end{cases} \tag{3.3}$$

where μ_i is the page fault rate, $r_{MEM}(j)$ denotes the memory space requested by job j, and $n_{MEM}(i)$ represents the memory space in bytes that is available to jobs running on node i.

While $r_{MEM}(j)$ is one of the major parameters of job j provided by users, $n_{MEM}(i)$ is a system parameter of node i that can be given by system administrators.

Job j's explicit I/O load, $l_{IO}(i, j)$, can be expressed as a function of I/O access rate λ_j and I/O buffer hit rate $h(i, j)$. The explicit I/O load can be approximately measured by the following expression:

$$l_{IO}(i, j) = \lambda_j [1 - h(i, j)]. \tag{3.4}$$

The buffer hit rate $h(i, j)$ can be affected by the re-access rate r_j (defined to be the average number of times the same data is repeatedly accessed by job j), the buffer size $d_{buf}(i, j)$ allocated to job j, and the amount of data $d_{data}(j)$ job j retrieves from or stores to the disk, given a buffer with infinite size. There are two approaches to obtaining the value of r_j. The first one is to execute job j multiple times using various inputs, and the second approach is through the use of online profiling mechanisms to dynamically calculate r_j. We will discuss the impacts of re-access rate and average data size on the proposed load-balancing schemes in Sections 3.4.4 and 3.4.6, respectively. The buffer hit rate is approximated by the following formula:

$$h(i, j) = \begin{cases} r_j / (r_j + 1) & \text{if } d_{buf}(i, j) \geq d_{data}(j), \\ \dfrac{r_j d_{buf}(i, j)}{(r_j + 1) d_{data}(j)} & \text{otherwise,} \end{cases} \tag{3.5}$$

The I/O buffer in a node is a resource shared by multiple jobs in the node, and the buffer size a job can obtain in node i at run time heavily depends on the jobs' access patterns, characterized by I/O access rate and average data size of I/O accesses. $d_{data}(j)$ linearly depends on access rate, computation time and average data size of I/O accesses d_j^{RW}, and $d_{data}(j)$ is inversely proportional to I/O re-access rate. $d_{buf}(i, j)$ and $d_{data}(j)$ are estimated using the following two equations, where t_j is the computation time of job j:

$$d_{buf}(i,j) = \frac{\lambda_j d_j^{RW}}{\sum_{k \in M_i} \{\lambda_k d_j^{RW}\}} d_{buf}(i), \tag{3.6}$$

$$d_{data}(j) = \frac{\lambda_j t_j d_j^{RW}}{r_j + 1}. \tag{3.7}$$

From (3.5), (3.6) and (3.7), hit rate *h(i, j)* becomes:

$$h(i,j) = \begin{cases} r_j / (r_j + 1) & \text{if } d_{buf}(i,j) \geq d_{data}(j), \\ \dfrac{r d_{buf}(i)}{t_j \sum_{k \in M_i} \{\lambda_k d_k^{RW}\}} & \text{otherwise.} \end{cases} \tag{3.8}$$

In practice, as the job executes, the operating system kernel can keep track of the I/O access rate and I/O buffer hit rate used in Expression (3.4) and therefore, the values of these two parameters can be dynamically obtained by maintaining a running average for λ_j and *h(i, j)*.

3.2.3 Expected Response Time of an I/O-Intensive Job

We now turn to the calculation of the response time of a job with local/remote I/O accesses and the migration cost, which will be utilized in Step(3) to decide if migration can improve the performance. When a job *j* accesses I/O locally on node *i*, its expected response time can be computed as follows:

$$r(i,j) = t_j E(L_i) + t_j \lambda_j \left[E(s_{disk}^i) + \frac{\Lambda_{disk}^i E[(s_{disk}^i)^2]}{2(1 - \rho_{disk}^i)} \right], \tag{3.9}$$

where λ_j is the I/O access rate of job *j*. $E(s^i_{disk})$ and $E[(s^i_{disk})^2]$ are the mean and mean-square I/O service time in node *i*, and ρ^i_{disk} is the utilization of the disk in node *i*. $E(L_i)$ represents the mean CPU queue length L_i, and Λ^i_{disk} denotes the aggregate I/O access rate in node *i*. The two terms on the right hand side of Equation (3.9) represent the CPU execution time and the I/O processing time, respectively. Round-robin scheduling (time-sharing) is employed as the CPU scheduling policy, and the disk of each node is modeled as a single M/G/1 queue [Lee et al., 2000].

It is assumed in our model that all I/O operations are blocking. In other words, computations and I/O operations are not overlapped. This assumption is reasonable because a vast majority of read operations is blocking in nature and, therefore, the assumption has been widely used [Surdeanu et al., 2002]. Thus, the response time of a job is the summation of CPU and I/O reponse time. This simplification is conservative in the sense that it makes the proposed I/O-aware load-balancing schemes less effective.

The aggregate I/O access rate, Λ^i_{disk}, is defined as:

$$\Lambda^i_{disk} = \sum_{k \in M_i} \lambda'_k, \quad \text{where } \lambda'_k = \frac{\lambda_k}{E(L_i)}. \tag{3.10}$$

In Equation (3.10), M_i is a set containing all the jobs that are assigned to node *i*, and λ_k' is the effective I/O access rate imposed on the disk by job *k*, taking the effect of time-sharing into account. To accurately estimate the effective I/O access rate, λ_k, measured in a non-shared environment, must be deflated by the time-sharing factor, which is $E(L_i)$. Based on λ_k', the disk utilization can be expressed as: $\rho^i_{disk} = \sum_{k \in M_i} \lambda'_k s^k_{disk}$.

Let p^k_{disk} be the probability of an I/O access being from job *k* on node *i*, we then have $p^k_{disk} = \lambda'_k / \Lambda_i$. Therefore, the mean I/O service time, used in Equation (3.9), can be calculated as follows:

$$E(s_{disk}^{i}) = \sum_{k \in M_i} \left(p_{disk}^{k} s_{disk}^{k} \right) = \frac{1}{\Lambda_i} \sum_{k \in M_i} \left(\lambda_k' s_{disk}^{k} \right) = \frac{\rho_{disk}^{i}}{\Lambda_{disk}^{i}}, \text{ and}$$

$$E[(s_{disk}^{i})^2] = \sum_{k \in M_i} \left(p_{disk}^{k} (s_{disk}^{k})^2 \right) = \frac{1}{\Lambda_{disk}^{i}} \sum_{k \in M_i} \left(\lambda_k' (s_{disk}^{k})^2 \right). \quad (3.11)$$

Let p_{CPU}^{k} denote the probability of a job *k* being executed by the CPU or waiting in the CPU queue, as opposed to waiting for I/O access. We have:

$$p_{CPU}^{k} = t_k / (t_k + t_k \lambda_k s_{disk}^{k}) = 1/(1 + \lambda_k s_{disk}^{k}).$$

Thus, the mean CPU queue length, used in Equation (3.9) and (3.10), becomes:

$$E(L_i) = \sum_{k \in M_i} p_{CPU}^{k} = \sum_{k \in M_i} \frac{1}{1 + \lambda_k s_{disk}^{k}}. \quad (3.12)$$

We now turn our attention to the response time of a job with remote I/O accesses. The network links are modeled as a single M/G/1 queue [Riska and Smirni, 2002]. Let *r(i,k,j)* be the response time of job *j* on node *i* remotely accessing data on node *k (k≠i)*. Thus we have:

$$r(i,k,j) = t_j E(L_i) + t_j \lambda_j \left[E(s_{disk}^{k}) + \frac{\Lambda_{disk}^{k} E[(s_{disk}^{k})^2]}{2(1 - \rho_{disk}^{k})} \right] + t_j \lambda_j \left[E(s_{net}^{ik}) + \frac{\Lambda_{net}^{ik} E[(s_{net}^{ik})^2]}{2(1 - \rho_{net}^{ik})} \right] \quad (3.13)$$

where $E(s_{net}^{ik})$ *and* $E[(s_{net}^{i})^2]$ are the mean and mean-square network service time, ρ_{net}^{ik} denotes the utilization of the network link, and Λ_{net}^{ik} is the aggregate communication request rate. The three terms on the right hand side of Equation (3.13) rerepresent CPU execution time, I/O processing time, and network communication time, respectively. For simplicity, we assume that, for a remote I/O operation, there is no overlap in time between I/O processing and communication [Surdeanu et al., 2002]. This simplification is conservative in nature, thereby making the proposed schemes less effective.

3.2.4 Migration Cost

Given a job *j* submitted to node *i*, the expected migration cost is estimated as follows,

$$c_j = \begin{cases} e & \text{if } M_{CM}(j)=k\neq i \text{ and } M_{IO}(j) = i, \\ d_j^{INIT}\left(\frac{1}{b_{net}^{il}}+\frac{1}{b_{disk}^{i}}+\frac{1}{b_{disk}^{l}}\right) & \text{if } M_{CM}(j) = i \text{ and } M_{IO}(j)=l \neq i, \\ e + d_j^{INIT}\left(\frac{1}{b_{net}^{il}}+\frac{1}{b_{disk}^{i}}+\frac{1}{b_{disk}^{l}}\right) & \text{if } M_{CM}(j)=k\neq i \text{ and } M_{IO}(j)=l \neq i. \end{cases} \quad (3.14)$$

where *e* is the fixed cost of migrating the job, b_{net}^{kl} is the available bandwidth of the link between node *k* and *l*, b_{disk}^{k} is the available disk bandwidth in node *k*. In practice, b_{net}^{kl} and b_{disk}^{k} can be measured by a performance monitor [Basney and Livny, 2000]. d_j^{INIT} represents the amount of data initially stored on disk to be processed by job *j*, and this amount of data is referred to as *initial data* throughout this paper. Thus, the second line of Equation (3.14) represents the migration time spent on transmitting data over the network and on accessing source and destination disks. In real world applications, there is no need to migrate all the initial data, since some data will be only read once. However, IOCM might not be able to know which portion of initial data will be read only once. We assume that the initial data of a job is transmitted if the job encounters a migration. This assumption is conservative, since IOCM can be further improved if the amount of initial data that must be migrated can be accurately predicted by a monitor at run time.

In Step (3), IOCM attempts to guarantee that the response time of the candidate migrant is less in expectation than it would be without migration. This guarantee is implemented by checking the following criterion based on Equation (3.9), (3.13), and (3.14).

$$r(i,j) > \begin{cases} r(k,i,j) + c_j, & \text{if } M_{CM}(j)=k\neq i \text{ and } M_{IO}(j)=i, \\ r(i,l,j) + c_j, & \text{if } M_{CM}(j)=i \text{ and } M_{IO}(j)=l\neq i, \\ r(k,l,j) + c_j, & \text{if } M_{CM}(j)=k\neq i \text{ and } M_{IO}(j)=l\neq i,\ k\neq l, \\ r(k,j) + c_j, & \text{if } M_{CM}(j)=k\neq i \text{ and } M_{IO}(j)=k\neq i. \end{cases} \quad (3.15)$$

Four migration cases in Expression (3.15) are:

(1) the job is migrated without its I/O portion;

(2) the job is executed locally, while its I/O portion is serviced in another node;

(3) both the job and its I/O portion are migrated, but to two different nodes; and

(4) both the job and its I/O portion are migrated to the same node, while I/O operations can still be processed locally in another node.

3.2.5 Load-Balancing with Preemptive Job Migrations

We are now in a position to study the Weighted-Average-Load based policy with Preemptive job Migrations (*WAL-PM*) that improves the performance by considering not only incoming jobs but also currently running jobs.

For a newly arrived job *j* at a node *i*, WAL-PM balances the load in the following six steps. First, the load of node *i* is updated by adding *j*'s load, assigning the newborn job to the local node. Second, a migration is to be initiated, if node *i* has the highest load and the ratio between node *i*'s load and the system's average load is greater than 1.25. Third, a candidate node *k* with the lowest load is chosen. If a candidate node is not available, WAL-PM will be terminated and no migration will be carried out. Fourth, WAL-PM determines a set *EM* of jobs eligible for migration such that the migration of each job in *EM* is able to potentially reduce the slowdown of the job. Fifth, a job *q* from *EM* is judiciously selected in such a way that the migration benefit is maximized. In fact, this step substantially improves the performance over the WAL-based load-balancing scheme with non-preemptive migration. Sixth, finally, job *q* is migrated to the remote node *k*, and the load of

nodes *i* and *k* is updated in accordance with job *q*'s load. An outline of the WAL-PM scheme is presented in Figure 3.3.

WAL-PM(**Input:** Job *j*, Node *i*)
Step1: assign job *j* to node *i*, and add the load of job *j* into the load of node *i*;
Step 2: **if** the weighted average load index indicates that node *i* is overloaded **then**
Step 2.1 select a candidate node *k* with the smallest value of load;
Step 2.2 **if** a candidate node is not available **then**
preemptive migration is terminated
Step 2.3 Determine a set of jobs *EM(i, k)*, in which jobs have been assigned to node *i* and are eligible for migration;
Step 2.4 **if** the set *EM* is not empty **then**
select a job *q* in *EM(i, k)* that gains a maximal benefit from migration;
migrate job *q* from node *i* to node *k*;

Figure 3.3 Pseudo code of the Weighted-Average-Load based policy with preemptive job migrations

WAL-PM estimates the weighted average load index in Step (1). Since there are three primary resources considered, the load index of each node *i* is the weighted average of CPU, memory and I/O load, thus:

$$load_{WAL}(i) = W_{CPU}\left(\frac{load_{CPU}(i)}{MAX_{j=1}^{n} load_{CPU}(j)}\right) + W_{MEM}\left(\frac{load_{MEM}(i)}{MAX_{j=1}^{n} load_{MEM}(j)}\right) + W_{IO}\left(\frac{load_{IO}(i)}{MAX_{j=1}^{n} load_{IO}(j)}\right), \quad (3.16)$$

where $load_{CPU}(i)$ is the length of the CPU waiting queue, $load_{MEM}(i)$ is expressed in Equation (3.1), and the derivation of $load_{IO}(i)$ can be found in Equation (3.2). Each weight indicating the significance of one resource is automatically configured by the load monitor. In practice, three weights are measured by the percentage of time spent on CPU, paging, and I/O accessing, respectively.

3.2.6 Selection of the Best Candidate Migrant

The WAL-PM scheme judiciously selects an eligible job in *EM* from the overloaded node to migrate, and the expected response time of an eligible migrant on the source node, by design, is greater than the sum of its expected response time on the destination node and the migration cost. In what follows, the expected response time of a candidate migrant *j* on node *i* is given in the following equation, where a_j is the age of job *j* and other variables assume the same meanings as in (3.9) and (3.13).

$$r_{PM}(i,j) = (t_j - a_j)E(L_i) + (t_j - a_j)\lambda_j \left[E(s^i_{disk}) + \frac{\Lambda^i_{disk} E[(s^i_{disk})^2]}{2(1-\rho^i_{disk})} \right]. \qquad (3.17)$$

Based on Equation (3.17), the set of eligible migrant jobs becomes:

$$EM(i,k) = \left\{ j \in M_i \,\middle|\, r_{PM}(i,j) > r_{PM}(k,j) + c_j \right\}, \qquad (3.18)$$

where *k* represents a destination node, and c_j is the migration cost (time) of job *j*. In other words, each eligible migrant's expected response time on the source node is greater than the sum of its expected response time on the destination node and the expected migration cost. This is modeled as follows

$$c_j = \begin{cases} e + d_j^{INIT}\left(\frac{1}{b^{ik}_{net}} + \frac{1}{b^i_{disk}} + \frac{1}{b^k_{disk}}\right) & \text{for remote execution,} \\ f + \frac{m_j}{b_{net}} + \left(d_j^{INIT} + d_j^W\right)\left(\frac{1}{b^{ik}_{net}} + \frac{1}{b^i_{disk}} + \frac{1}{b^k_{disk}}\right) & \text{for preemptive migration,} \end{cases} \qquad (3.19)$$

where *f* is the fixed cost for preemptive migration and m_j is the memory size of the migrant job. Like equation (3.14), the last three terms of both the upper and the bottom line of Equation (3.19) represent the migration time spent on transmitting data over the network and on accessing source and destination disks, respectively. The second term of the bottom

line of Equation (3.19) is the memory transfer cost. d_j^W and m_j in Equation (3.19) denote the amount of disk (I/O) data and of main memory data generated at the runtime by the job, respectively. Disk data d_j^W is proportional to the number of write operations that has been issued by the job at the runtime and the average amount of data d_j^{RW} stored by the write operations. d_j^W is inversely proportional to the data re-access rate r_j. Thus, d_j^W is defined by:

$$d_j^W = a_j \lambda_j w_j d_j^{RW} / (r_j + 1), \tag{3.20}$$

where w_j is the percentage of I/O operations that store data to the local disk, and the number of write operations is a product of a_j, λ_j, and w_j in the numerator. In some I/O-intensive applications, numerous snapshots are spawned by write-only operations. Since the permanent data of snapshots will not be read again by the same job, there is no need to move such write-only data when the job is migrated. Hence, w_j in Expression (3.20) does not consider write-only operations.

In Step (5), WAL-PM chooses one job j from set *EM(i, k)* in such a way that the benefit of migration is maximized. To find a maximizing factor, we define an objective function, called the *migration cost-effectiveness (MCE)*, which measures the amount of I/O load migrated per unit migration cost. More specifically, for job j, we have:

$$MCE(j) = a_j \lambda_j / c_j ,$$

where the numerator represents the I/O load of job j while the denominator indicates migration cost of the job. Thus, the best job in *EM* to choose for migration is the one with the maximum *MCE* value, as shown in Equation (3.21),

$$MCE(j) = \underset{p \in EM(i,k)}{MAX} \{MCE(p)\}, \text{ where } j \in EM(i,k) \tag{3.21}$$

3.3 The Simulator and Simulation Parameters

The performance metric used in our simulations is *slowdown* [Harchol-Balter and Downey, 1997][Zhang et al., 2000]. The slowdown of a job, which reflects the performance degradation of the job due to resource sharing with other jobs and migration overhead, is defined as the ratio between the job's execution time in a resource-shared setting and its execution time running in the same system but without any resource sharing.

Since the definition of slowdown in [Harchol-Balter and Downey, 1997][Zhang et al., 2000] does not consider time spent on I/O accesses, it has to be extended by incorporating I/O access time. The definition of slowdown for a job *j* is given as:

$$slowdown(j) = T_{wall}(j) / \left[T_{CPU}(j) + T_{IO}(j) \right], \tag{3.22}$$

where $T_{wall}(j)$ is the total time the job spends executing, accessing I/O, waiting, or migrating in a resource-shared setting, and $T_{CPU}(j)$ and $T_{IO}(j)$ are the times spent by *j* on CPU and IO, respectively, without any resource sharing.

To study dynamic load balancing, Harchol-Balter and Downey implemented a trace-driven simulator for a distributed system with 6 nodes in which round-robin scheduling is employed [Harchol-Balter and Downey, 1997]. The load balancing policy studied in that simulator is CPU-based. Six daytime intervals from the traces were selected, and the start times in addition to CPU durations of the processes were extracted. The job arrival process of the traces is burstier than a Poisson process, and the serial correlation in interarrival times is higher than one would expect from a Poisson process [Harchol-Balter and Downey, 1997].

Zhang et. al extended the simulator, incorporating memory resources into the simulation system [Zhang et al., 2000]. Based on the simulator presented in [Zhang et al., 2000], our simulator incorporates three new features:

(1) The IOCM, IO-RE, IO_PM, WAL-RE and WAL-PM schemes are implemented;

(2) a fully connected network is simulated;

(3) a simple disk model is added; and

(4) an I/O buffer is implemented.

Table 3.2 Data Characteristics.

Parameter	Value (Fixed) (Varied)
CPU Speed	(800 million instructions/second or MIPS) -
RAM Size	(640Mbytes) -
Buffer Size	(160Mbytes) -
Network Bandwidth(point to point)	(100Mbps) (50, 100, 250, 500, 750, 1000 Mbps)
Page fault service time	(8.1 ms) -
Page fault rate	(0.625 No./Million Instructions or No./MI) – (9.0, 9.25, …, 11.0 No./MI)
Time slice of CPU time sharing	(10 ms) -
Context switch time	(0.1 ms) -
Disk seek time and rotation time	(8.0 ms) -
Disk transfer rate	(40 MB/s) -
AR (Maximal I/O access rate)	(0.125No./MI) - (3.0, 3.125, …, 3.625No./MI)
I/O access rate	Uniformly distributed between 0 and AR
Data Re-access rate, *r*	(5) - (1, 2, 3, 4)
Average Initial Data Size	(60MB) – (50, 100, 150, 200, 250, 300, 350 MB)

In all experiments, the simulated system is configured with parameters listed in table 3.2. Each measurement is repeatedly made 10 times and the average value is computed after discarding the highest and lowest values. Note that this method applies to all the experiments presented in the subsequent chapters.

The average page fault rate caused by all jobs on a node is measured by the number of page faults per million instructions (No./MI) under circumstance where the allocated memory space equals the memory load [Xiao et al., 2002][Zhang et al., 2000]. Note that this is a system independent way to measure page fault rates. Similarly, the disk I/O access rate of a job is measured by the number of disk I/O requests issued by the job per million instructions (No./MI).

Disk accesses from each job are modeled as a Poisson Process with a mean I/O access rate λ. Although the durations and memory requirements of the jobs come from trace data,

the I/O access rate of each job is randomly generated according to a uniform distribution between 0 and AR, where AR represents the maximal I/O access rate. This simplification deflates any correlations between I/O requirement and other job characteristics, but we are able to control the I/O access rate as a parameter and examine its impact on system performance. Data sizes of the I/O requests are randomly generated based on a Gamma distribution with the mean size of 256KByte and the standard deviation of 128Kbyte. The sizes chosen in this way reflect typical data characteristics for many data-intensive applications, such as a fine-grained regional atmospheric climate model [Roads, et al., 1992] and an ocean global circulation model [CASA, 1991], where the vast majority of I/O requests are small [Kotz and Nieuwejaar, 1994][Pasquale and Polyzos, 1994].

3.4 Experimental Results

In this section we compare the performance of IOCM, IO-PM, and WAL-PM with four existing schemes, namely, the CPU-Memory-based load balancing schemes with remote execution and preemptive migration (CM-RE and CM-PM), the I/O-based load balancing with remote execution (IO-RE), and the Weighted-Average-Load-based balancing with remote execution (WAL-RE). In what follows, we give a brief description of these policies.

(1) CPU-Memory-based load balancing schemes CM-RE (with remote execution) and CM_PM (with preemptive migration) were introduced in [Zhang et al., 2000]. When a node has sufficient memory space, the CM schemes balance the system using CPU load index. When the system encounters a large number of page faults due to insufficient memory space for the running jobs, memory load index $load_{MEM}(i)$ is used by CM to balance the system.

(2) The I/O-based load balancing with remote execution (IO-RE) [Lee et al., 2000] uses a load index that represents only the I/O load. For a job arriving in node *i*, the IO scheme

greedily assigns the computational and I/O tasks of the job to the node that has the least accumulated I/O load.

(3) The Weighted-Average-Load-based balancing with remote execution (WAL-RE), proposed in [Surdeanu et al., 2002], assigns jobs to a node that is not overloaded. If such a node is not available, WAL-RE dispatches the job to a node with the smallest value of the load index.

3.4.1 Performance on I/O-Intensive Workload Conditions

To stress the synthetic I/O-intensive workload in this experiment, the page fault rate is fixed at a relatively low value of 0.625 No./Million Instructions (No./MI). This workload reflects a scenario where memory-intensive jobs exhibit high temporal and spatial locality of access.

A realistic system is likely to have a mixed workload, where some jobs are I/O-intensive and other jobs are either CPU or memory intensive. Therefore, we randomly choose 10% of jobs from the traces to be non-I/O-intensive by setting their I/O access rate to be 0 No./MI. Among these non-I/O-intensive jobs, 50% of jobs are made to be CPU-intensive by scaling their execution time by a factor of 10, and other jobs are modified to be memory-intensive with page fault rate set to 10 No./MI.

Figure 3.4 plots slowdown as a function of the maximal I/O access rate in the range between 3 No./Million Instructions (No./MI) and 3.625 No./MI with increments of 0.125 No./MI. The mean slowdowns of IO-RE, IO_PM, and CM-RE are almost identical to those of WAL-RE, WAL-PM, and CM_PM respectively, and therefore are omitted from Figure 3.4. Four observations were made from this experiment.

First, Figure 3.4 reveals that the mean slowdowns of the five policies all increase with the I/O load. This is because as CPU load and memory demands are fixed, high I/O load leads to a high utilization of disks, causing longer waiting time on I/O processing.

Second, the results show that the WAL_RE scheme significantly outperforms the CM-RE and CM_PM policies, suggesting that the CM-RE and CM-PM policies are not suitable for I/O intensive workloads. This is because CM-RE and CM-PM only balance CPU and memory load, ignoring the imbalanced I/O load under the I/O intensive workloads.

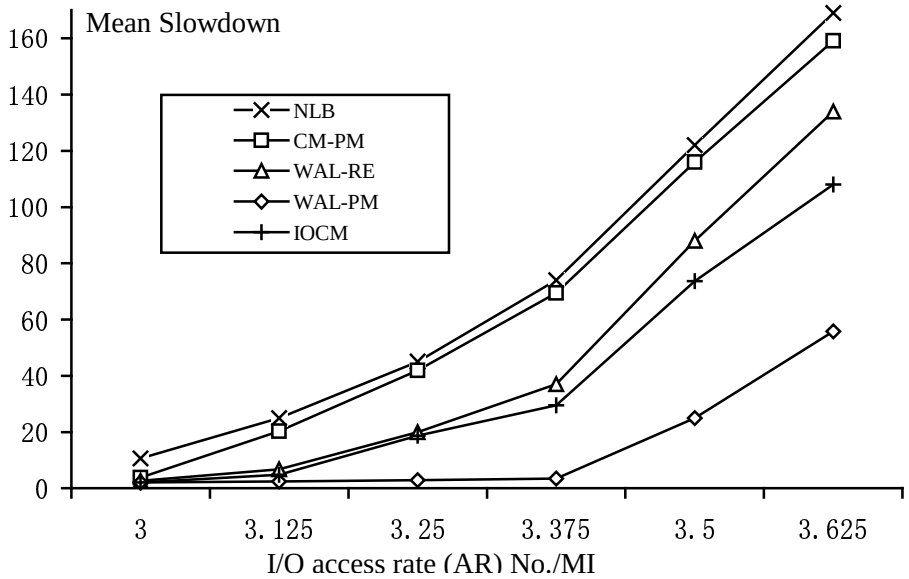


Figure 3.4 Mean slowdown as a function of I/O access rate, on six traces with a page fault rate of 0.625 No./MI

Third, the results further reveal that the IOCM scheme outperforms CM-RE, CM-PM, IO-RE, and WAL-RE. This is because IOCM partitions each job into a computational task and an I/O task, and individually improves the utilizations of three resources by allowing the computational and I/O tasks of each job to be assigned to different nodes.

Finally, the proposed WAL-PM policy improves the performance even over WAL-RE by virtue of preemptive migration strategy, suggesting that preemptive migration outperforms remote execution for I/O-based schemes under an I/O-intensive workload. Consequently, the slowdowns of CM-RE, CM-PM and WAL-RE are more sensitive to I/O

access rate than WAL-PM. This performance improvement of WAL-PM over WAL-RE can be explained by the following reasons. First, one problem encountered in the WAL-RE policy is that the I/O demand of a newly arrived job may not be high enough to offset the migration overhead. However, WAL-PM provides better migratory opportunities by considering all existing jobs on a node, in addition to the newly arrived job. Second, unlike the preemptive scheme, in the non-preemptive scheme, once a job with high I/O demand misses the opportunity to migrate it will never have a second chance.

3.4.2 Performance on CPU-Memory Intensive Workload Conditions

This section shows the worst-case scenario for IOCM and WAL-PM, namely, subjecting them to a synthetic CPU-memory-intensive workload. To simulate a memory intensive workload, the maximal I/O access rate is fixed at a low value of 0.125 No./MI, keeping the I/O demands of all jobs at a very low level.

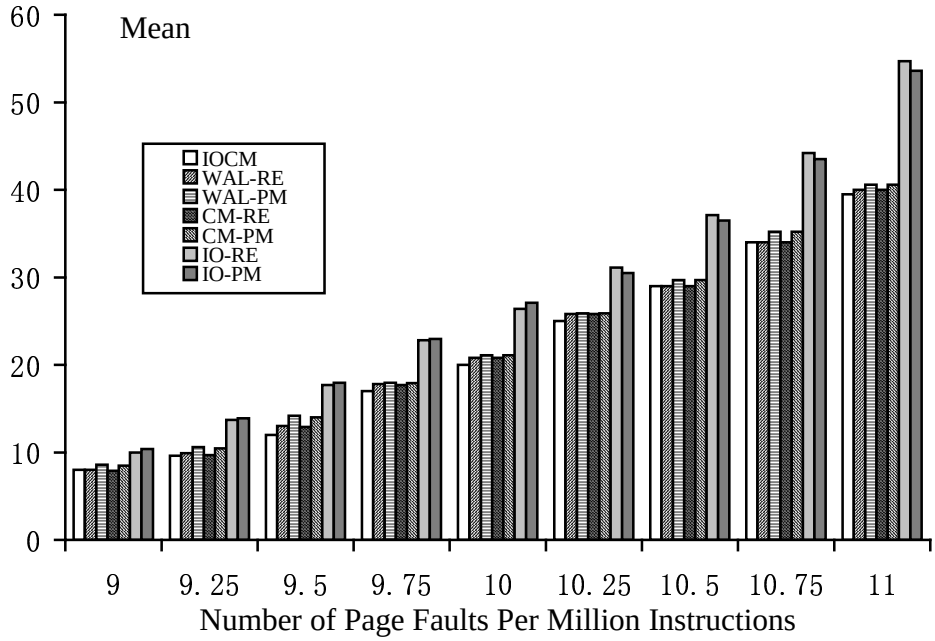


Figure 3.5 Mean slowdown as a function of page fault rate, on nine traces with an I/O access rate of 0.125 No./MI

Again, to simulate a mixed workload, we randomly choose 10 percent jobs to be I/O intensive by setting their I/O access rate to 3.5 No./MI. In [Zhang et al., 2000], the page fault-rate is scaled according to the upper limit vertical axis presenting the mean slowdown. Similarly, we set the upper limit of the mean slowdown at 60, and scale the page fault rate from 9.0 No./Million Instructions (No./MI) in increments of 0.25 No./MI. In practice, the page fault rates of applications range from 1 to 10 No./MI. [Zhang et al., 2000].

The results of the mean slowdown as a function of the page fault rate are summarized in Figure 3.5. The general observations in this experiment are in agreement with [Zhang et al., 2000], where the impact of page fault rate on the mean slowdown is quantitatively evaluated. Therefore, we only present new results about the proposed schemes and their comparison with the existing schemes.

As can be seen in Figure 3.5, when page fault rate is higher and I/O rate is very low, IOCM, CM-RM, CM-PM, WAL-RE, and WAL-PM outperform the IO-RM and IO-PM schemes considerably. These results can be explained by the following reasons. First, IOCM, CM-RE, CM-PM, WAL-RE, and WAL-PM consider the effective usage of global memory, attempting to balance implicit I/O load, which makes the most significant contribution to the overall system load when page fault rate is high and explicit I/O load is low. Second, the IO-RE and IO-PM schemes improve the utilization of disks based only on explicit I/O load, ignoring implicit I/O load resulted from page faults. Again, Figure 3.5 illustrates that IOCM outperforms CM-RE, CM-PM, and WAL-RE, by up to 18.3%. The reason for this phenomenon is that besides balancing memory load and implicit I/O load generated by page faults, IOCM further balances explicit I/O load measured by I/O access rates.

Figure 3.5 shows that the mean slowdowns of WAL-RE and WAL-PM are nearly identical to those of the CM-RE and CM-PM policies. Since the weighted load index is dynamically configured in accordance with the CPU-memory intensive workload, the WAL-RE and WAL-PM policies gracefully reduce to CM-RE and CM_PM, respectively.

As expected, CM_PM and WAL-PM consistently perform slightly worse than CM-RE and WAL-RE because remote execution results in lower data movement cost during migration than that of preemptive strategy for memory-intensive jobs. This experiment is consistent with the results reported in [Zhang et al., 2000]. However, the opposite is true for I/O-based policies when memory demand is comparatively high. The explanation is that high memory demand implies an I/O-intensive workload due to a large number of page faults, and a preemptive strategy is effective for an I/O-intensive workload.

3.4.3 Impact of Varying Initial Data Size

The migration cost of non-preemptive and preemptive policies depend in part on the size of initial data. Figure 3.6 shows the impact of initial data size on slowdowns under I/O-intensive workloads. We only present the results of WAL-RE, WAL-PM, and IOCM, since the performance patterns of other policies are similar to these three policies.

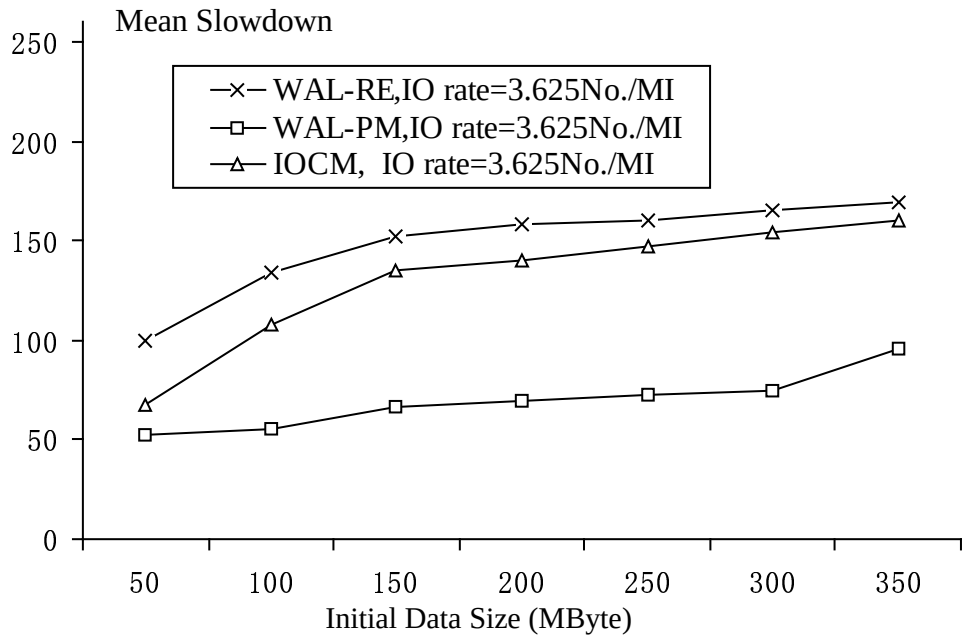


Figure 3.6 Mean slowdown as a function of the size of initial data, on seven traces with a page fault rate of 0.625 No./MI

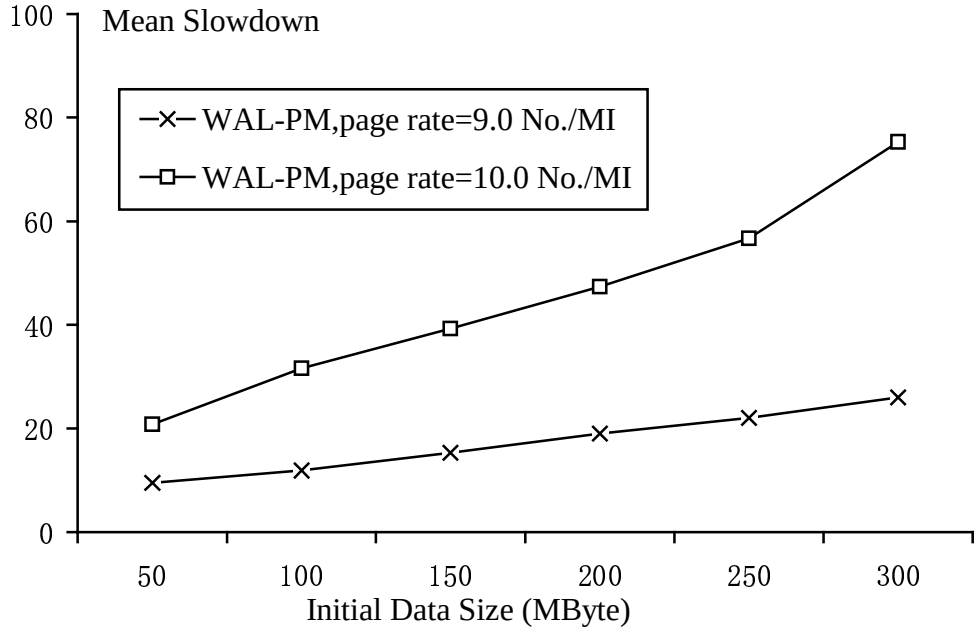


Figure 3.7 Mean slowdown of WAL-PM as a function of the size of initial data, on seven traces with an I/O access rate of 0.125 No./MI

First, Figure 3.6 shows that the IOCM and WAL-PM policies consistently outperform the WAL-RE policy. Second, the slowdowns increase with the increasing size of the initial data size. The reason is that the large initial data size results in a high migration time, which in turn reduces benefits gained from migrations. Third, it is observed that the slowdowns of WAL-RE and IOCM are much more sensitive to the initial data size than that of WAL-PM. This result indicates that the performance gain by WAL-PM over existing policies becomes more pronounced when the initial data size is large.

Figure 3.7 illustrates the impact of the initial data size on the performance of WAL-PM under a memory-intensive workload. An observation is that the sensitivity of WAL-PM to the initial data size is heightened by increasing the page fault rate. This is because migrating the initial data of a job involves two disk accesses, reading initial data from the

source disk and writing it to the target disk. The average response times at the source and the target disks are likely to be long when the I/O load of two disks is heavy due to the high page fault rate, thereby leading to a large overhead of migrating initial data. This result suggests that the slowdown of the WAL-PM policy does not suffer significantly from a large initial data size when the page fault rate is low. Likewise, WAL-PM can tolerate a relatively high page fault rate if the initial data size is small.

3.4.4 Impact of Varying Average Data Size

I/O load depends on I/O access rate and the average data size of each I/O access, which in turn depends on I/O access patterns. The purpose of this experiment, therefore, is to show the impact of average data size on the performance of load balancing policies.

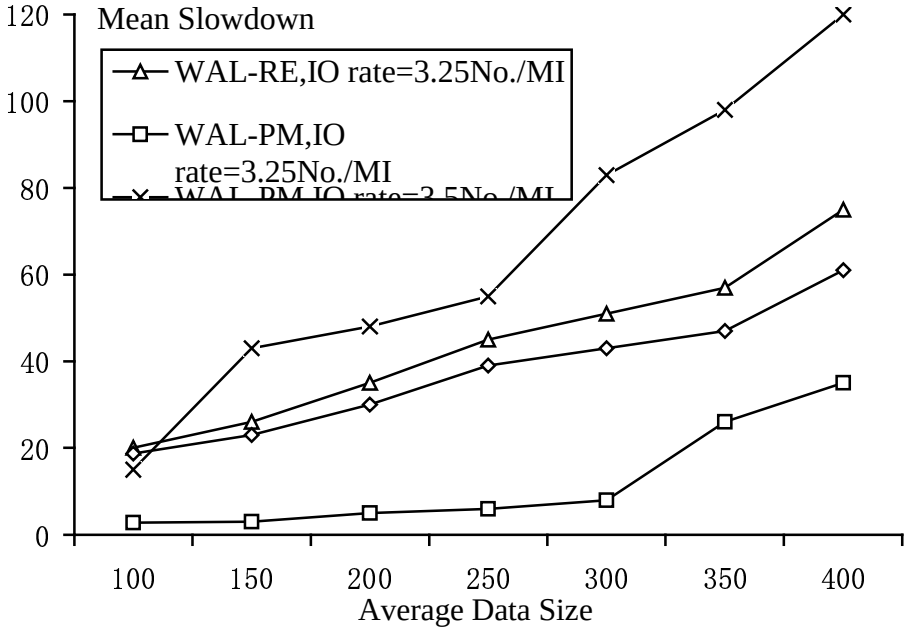


Figure 3.8 Mean slowdown as a function of the size of average data size. Page fault rate is 0.625 No./MI, and I/O rate is 3.25 No./MI

Figure 3.8 shows that the mean slowdown increases as the average data size of each I/O access increases. The reason is that, as I/O access rate is fixed, a large average data size yields a high utilization of disks, causing longer waiting times on I/O processing. A second observation from Figure 3.8 is that the slowdown performance of WAL-PM at a high I/O access rate is more sensitive to average data size than that at a low I/O access rate. This is because the higher the I/O access rate, the higher the disk utilization, which results in longer waiting time in disk queue.

3.4.5 Impact of Varying Network Bandwidth

Let us now consider the impact of network bandwidth on the mean slowdowns of the CM_PM, WAL-RE, IOCM, and WAL-PM policies. In order to explore this issue, we set the network bandwidth in the range between 50Mbps and 1Gbps.

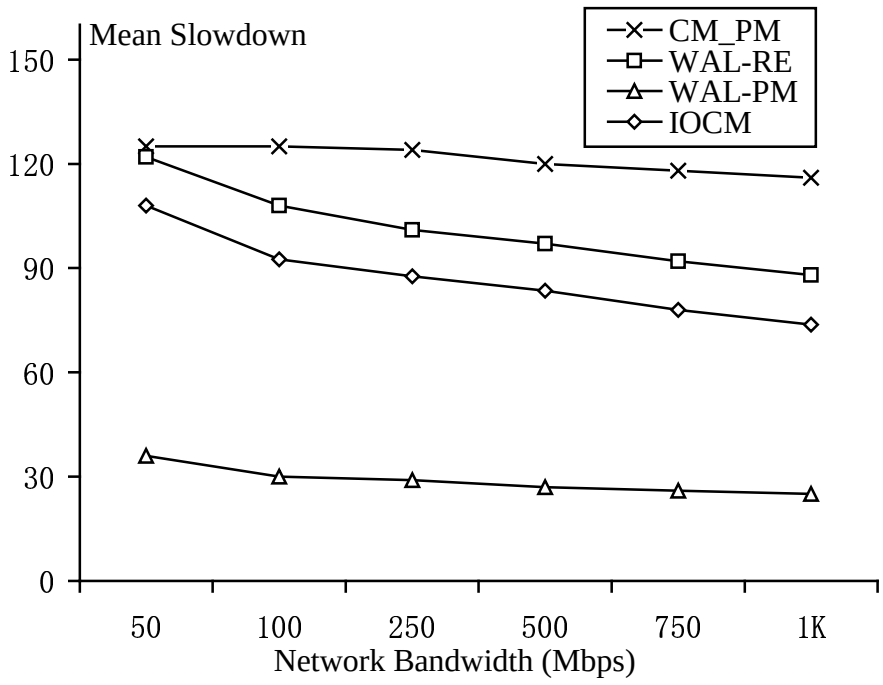


Figure 3.9 Mean slowdown as a function of the network bandwidth. Page fault rate is 0.625 No./MI, and I/O rate is 3.5 No./MI

Since the performance of CM-RE, IO-RE, and IO_PM are almost identical to those of CM_PM, WAL-RE, and WAL-PM, respectively, Figure 3.9 only shows the results for the CM_PM, WAL-RE, IOCM, and WAL-PM policies.

As shown in Figure 3.9, CM_PM is not sensitive to network bandwidth. Because when the workload is I/O intensive, CM_PM degrades to no-load-balancing policy that is not affected by the network speed.

The slowdowns of the other four policies share a common feature in the sense that when the network bandwidth increases, the slowdown slightly drops. The reason is that a network with high bandwidth results in a low migration cost in these four load-balancing policies.

Figure 3.9 further reveals that WAL-RE and IOCM are more sensitive to the network bandwidth than WAL-PM. This result can be explained by the fact that the preemptive schemes tend to select a job with the minimal migration cost and reduce the network traffic overhead, thereby deflecting the impact of network speed on the performance of the preemptive migration schemes.

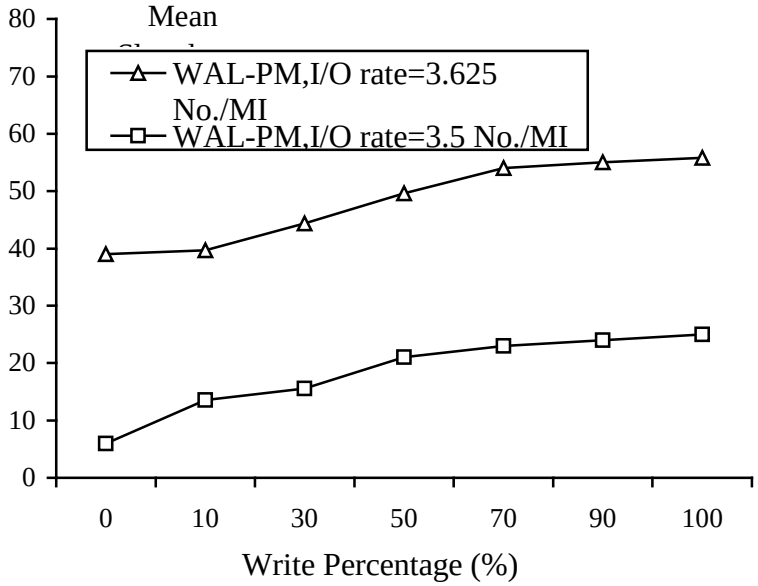


Figure 3.10 Impact of write percentage on mean slowdowns. Page fault rate is 0.625 No./MI.

3.4.6 Impacts of Varying Write Percentage and Re-access Rate

Figure 3.10 and Figure 3.11 illustrate the mean slowdowns of WAL-PM as functions of write percentage and re-access rate, respectively.

We observe from Figure 3.10 that the increase in the write percentage worsens the performance. The reason is that a high write percentage means a large amount of data to be migrated (See Equation (3.20)), implying a high migration cost as given in Equation (3.19). Consequently, the high migration cost yields a high slowdown.

Figure 3.11 indicates that the mean slowdowns decrease as the value of re-access rate increases. This is because if the I/O access rate of a job is fixed, increasing re-access rate implies a smaller amount of data stored in disk for the job, and thus a smaller amount of migrated data. As mentioned earlier, the migration cost is proportional to the amount of migrated data, and reducing the amount of migrated data results in a reduced cost for migration.

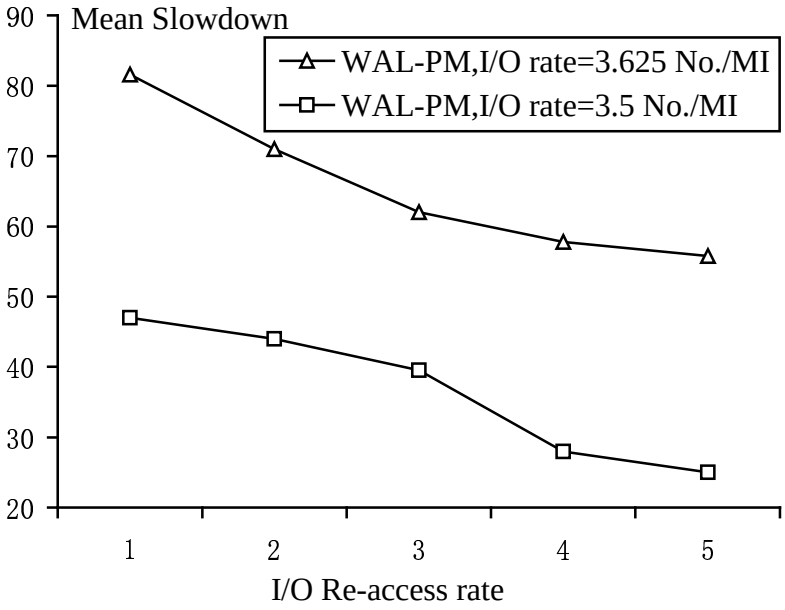


Figure 3.11 Impact of I/O re-access rate on mean slowdowns. Page fault rate is 0.625 No./MI.

The results strongly suggest that the overall performance depends on I/O access patterns. Thus, I/O intensive jobs in which either the I/O operations are dominated by read, or data are most likely to be re-accessed, can potentially benefit more from the WAL-PM scheme.

3.5 Summary

In this chapter we have proposed two dynamic disk-I/O-aware load-balancing policies, referred to as *IOCM* (load balancing for I/O, CPU, and Memory) and *WAL-PM* (Weighted-Average-Load based policy with Preemptive Migration), for cluster systems.

IOCM employs remote I/O execution facilities to improve system performance, whereas WAL-PM utilizes a preemptive job migration strategy to boost performance. More specifically, IOCM allows applications and their data to be located on different nodes, if well-balanced I/O load is able to offset the communication overhead imposed by remote I/O execution. WAL-PM considers not only the newly arrived jobs but also older, currently running jobs as candidate migrant jobs, and migrates jobs that are the most migration cost-effective. In addition to CPU and memory utilization, both IOCM and WAL-PM consider I/O load, leading to a performance improvement over existing I/O-based and CPU-Memory-based policies under the I/O-intensive workload.

We compare IOCM and WAL-PM with four existing approaches, namely, (1) CPU-Memory-based policy with preemptive migration *(CM-PM)*, (2) CPU-Memory-based policy with non-preemptive migration *(CM-RE)*, (3) I/O-based policy with non-preemptive migration *(IO-RE)*, and (4) Weighted-Average-load based policy with non-preemptive migration *(WAL-RE)*. For comparison purposes, I/O-based policy with preemptive migration (IO_PM) is also simulated and compared with the proposed schemes. A trace-driven simulation demonstrates that applying IOCM and WAL-PM to clusters for I/O-intensive workloads is highly effective.

In particular, the proposed schemes improve performance with respect to mean slowdown over existing non-preemptive I/O-aware schemes by up to a factor of 10. On the other hand, when the workload is I/O intensive, our schemes achieve improvement in slowdown over existing CPU-Memory-based schemes by up to a factor of 20.

Chapter 4

Disk-I/O-aware Load Balancing for Parallel Applications

In the previous chapter, we have designed two load-balancing schemes for sequential jobs running on homogeneous clusters. However, it is often the case that applications executing on clusters are parallel in nature. This is especially true for large-scale scientific applications. To make the load-balancing policies presented in Chapter 3 more practical, we have to address the issue of load balancing for parallel data-intensive applications.

This chapter is also devoted to the problem of distributed dynamic load balancing among a cluster of homogeneous nodes connected by a high-speed network. Specifically, this chapter explores two simple yet effective I/O-aware load-balancing schemes for parallel jobs in homogeneous clusters. Using a set of real world parallel applications and synthetic parallel jobs with various disk-I/O characteristics, we show that the proposed schemes are capable of balancing the load of a cluster in such a way that CPU, memory, and I/O resources at each node can be simultaneously well utilized under a wide spectrum of workload.

This chapter is organized as follows. Section 4.1 proposes a disk-I/O-aware load-balancing policy with remote execution (IOCM-RE) for parallel jobs, and Section 4.2 evaluate the performance of the IOCM-RE scheme. In Section 4.3, a second load-balancing

policy with preemptive migrations (IOCM-PM) is developed and evaluated. Section 4.4 presents the empirical results from extensive simulations based on real world applications. Finally, Section 4.5 summarizes the main contributions of this chapter.

4.1 A load balancing policy with remote execution

In this section we present an IO-CPU-Memory based load-balancing policy (IOCM-RE), which is an effective dynamic I/O-aware load-balancing scheme with remote execution. Each parallel job consists of a number of tasks, and the terms process and task are used interchangeably throughout this paper. The tasks of a parallel job are assumed to synchronize with one another [Dusseau et al., 1996][Ryu and Hollingsworth, 2000] [Valiant, 1990]. Specifically, each task of a parallel job serially computes for some period of time, then a barrier is performed so that each task starts exchanging messages with other processes of the parallel job. Each task is described by its requirements for CPU, memory, and I/O, measured, respectively, by the total time spent on CPU, Mbytes, and number of disk accesses per ms. Each workstation serves several tasks in a time-sharing fashion so that the tasks can dynamically share the cluster resources. A detailed model of parallel applications is described in Section 7.2.1.

For a parallel job, arriving in its home workstation via the client services, the IOCM-RE scheme attempts to balance three different resources simultaneously in the following manner.

(1) When the I/O load of a node is greater than zero, tasks with high I/O and memory demands are likely to experience waiting time on disk I/O processing. To alleviate the problem of unevenly distributed I/O load, IOCM-RE selects a group of nodes with lighter I/O load. If there are a number of choices, the one with the smallest value of memory load will be chosen to break the tie. It is noted that in this study, the proposed load balancing schemes utilize an I/O load index to quantitatively measure two types of I/O accesses: implicit I/O load (See Expression 3.3) induced by page faults and explicit I/O requests (See

Expression 3.4) resulting from tasks accessing disks. The I/O load index of node *i* has been defined by Equation 3.2 in Section 3.2.2.

The tasks of the parallel job are assigned to the selected remote nodes satisfying a criterion based on remote execution cost, in addition to load distribution. The criterion guarantees that the response time of the expected execution on the selected remote node is less than the local execution. Formally, the criterion is described as: $r(i, j) > r(k, j) + c_j(i, k)$, where $r(i, j)$ and $r(k, j)$ are the expected response times of task *j* on the local node *i* and on the remote node *k*, respectively, and $c_j(i, k)$ is the remote execution cost (i.e., migration cost). $r(i, j)$ can be obtained using Equation 3.9, and $c_j(i, k)$ is computed as:

$$c_j(i,k) = e + d_j^{INIT}\left(\frac{1}{b_{net}^{ik}} + \frac{1}{b_{disk}^{i}} + \frac{1}{b_{disk}^{k}}\right). \tag{4.1}$$

where *e* is the fixed cost of migrating the job and executing it on another node, b_{net}^{ik} denotes the available bandwidth of the network link between node *k* and *l*, b_{disk}^{i} is the available disk bandwidth in node *i*.

(2) If no I/O load is imposed on the node, the IOCM-RE scheme considers the node's memory load, defined as the sum of the memory space allocated to the tasks running on the node (See Expression 3.1). When the memory load exceeds the amount of available memory space, the IOCM-RE policy transfers the tasks of the newly arrived parallel job from the overloaded node to the remote nodes that are lightly loaded with respect to memory.

(3) If both the disk I/O and memory resources of the node are well balanced, IOCM-RE attempts to evenly distribute the CPU load. Specifically, if the node is overloaded in terms of CPU resource, the IOCM-RE policy transfers the tasks of the newly arrived job to the remote node with the lightest CPU load. Therefore, IOCM-RE is capable of resulting in a balanced CPU load distribution for systems under a CPU-intensive workload.

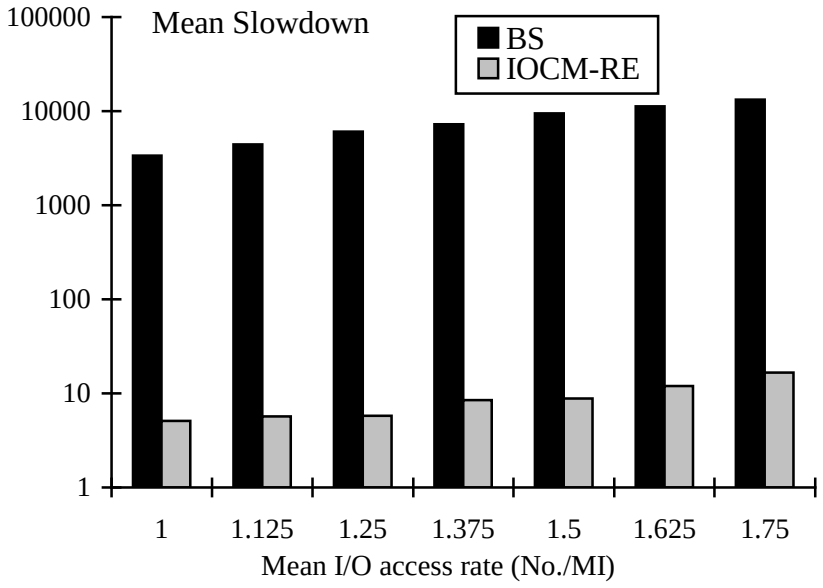
4.2 Evaluation of the IOCM-RE Scheme

To evaluate the performance of the disk-I/O-aware load-balancing scheme presented above, we have conducted a large number of trace-driven simulations based on a time-sharing cluster with 32 nodes. The workload we used is represented by trace files extrapolated from those reported in [Zhang et al., 2000]. These traces consist of the arrival time, arrival node, request memory size, and the actual running time. To simulate a multi-user time-sharing environment where a mixture of sequential and parallel jobs are running, the number of parallel jobs in each trace are chosen, respectively, to be 30% and 60% of the total number of jobs in the trace. The number of tasks in each parallel job is randomly generated according to a uniform distribution between 2 and 32. We simulate a bulk-synchronous style of communication, where processes concurrently compute during a computation phase, and then processes will be synchronized at a barrier so that messages can be exchanged among these processes during the communication phase [Dusseau et al., 1996] [Ryu and Hollingsworth, 2000][Valiant, 1990]. In our simulation, the time interval between two consecutive synchronization phases is 100 ms [Ryu and Hollingsworth, 2000].

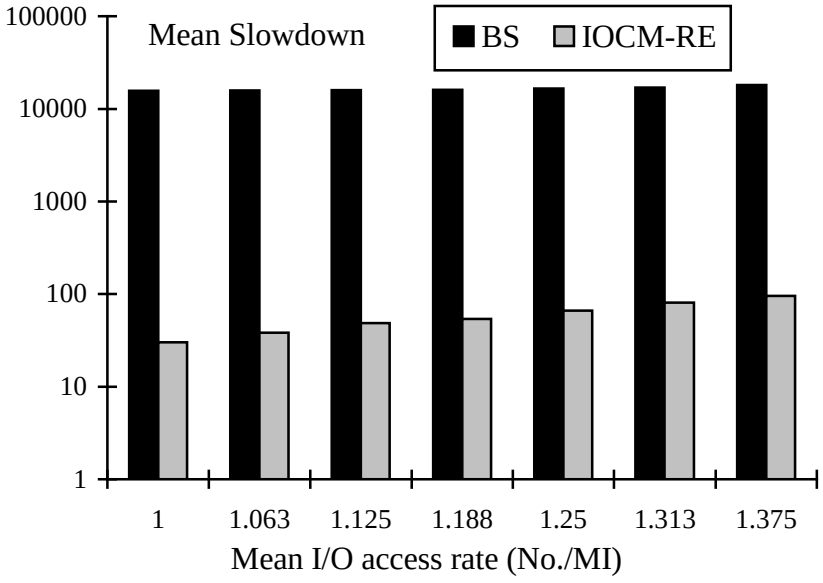
Similar to the experiment setting described in Section 3.3, data sizes of the I/O requests are randomly generated based on a Gamma distribution with the mean size of 256Kbyte. The performance metric used in this chapter is the mean *slowdown* (See Expression 3.22) of all the jobs in a trace.

4.2.1 IOCM-RE vs. a Centralized Load Balancing Approach

In this section, we compare IOCM-RE against a simple centralized load balancing approach used in a space-sharing cluster, where the nodes of the cluster are partitioned into disjoint sets and each set can only run one parallel job at a time. Since this approach is commonly used for batch systems [IBM, 2001], we term this load-balancing scheme as BS (Batch System), or PBS-like [Bode, 2000].



(a)



(b)

Figure 4.1 Mean slowdown vs. I/O access rate. Mean slowdown of parallel jobs running on a cluster of 32 nodes using the IOCM-RE and BS policies when (a) traces only contain sequential jobs, and (b) traces have 30% parallel jobs. For two graphs, page fault rate is 0.625 No./MI.

To stress the synthetic I/O-intensive workload in this experiment, the page fault rate is fixed at a relatively low value of 0.625 No./Million Instruction (No./MI). Figure 4.1 plots mean slowdown as a function of I/O access rate. While Figure 4.1(a) reports the results for seven traces that only contain sequential jobs, Figure 4.1(b) illustrates the mean slowdown of other seven traces where 30% of the jobs in each trace are parallel. Figures 4.1(a) and 4.1(b) indicate that both IOCM-RE and BS experience an increase in the mean slowdown when the I/O access rate increases. This is because in a scenario where CPU load and memory demands are fixed, high I/O load gives rise to a high utilization of disks, causing longer waiting time on I/O processing.

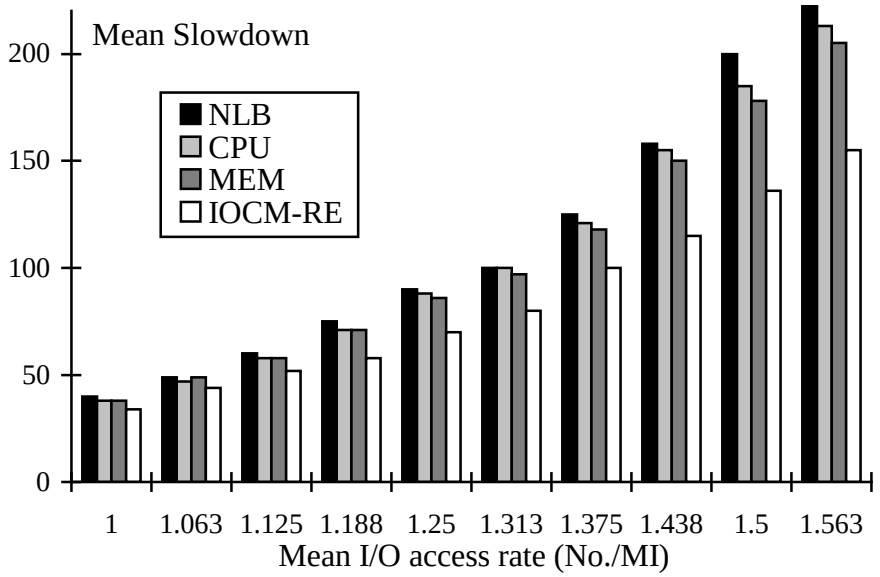
We observe from Figure 4.1 that under I/O-intensive workload situations, IOCM-RE is significantly better than BS. The main reason is that it is more difficult to utilize dedicated clusters as efficient, multiuser time-sharing platforms to execute I/O-intensive jobs. Figure 4.1(b) shows that the performance of I/O-intensive jobs drops considerably when a number of parallel jobs are waiting in the queue of the centralized node to be executed. This is because the synchronizations among processes of parallel jobs further decrease the utilization of resources in the system.

4.2.2 Performances under I/O-intensive Workload Conditions

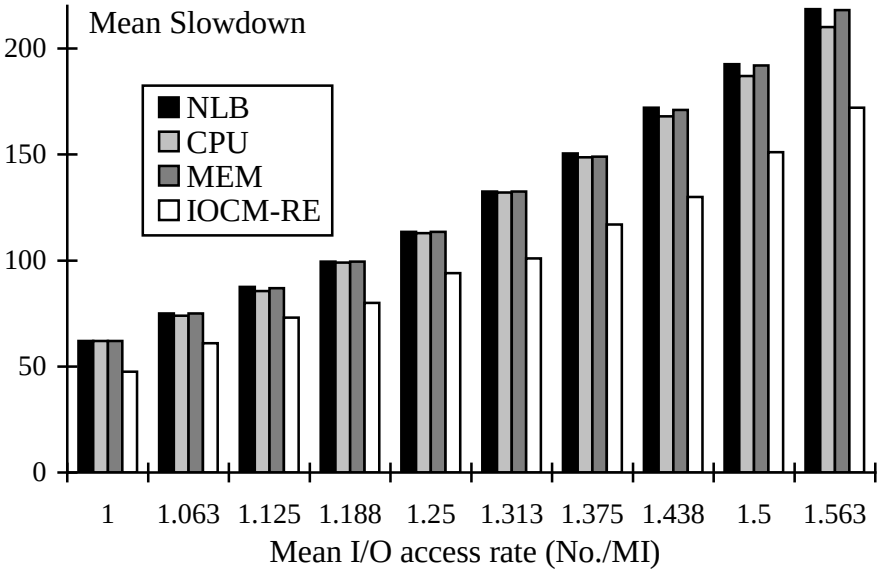
In what follows, we compare the performance of IOCM-RE with two existing schemes, namely, CPU-based (CPU) [Eager et al., 1986][Harchol-Balter and Downey, 1997] and memory-based (MEM) [Zhang et al., 2000] policies. For the purpose of comparison, we have also simulated a so-called NLB (Non-Load Balancing) policy that makes no effort to alleviate the problem of imbalanced load in any resource.

Figure 4.2 plots mean slowdown as a function of the mean I/O access rate in the range between 1.0 and 1.563 No./MI with increments of 0.0625 No./MI. First, Figure 4.2 reveals that the mean slowdowns of the four policies all increase with the increasing I/O load. Second, the results show that IOCM_RE greatly outperforms the CPU-based and memory-

based policies. The reason is because CPU-based and Memory-based policies only balance CPU and memory load, ignoring the imbalanced I/O load under the I/O intensive workload.



(a)



(b)

Figure 4.2 Compare IOCM-RE with the CPU-based and memory-based policies under I/O-intensive workload conditions when (a) traces have 30% parallel jobs, and (b) traces have 60% parallel jobs. For two graphs, page fault rate is 0.625 No./MI.

After comparing Figure 4.2(a) with Figure 4.2(b), we realize that if the mean I/O access rate is fixed, the mean slowdowns of the four policies all increase with the percentage of parallel jobs. This is because a higher percentage of parallel jobs leads to more tasks that are concurrently executed, causing more synchronization overhead, and competing with the resources of the cluster.

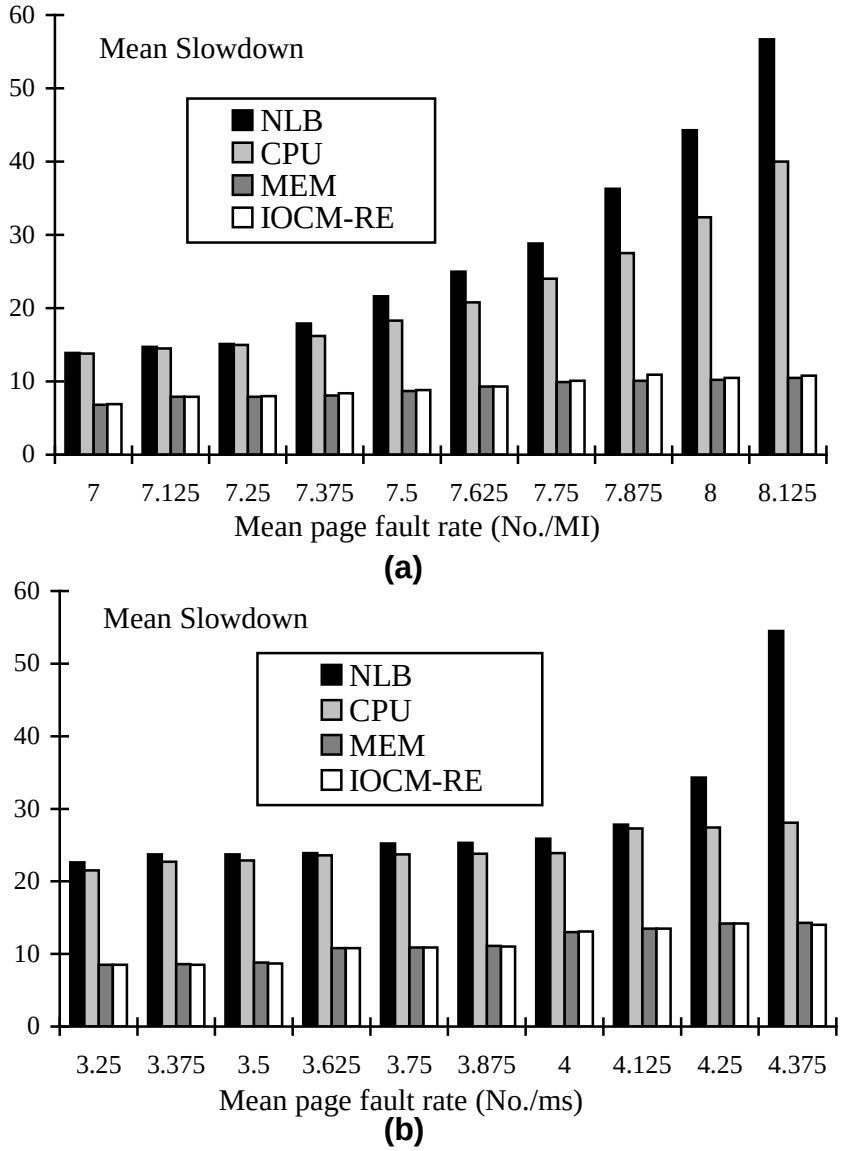


Figure 4.3 Compare IOCM-RE with the CPU-based and memory-based policies under memory-intensive workloads when (a) traces have 30% parallel jobs, and (b) traces have 60% parallel jobs. Mean I/O access rate is 0.0125 No./MI.

4.2.3 Performances under Memory-Intensive Workload Conditions

We now turn our attention to memory-intensive workload conditions. To simulate memory intensive workloads, the mean I/O access rate is fixed at a low value of 0.0125 No./MI. In practice, the page fault rates of applications range from 1 to 10 No./MI [Xiao et al., 2002][Zhang et al., 2000]. The results of the mean slowdown as a function of the page fault rate are summarized in Figure 4.3.

As can be seen in Figure 4.3, when page fault rate is high and I/O rate is very low, IOCM-RE and MEM outperform the CPU-based and NLB schemes considerably. These results can be explained by the following reasons. First, IOCM-RE and MEM consider the effective usage of global memory, attempting to balance the implicit I/O load, which makes the most significant contribution to the overall system load when page fault rate is high and the explicit I/O load is low. Second, the CPU-based scheme improves the utilization of CPU, ignoring the implicit I/O load resulting from page faults.

4.3 A Load Balancing Policy with Preemptive Migrations

We are now in a position to study IOCM-PM, another I/O-aware load-balancing scheme that improves the performance by considering not only incoming jobs but also currently running jobs.

4.3.1 Design of the IOCM-PM Scheme

For a newly arrived job at its home node, the IOCM-PM scheme balances the system load in the following manner. First, IOCM-RE will be invoked to assign the tasks of the newly arrived parallel job to a group of suitable nodes. Second, if the home node is still overloaded, IOCM-PM determines a set of currently running processes that are eligible for migration. The migration of an eligible task is able to potentially reduce the slowdown of the task, and this step substantially improves the performance over the IOCM-RE scheme with non-preemptive migration. The set of eligible migrant tasks is:

$EM(i,k) = \left\{ j \in M_i \,\middle|\, r_{PM}(i,j) > r_{PM}(k,j) + c_j(i,k) \right\}$, where $r_{PM}(i, j)$ and $r_{PM}(k, j)$ are the expected response time of task j on the local node i and on the remote node (computed by Equation 3.17 in Section 3.2.6), respectively, and $c_j(i, k)$ is the migration cost of task j (See Expression 3.19). In other words, each eligible migrant's expected response time on the source node is greater than the sum of its expected response time on the destination node and the expected migration cost. Finally, the eligible processes are preempted and migrated to a remote node with lighter load, and the load of the home node and remote nodes is updated accordingly.

4.3.2 Evaluation of the IOCM-PM Scheme

To evaluate the performance of IOCM-PM, we compare it against CPU, MEM, and IOCM-RE under 20 I/O-intensive traces, which use the same configurations given in Section 4.2. It is noted that in this experiment, the traces encompass synthetic I/O-intensive parallel jobs. The experiments with real world I/O-intensive applications will be discussed in the next section (Section 4.4).

The results of 10 traces, in which 30 percent of the jobs in each trace are parallel, are plotted in Figure 4.4(a). Similarly, Figure 4.4(b) illustrates the mean slowdowns of other 10 traces where 60 percent of the jobs in each trace are parallel.

It is observed from Figure 4.4 that IOCM-PM consistently outperforms all other schemes. For example, IOCM-PM improves the performance over IOCM-RE, memory-based and CPU-based schemes by up to 112.8%, 142.6%, and 146.1%, respectively. These results indicate that load-balancing schemes with preemptive migration outperform those schemes with non-preemptive migration under I/O-intensive workloads. Consequently, the slowdowns of the CPU-based, memory-based, and IOCM-RE are more sensitive to I/O access rate than IOCM-PM.

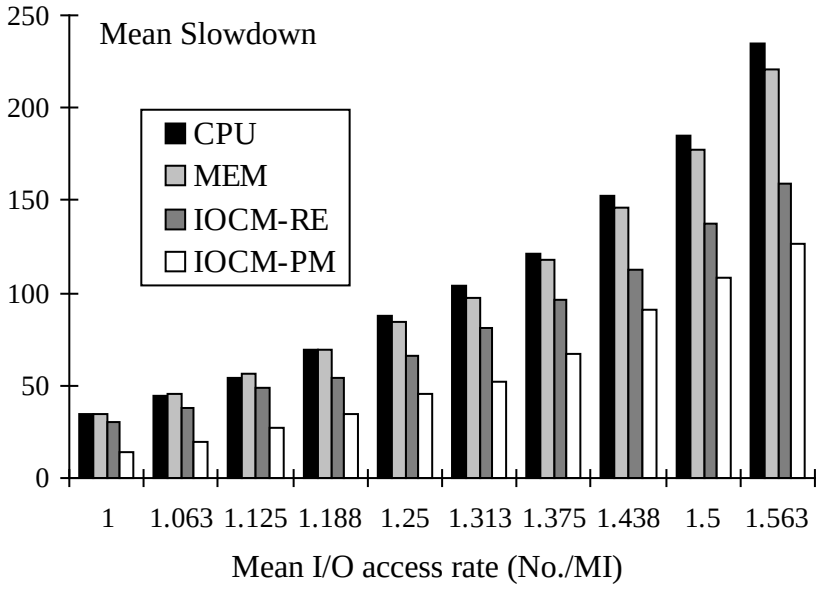


Figure 4.4 (a)

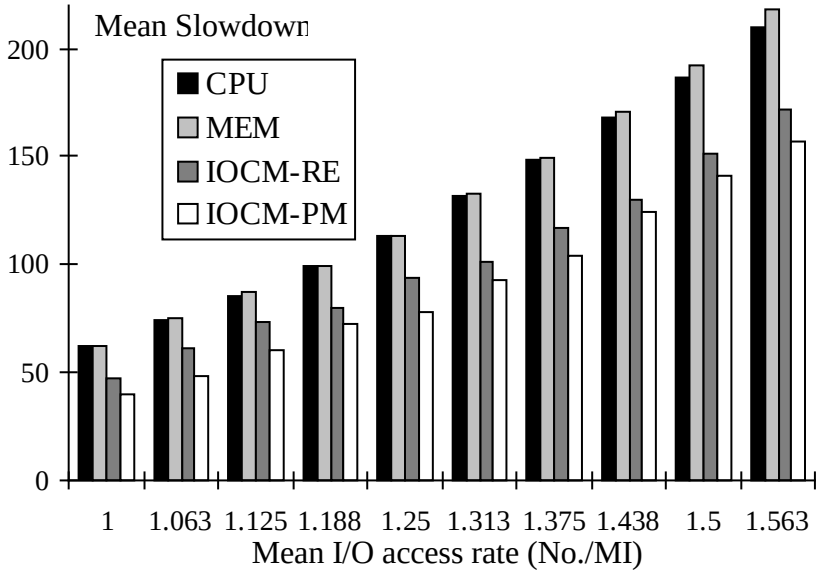


Figure 4.4(b)

Figure 4.4 Compare IOCM-PM with the IOCM-RE policy under I/O-intensive workload conditions when (a) traces have 30% parallel jobs, and (b) traces have 60% parallel jobs. For two graphs, page fault rate is 0.625 No./MI.

The performance improvement of IOCM-PM over the IOCM-RE scheme can be explained by the following reasons. First, one problem encountered in the IOCM-RE policy is that IOCM-RE only considers newly arrived jobs for migration, completely ignoring the running tasks that might take advantages of migration from their overloaded node to a remote node with lighter load. In other words, in the non-preemptive schemes, once a task with high I/O demand misses the opportunity to migrate it will never have a second chance. Second, I/O demand of the tasks in a newly arriving job may not be high enough to offset the migration overhead. Third, IOCM-PM provides better migratory opportunities by considering all the running tasks on a node, in addition to the tasks of a newly arrived job.

4.4 Experiments with Real I/O-intensive Parallel Applications

The experimental results reported in the previous sections are obtained from parallel jobs with synthetic I/O requests. To validate the results based on the synthetic I/O workload, we have conducted experiments based on real world I/O-intensive applications. We have measured the impact of the IOCM-RE and IOCM-PM schemes on the performance of these applications.

4.4.1 An Overview of Real I/O-Intensive Applications

We have simulated a number of real I/O-intensive parallel applications using five sets of I/O traces collected from the University of Maryland [Uysal et al., 1997]. These sets of traces reflect both non-scientific and scientific applications with diverse disk I/O demands.

We evaluate the mean slowdowns of the following five applications:

(1) Data mining (Dmine): This application extracts association rules from retail data [Mueller 1995].

(2) Parallel text search (Pgrep): This application is used for partial match and approximate searches, and it is a modified parallel version of the *agrep* program from the University of Arizona [Wu and Manber, 1992].

(3) Titan: This is a parallel scientific database for remote-sensing data [Chang et al., 1997].

(4) DB2: This is a parallel relational database management system from IBM [IBM DB2]. Due to long run times, only a part of the traces were executed.

(5) LU decomposition (LU): This application tries to compute the dense LU decomposition of an out-of-core matrix [Hendrickson and Womble, 1994].

(6) Sparse Cholesky (Cholesky): This application is capable of computing Cholesky decomposition for sparse, symmetric positive-definite matrices [Acharya et al., 1996].

To simulate these I/O-intensive parallel applications, we generate six job traces where the arrival patterns of jobs are extrapolated based on the job traces collected from the University of California at Berkeley [Harchol-Balter and Downey, 1997]. The main purpose of conducting this experiment is to measure the impact of the I/O-aware load balancing schemes on a variety of real applications and, therefore, each job trace consists of one type of I/O-intensive parallel application described above. A 32-node cluster is simulated to run the applications with different I/O demands in each trace.

4.4.2 Single-Application Experiments

Figure 4.5 shows the mean slowdowns of the six job traces scheduled by four load-sharing policies. We make three observations. First, the I/O-aware load balancing schemes benefit all I/O intensive applications, and offer a 23.6-88.0% performance improvement in mean slowdown over the non-I/O-aware policies. The performance gain is partially attributed to the low migration cost by virtue of duplicating read-only data. Note that these applications present a very small I/O demand for writes, and the I/O request rates for writes are uniformly low.

Second, IOCM-RE and IOCM-PM yield approximately identical performances. We attribute this result to the fact that, since all jobs running on the cluster in this experiment belong to the same application and have nearly identical CPU and I/O demands, the tasks of a newly arrived parallel job are most likely to become the suitable task for migration due to their low migration cost. In other words, both IOCM-RE and IOCM-PM attempt to migrate the tasks of newly arrived jobs when the local node in the cluster is overloaded and, as a result, IOCM-PM reduces to IOCM-RE when the variance in CPU and I/O demand is minimum.

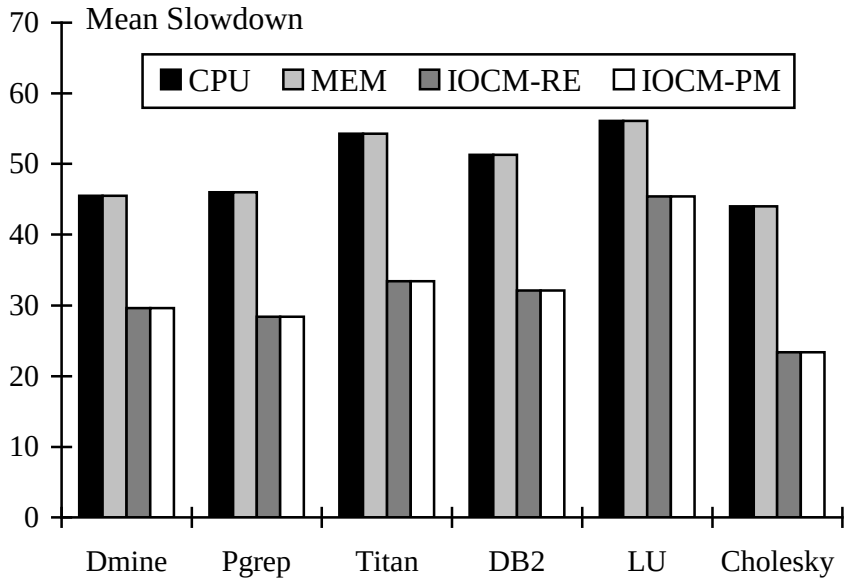


Figure 4.5 Mean slowdowns of four load-balancing policies on six applications.

Third, the trace with LU applications exhibits a larger mean slowdown than the other five traces. Given a fixed job arrival pattern, the mean slowdowns of jobs in a trace depends partially on jobs' total execution time, which in turn is affected by the CPU and I/O execution times of jobs running on a dedicated cluster. Figure 4.6 plots the total execution time of the five applications on a dedicated cluster. As can be seen from Figure

4.6, the total execution time of LU is considerably longer than the other applications, implying that an LU application is expected to spend more time than other applications sharing resources with other jobs running on the cluster. Consequently, there is a strong likelihood that each LU job in the trace will experience a higher slowdown.

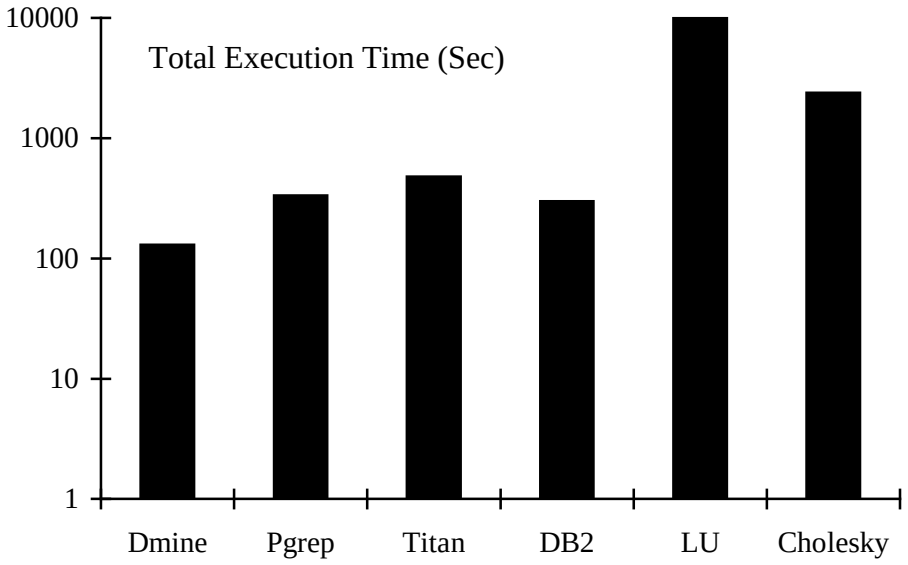


Figure 4.6 The total execution time of six applications on a dedicated cluster

4.4.3 Performance Improvements by I/O-Aware Schemes

Before comparing the performance improvement in terms of mean slowdown, we illustrate the contribution of CPU and I/O execution time to the total execution time of each application in a dedicated computing environment.

As can be seen from Figure 4.7, the total execution time of LU is dominated by I/O processing, thereby giving rise to a low utilization of CPU resources, which in turn leads to a high value of mean slowdown for the trace with LU applications (See Figure 4.5). Unlike the workload with the LU applications, the workload with the other five applications sustains a reasonably high utilization of CPU and disk I/O. This is because, for these

applications, neither CPU time nor I/O time dominates the total execution time. Hence, the trace of the LU applications has the highest slowdown value among all application traces for the four load-balancing policies (See Figure 4.5).

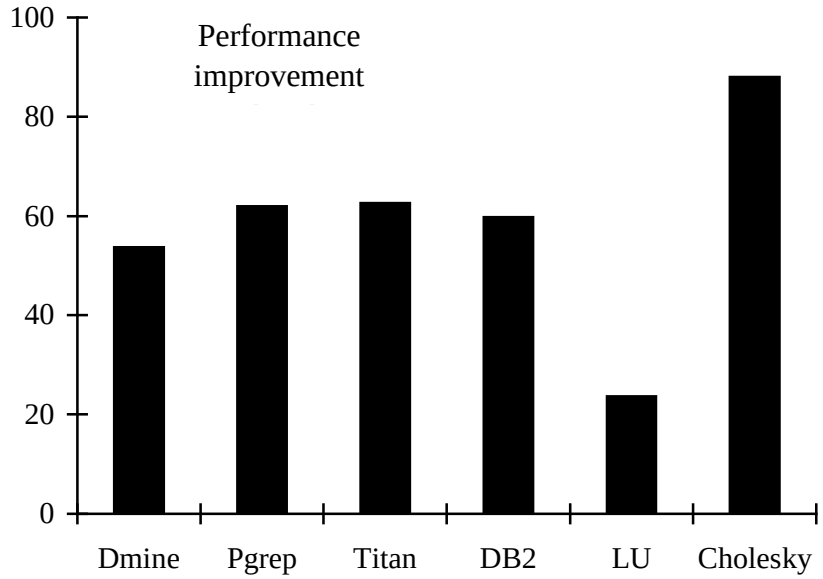


Figure 4.7 The execution time of CPU and I/O as the components of the total execution time of six applications.

Figure 4.8 shows the performance improvement of the I/O-aware policies over non-I/O-aware policies. It is observed that all the six applications benefit from cluster load balancing that dynamically distributes I/O load among all nodes in the cluster. The benefits are pronounced for Cholesky, Titan, Pgrep, DB2, and Dmine, and the performance improvements for these applications are more than 53%. In contrast, the proposed approaches only improve performance in slowdown by 23.6% for the workload with the LU applications. The reason behind this observation is that LU exhibits a low CPU utilization, thus making most of the running LU jobs in the cluster compete for disk I/O resources.

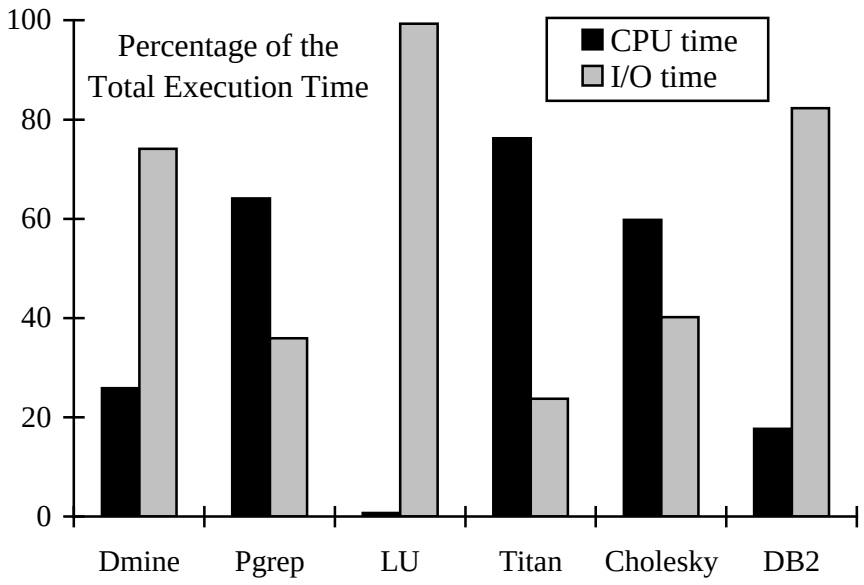


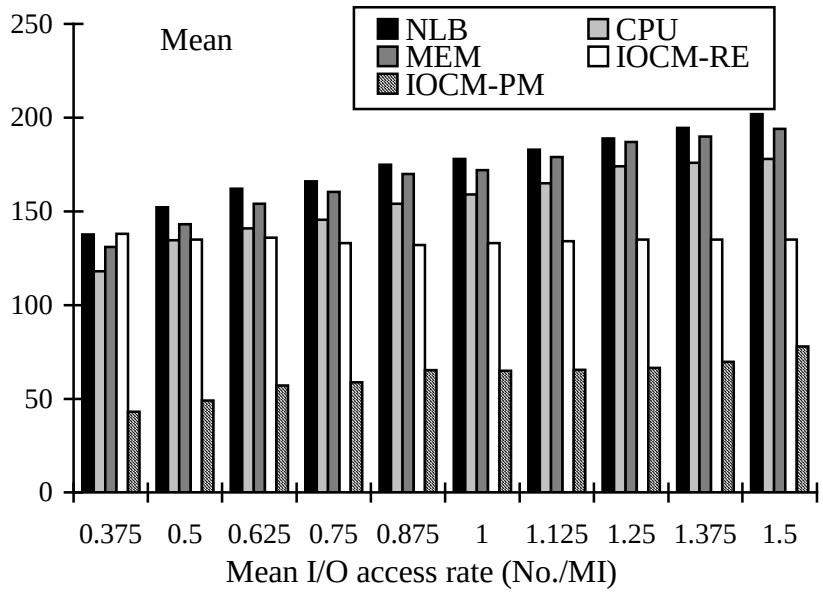
Figure 4.8 Comparison of performance Im-provement in mean slowdown on five traces.

4.4.4 Experiments with Multiple Applications Running Simultaneously

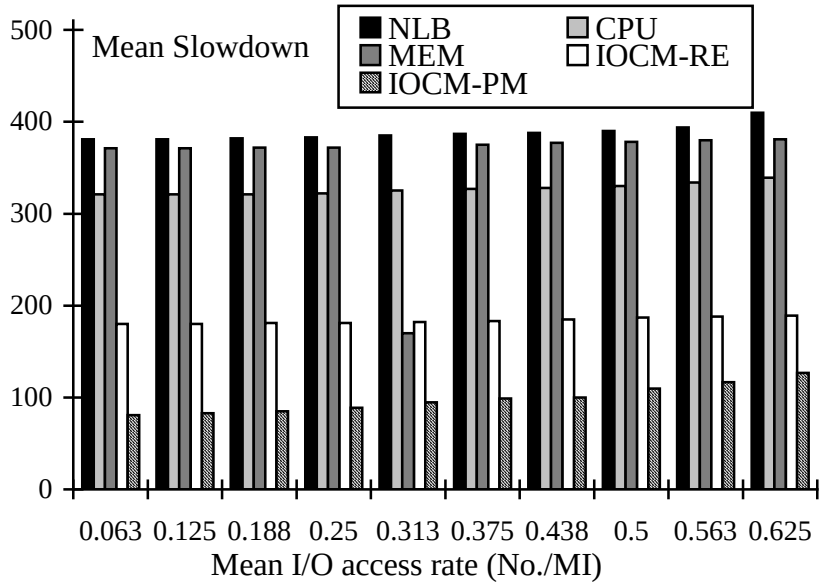
In what follows, we measure the impact of the I/O-aware load-balancing policies using traces, which comprise sequential jobs and a combination of the five I/O-intensive parallel jobs used in the previous experiment. The I/O demands of parallel jobs are accurately determined by the I/O traces of real applications, whereas the I/O access rates of sequential jobs are randomly generated based on a uniform distribution.

The Figure 4.9 plots mean slowdown as a function of the mean I/O access rate of sequential jobs when 30 and 60 percent of the jobs in each trace are parallel.

We make the following three observations from Figure 4.9. First, the high percentage of I/O-intensive parallel jobs results in a high I/O intensity, which in turn gives rise to a high value of mean slowdown under all load-balancing policies.



(a)



(b)

Figure 4.9 Mean slowdowns of the traces with multiple applications. Page fault rate of 0.625 No./MI. (a) The traces have 30% parallel jobs. (b) The traces have 60% parallel jobs.

Second, Figure 4.9 reveals that the mean slowdowns of the five policies increase with the I/O load. This result indicates that even when the I/O demands of parallel I/O-intensive applications remain unchanged, the performance depends hugely on the I/O intensity of the workload, which in turn is partially affected by both parallel and sequential jobs. When the workload is highly I/O-intensive, the parallel I/O-intensive applications spend more time waiting for CPU and disk I/O. Third, for all cases, the IOCM-PM scheme is substantially better than the other policies with non-preemptive migration strategy. Figure 4.9(a) shows that for most cases, the IOCM-RE scheme is the second best load-balancing policy. Interestingly, when the mean I/O access rate is as low as 0.375 No./MI for the workload where 30 percent of jobs are parallel, the performance of IOCM-RE is slightly worse than that of CPU-based and memory-based schemes. The performance deterioration of IOCM-RE comes from the inaccurate estimation of remote execution cost when I/O intensity is relatively low. In general, the two I/O-aware load-balancing schemes are less sensitive to I/O intensity than those non-I/O-aware policies.

4.5 Summary

In this chapter we have proposed two efficient load-balancing policies for parallel data-intensive applications. The first one is referred to as *IOCM-RE* (with remote execution), and the second one is called *IOCM-PM* (with preemptive migration). IOCM-RE employs remote execution facilities to improve system performance, whereas IOCM-PM utilizes a preemptive migration strategy to boost the performance.

The IOCM-RE and IOCM-PM policies consider both explicit and implicit I/O load, leading to a performance improvement over the existing CPU- and Memory-based policies under workloads with high disk I/O demands. Using five real I/O-intensive parallel applications in addition to a set of synthetic parallel jobs with a wide variety of I/O demands, we have demonstrated that applying IOCM-RE and IOCM-PM to clusters of workstations for I/O-intensive workloads is highly effective. Specifically, the proposed

schemes offer 23.6-88.0% performance improvements in mean slowdown for I/O-intensive applications including LU decomposition, Sparse Cholesky, Titan, Parallel text searching, and Data Mining.

Chapter 5

Load Balancing for Heterogeneous Clusters

In the previous chapter, we have addressed the issue of load balancing for homogeneous clusters, which comprise a set of identical or similar nodes with a given set of performance characteristics in computing power, memory capacity, and disk speed. In this chapter we will explore load-balancing schemes for heterogeneous clusters, where nodes may be of different capabilities and characteristics, under I/O- and memory-intensive workload situations.

This chapter is organized as follows. Section 5.1 presents the motivation of this study. A heterogeneity-aware load-balancing policy, referred to as IOCM-RE, is outlined in Section 5.2. In Section 5.3, the experiment setup is described. Empirical results based on a number of simulated heterogeneous clusters are discussed in Section 5.4. Section 5.5 provides a summary of this chapter.

5.1 Motivation

Although it is common that a new and stand-alone cluster system is homogeneous in nature, upgraded clusters or networked clusters are likely to be heterogeneous in practice. In other words, heterogeneity of a variety of resources such as CPU, memory, and disk

storage, may exist in cluster systems. This is because, to improve performance and support more users, new nodes that might have different characteristics than the original ones may be added to the systems or several smaller clusters of different characteristics may be connected via a high-speed network to form a bigger one. Accordingly, heterogeneity may exist in a variety of resources such as CPU, memory, and disk storage.

Heterogeneity in disks tends to induce more significant performance degradation when coupled with imbalanced load of memory and I/O resources and therefore and, therefore, we have developed a heterogeneity-aware load-balancing scheme that is able to sustain high performance for a wide spectrum of workload conditions on clusters with heterogeneous resources.

This chapter presents a load balancing scheme that is capable of hiding the heterogeneity of resources, especially that of I/O resources, by judiciously balancing I/O work across all the nodes in a cluster. The experimental results, generated from extensive simulations driven by both synthetic and real-application traces, indicate that our proposed schemes significantly improve the performance of the existing load balancing schemes that only consider CPU and memory.

5.2 A Heterogeneity-Aware Load Balancer

5.2.1 Heterogeneity Level

In this part of our study, it is imperative to introduce an efficient way to quantitatively estimate the heterogeneity level of each resource, since the heterogeneity of resources is expected to have a noticeable impact on the system performance.

The nodes may have a wide variety of operating systems, network interfaces, and internal architectures. However, we only address heterogeneity with respect to a diverse set of disks, CPUs, and memories. Specifically, we characterize each node N_i by its CPU speed C_i, memory capacity M_i, and disk performance D_i. Let $B^{disk}{}_i$, S_i, and R_i denote the disk

bandwidth, average seek time, and average rotation time of the disk in node *i*, then the disk performance can be approximately measured as the following equation:

$$D_i = 1/\left(S_i + R_i + d/B_i^{disk}\right), \tag{5.1}$$

where *d* is the average size of data stored or retrieved by I/O requests.

The weight of a disk performance W^{disk}_i is defined as a ratio between its performance and that of the fastest disk in the cluster. Thus, we have:

$$W_i^{disk} = D_i / MAX_{j=1}^{n}(D_j). \tag{5.2}$$

The disk heterogeneity level, referred to as H_D, can be quantitatively measured by the standard deviation of disk weights. Formally, H_D is expressed as:

$$H_D = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(W_{avg}^{disk} - W_i^{disk}\right)^2}, \tag{5.3}$$

where W_{avg}^{disk} is the average disk weight. Likewise, the CPU and memory heterogeneity levels are defined as follows:

$$H_C = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(W_{avg}^{CPU} - W_i^{CPU}\right)^2}, \tag{5.4}$$

$$H_M = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(W_{avg}^{mem} - W_i^{mem}\right)^2}, \tag{5.5}$$

where W_i^{CPU} and W_i^{mem} are the CPU and memory weights, and W_{avg}^{CPU} and W_{avg}^{mem} are the average weights of CPU and memory resources [Xiao et al., 2000].

5.2.2 Description of the Heterogeneity-Aware Load Balancing Scheme

We now turn our attention to a heterogeneity-aware load balancing scheme for I/O-intensive workload conditions. We refer to this policy as IO-RE, which is heuristic and greedy in nature. The key objective of IO-RE is to achieve a well-balanced I/O load under an I/O-intensive workload. Instead of using CPU and memory load indices, the IO-RE policy relies heavily on the I/O load index (See Equation 3.2 in Section 3.2.2) to measure system workloads and distribute I/O load accordingly.

An I/O threshold, $threshold_{IO}(i)$, is introduced to identify whether node i's I/O resource is overloaded. Thus, node i's I/O resource is considered overloaded, if the load index $load_{IO}(i)$ is higher than $threshold_{IO}(i)$. Specifically, $threshold_{IO}(i)$, which reflects the I/O processing capability of node i, is expressed by the following equation:

$$threshold_{IO}(i) = \left(\sum_{j=1}^{n} load_{IO}(j) \right) \times \left(D_i \Big/ \sum_{j=1}^{n} D_j \right), \tag{5.6}$$

where the term in the first parenthesis gives the accumulative I/O load imposed by the running tasks in the heterogenesou cluster, and the term in the second parenthesis corresponds to the fraction of the total I/O processing power on node i. The I/O threshold associated with node i can be calculated using equations 3.2 and 5.1 to substitute for the terms in the first and second parentheses.

Recall that a parallel job comprises a number of tasks, which are either dependent or independent of one another. For a task j of the given job arriving at a local node i, the IO-RE scheme attempts to balance I/O resources in the following four main steps. First, the I/O load of node i is updated by adding task j's explicit and implicit I/O load. Second, the I/O threshold of node i is computed based on Equation 5.6. Third, if node i's I/O load is less than the I/O threshold, the I/O resource of node i is considered under-loaded. Consequently, task j will be executed locally on node i. Otherwise, the node is overloaded with respect to I/O resources and, thus, IO-RE chooses a remote node k as task j's

destination node, subject to the following two conditions: (1) The I/O resource is not overloaded. (2) The I/O load discrepancy between node *i* and *k* is greater than the I/O load induced by task *j*, to avoid useless migrations. If such a remote node is not available, task *j* has to be executed locally on node *i*. Otherwise, task *j* is transferred to the remote node *k*, and the I/O load of nodes *i* and *k* is updated according to *j*'s load.

Since the main target of the IO-RE policy is exclusively I/O-intensive workloads, IO-RE is unable to maintain a high performance when the workloads tend to be CPU- or memory-intensive. To overcome this limitation of IO-RE and develop a load-balancing scheme for a broad range of applications, IOCM-RE, which is similar to the one presented above, is studied to achieve the effective usage of CPU and memory in addition to that of I/O resources in heterogeneous clusters. More precisely, IOCM-RE leverages the I/O-RE policy as an efficient means to make load-balancing decisions in the presence of explicit I/O load in a node. If the node exhibits implicit I/O load due to page faults, load-balancing decisions are made by the memory-based policy. Otherwise, the CPU-based policy is used when the node is able to fulfill the accumulative memory requirements of all tasks running on it.

Table 5.1 Characteristics of System Parameters. CPU speed is measured by Millions Instruction Per Second (MIPS).

Parameter	Values assumed
CPU Speed	100-400 MIPS
RAM Size	32-256 Mbytes
Buffer Size	64 Mbytes
Network Bandwidth	100 Mbps
Page fault service time	8.1 ms
Time slice of CPU time sharing	10 ms
Context switch time	0.1ms
Data re-access rate, *r*	5

5.3 Simulation Parameters

To study the performance of the two schemes presented above, we have conducted a series of trace-driven simulations. In this section, the important simulation parameters will be discussed.

We have simulated a cluster that comprises six workstations with the configuration parameters listed in Table 5.1. The parameters are chosen in such a way that they resemble some workstations such as Sun SPARC-20 and Sun Ultra 10.

Table 5.2 Characteristics of Disk Systems

Age (years)	Avg. Seek time(ms)	Avg. Rotation time (ms)	Bandwidth (MB/sec)
1	5.30	3.00	20.0
2	5.89	3.33	14.3
3	6.54	3.69	10.2
4	7.29	4.11	7.29
5	5.21	4.56	5.21
6	3.72	5.07	3.72

The configuration of disks used in our simulated environment is based on the assumption of device aging and performance-fault injection. Specifically, IBM 9LZX is chosen as a base disk and its performance is aged over years to generate a variety of disk characteristics [Forney et al., 2001] shown in Table 5.2.

Table 5.3 Characteristics of traces

	Trace 1	**Trace 2**	**Trace 3**	**Trace 4**
Trace Type	I/O intensive	I/O intensive	I/O intensive	Memory intensive
Mean I/O request size	256Kbyte	256Kbyte	1 Mbyte	64Kbyte
I/O data size distribution	Gamma	Gamma	Gamma	Uniform
Mean I/O access rate	20 No./MI	20 No./MI	20 No./MI	0.1 No./MI
I/O request distribution	Exponential	Exponential	Exponential	Uniform
Mean initial data size	50 Mbyte	0 Kbyte	0 Kbyte	100 KByte

We use the same method described in Section 3.3 to generate a group of traces. To show that our approach sustains high performance under a diversity of workload, we have used the following four traces (See Table 5.3) with a mix of CPU-, memory- and I/O-intensive tasks. In Trace 1-3, 10% tasks are either CPU-intensive or memory-intensive. Trace 4 comprises 10% I/O-intensive tasks. The data sizes in Trace 1-3 reflect typical data characteristics for many data-intensive applications, where the vast majority of I/O requests are small.

Table 5.4 Characteristics of Five Heterogeneous Clusters. CPU and memory are measured by MIPS and MByte. Disks are characterized by bandwidth measured in MByte/Sec. HL- Heterogeneity Level

	Node	**1**	**2**	**3**	**4**	**5**	**6**	**HL**
System A	CPU	100	100	100	100	100	100	0
	Memory	480	480	480	480	480	480	0
	Disk	20	20	20	20	20	20	0
System B	CPU	100	150	150	50	100	50	0.27
	Memory	480	640	640	320	480	320	0.2
	Disk	20	20	20	20	20	20	20
System C	CPU	100	150	150	50	100	50	0.27
	Memory	480	640	640	320	480	320	0.2
	Disk	10.2	20	20	10.2	20	10.2	0.25
System D	CPU	50	200	200	50	50	50	0.35
	Memory	320	800	800	320	320	320	0.28
	Disk	10.2	20	20	14.3	14.3	10.2	0.2
System E	CPU	50	200	200	50	50	50	0.35
	Memory	320	800	800	320	320	320	0.28
	Disk	5.21	14.3	20	5.21	7.29	3.72	0.3

5.4 Performance Results

In this section, we experimentally compare the performance of IOCM-RE and IO-RE against that of three other schemes: CPU-RE [Eager et al., 1986][Harchol-Balter and

Downey, 1997] and MEM-RE [Zhang et al., 2000]. As mentioned earlier, the performance metric by which we judge system performance is the mean slowdown of all the jobs in a trace.

5.4.1 Impact of Heterogeneity on the Performance of Load Balancers

The goal of this experiment is to evaluate the performance improvement of the proposed load-balancing policies over the existing schemes, and to understand the sensitivity of the proposed schemes to heterogeneity level.

The five configurations of increasing heterogeneity of a heterogeneous cluster with 6 nodes are summarized in Table 5.4. For the purpose of comparison, system A is homogenous, and system B is homogenous in terms of disk I/O.

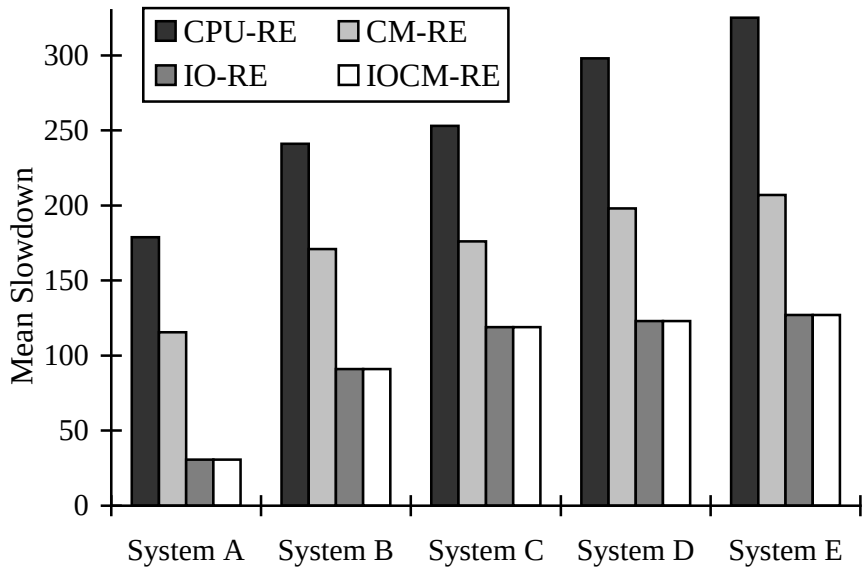


Figure 5.1 Mean slowdown when trace 1 is used on five heterogeneous systems

As can be seen from Figure 5.1, IO-RE and IOCM-RE significantly outperform the other three policies. For example, IOCM-RE improves the performance over CPU-RE and MEM-RE by up to a factor of 5 and 3, respectively.

Importantly, Figure 5.1 shows that the mean slowdowns of almost all policies increase consistently as the system heterogeneity increases. An interesting observation from this experiment is that the mean slowdowns of IO-RE and IOCM-RE are more sensitive to changes in CPU and memory heterogeneity than the other three policies. Recall that system B's CPU and memory heterogeneities are higher than those of system A, and both systems A and B are homogeneous with respect to disk power. Compared the performance of system A with that of system B, the mean slowdowns of IO-RE and IOCM-RE are increased by 196.4%, whereas the slowdowns of CPU-RE and MEM-RE are increased approximately by 34.7% and 47.9%, respectively. The reason can be explained by the fact that I/O-aware policies ignore the heterogeneity in CPU resources. A second observation is that, when the heterogeneity of CPU and memory remain unchanged, the performance of IO-RE and IOCM-RE is less sensitive to the change in disk I/O heterogeneity than the other three policies. For example, when the slowdowns of system D and system E are compared, the mean slowdowns of IO-RE and IOCM-RE increase by approximately 3%, the slowdown of MEM-RE increases by around 5%, and the slowdowns of CPU-RE increase by nearly 9%. This is because both IO-RE and IOCM-RE consider disk heterogeneity as well as the effective usage of I/O resources.

5.4.2 Effects of Mean Initial Data Size

This section investigates the effects of mean initial data size, a key source of the migration overhead, on the performance of heterogeneous clusters.

Figure 5.2 plots the mean slowdowns of CPU-RE, MEM-RE, and IOCM-RE for Trace 1 and Trace 2, as a function of increasing heterogeneity. While Trace 1 represents a workload where the initial data of each remotely executed task is not available at the remote node, Trace 2 illustrates a scenario where the migration overhead is considerably

reduced by replicating initial data across all the nodes in advance. The mean slowdowns of NLB and IO-RE are similar to those of CPU-RE and IOCM-RE, respectively. Thus, the experimental data for NLB and IO-RE is omitted from Figure 5.2.

Figure 5.2 shows that IOCM-RE consistently outperforms the CPU-RE and MEM-RE policies, a finding that is consistent with the results reported in Section 5.4.1. Moreover, Figure 5.2 shows that the slowdowns of CPU-RE for two traces are roughly identical, implying that the sensitivity of CPU-RE to initial data size is not noticeable when the workload is I/O-intensive. The explanation of this result is that CPU-RE makes no effort to balance disk resources and, as a result, very few remote executions occur under I/O-intensive workload conditions.

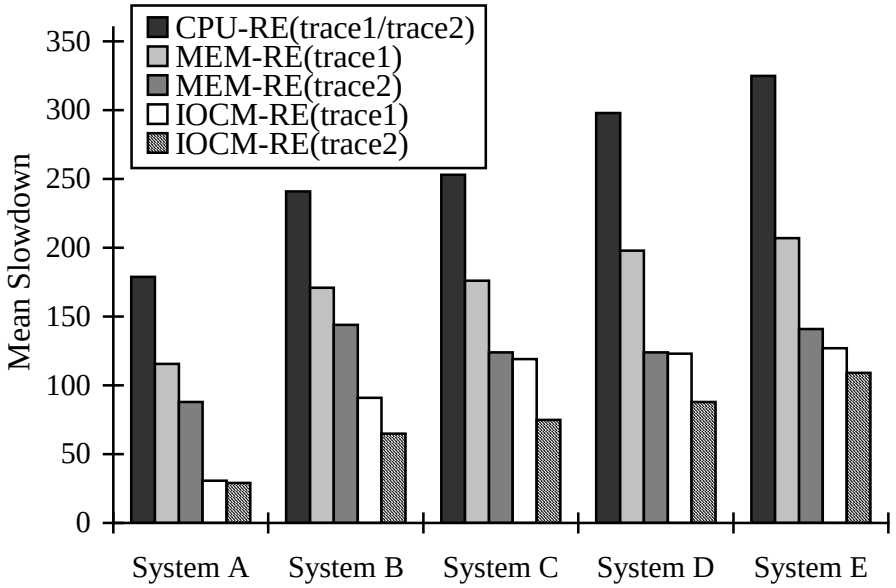


Figure 5.2 Mean slowdown when trace 1 and 2 are used on five heterogeneous systems.

A third observation from Figure 5.2 is that, for the MEM-RE and IOCM-RE policies, the slowdowns of Trace 2 are noticeably lower than those of Trace 1, indicating that the system performance improves dramatically as the average initial data size is decreased. The

reason can be explained as follows. For a remotely executed task, its initial data has to be available in the corresponding remote node. Hence, if the required initial data is not initially provided by the remote node, the overhead of moving initial data offsets the benefits gained from load balancing schemes. In other words, small amount of migrated data results in low overhead of remote executions, which in turn helps alleviate the network burden. This result suggests that combining data replication algorithms for I/O-intensive applications, our approach can achieve additional performance improvement by reducing the overhead of data migration.

5.4.3 Sensitivity to I/O Demands

In principle, I/O demand of an application can be characterized by the average data size and the access rate of I/O requests, which in turn heavily depend on I/O access patterns. In this section, we first evaluate how the average data size affects the performance of the load-balancing policies.

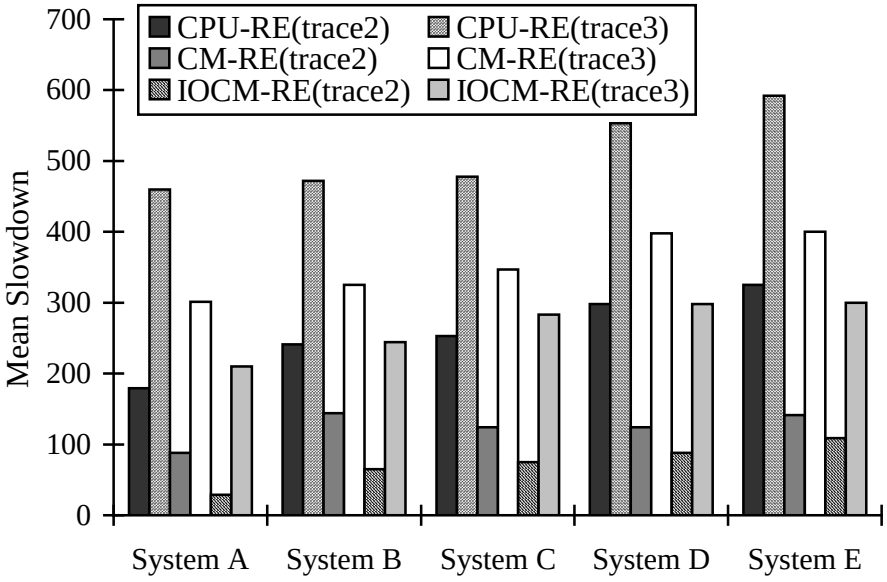


Figure 5.3 Mean slowdown when trace 2 and 3 are used on five heterogeneous systems.

Figure 5.3 plots the performance of CPU-RE, MEM-RE, and IOCM-RE for Trace 2 and Trace 3 on five heterogeneous clusters. Figure 5.3 shows that, for all policies on each cluster, the mean slowdown under Trace 2 is significantly lower than that under Trace 3. This is because the average data size of Trace 3(1MByte/Sec) is four times as large as that of Trace 2 (256KByte/Sec), and the larger average data size leads to lower buffer hit rate and longer waiting times on I/O processing.

Figure 5.4 shows the sensitivity of the CPU-RE, MEM-RE, and IOCM-RE polices to I/O access rate on system A under a modified Trace 2, as a function of I/O access rate. The workload parameters of the modified trace are identical to those of Trace 2, except that the average I/O access rate is increased from 14 to 20 No./MI with increments of 1.0 No./MI.

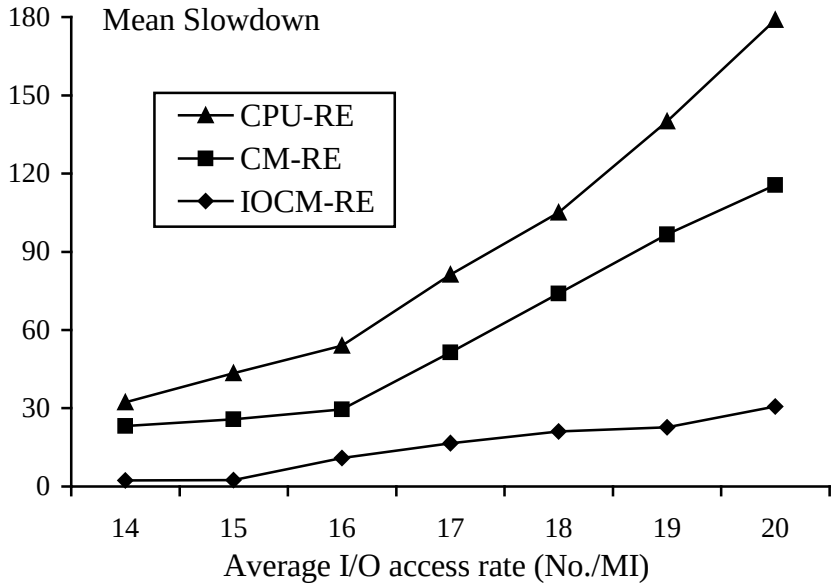


Figure 5.4 Mean slowdown under a modified trace 2 on System A.

Figure 5.4 reveals that the mean slowdowns of the three policies increase with the increasing value of I/O access rate, because high I/O access rate results in long waiting

time on I/O processing. Additionally, the slowdowns of CPU-RE and MEM-RE increase much more quickly than that of IOCM with the I/O access rate. This result indicates that when the workload is I/O-intensive, the CPU-RE and MEM-RE policies are more sensitive to changes in I/O access rate than IOCM-RE. The explanation is that IOCM-RE achieves a highly effective usage of global disk I/O resources by making I/O load well balanced.

5.4.4 Performance under Memory-Intensive Workloads

We conduct this experiment with an examination of the proposed policies applied to a memory-intensive workload. We use Trace 4, which represents a memory-intensive workload, to drive our simulation. Figure 5.5 shows the mean slowdowns of the five load-balancing policies applied to the five clusters.

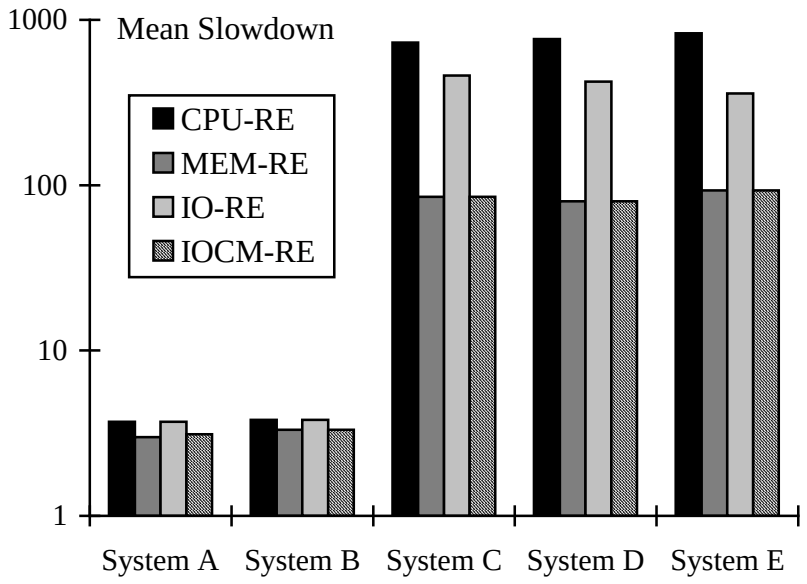


Figure 5.5 Mean slowdowns of five systems under a memory-intensive workload. Trace 4 is used.

As can be seen in Figure 5.5, when the workload is highly memory-intensive, MEM-RE and IOCM-RE perform equally well, outperforming the rest of the policies investigated. This indicates that the IOCM-RE policy maintains the same level of performance as MEM-RE under memory-intensive workloads. For example, MEM-RE and IOCM-RE improve the performance of System D over CPU-RE and IO-RE by a factor of 8 and 5, respectively. This is because MEM-RE and IOCM-RE consider the effective usage of global memory, which helps alleviate the problem of imbalance in implicit I/O load.

Another interesting observation from Figure 5.5 is that MEM-RE and IOCM-RE are less sensitive to the heterogeneity level than the other three policies. For example, the slowdown of CPU-RE for System E is more than 223 times higher than that for System A, whereas the IOCM-RE's slowdown for System E is only 29 times higher than that for System A. The reason for this phenomenon is that MEM-RE and IOCM-RE can mask the heterogeneity effects by migrating tasks with high memory demands to the nodes with adequate free memory resources.

5.4.5 Effectiveness of Improving I/O Buffer Hit Rate

The IOCM-RE policy is capable of boosting the utilization of I/O buffers, which in turn decreases I/O access frequency. This experiment focuses on I/O buffer hit rate, which reflects I/O buffer utilization. Using Trace 2 and Trace 3 on five heterogeneous clusters, Figure 5.6 compares the I/O buffer hit rates of the CPU-RE, MEM-RE, and IOCM-RE policies.

Three observations can be made from this experiment. First, the buffer hit rate of IOCM-RE is consistently higher than those of CPU-RE and MEM-RE in all cases. For example, when Trace 3 is evaluated on System B, IOCM-RE improves the buffer hit rate over CPU-RE and MEM-RE by 28% and 7%, respectively. These improvements then enable the overall performance to be increased by 93.4% and 33.2% (See Figure 5.3), respectively. The overall performance gains are due to the high buffer hit rates that help reduce both paging time and I/O processing time.

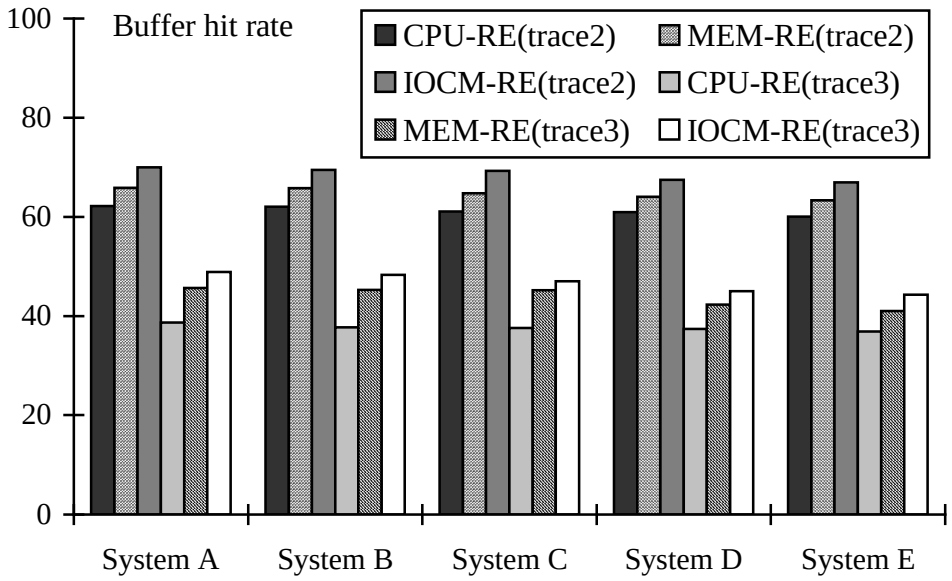


Figure 5.6 Buffer hit ratios of five systems when trace 2 and trace 3 are used.

Second, increasing the system heterogeneity results in a slight reduction in the buffer hit ratio, which in turn worsens the overall performance.

Third, Figure 5.6 shows that for all the three load-balancing policies, the average data size of I/O request significantly affects the I/O buffer hit rate. The larger the average I/O request size, the lower the I/O buffer hit rate. For example, recall that the average data size of Trace 3 is larger than that of Trace 2. The buffer hit rate of IOCM-RE for Trace 3 on System C is 47%, whereas the buffer hit rate of Trace 2 on the same platform is more than 69%.

5.4.6 Performance of Load-Balancing Real I/O-Intensive Applications

While the previous results were obtained from traces containing jobs with synthetic I/O demands, we use the same method described in Section 4.4 to simulate six traces with real I/O-intensive applications.

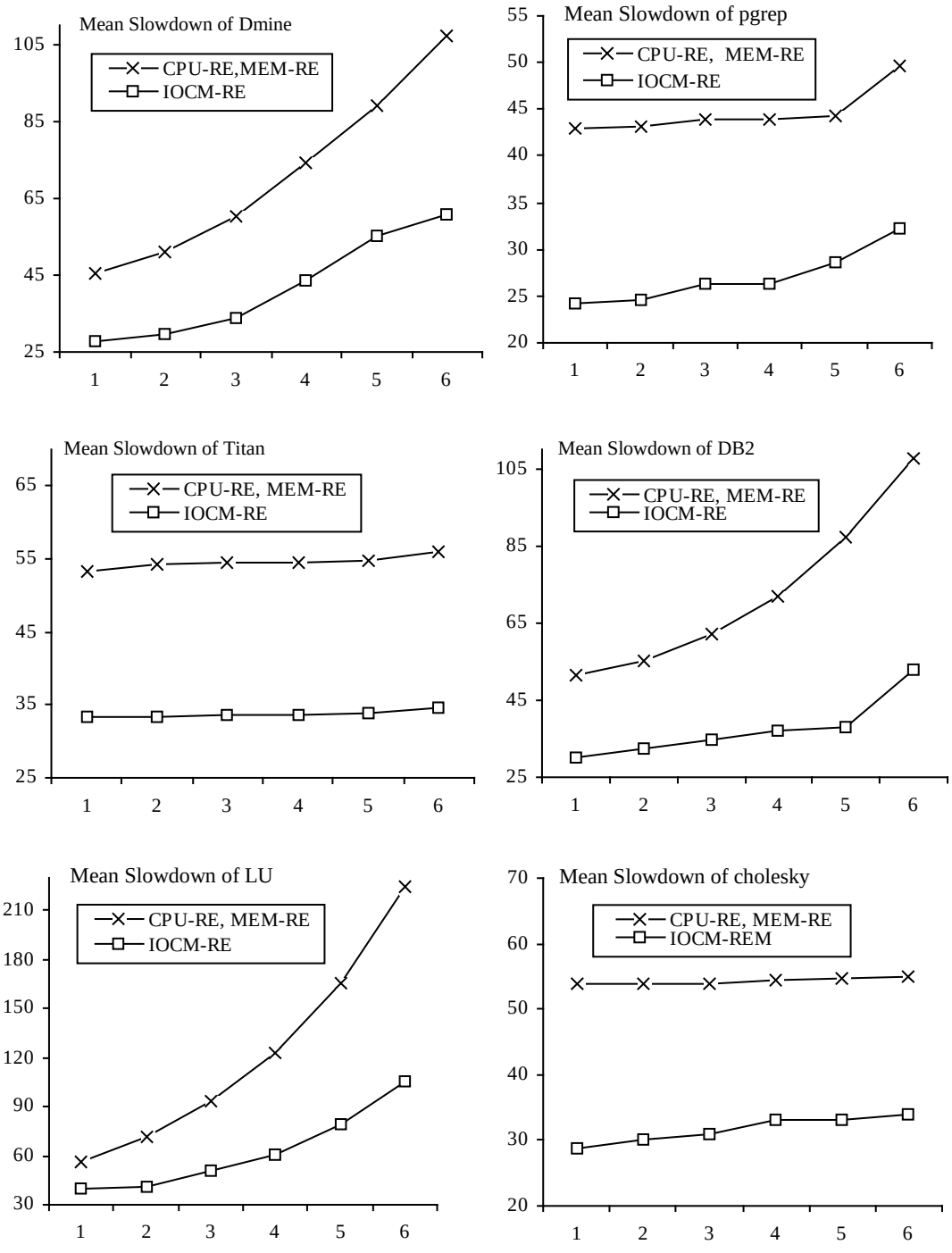


Figure 5.7 Mean slowdowns as a function of the age of a single disk

In this experiment disks are configured such that five nodes possess fast disks that are one year old, and a sixth node has a slower disk assumed an age ranging from 1 to 6 years. The x-axis in Figure 5.7 denotes the age of the slow disk, whereas the y-axis represents the mean slowdown of each trace with a particular application.

Figure 5.7 clearly shows that compared with the CPU-RE and MEM-RE schemes, IOCM-RE is more beneficial for all six I/O intensive applications. For example, IOCM-RE achieves a performance improvement ranging from 39.6% to 129.2%.

The performance improvement of IOCM-RE largely relies on the technique that balances I/O load by migrating I/O-intensive tasks from overloaded nodes to underloaded ones. It is interesting that that for all six applications, the mean slowdowns of three schemes increase as the slow disk ages. This is because aging a disk results in a higher level of disk I/O heterogeneity, which in turn gives rise to longer I/O processing time.

Figure 5.7 illustrates that the performance of IOCM-RE is less sensitive to the change in the age of the slow disk than the CPU-RE and MEM-RE policies. For example, when the trace with DB2 applications is simulated, the slowdowns of CPU-RE and MEM-RE increase by 70% as the slow disk is aged from 1 to 5 years old; whereas the slowdown of IOCM-RE merely increase by approximately 26%. This results shows that IOCM-RE delivers better performance by hiding disk heterogeneity as well as the effective usage of I/O resources.

More interestingly, the traces with Dmine, DB2, and LU are more sensitive to the age of the slow disk than those with pgrep, Titan, and Cholesky. Given a fixed job arrival pattern, the sensitivity to disk I/O heterogeneity level depends partially on the ratio of a job’s I/O execution time to its total execution time, which can be used to quantitatively measure the I/O-intensive level of applications. Recall the results summarized in Figure 12, for the Dmine, DB2, and LU application, the percentages of total execution time spent in performing I/O operations are 74%, 82%, and 99%, respectively. In contrast, such percentages for the other three applications are as relatively low as 36%, 24%, and 40%, respectively. This result indicates that Dmine, DB2, and LU are more I/O-intensive than the

other three applications, implying that Dmine, DB2, and LU are expected to spend more time in sharing disk I/O resources with other jobs running on the cluster. Consequently, the traces with the Dmine, DB2, and LU applications are more sensitive to the disk I/O heterogeneity level.

5.5 Summary

In this chapter we explored load balancing for heterogeneous clusters under I/O- and memory-intensive workloads. In particular, we studied two I/O-aware load-balancing policies: *IO-RE* (I/O-based policy) and *IOCM-RE* (load balancing for I/O, CPU, and Memory). This section summarizes several interesting findings observed from our extensive simulations:

(1) The slowdowns of all the policies considerably increase as one of the disks ages, and the performance with two slow disks is as poor as that with all disks being slow.

(2) The slowdowns of almost all the policies increase consistently with the system heterogeneity. The slowdowns of IO-RE and IOCM-RE are more sensitive to changes in CPU and memory heterogeneity than the other three policies, whereas IO-RE and IOCM-RE are less sensitive to changes in disk I/O heterogeneity than non I/O-aware load balancing policies.

(3) The system performance improves substantially with the decreasing value of the average initial data size, suggesting that our approach could achieve additional performance improvements when combined with data replication algorithms to replicate initial data. Moreover, the slowdowns of all the policies increase consistently with the I/O access rate.

(4) The I/O buffer hit rate resulting from IOCM-RE is consistently higher than those from the existing schemes in all cases, and large average I/O request size leads to low I/O buffer hit rate.

(5) If a workload is highly memory-intensive, the IOCM-RE policy significantly outperforms IO-RE, and IOCM-RE maintains the same level of performance as MEM-RE. Furthermore, under memory-intensive workload situations MEM-RE and IOCM-RE are less sensitive to changes in heterogeneity than the other three policies.

Chapter 6

A Feedback Control Mechanism

In this chapter, a new feedback control algorithm (BFC) for managing I/O buffers is presented to improve the overall performance of a cluster. This algorithm is also shown to be effective in complementing and enhancing the performance of a number of existing dynamic load-balancing schemes including those presented in the previous chapters. The primary objective of this mechanism is to minimize the number of page faults for memory-intensive jobs while increasing the utilization of I/O buffers for I/O-intensive jobs. The BFC algorithm dynamically adjusts I/O buffer sizes relative to total main memory space based on a feedback algorithm that captures current and past workload characteristics.

The chapter is organized as follows. Section 6.1 motivates this research by describing the buffer management in existing Linux systems. Section 6.2 describes an approach that uses a feedback control mechanism to configure I/O buffer sizes at the runtime. Section 6.3 discusses the experimental results by comparing the performance of three load balancing schemes with the feedback controller against those without using BFC. Section 6.4 summarizes the chapter.

6.1 Motivation

The goal of buffer caches in an operating system is to reduce the number of slow disk accesses, making disk I/O operations exhibit minimal slowdowns on the system. Linux is an open-source operating system running on commodity PCs. Linux systems cache disk data in virtual memory pages, and allow buffer cache to compete with other virtual memory segments for the entire space of main memory [Bovet and Cesati, 2001]. The effectiveness of buffer caches mainly depends on their size. For example, if the size of a buffer cache is too small, all cached data may be flushed from the cache before being reused. If the size is very big, it may make the free memory too small and cause swapping. Linux automatically use all free memory for buffer cache while keeping the buffer cache small for circumstances under which programs need more memory. In practice, a Linux system specifies the maximal and minimal bounds of the buffer cache size. The optimal bounds largely depend on the system workload, which may be determined by experiments. Although the static configuration of buffer size bounds is an approach to tuning the performance of clusters when workload conditions can be modeled and predicted, this approach performs poorly and inefficiently for highly dynamic environments where workloads are unknown at compile time. In addition, the buffer mechanisms in Linux systems only consider present buffer cache usages in a passive fashion, making it difficult for buffer subsystems to optimize the buffer cache usability.

To alleviate the above problems, we propose in this chapter a feedback control mechanism that is able to adjust buffer sizes in a speculative and proactive way. Specifically, we use the feedback control theory to design a closed-loop buffer management subsystem, where the loop is closed by feeding back slowdowns. Furthermore, our approach attempts to dynamically balance the memory usage when both memory-intensive and I/O-intensive applications are in execution, in turn minimizing the mean slowdown. Compared with existing buffer management subsystems in Linux, the buffer feedback control mechanism encompasses the following two appealing features.

First, there is no need to determine the optimal bounds for buffer cache sizes. Second, the feedback control mechanism relaxes the requirement on a known and predictable workload.

6.2 A Feedback Control Algorithm

In this section, a general algorithm is presented so that the buffer size of each node in a cluster can be dynamically manipulated in accordance with the unpredictable workload. The main goals of the algorithm are:

(1) To improve the buffer utilization and the buffer hit rate; and

(2) To reduce the number of page faults by maintaining an effective usage of memory space for running jobs and their data.

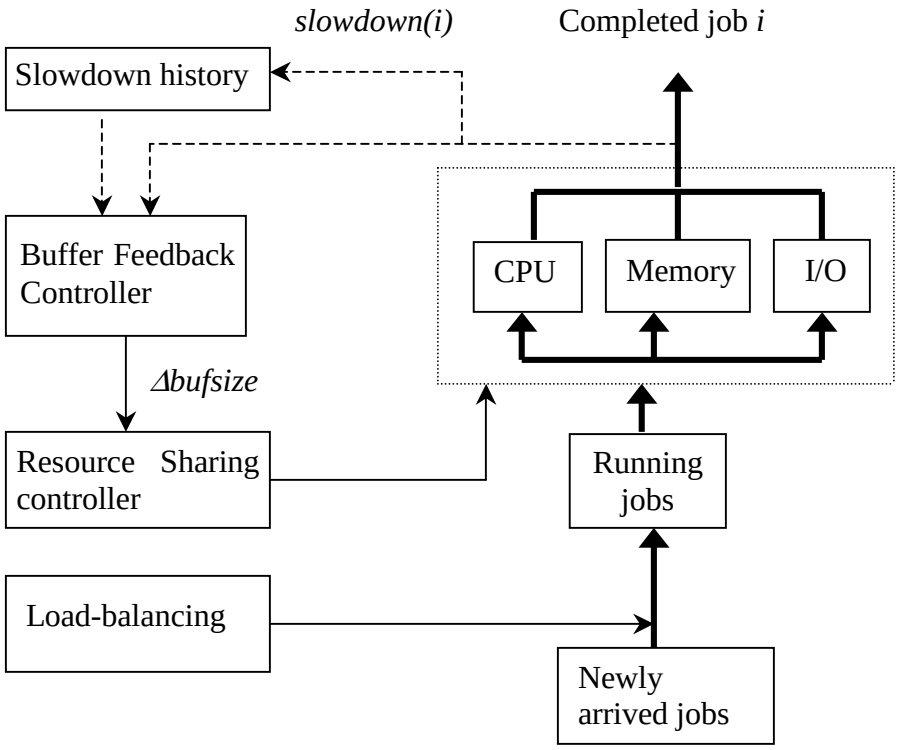


Figure 6.1 Architecture of the feedback control mechanism

A high level view of the architecture for one node of a cluster is presented in Figure 6.1. The above architecture comprises a load-balancing scheme, a resources-sharing controller, and a buffer feedback controller that is referred to as *BFC*. The resource-sharing controller consists of a CPU scheduler, a memory allocator and an I/O controller. The slowdown of a newly completed job and the history slowdowns are fed back to the BFC controller. The BFC then determines the required control action *ΔbufSize*. (*ΔbufSize > 0* means the buffer size need to be increased, and *ΔbufSize < 0* indicates that the buffer size required to be decreased.)

The BFC mechanism makes noticeable impacts on both the memory allocator and I/O controller, which in turn affect the number of page faults and buffer hit rate, on which the overall performance heavily depends.

We can derive the slowdown based on a model that captures the correlation between the buffer size and the slowdown. For simplicity, the model can be constructed as follows,

$$slowdown(z) = -bg(L)bufsize(z) + bd(L), \tag{6.1}$$

where *bg(L)* and *bd(L)* are the buffer size gain factor and disturbance factor under workload *L*, respectively. The control rule for buffer sizes is formulated as,

$$\Delta bufsize_u = G_b(\bar{s}_{u-1} - \bar{s}_u)\frac{\Delta bufsize_{u-1}}{|\Delta bufsize_{u-1}|}, \tag{6.2}$$

where $\Delta bufsize_u$ is the control action, $\Delta bufsize_{u-1}/|\Delta bufsize_{u-1}|$ indicates whether the previous control action has increased or decreased the resource weight, and C_b denotes the controller gain. G_w is tuned to be 0.5 in order to deliver better performance. Let $bufsize_{u-1}$ be the current buffer size, the buffer size is calculated as $bufsize_u = bufsize_{u-1} + \Delta bufsize_u$.

As can be seen from Figure 6.2, the feedback control generates control action *Δbufsize*. The adaptive buffer size makes noticeable impacts on both the memory allocator and I/O controller, which in turn affect the overall performance (See Figure 6.1). The feedback

controller generates a control action *Δbufsize* based on the previous control action along with the comparison between $\bar{s}_u$ and $\bar{s}_{u-1}$. Specifically, $\bar{s}_{u-1} > \bar{s}_u$, means the performance is improved by the previous control action, thereby increasing the buffer size if it has been increased by the previous control action, otherwise the buffer size is reduced. Likewise, $\bar{s}_{u-1} < \bar{s}_u$, indicates that the latest buffer control action leads to a worse performance, implying that the buffer size has to be increased if the previous control action has reduced the buffer size, otherwise the controller decreases the buffer size.

The extra time spent in performing feedback control is negligible and, therefore, the overhead introduced by the feedback control mechanism is ignored in our simulation experiments. The reason is because the complexity of the mechanism is low, and it takes a constant time to make a feedback control decision.

6.3 Experiments and Results

To evaluate the performance of the proposed load-balancing scheme with a feedback control mechanism, we have conducted a trace-driven simulation, in which the performance metric used is slowdown that is defined earlier in expression 3.22. Assume a cluster with six identical nodes, to which a disk-IO aware load-balancing policy is applied. The average page-fault rate and I/O access rate are chosen to be 2.5 No./MI (Number/ Million Instructions) and 3.5 No. /MI, respectively. The total memory size for each node is 640Mbyte, and other parameters of the cluster are given in Table 3.2. In the traces used in our experiments, the CPU and memory demands are the same as those used in [Zhang et al., 2000], and we add a randomly generated I/O access rate to each job. Without loss of generality, we assume that the buffer sizes of six nodes are identical. We have evaluated the performance of the following load-balancing policies:

(1) CM: the CPU-memory-based load-balancing policy [Xiao et al., 2002] without using buffer feedback controller. If the memory is imbalanced, CM assigns the newly

arrived job to the node that has the least accumulated memory load. When CPU load is imbalance and memory load is well balanced, CM attempts to balance CPU load.

(2) IO: the I/O-based policy [Qin et al., ICPPW2003][Surdeanu et al., 2002] without using the BFC algorithm. The IO policy uses a load index that represents only the I/O load. For a job arriving in node *i*, the IO scheme greedily assigns the job to the node that has the least accumulated I/O load.

(3) WAL: the Weighted Average Load-balancing scheme without the feedback controller.

(4) CM-FC: the CPU-memory-based policy in which the feedback controller is incorporated.

(5) IO-FC: the I/O-based load-balancing policy to which the feedback controller is applied.

(6) WAL-FC: the Weighted Average Load-balancing scheme with the feedback control mechanism.

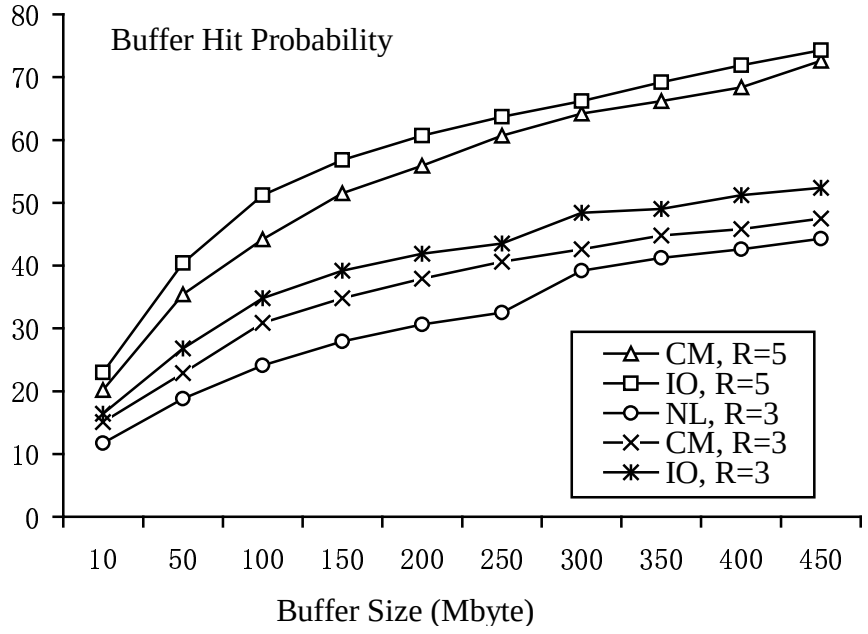


Figure 6.2 Buffer Hit Probability vs. Buffer Size. Page-fault rate is 5.0 No./MI, and I/O access rate is 2.75 No./MI

6.3.1 Buffer Hit Rate

To study feedback control mechanism, we have simulated a cluster with 6 nodes where round-robin scheduling is employed. The simulated cluster system is configured with parameters listed in Table 3.2.

Since buffer can be used to reduce the disk I/O access frequency, we have used equation 3.5 to approximately model the buffer hit probability of I/O access for a running job. Note that the I/O buffer of a node is a resource shared by multiple jobs in the node, and the buffer size a job can obtain at run time largely depends on the jobs' access patterns, characterized by I/O access rate and average data size of I/O accesses.

Figure 6.2 shows the effects of buffer size on the buffer hit probabilities of the NLB, CM and IO policies. When buffer size is smaller than 150Mbyte, the buffer hit probability increases almost linearly with the buffer size. The increasing rate of the buffer hit probability drops when the buffer size is greater than 150Mbyte, suggesting that further increasing the buffer size can not significantly improve the buffer hit probability when the buffer size approaches to a level at which a large portion of the I/O data can be accommodated in the buffer. *R* in Figure 6.2 represents data re-access rate, which is defined to be the average number of times the same data is repeatedly accessed by job *j* (See Equation 3.5).

6.3.2 Performance under Memory-Intensive Workloads

To simulate a memory intensive workload, the I/O access rate is fixed to a comparatively low level of 0.125 No./MI. The page-fault rate is set from 9.0 No./MI to 11.0 No./MI with increments of 0.25 No./MI. The performances of CM and CM-FC are omitted, since they are very close to those of WAL and WAL-FC.

Figure 6.3 reveals that the mean slowdown of all the policies increase with the page-fault rate. This is because as I/O demands are fixed, high page-fault rate leads to a high utilization of disks, causing longer waiting time on I/O processing.

In a scenario where the page-fault rate is high, WAL outperforms the IO scheme, and the WAL-FC has better performance than IO-FC. For example, the WAL policy reduces slowdowns over the IO policy by up to 30.6% (with an average of 23.7%), and the WAL-FC policy improves the performance in terms of mean slowdown over IO-FC by up to 39.7% (with an average of 32.0%). The reason is that the IO and IO-FC policies only attempt to balance explicit I/O load, ignoring the implicit I/O load that resulted from page faults. When the explicit I/O load is low, balancing explicit I/O load does not make a significant contribution to balancing the overall system load.

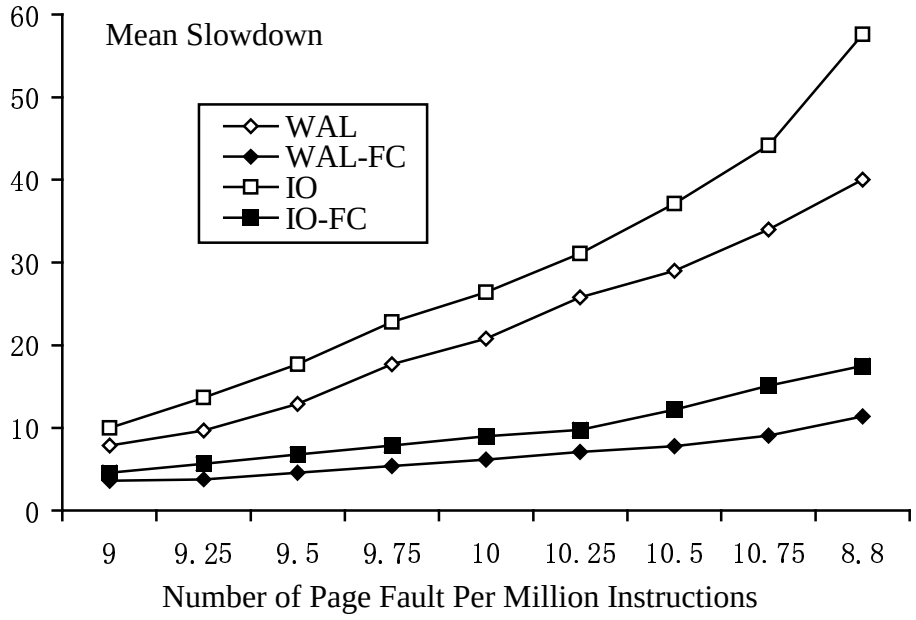


Figure 6.3 Mean slowdowns vs. page-fault rate. To simulate a memory intensive workload, the I/O access rate is fixed to a comparatively low level of 0.125 No./MI.

More interestingly, the policies that use the feedback control mechanism algorithm considerably improve the performance over those without employing the feedback controller. For example, WAL-FC reduces the slowdown of WAL by up to 73.2% (with an average of 67.7%), and IO-FC experiences a decrease of mean slowdown over the IO

policy by up to 69.6% (with an average of 64.9%). Consequently, the slowdowns of WAL, and IO are more sensitive to the page-fault rate than WAL-FC, and IO-FC.

6.3.3 Performance under I/O-Intensive Workloads

To stress the I/O-intensive workload in this experiment, the I/O access rate is fixed at a high value of 3.5 No./MI, and the page-fault rate is chosen from 2.0 No./MI to 2.625 No./MI with increments of 0.125 No./MI. This workload imply that, even when the requested memory space is larger than the allocated memory space, page faults do not occur frequently due to high temporal and spatial locality of access.

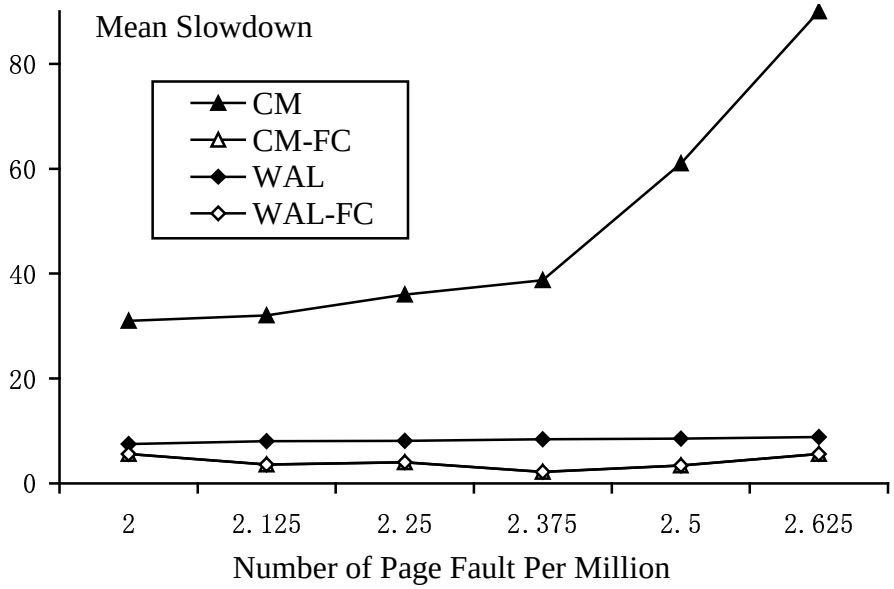


Figure 6.4 Mean slowdown vs. page-fault rate. To stress the I/O-intensive workload, the I/O access rate is fixed at a high value of 3.5 No./MI.

Figure 6.4 plots slowdown as a function of the page-fault rate. The results of IO and IO-FC are omitted from Figure 6.4, since they are nearly identical to those of WAL and WAL-FC. The mean slowdown of WAL-FC is almost identical to that of CM-FC. The reason that the performance of the WAL-FC policy is similar to that of the CM-FC policy

might be explained by the fact that the feedback controller increases the buffer sizes to a high level at which the buffer hit probability approximately approaches 100%. Hence, the CM-FC policy does not suffer from the imbalanced I/O load, which is alleviated by the high buffer hit rate.

First, the results show that the WAL scheme significantly outperforms the CM policies, indicating that CM is not suitable for an I/O intensive workload. For example, as shown in Figure 6.4, WAL reduces the mean slowdown of CM by up to 90.2% (with an average of 80.5%). This is because the CM policies only balance CPU and memory load, ignoring the imbalanced I/O load of clusters under the I/O intensive workload.

Second, Figure 6.4 shows that CM-FC and WAL-FC significantly outperform CM and WAL, respectively. For example, CM-FC reduces the slowdown of CM by up to 94.4% (with an average of 90.3%) and WAL-FC decreases the slowdown of WAL by up to 73.8% (with an average of 50.2%). Again, this is because CM-FC, and WAL-FC apply the feedback controller to meet the high I/O demands by changing the weights and the I/O buffer sizes to achieve a high buffer hit probability.

Third, Figure 6.4 shows that the performance of CM-FC is even noticeably better than WAL. This is because, as mentioned earlier, if the buffer hit rate is close to 100% due to the dynamic configuration of the buffer sizes, the imbalanced I/O load may not have a negative impact on CM-FC. However, this is not a case for the CM policy, since the I/O load in the cluster is extremely imbalanced. This result suggests that improving the I/O buffer utilization by using the buffer feedback control algorithm can potentially alleviate the performance degradation resulted from the imbalanced I/O load.

Finally, the results further show the slowdowns of CM are very sensitive to the page-fault rate. In other words, the mean slowdowns of CM all increase noticeably with the increasing value of I/O load. One reason is, as I/O load are fixed, a high page-fault rate leads to high disk utilization, causing longer waiting time on I/O processing. A second reason is, when the I/O load is imbalanced, the explicit I/O load imposed on some node will be very high, leading to a longer paging fault processing time. Conversely, the page-fault rate makes

insignificant impact on the performance of CM-FC, WAL, and WAL-FC. Since the high I/O load imposed on the disks is diminished either by balancing the I/O load or by improving the buffer utilization. This observation suggests that the feedback control mechanism is capable of boosting the performance of clusters under an I/O-intensive workload even in the absence of any dynamic load-balancing schemes.

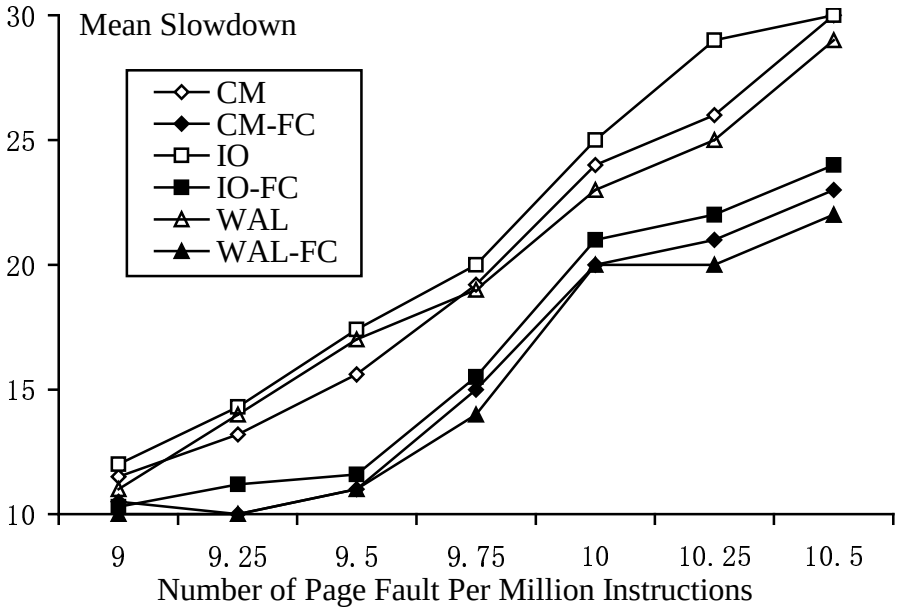


Figure 6.5 Mean slowdowns vs. page-fault rate. Simulate workloads with both high memory and I/O intensive jobs. I/O access rate is o of 1.875 No./MI.

6.3.4 Performance under Memory- and I/O-Intensive Workloads

The two previous sections presented the best cases for the proposed scheme since the workload was either highly memory-intensive or I/O-intensive but not both. In these extreme scenarios, the feedback control mechanism provides more benefits to clusters than load-balancing policies do. This section attempts to show another interesting case in which the cluster has a workload with both high memory and I/O intensive jobs. The I/O access

rate is set to 1.875 No./MI. The page fault rate is from 9.0 No./MI to 10.5 No./MI with increments of 0.25 No./MI.

Figure 6.5 shows that the performances of CM, IO, and WAL are close to one another, whereas the performance of CM-FC is similar to those of IO-FC and WAL-FC. This is because the trace, used in this experiment, comprises a good mixture of memory-intensive and I/O-intensive jobs. Hence, while CM and CM-FC take advantage of balancing CPU-memory load, IO and IO-FC can enjoy benefits of balancing I/O load. Interestingly, under this specific memory and I/O intensive workload, the resultant effect of balancing CPU-memory load is almost identical to that of balancing I/O load.

A second observation is that incorporating the feedback control mechanism in the existing load-balancing schemes is able to further boost the performance. For example, compared with WAL, WAL-FC further decreases the slowdown by up to 27.3% (with an average of 21.2%), and IO-FC further decreases the slowdown of IO by up to 24.7% with an average of 20.4%. This result suggests that to sustain a high performance in clusters, compounding a feedback controller with an appropriate load-balancing policy is desirable and strongly recommend.

Third, we also notice that under the memory and I/O intensive workload, load-balancing schemes achieve higher level of improvements over the cluster that does not consider load balancing. The reason is that when both memory and I/O demands are high, the buffer sizes in a cluster are unlikely to be changed, as there is a memory contention among memory-intensive and I/O-intensive jobs. Thus, instead of fluctuating widely to optimize the performance, the buffer sizes finally converge to a value that minimizes the mean slowdown.

6.4 Summary

It is conventional wisdom that the static configuration of buffer size bounds can be used to tune the performance of clusters when workload conditions can be modeled and predicted.

However, this approach performs poorly for circumstances under which workloads are unknown in advance. To solve this problems, in this chapter we have proposed a feedback control mechanism to dynamically adjust the weights of recourses and the buffer sizes in a cluster with a general and practical workload that includes memory and I/O intensive load.

The primary objective of the proposed mechanism is to minimize the number of page faults for memory-intensive jobs while improving the buffer utilization of I/O-intensive jobs. The feedback controller judiciously configures I/O buffer sizes to achieve an optimal performance. Thus, under a workload where the memory demand is high, the buffer sizes are decreased to allocate more memory for memory-intensive jobs, thereby leading to a low page-fault rate. The feedback controller was incorporated in three existing load-balancing schemes.

To evaluate the performance of the mechanism, we compared WAL-FC, CM-FC, and IO-FC with WAL, CM, and IO. A trace-driven simulation provides extensive empirical results demonstrating that the proposed algorithm is not only effective in enhancing performance of existing dynamic load-balancing algorithms, but also capable of significantly improving system performance under a memory-intensive or I/O-intensive workload even in the absence of any dynamic load-balancing schemes. In particular, Results from a trace-driven simulation for memory-intensive workloads show that compared with the two existing schemes with fixed buffer sizes, the feedback controller reduces the mean slowdown by up to 73.2% (with an average of 65.9%). In addition, when the workload is I/O-intensive, the proposed scheme reduces the mean slowdown from the CPU-memory-based policy by up to 94.4% (with an average of 90.3%).

Additionally, we have made the following observations:

(1) When the page-fault rate is high and the I/O rate is very low, WAL-FC and CM-FC have better performance than IO-FC;

(2) When I/O demands are high, WAL-FC, IO-FC, and CM-FC have noticeably better performance than that of IO;

Future studies in this issue may be performed in several directions. First, the feedback control mechanism will be implemented in a cluster system. Second, we will study the stability of the proposed feedback controller. Finally, it will be interesting to study how quickly the feedback controller converges to the optimal value in clusters.

Chapter 7

Load Balancing for Communication-Intensive Parallel Applications

In the previous chapters, we have tackled the issue of balancing I/O load in terms of disk storage, and the proposed disk-I/O-aware load-balancing schemes can meet the needs of a cluster system with high disk I/O demands. Although the approaches presented in Chapters 3, 4, and 5 take into account network I/O load as a measure to determine the migration cost, balancing network I/O load to improve network usability has not been addressed in the previous chapters or in the literature to the best of our knowledge. For this reason, we propose in this chapter a communication-aware load balancing technique (COM-aware for short) that is capable of improving the performance of communication-intensive applications by increasing the effective utilization of network bandwidth in cluster environments.

The rest of the chapter is organized as follows. In the section that follows, the motivation for this research is presented. Section 7.2 introduces an application behavioral model and the system model to capture the requirements for applications and the simulated cluster environment, and section 7.3 focuses on a communication-aware load-balancing scheme. Section 7.4 introduces the simulation model and methods for performance evaluations, while Section 7.5 discusses performance measurements of the proposed

scheme by simulating a cluster of 32 nodes with both synthetic bulk synchronous and real communication-intensive parallel applications. Finally, Section 7.6 summarizes the main contributions of this chapter.

7.1 Motivation

Scheduling [Spring and Wolski, 1998] and load balancing [Qin et al., Cluster2003] are two key techniques used to improve the performance of clusters for scientific applications by fully utilizing machines with idle or under-utilized resources. A number of distributed load-balancing schemes for clusters have been developed, primarily considering a variety of resources, including CPU [Harchol-Balter and Downey, 1997], memory [Acharya and Setia, 1999], disk I/O [Qin et al., ICPPW2003], or a combination of CPU and memory [Xiao et al., 2002]. These approaches have been proven effective in increasing the utilization of resources in clusters, assuming that networks connecting nodes are not potential bottlenecks in clusters. However, most of existing load balancing schemes ignore network resources, leaving open the opportunity for significant performance bottleneck to form for communication-intensive parallel applications. This is because a recent study demonstrated that the need to move data from one component to another in clusters is likely to result in a major performance bottleneck [Feng et al., 2003], indicating that data movement and message passing through the interconnection network of a cluster can become primary bottlenecks [Cho, 1999]. Thus, if the opportunity for improving effective bandwidth of networks is fully exploited, the performance of parallel jobs executing on clusters can be enhanced.

A large number of scientific applications have been implemented for executions on clusters and more are underway. Many scientific applications are inherently computationally and communicationally intensive [Cypher et al., 1993]. Examples of such applications include 3-D perspective rendering, molecular dynamics simulation, quantum chemical reaction dynamics simulations, and 2-D fluid flow using the vortex method, to

name just a few [Cypher et al., 1993]. The above bottleneck becomes more severe if the resources of a cluster are time-/space-shared among multiple scientific applications and communication load is not evenly distributed among the cluster nodes. Furthermore, the performance gap between CPU and network continues to widen at a faster pace, which raises a need of increasing the utilization of networks on clusters using various techniques.

The main motivation for our study is to improve the efficiency and usability of networks in cluster computing environments with high communication demands. The effort to improve the utilization of networks can be classified into hardware and software approaches. In this chapter we focus on an approach designed at software level. In particular, we develop a communication-aware load-balancing scheme (referred to as *COM-aware*) to achieve high effective bandwidth communication without requiring any additional hardware.

To facilitate the proposed load-balancing scheme, we introduce a behavioral model for parallel applications to capture the typical characteristics of various requirements of CPU, memory, network, and disk I/O resources. The proposed load-balancing scheme can make full use of this model to quickly and accurately determine the load induced by a variety of parallel applications. Our approach can substantially improve the performance of parallel applications running on a large-scale, time-/space-shared cluster, where a number of parallel jobs share various resources. COM-aware enables a cluster to utilize most idle or under-utilized network resources while keeping the usage of other type of resources reasonably high.

Typical load-balancing approaches that only focus on maximizing the utilization of one type of resource or another are not sufficient for a wide range of applications. Increasing evidence shows that high-performance computing requires clusters to be capable of executing multiple types of applications submitted by users [Surdeanu et al., 2002] simultaneously. For this reason, the COM-aware scheme is designed in a more sophisticated way that delivers high performance under a large variety of workload scenarios to provide one possible solution to the problem.

7.2 System Models and Assumptions

Since communication and I/O demands of applications may not be known in advance, in this section we introduce an application model, which aims at capturing the typical characteristics of communication, disk I/O, and CPU activity within a parallel application. The parameters of the model for a parallel application can be obtained by repeatedly executing the application off-line for multiple inputs. Alternatively, a light-weight and on-line profiling can be used to monitor the application behavior, thereby updating the model accordingly. Note that this application model is used in our trace-driven simulation.

7.2.1 The Application Model

Because the *Message Passing Interface* (MPI) programming environment simplifies many complex issues for scientific application development, application designers and developers have more or less adopted the MPI programming model [Vetter and Mueller, 2002][Vetter and Yoo, 2002] as the *de facto* standard for parallel programming on clusters. Consequently, the development of numerous scientific applications for clusters has been largely based on MPI [Grop and Lusk, 2001]. In the MPI-based application model, large data sets are decomposed across a group of nodes in a cluster. Processes (the terms process and task are used interchangeably throughout this paper) in a parallel application communicate with one another through the message-passing interface (Figure 7.1).

In this study we focus on a bulk-synchronous style of communication, which has been used in the popular Bulk-Synchronous Parallel model [Dusseau et al., 1996] [Ryu and Hollingsworth, 2000][Valiant, 1990] and conforms with the MPI style of inter-process communication patterns. This model is good fit with a variety of bulk-synchronous parallel applications, especially MPI-based parallel programs, in which a master creates a group of slave processes across a set of nodes in a cluster (Figure 7.1a). The slave processes concurrently compute during a computation phase, and then processes will be synchronized

at a barrier so that messages can be exchanged among these processes during the communication phase (Figure 7.1b).

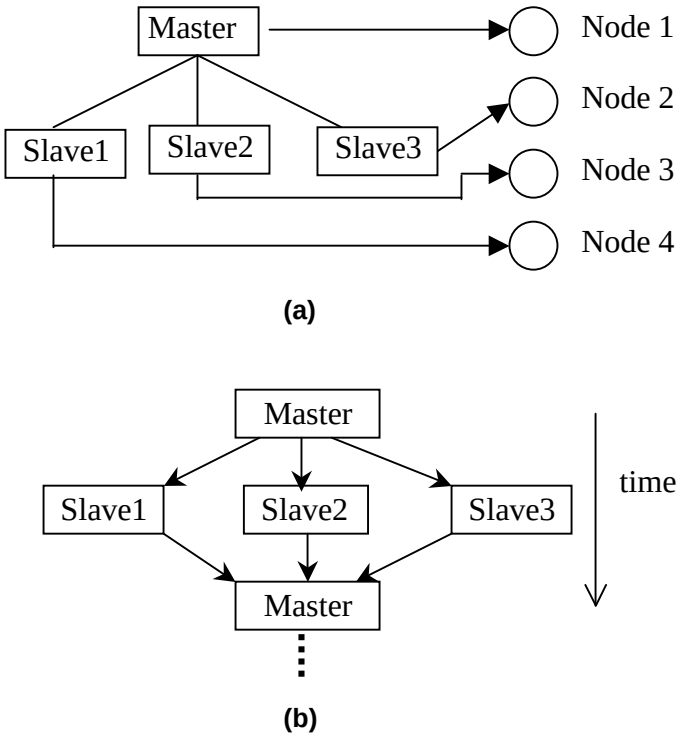


Figure 7.1 MPI-based application model. A master-Slave communication topology. (a) Four processes are allocated to four nodes, (b) The master periodically communicates to and from the slaves.

To design a load-balancing scheme that improves the effective usage of network resources in a cluster, we have to find a way to quantitatively measure the communication requirements of parallel applications. Based on the communication requirements of all the applications running on the cluster, we can approximately estimate the communication load of each node.

In what follows, we introduce an application behavioral model, which captures the requirements of CPU and network activities as well as that of disk I/O. This model is

important and desirable because the load-balancing scheme to be proposed shortly in Section 7.3 makes full use of this model to quickly calculate the load induced by parallel applications running on the system. Our model is reasonably general in the sense that it is applicable for both communication- and I/O-intensive parallel applications. Let N be the number of phases for a process (slave task) of a parallel job running on a node, and T_i be the execution time of the ith phase. T_i can be obtained by the following equation, where T^i_{CPU} , T^i_{COM} , and T^i_{Disk} are the time spent on CPU, communication, and disk I/O in the ith phase:

$$T^i = T^i_{CPU} + T^i_{COM} + T^i_{Disk} . \qquad (7.1)$$

Therefore, the total execution time of the process is estimated at $T = \sum_{i=1}^{N} T^i$. Let R_{CPU}, R_{COM}, and R_{Disk} be the requirements of CPU, communication, and disk I/O, and these requirements can be quantitatively measured as:

$$R_{CPU} = \sum_{i=1}^{N} T^i_{CPU} \ , \ R_{COM} = \sum_{i=1}^{N} T^i_{COM} \ , \text{ and } R_{Disk} = \sum_{i=1}^{N} T^i_{Disk} .$$

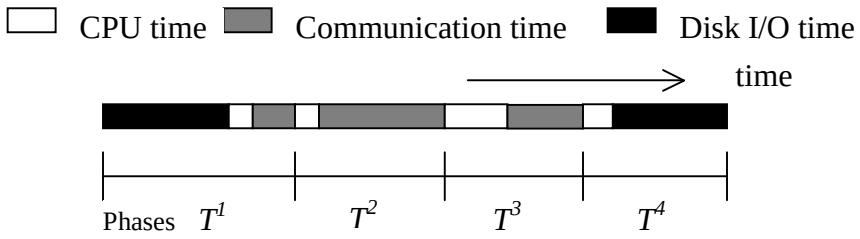


Figure 7.2 Example for the application behavior model. Number of phases is: N = 4.

An example of the above notation is illustrated in Figure 7.2, where the process has four phases. In the first phase, the application spends a large fraction of time in reading data from the disk. The second phase spends a small amount of time on computation and

the rest on communication. Thus, phase two is communication-intensive. The third phase is comprised of balanced CPU and communication requirements. The last phase is I/O-intensive in the sense that it spends a large fraction of time writing data to the disk.

We are now in a position to consider the execution time model for a parallel job (the terms job, application, and program are used interchangeably) running on a dedicated (non-shared) cluster environment. Given a parallel job with *p* identical processes, the execution time of the job can be calculated as:

$$T_p = s_p + \sum_{i=1}^{N} \left\{ MAX_{j=1}^{p} \left(T_{j,CPU}^{i} + T_{j,COM}^{i} + T_{j,Disk}^{i} \right) \right\} \qquad (7.2)$$

where $T_{j,CPU}^{i}$,$T_{j,COM}^{i}$, and $T_{j,Disk}^{i}$ denote the execution time of process *j* in the *i*th phase on three resources, respectively. The first term on the right-hand side of the equation represents the execution time of the sequential components including synchronization delay and computation in the master process; the second term corresponds to the duration of the parallel components. Let $s_{p,CPU}$, $s_{p,COM}$ and $s_{p,Disk}$. denote the sequential components' execution time on CPU, communication, and disk I/O accesses, we have:

$$s_p = s_{p,CPU} + s_{p,COM} + s_{p,Disk}. \qquad (7.3)$$

7.2.2 The Platform Model

Our goal is to model a non-dedicated cluster, where each job has a "home" node that it prefers for execution [Lavi and Barak, 2001]. A cluster in the most general form is comprised of a set of nodes each containing a combination of multiple types of resources, such as CPU, memory, network connectivity, and disks. Each node has a load balancer that is responsible for balancing the load of available resources. The load balancer periodically receives reasonably up-to-date global load information from resource monitor.

The network in our model provides full connectivity in the sense that any two nodes are connected through either a physical link or a virtual link. This assumption is arguably

reasonable for modern interconnection networks such as Myrinet [Boden et al., 1995] and InfiniBand [Wu et al., 2004]) that are widely used in high-performance clusters. Note that both Myrinet and Infiniband networks provide pseudo-full connectivity, allowing simultaneous transfers between any pair of nodes.

7.3 Communication-Aware Load Balancing

In this section, we propose an effective, dynamic communication-aware load-balancing scheme for non-dedicated clusters, where each node serves multiple processes in a time-sharing fashion so that these processes can dynamically share the cluster resources. To facilitate the new load-balancing technique, we need to measure the communication load imposed by these processes.

Let a parallel job be formed by p processes $t_0, t_1, \ldots, t_{p-1}$, and n_i the node to which t_i is assigned. Without loss of generality, we assume that t_0 is a master process, and t_j $(0 < j < p)$ is a slave process. Let $L_{j,p,COM}$ denote the communication load induced by t_j, which can be computed with the following formula, where $T^i_{j,COM}$ is the same as the one used in Equation (7.2).

$$L_{j,p,COM} = \begin{cases} \sum_{i=1}^{N} T^i_{j,COM} & j \neq 0, n_j \neq n_o, \\ 0 & j \neq 0, n_j = n_0, \\ \sum_{1 \leq k < p, n_k \neq n_j} \sum_{i=1}^{N} T^i_{k,COM} & j = 0. \end{cases} \tag{7.4}$$

The first term on the right hand side of the equation corresponds to the communication load of a slave process t_j when the process t_j and its master process are allocated to different nodes. The second term represents a case where the slave and its master process are running on the same node and, therefore, the slave process exhibits no communication load. Similarly, the third term on the right hand side of the equation (7.4) measures the

network traffic in and out of the master node, which is the summation of the communication load of the slave processes that are assigned to nodes different than the one executing the master.

Intuitively, the communication load on node *i*, $L_{i,COM}$, can be calculated as the accumulative load of all the processes currently running on the node. Thus, $L_{i,COM}$ can be estimated as follows:

$$L_{i,COM} = \sum_{\forall All\ j:n_j=i} L_{j,p,COM} \tag{7.5}$$

Given a parallel job arriving at a node, the load-balancing scheme attempts to balance the communication load by allocating the job's processes to a group of nodes with lower utilization of network resources. Before dispatching the processes to the selected remote nodes, the following two criterions must be satisfied to avoid useless migrations.

Criterion 1: Let nodes *i* and *j* be the home node and candidate remote node for process *t*, the communication load discrepancy between nodes *i* and *j* is greater than *t*'s communication load. We can formally express this criterion as:

$$\left(L_{i,COM} - L_{j,COM}\right) > L_{t,p,COM} \tag{7.6}$$

Criterion 1 guarantees that the load on the home node will be effectively reduced without making other nodes overloaded.

Criterion 2: Let nodes *i* and *j* be the home node and candidate remote node for process *t*, then the estimated response time of *t* on node *j* is less than its local execution. Hence, we have the following inequality, where R_t^i and R_t^j are the estimated response time of *t* on nodes *i* and *j*, respectively. $R_{t,mig}^{i,j}$ is the migration cost for process *t*.

$$R_t^j < R_t^i + R_{t,mig}^{i,j} \tag{7.7}$$

The migration cost, $R_{t,mig}^{i,j}$, is the time interval between the initiation and the completion of *t*'s migration, which is comprised of a fixed cost for running *t* at the remote node *j* and the process transfer time that largely depends on the process size and the available network bandwidth measured by the resource monitor.

```
if (L_i,COM = MAX (L_i,COM , L_i,CPU , L_i,Disk )) begin
    if (the submitted job is communication-intensive) begin
        node i makes an effort to balance the communication load;
    else if (the submitted job is I/O-intensive) begin
            node i keeps disk I/O resources well balanced;
        else if (the submitted job is CPU-intensive)
            balance CPU resources;
        end
    end
else if (the workload is memory-intensive) begin
        Balance memory resources;
    else if (the submitted job is I/O-intensive) begin
            node i keeps disk I/O resources well balanced;
        else if (the submitted job is CPU-intensive)
            balance CPU resources;
        end
    end
end
```

Figure 7.3 Pseudocode of the communication-aware load balancing scheme

It is a known fact that in practice clusters are likely to have a mixed workload with memory-, I/O-, and communication-intensive applications. Therefore, our communication-aware load-balancing approach is designed in such a way to achieve high performance under a wide spectrum of workload conditions by considering multiple application types. Specifically, to sustain a high and consistent performance when the communication load becomes relatively low or well balanced, our scheme additionally and simultaneously considers disk I/O and CPU resources. In other words, the ultimate goal of our scheme is to

balance three different resources simultaneously under a wide spectrum of workload conditions. In practice, a resource is likely to become a bottleneck if the fraction of the execution time spent on this resource is substantially higher than that on the other two resources. For this reason, our scheme envisions a type of resource that exhibits higher load than that of any other resources as the first-class resource, and the scheme attempts to first balance the first-class resource. Let $L_{i,CPU}$ and $L_{i,Disk}$ be the load of node i in terms of CPU and disk I/O activities. The pseudocode of our communication-aware scheme is presented in Figure 7.3.

7.4 Performance Evaluation

To evaluate the performance of the proposed load-balancing scheme, we have conducted extensive trace-driven simulations. This section describes our simulation model, workload conditions, and performance metrics.

We simulate a non-dedicated cluster with 32 nodes under a variety of workload conditions. The simulated cluster is configured by the various parameters listed in Table1. The parameters for CPU, memory, disks, and network are chosen in such a way that they resemble a typical modern day cluster.

Table 7.1 System Characteristics

Parameter	Values assumed
CPU Speed	1000 MIPS (million instructions/second)
RAM Size	1 Gbytes (Gegabytes)
Network Bandwidth (point to point)	1Gbps
Disk seek time and rotation time	8.0 ms
Disk transfer rate	40 Mbytes/Sec. (Megabytes/Second)
Time slice of CPU time sharing	10 ms
Context switch time	0.1 ms

7.4.1 Workload Scenarios

It is well understood that the performance of a cluster system is affected by the workload submitted to the system. Therefore, designing realistic workload scenarios plays an important role in our performance evaluation. To evaluate the performance impacts of the communication-aware load-balancing scheme, we extrapolate traces from those reported in [Harchol-Balter and Downey, 1997][Zhang et al., 2000].

To simulate a multi-user time-sharing environment, the traces contain a collection of parallel jobs. Without loss of generality, we first consider a bulk-synchronous style of communication (See Section 7.2.1), where the time interval between two consecutive synchronization phases is determined by the message arrival rate. The experimental results reported in Section 7.5 validate our argument that the communication-aware load-balancing scheme can also be applied to communication-intensive applications with a variety of communication patterns.

For jobs with communication requirements, messages issued by each task are modeled as a *Poisson Process* with a mean message arrival rate. The message size is randomly generated according to a Gamma distribution with a mean size of 512Kbyte, which reflects typical data characteristics for many applications, such as 3-D perspective rendering [Li and Curkendall, 1992].

To validate the results based on synthetic parallel applications, we simulate five real scientific applications [Cypher et al., 1993] which have various computation, memory, disk I/O, and communication requirements (See Section 7.5.5).

7.4.2 Performance Metrics

The goal of the proposed load-balancing scheme is to improve the performance of submitted jobs and, thus, we need to choose good metrics to intuitively capture the performance of jobs running on a cluster. As such, the first performance metric considered in our experiments is the job *turn-around time* [Cirne and Berman, 2003]. The turn-around time of a job is the time elapsed between the job's submission and its completion. The turn-

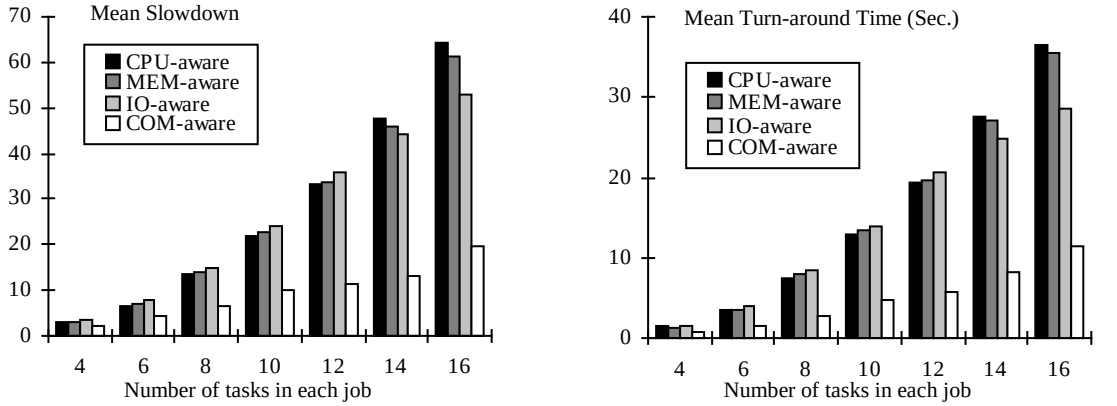
around time is a natural metric for the job performance due to its ability to reflect a user's view of how long a job takes to complete.

A second important performance metric to be introduced in our study is *slowdown.* The slowdown of a parallel job running on a non-dedicated cluster is a commonly used metric for measuring the performance of the job [Harchol-Balter and Downey, 1997]. The slowdown of a job is defined by: $S_p = T_p' / T_p$, where T_p' is the job's turn-around time in a resource-shared setting and T_p is the turn-around time of running in the same system but without any resource sharing. It is to be noted that T_p can be estimated by either using equation (7.2) given in Section 7.2.1 or offline execution using multiple inputs, and T_p' can be measured at run time.

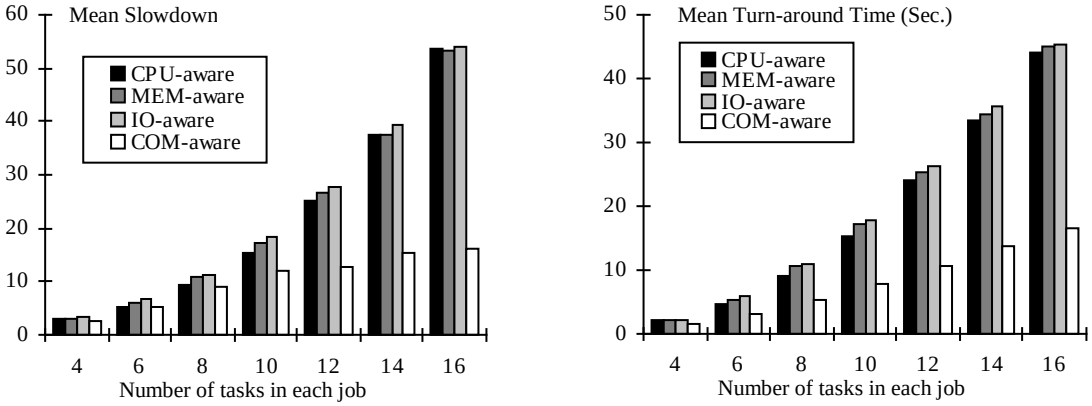
7.5 Experimental Results

To empirically evaluate the performance of the proposed communication-aware load-balancing scheme (referred to as *COM-aware*), we compare our scheme against three existing load-balancing policies that can achieve high CPU, memory, and disk I/O utilizations, respectively. We refer to these three approaches as CPU-aware [Harchol-Balter and Downey, 1997], MEM-aware [Zhang et al., 2000], and I/O-aware [Qin et al., Cluster2003].

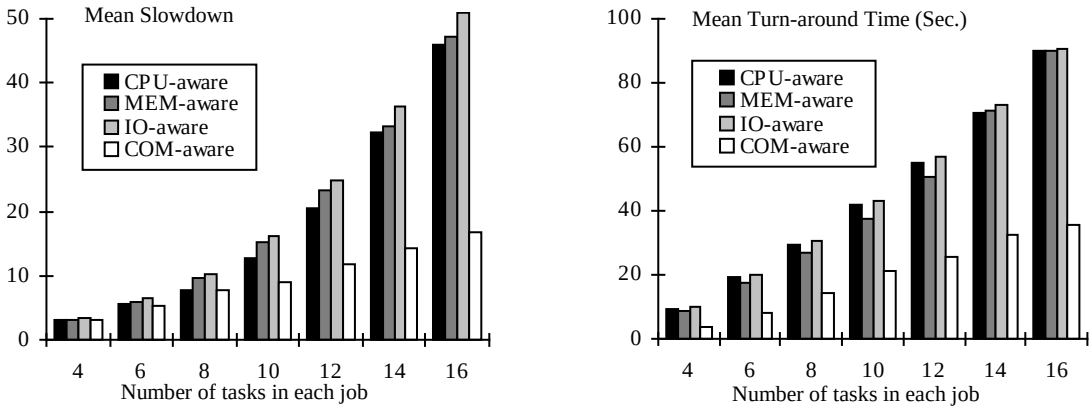
This section presents experimental results obtained from feeding a simulated cluster of 32 nodes with various traces that reflect a wide range of workload conditions. We make use of the simulated cluster to run five experiments. First, in Section 7.5.1, the performance under communication-intensive workload situations is studied. Next, in Sections 7.5.2 and 7.5.3, we investigate the impacts of network bandwidth and message size on the performance. In Section 7.5.4, we measure the performance of the cluster under mixed workload conditions with I/O-, memory-, and communication-intensive applications. Finally, in Section 7.5.5, we simulate several real communication-intensive parallel applications to validate the results from the synthetic application cases.



(a) Message arrival rate = 0.1 No./MI



(b) Message arrival rate = 0.3 No./MI



(c) Message arrival rate = 0.5 No./MI

Figure 7.4 Mean slowdown and mean turn-around time vs. number of tasks. Message arrival rate is (a) 0.1No./MI, (b) 0.3 No./MI, and (c) 0.5 No./MI.
Parallel jobs are running on a cluster of 32 nodes using four load-balancing schemes when the network bandwidth is 1Gbps, and the average message size is 512 Kbytes.

7.5.1 Performance under Communication-Intensive Workloads

In the first experiment we intend to stress the communication-intensive workload by setting the message arrival rate at values of 0.1 No./MI (Number/Million Instruction), 0.3 No./MI, and 0.5 No./MI, respectively. Both page fault rate and I/O access rate are set at a low value of 0.01 No./MI. This workload reflects a scenario where jobs have high communication-to-computation ratios.

Figure 7.4 plots the mean slowdown and turn-around time (measured in Seconds) as functions of the number of tasks in each parallel job, and each graph in Figure 7.4 reports the results for communication-intensive jobs running on a cluster of 32 nodes using four different load-balancing schemes. Figure 7.4 indicates that all the load-balancing schemes experience an increase in the mean slowdown and turn-around time when the number of tasks in jobs increases. This is because when CPU, memory, and disk I/O demands are fixed, the increasing number of tasks in each parallel job leads to high communication demands, causing longer waiting time on sending and receiving data.

Importantly, we observe from Figure 7.4 that COM-aware is significantly better than CPU-aware, MEM-aware, and I/O-aware under the communication-intensive workload situations. For example, when each parallel job consists of 16 tasks and the message arrival rate is 0.5 No./MI, the COM-aware approach improves the performance in terms of mean slowdowns over CPU-aware, MEM-aware, and I/O-aware by more than 174%, 182%, and 206%, respectively. We attribute this result to the fact that the CPU-aware, MEM-aware, and I/O-aware schemes only balance CPU, memory, and disk I/O resources, respectively. Consequently, these schemes totally ignore the imbalanced communication load resulting from those parallel applications that are communication-intensive.

A third observation drawn from Figure 7.4 is that, when most parallel applications running on the cluster are small-scale, the COM-aware scheme only marginally outperforms CPU-aware and MEM-aware. For example, if the number of tasks in each parallel job is 4 and the message arrival rate is set to 0.1 No./MI, the performance gains in terms of mean slowdown and turn-around time achieved by the COM-aware scheme are as

low as 3% and 4%, respectively. When parallel applications in the system scale up, the performance gap between COM-aware and the other three schemes is rapidly widening. The implication of this result is that large-scale parallel applications with high communication demands can greatly benefit from the proposed communication-aware load-balancing technique.

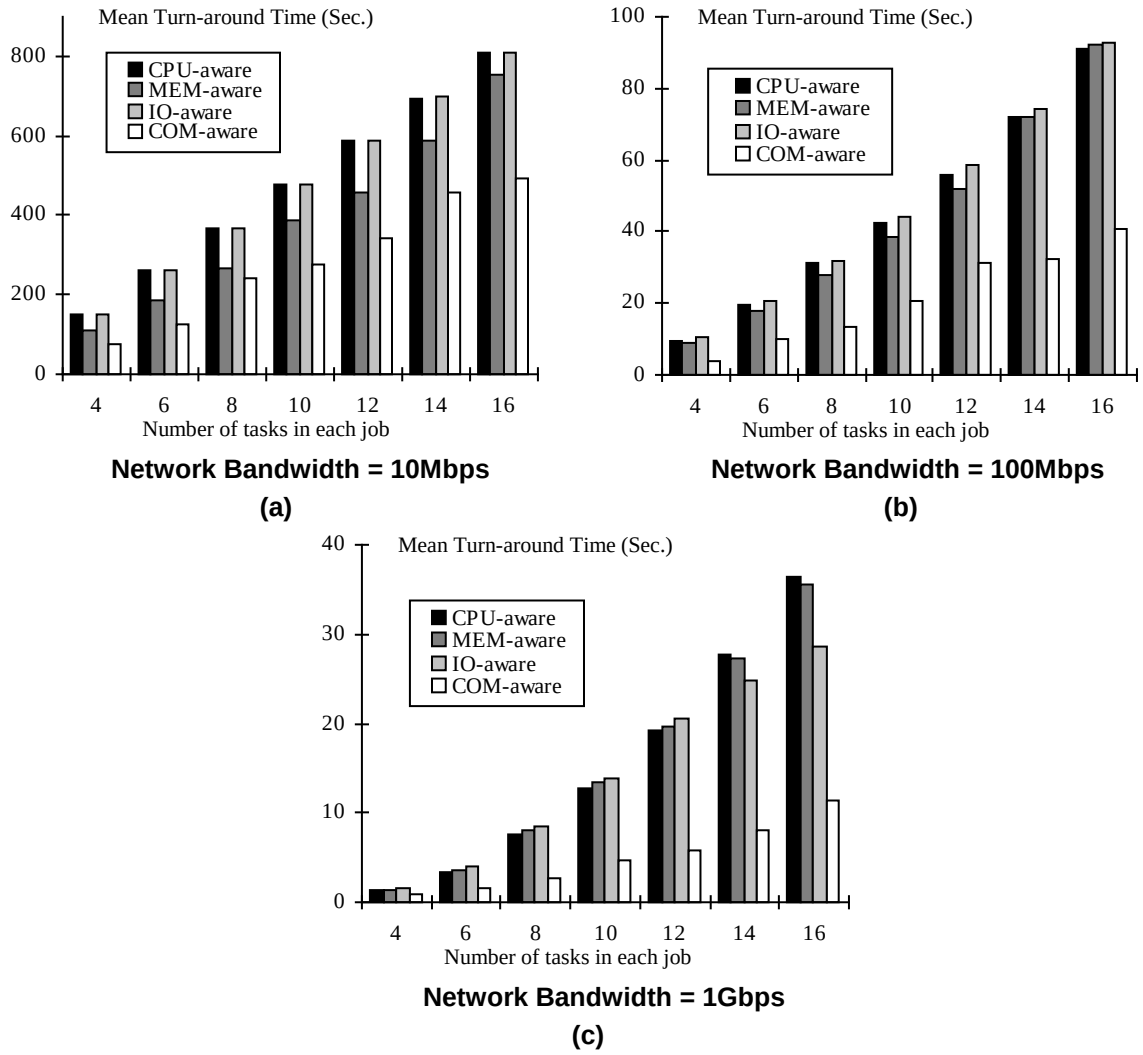


Figure 7.5 Mean turn-around time vs. network bandwidth. Network bandwidth is (a) 10Mbps, (b) 100Mbsp, and (c) 1Gbps. The mean turn-around time of parallel jobs running on a cluster of 32 nodes using four load-balancing policies when the message arrival rate is 0.1 No./MI and the average message size is 512 Kbytes.

7.5.2 Impact of Varying Network Bandwidth

While the first experiment assumes that all messages are transmitted over a network at an effective rate of 1Gbps, the second set of results is obtained by varying the network bandwidth of the cluster. As a result, we will investigate the impact of network bandwidth on the performance of the four load balancing schemes.

In order to explore this issue, we set the network bandwidth to 10Mbps, 100Mbps and 1Gbps, respectively. Since the mean slowdown has similar patterns as those of the mean turn-around time, Figure 7.5 only shows the performance with respect to the turn-around time for the CPU-aware, MEM-aware, I/O-aware, and COM-aware policies.

As shown in Figure 7.5, the mean turn-around time the four policies share a common feature in the sense that turn-around time is inversely proportional to network bandwidths. This result can be explained by the fact that when the communication demands are fixed, increasing the network bandwidth effectively reduces communication-intensive applications' time spent transmitting data. In addition, a high network bandwidth results in less synchronization time among tasks of a parallel job and low migration cost in these four load-balancing policies.

Figure 7.5 further reveals that the performance improvement achieved by the COM-aware scheme becomes more profound when the network bandwidth is very high. For example, if the network bandwidth of the cluster is set to 10Mbps, 100Mbps, and 1Gbps, the average performance improvements gained by COM-aware over the three existing policies are 61.4%, 116.1%, and 182.9%, respectively.

7.5.3 Impact of Varying Average Message Size

Communication load depends on message arrival rate and the average message size, which in turn depends on communication patterns. The purpose of this experiment, therefore, is to show the impact of average message size on the performance of the four load balancing schemes.

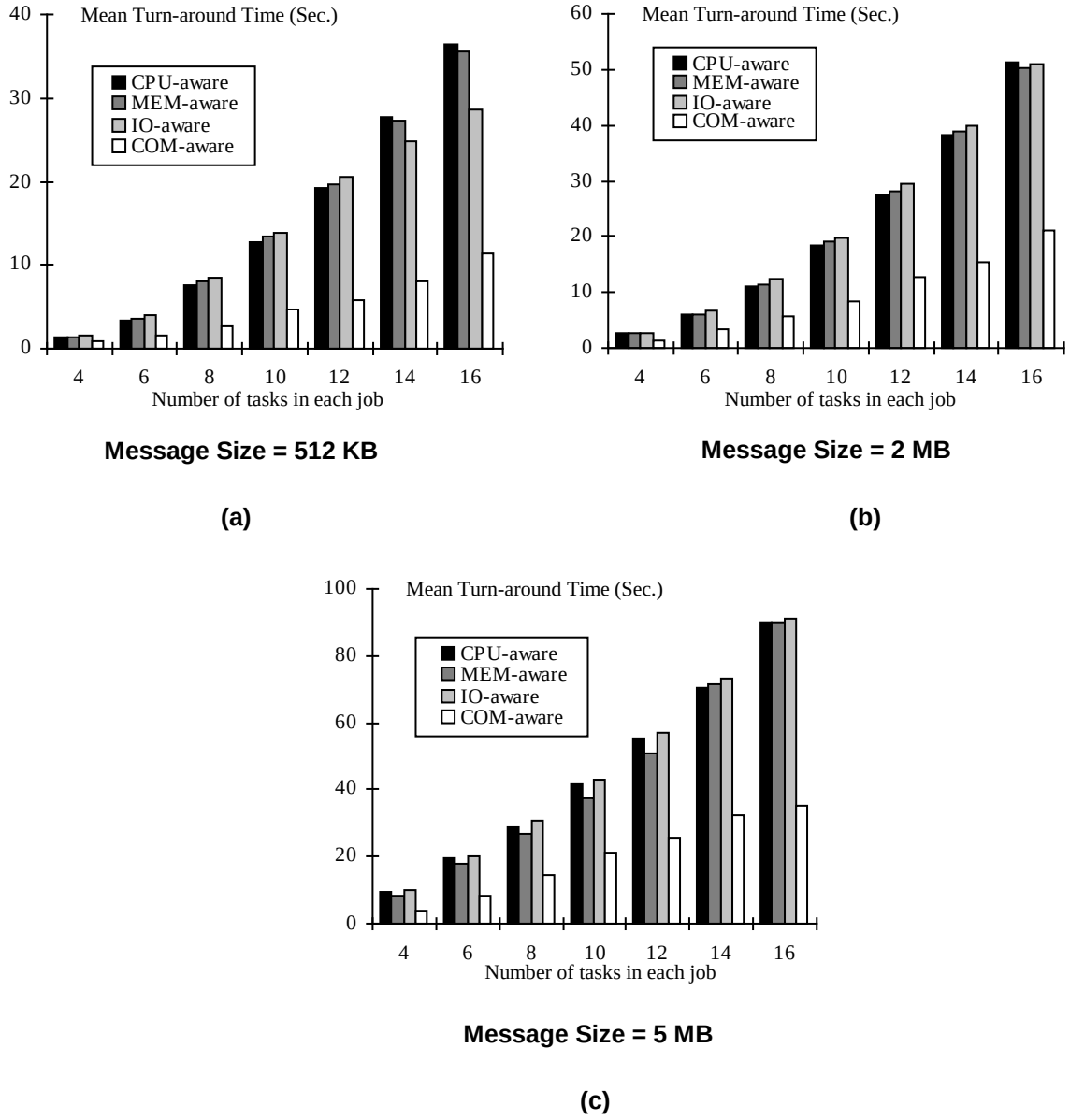


Figure 7.6 Mean turn-around time vs. message size. Average message size is (a) 512 KB, (b) 2MB, and (c) 5MB. The mean turn-around time of parallel jobs running on a cluster of 32 nodes using four load-balancing policies when the message arrival rate is 0.1 No./MI, and the network bandwidth is 1Gbps.

Figure 7.6 shows that the mean turn-around time increases as the average message size increases. The reason is that as message arrival rate is unchanged, a large average message size yields high network utilizations in the system, causing longer waiting times on message transmissions.

A second observation from Figure 7.6 is that the benefits of the COM-aware scheme become increasingly significant when communication-intensive applications running on the cluster tend to send and receive larger data volumes on the network. This is because the larger the average message size, the higher the network utilization, which in turn results in longer waiting time in network queues.

7.5.4 Performance under Workloads with a Mix of Job Types

In the results provided in Sections 7.5.1-7.5.3, memory- and I/O-intensive jobs are omitted because the focus of the pervious experiments is more on communication-intensive workload situations. However, the results obtained for workloads with only communication-intensive jobs are not always valid for clusters where multiple types of jobs are submitted. The reason is that realistic clusters may have to execute a mix of job types, and different job types in the systems affect one another. Therefore, in this section we measure the impact of the proposed scheme on a cluster of 32 nodes where memory- and I/O-intensive jobs are also submitted along with communication-intensive jobs.

Again, we compare the mean slowdown and turn-around time of COM-aware against those of the three existing load-balancing policies. The message arrival rate and network bandwidth in this experiment are set to 0.1 No./MI and 1Gbps.

Figure 7.7 shows the mean slowdown and turn-around time as functions of the percentage of communication-intensive jobs. It is noted that the eleven traces in Figure 7.7 contain a collection of I/O- and communication-intensive jobs. In general, the results show that our COM-aware policy performs the best in all cases. In particular, both COM- and I/O-aware noticeably outperform CPU-aware and MEM-aware when the workload is I/O-

intensive, indicating that COM-aware can maintain the same level of performance as the I/O-aware scheme for I/O-intensive applications.

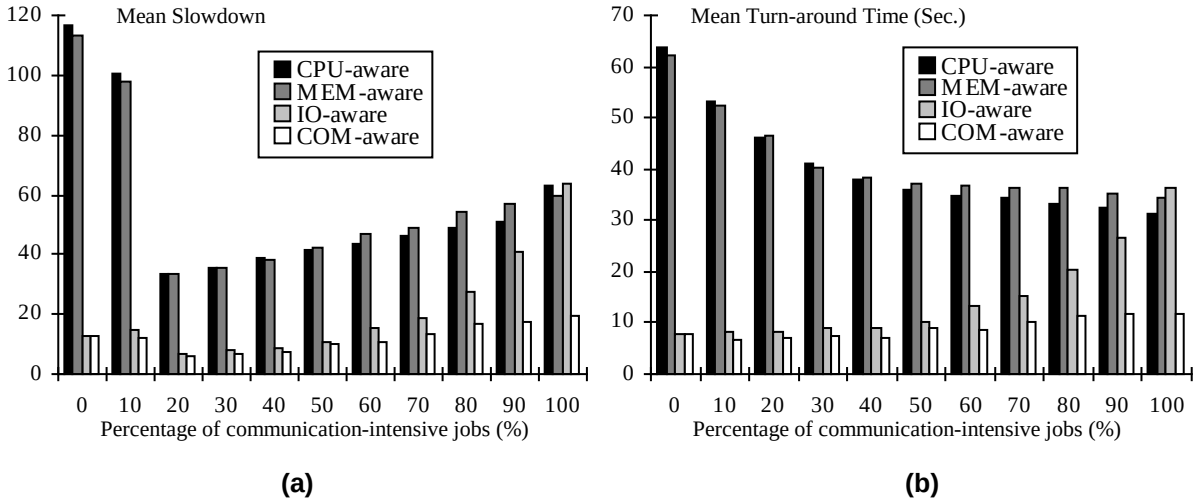


Figure 7.7 Performance vs. percentage of communication-intensive jobs. (a) The mean slowdown, and (b) the mean turn-around time of I/O- and communication-intensive jobs. For each graph, a cluster of 32 nodes makes use of four load-balancing policies when the message arrival rate is 0.1 No./MI and the network bandwidth is 1Gbps.

Interestingly, Figure 7.7 shows that the performance improvement of I/O-aware over CPU- and MEM-aware consistently decreases with the increase in the percentage of communication-intensive jobs. The reason is that I/O-aware does not view network devices as important components of clusters even when vast majority of running applications have high communication demands. In contrast, the mean slowdown and turn-around time of COM-aware are not sensitive to the percentage of communication-intensive jobs.

We now turn to study another mixed workload with memory- and communication-intensive jobs. Similarly, Figure 7.7 shows the mean slowdown and turn-around time as functions of the percentage of communication-intensive jobs. Again, the first observation is that the COM-aware policy performs the best in all cases. When the workload is memory-

intensive, the performance of COM-aware is very close to that of the MEM-aware scheme, and MEM-aware is better than CPU-aware and I/O-aware if the percentage of communication-intensive jobs is less than 40%, implying that the CPU-aware and I/O-aware schemes are not suitable for applications with high memory demands. This is because both MEM-aware and COM-aware take efforts to achieve high usage of global memory in the cluster.

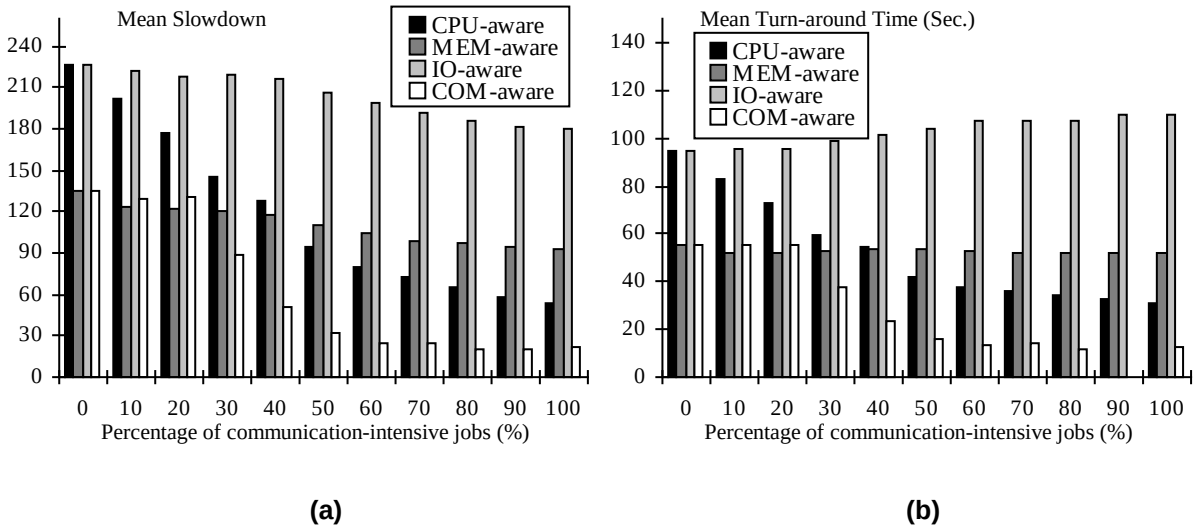


Figure 7.8 Performance vs. percentage of communication-intensive jobs. (a) The mean slowdown, (b) the mean turn-around time of memory- and communication-intensive jobs. For each graph, a cluster of 32 nodes makes use of four load-balancing policies when the message arrival rate is 0.1No./MI and the network bandwidth is 1Gbps.

The results also show that the performance of MEM-aware is worse than that of CPU- and COM-aware when the percentage of communication-intensive jobs is larger than 40%. The experimental results are expected because MEM-aware does not attempt to share network services among the different nodes in the cluster.

We have obtained similar results under other workload with a mix of CPU-, memory-, I/O-, and communication-intensive jobs. Due to the space limits, we present only the partial results in this section.

7.5.5 Experiments on Real Parallel Applications

To validate the experimental results based on the synthetic parallel application case, we simulate a number of real communication-intensive parallel applications [Cypher et al., 1993]. In this section we evaluate the impact of the COM-aware approach on five real communication-intensive applications, which have different computation, disk I/O, and communication patterns. Table 7.2 provides a description of these applications.

Table 7.2 Descriptions of Real Communication-intensive Applications

Application	Description
Render	This application integrates a 6000×6000 pixel 24-bit color image with 16-bits of elevation data per pixel to obtain 480×640 pixel 24-bit output images.
Molecular	This application is used to compute properties for liquids and polymers and to calculate conformational properties of organic molecules.
Semi	This application is a 3-D semiconductor device simulator that utilizes a parallelized 3-D Poisson solver.
React	This application is used to predict the behavior of chemical reactions based on first principles.
Vortex	This application models the evolution of vortices in a 2-dimensional fluid.
ALL	The workload of a cluster of 32 nodes where all the five applications are submitted.

To simulate the above communication-intensive applications, we generate six job traces where the arrival patterns of jobs are extrapolated based on the job traces collected from the University of California at Berkeley [Harchol-Balter and Downey, 1997]. Each of the five traces consists of one type of real application described in Table 7.2, whereas the

sixth trace is a collection of the five different types of applications. A 32-node cluster is simulated to run the applications.

Figure 7.9(a) shows the mean slowdowns of the real applications running on the cluster using four load-balancing policies. It is observed from Figure 7.9 that the COM-aware load-balancing approach works quite well for all communication-intensive applications. For example, in the case of single application executions the COM-aware scheme exhibits a performance improvement by up to 75.9% (with an average of 42.2%, See Figure 7.9(b)) in mean slowdown over the three existing policies. The performance gain of COM-aware is mainly attributed to the effective usage of network resources in addition to CPU, memory, and I/O load balancing in the cluster.

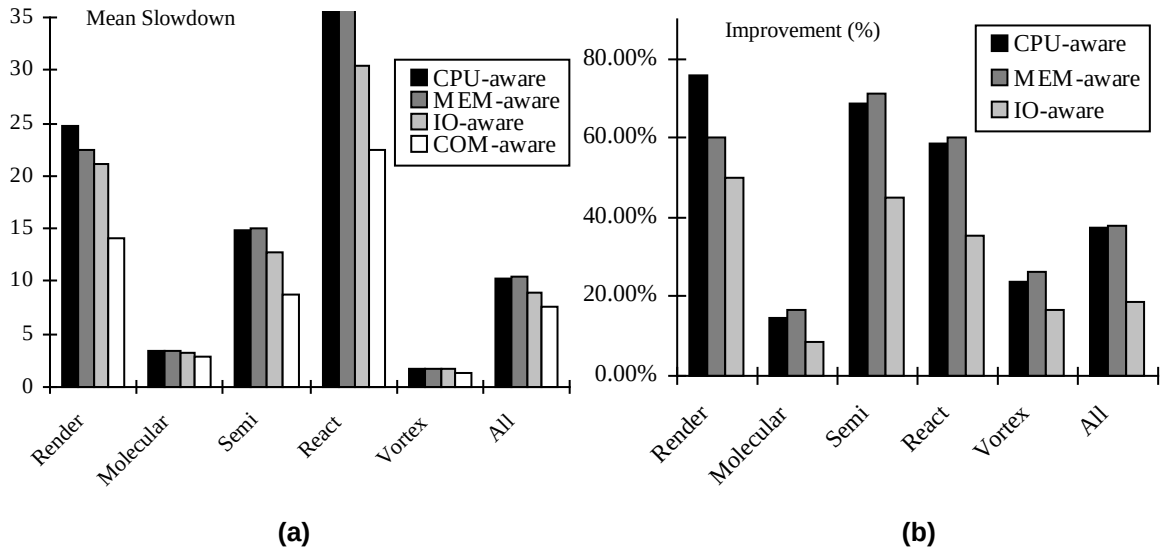


Figure 7.9 (a) The mean slowdowns of real applications. (b) performance improvements of COM-aware over CPU-, MEM-, and I/O-aware policies.
A cluster of 32 nodes makes use of four load-balancing policies.

7.6 Summary

This chapter introduced a behavioral model for parallel applications with large requirements of CPU, network, and disk I/O resources. The model is particularly beneficial to clusters where resource demands of applications may not be known in advance. Thus, load balancers can make full use of this model to quickly and accurately determine load induced by a variety of parallel applications.

Furthermore, we addressed the issue of improving effective bandwidth of networks on clusters at software level without requiring any additional hardware. Specifically, we have proposed a new communication-aware load-balancing scheme, referred to as *COM-aware*, for non-dedicated clusters where the resources of the cluster are shared among multiple parallel applications being executed concurrently. A simple yet efficient means of measuring communication load imposed by processes was presented. We reported the preliminary results based on trace driven simulations. The traces used in our study consist of both synthetic bulk synchronous and real parallel applications. The experimental results show that the *COM-aware* approach can improve the system performance over three existing schemes by up to 206% and 235%, in terms of slowdown and turn-around time respectively, under a workload with high communication demands.

As a future research direction, we plan to measure the performance of the proposed scheme based on more realistic communication-intensive applications. In addition, we intend to extend the approach to a more widely distributed environment such as a peer-to-peer system.

Chapter 8

Conclusions and Future Work

In this dissertation, we have developed a number of disk-I/O-aware and communication-aware load balancing schemes for data-intensive applications executing in a cluster-computing environment. 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 load-balancing support for data-intensive applications. Finally, Section 8.3 summarizes the results and their implications.

8.1 Main Contributions

Over the last ten years, cluster computing has become the fastest growing platform in high-performance computing. Due to the rapidly widening performance gap between processors and I/O systems in modern clusters, storage and network subsystems tend to become performance bottlenecks, which induce much more pronounced performance degradation for data-intensive applications, such as long running simulations, remote-sensing database systems, and biological sequence analysis. To alleviate the I/O bottlenecks, our research has investigated load-balancing policies that are capable of achieving high utilization for

disks and networks in addition to those of CPU and memory resources. In what follows, we summarize the main contributions of this dissertation.

8.1.1 A New I/O-aware Load-balancing Scheme

In the first phase of this study, we have developed a new I/O-aware load-balancing scheme that significantly improves the overall performance of clusters under a broad range of workload conditions [Qin et al., ICPPW2003]. The new scheme relies on a technique that allows a job's I/O operations to be conducted by a node that is different from the one in which the job's computation is assigned, thereby permitting the job to access remote I/O.

The proposed scheme dynamically detects I/O load imbalance among nodes of a system and determines whether to migrate the I/O requests of some jobs from overloaded nodes to other less- or under-loaded nodes, depending on data migration cost and remote I/O access overhead. Besides balancing I/O load, the scheme judiciously takes into account both CPU and memory load sharing in the system, thereby maintaining the same level of performance as the existing schemes when I/O load is low or well balanced.

Compared with the existing schemes that only consider CPU and memory, the proposed scheme significantly improves system performance by reducing mean slowdowns. Furthermore, the proposed scheme improves the performance in the mean slowdown over the existing approaches that only consider I/O. More importantly, our study shows that the new scheme improves over a very recent algorithm found in the literature that considers all the three resources mentioned above.

8.1.2 Preemptive Job Migrations

We have addressed the issue of whether preemptive job migrations can improve the performance of a cluster over non-preemptive schemes in the second phase of the study [Qin et al., SBAC-PAD2003]. Specifically, we have proposed a second load-balancing scheme where a running job is eligible to be migrated only if it is expected to improve the overall performance.

Unlike the existing I/O-aware load balancing schemes, the technique utilized in this scheme tackles the problem by considering both explicit I/O invoked by application programs and implicit I/O induced by page faults. The results from a trace-driven simulation show that, compared to the existing schemes that consider I/O without using preemptive job migrations, the proposed approach achieves the improvement in mean slowdown by a factor of up to 10.

8.1.3 Consideration of the Heterogeneity of Resources

Although it is reasonable to assume that a new and stand-alone cluster system may be configured with a set of homogenous nodes, there is a strong likelihood for upgraded clusters or networked clusters to become heterogeneous in terms of resources such as CPU, memory, and disk storage. This is because, to improve performance and support more users, new nodes that might have divergent behaviors and properties may be added to the systems or several smaller clusters of different characteristics may be connected via a high-speed network to form a bigger one.

Since heterogeneity in disks, when coupled with an imbalanced load of I/O and memory resources, inevitably imposes significant performance degradation, we have devised an approach for hiding the heterogeneity of resources, especially that of I/O resources, by judiciously balancing I/O work across all the nodes in a cluster [Qin et al., HiPC2003]. An extensive simulation study provides us with empirical results to show that the performance of the proposed policy is more sensitive to the changes in CPU and memory heterogeneity than the existing policies. Conversely, the results prove that our approach is less sensitive to the changes in disk I/O heterogeneity than non I/O-aware load balancing policies.

8.1.4 Support for Data-Intensive Parallel Applications

Most recently, we have extended our work in load balancing for sequential workloads by developing two simple yet effective I/O-aware load-balancing schemes for parallel jobs running on clusters [Qin et al. Cluster2003].

Each parallel job consists of a number of tasks, which are assumed to synchronize with one another. Each workstation serves several tasks in a time-sharing fashion so that the tasks can dynamically share the cluster resources. Extensive simulations show that the proposed schemes are capable of balancing the load of a cluster in such a way that CPU, memory, and I/O resources at each node can be simultaneously well utilized under a wide spectrum of workload conditions.

8.1.5 Improve Buffer Utilization

In addition to our research in issues related to load balancing, we have developed a feedback control mechanism to improve the performance of a cluster by adaptively manipulating the I/O buffer size [Qin et al., Euro-Par2003]. Results from a trace-driven simulation show that this mechanism is effective in enhancing the performance of a number of existing load-balancing schemes.

8.1.6 A Communication-Aware Load-Balancing Scheme

Clusters have emerged as a primary and cost-effective infrastructure for parallel applications, including communication-intensive applications that transfer a large amount of data among nodes of a cluster via the interconnection network.

Most of the existing load balancing schemes ignore network resources, leaving open the opportunity for significant performance bottleneck to form for communication-intensive parallel applications due to unevenly distributed communication load. To remedy this problem, we have proposed a communication-aware load balancing technique that is capable of improving the performance of communication-intensive applications by increasing the effective utilization of network resources in clusters.

To facilitate the proposed load-balancing scheme, we introduce a behavior model for parallel applications with large requirements of CPU, memory, network, and disk I/O resources. The proposed load-balancing scheme can make full use of this model to quickly and accurately determine the load induced by a variety of parallel applications.

Simulation results on executing a diverse set of both synthetic bulk synchronous and real parallel applications on a cluster show that the proposed scheme can significantly improve the performance both in slowdown and turn-around time over two existing schemes by up to 206% (with an average of 74%) and 235% (with an average of 82%), respectively.

8.2 Future Work

In the course of designing and evaluating I/O-aware load balancing schemes for clusters, we have found several interesting issues that are still unresolved. This section overviews some of these open issues that need further investigation. In addition, this section presents opportunities for future work by addressing load-balancing issues in other application domains that have to be studied in detail.

8.2.1 Optimization of Data Movement

Data movement has an enormous impact on the overall performance of load balancing polices, and this is especially true for data-intensive applications that tend to have a huge amount of frequently transferred data. This is because moving data from one node to another means reading data from local storage system, transferring it through network, and then writing it to the remote storage system, thereby imposing undesired load on both storage and network at local nodes.

To efficiently alleviate such a burden resulting from data movements, we will propose a predictive model to speculatively move data without compromising the performance of

applications running on local nodes in a cluster. The new model largely depends on data distribution, the amount of data, data access pattern, and network traffic.

More specifically, transferred data can be classified into two groups: synchronized and asynchronized data. We intend to propose a protocol that moves asynchronized data when the loads of a source node and destination node are below a certain threshold, implying that asynchronized data is to be transferred only when the local and remote nodes are lightly loaded in terms of network and disk resources. While designing the protocol, we have to address starvation cases where asynchronized data has no opportunity to be delivered due to consistently high load of source or destination nodes. To avoid such a potential starvation, we set up a rule to enforce asynchronized data to be transferred if its queuing time at source nodes exceeds a threshold, which is an experimental parameter to be dynamically tuned to optimize the system performance.

Compared against existing data movement approaches that treat synchronized and asynchronized data equally, the above approach is expected to boost the performance of load balancing policies when disk I/O and network workload situations are bursty in nature. We will conduct extensive experiments to demonstratively show that in the worst case where load is constantly high or low, the new data movement approach is expected to sustain the same level of performance as existing protocols.

8.2.2 Prefetching Synchronized Data

We notice that data prefetching technique, which complements the new data movement approach envisioned in Section 8.2.1, can optimize the performance of disk-I/O-aware load balancing policies. In case where required data that is considered synchronized data has to be moved towards to a remote node to which its job is migrated, the required data may be transferred in advance before the job is accessing the data locally.

A prefetching scheme has to be aware of the relevant information about data-intensive applications, in order to efficiently identify useful data that is most likely to be accessed in the nearest future. To achieve this goal, we can potentially leverage a meta-data language

to explicitly specify applications' required files, thereby applying a static prefectching scheme to make these files available before applications need to locally access them.

The disadvantage of the static prefetching approach discussed above is that to efficiently transfer necessary files in advance, static prefetching mechanisms must be aware of applications' requirements related to file access. To overcome this constraint, a new dynamic prefetching algorithm will be explored. Specifically, the prefetching algorithm dynamically predicts files to be retrieved in future by establishing correlations among files based on a statistic model. The dynamic prefetching is appealing because applications do not have to specify their required files. In addition to identifying files to be prefetched, the dynamic prefetching algorithm makes an effort to predict destinations for those prefetched files. Note that this feature distinguishes the proposed algorithm from traditional file prefetching algorithms.

8.2.3 Data Replication

Both scientific and non-scientific applications may be read-intensive, and data has to be read from nodes that store the data. As a result, the idea of replicating frequently accessed data on multiple nodes in clusters has been eagerly pursued. Data replications play an important role both in improving access locality and in increasing reliability of large-scale clusters, since there exists a strong likelihood for some nodes to become unavailable due to system failures.

The drawback of most existing replication techniques is that as the replication granularity is at a file level, a large file might be replicated even if only a small portion of the file is frequently accessed. This problem is analogous to false sharing in distributed shared memory system. To tackle the false replicating problem, we plan to investigate a so-called dynamic replication granularity model that can dynamically adjust replication granularities based on tradeoffs between benefits gained from replicas and overheads of generating replications.

Additionally, we will design and implement a data-caching algorithm to enhance performance for clusters in the context of data-intensive computing. Note that files cached on local disks are referred to as temporary replicas, and a cached file is different from a regular replica in that the cached file may be removed from its local disk when the disk cache is unable to accommodate newly arrived files. The algorithm specifies which cached file should be removed when new files enter into a full cache space, and this selection process ensures that cached files are likely to be read in the near future. We plan to use extensive simulations to study a number of possible file cache replacement policies, including locality-based algorithms and frequency-based algorithms. Based on performance comparisons, we can identify a replacement policy that achieves the best performance for cluster computing environments.

8.2.4 Other Application Domains

The load balancing schemes proposed in this dissertation were motivated by data-intensive applications running on cluster computing environments. However, our load balancing approaches can be applicable to other application domains. In the long term, we plan to explore the following areas as extensions to our current research.

8.2.4.1 Load-balancing Schemes for Data Mining

One of the most important issues in data mining is the process of extracting useful and previously unknown rules in large database. Since many data mining applications are highly I/O-intensive, we will extend our previous work to develop a parallel algorithm based on I/O-aware load balancing policies to provide sufficient support for the I/O-intensive data mining applications on a parallel architecture as scalable and cost-effective as clusters. The goals of this research are to reduce the time spent on accessing disk I/O and achieve well-balanced workload distribution on a large-scale cluster, thereby delivering high performance for data mining applications.

8.2.4.2 Data Grid

A data grid is a collection of geographically dispersed storage resources over a wide area network. The goal of a data grid system is to provide a large virtual storage framework with unlimited power through collaboration among individuals, institutions, and resources [Chervenak et al., 2000].

We expect that heterogeneity will be a big challenge for data-intensive applications running on computational grids, where interconnects are relatively slow and network latencies are generally high [Mullmann et al., 2001]. In designing an efficient load balancing mechanism for grids, we plan to evaluate the performance of our proposed load-balancing policies in an environment as large and widely distributed as a computational grid.

Most existing consistency models in data grid work well for read only data, but little attention has been paid to the consistency issues for data that needs to be written multiple times. In an environment as large and diverse as data grid, many datasets are multiple-written in nature. Therefore, we intend to develop a relaxed consistency model to effectively handle the consistency problems in datasets written multiple times.

8.3 Conclusions

This dissertation has presented a series of load balancing techniques in cluster computing environments. The experimental results reported in the dissertation have shown that the proposed disk-I/O-aware and communication-aware load balancing policies can deliver significant performance improvements by achieving high utilization of various resources for clusters running a diverse set of applications including data-intensive applications. In particular, the proposed schemes improve performance over existing schemes under I/O- or communication-intensive workload situations. In a scenario where workloads are CPU or memory intensive, our schemes can maintain the same level of performance as existing CPU- and memory-aware policies. Furthermore, we have presented a feedback control

mechanism to adaptively manipulate I/O buffer sizes. The empirical results have demonstrated that the feedback control mechanism not only can be leveraged to enhance the performance of load-balancing schemes, but also can be applied to clusters where workload conditions exhibit dynamic behaviors.

Bibliography

[Acharya and Setia, 1999] A. Acharya and S. Setia, “Availability and Utility of Idle Memory in Workstation Clusters,” *Proceedings of the ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems*, May 1999.

[Acharya et al., 1996] A. Acharya et al., “Tuning the performance of I/O-intensive parallel applications,” *Proc. of the 4th IOPADS*, pages 15–27, Philadelphia, PA, May 1996.

[Anglano, 2000] C. Anglano, “A Fair and Effective Scheduling Strategy for Workstation Clusters,” *Proceedings of the International Conference on Cluster Computing*, 2000.

[Arpaci-Dusseau and Culler, 1997] A. Arpaci-Dusseau and D. Culler, “Extending Proportional Share Scheduling to a Network of Workstations,” *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, June, 1997.

[Arpaci-Dusseau et al., 1999] R. Arpaci-Dusseau, E. Anderson, N. Treuhaft, D. Culler, J. Hellerstein, D. Paterson, and K. Yelick, “Cluster I/O with River: Making the Fast Case Common,” *Proceedings of the Sixth Workshop on I/O in Parallel and Distributed Systems*, Atlanta, Georgia, 1999.

[Barak et al., 1999] A. Barak, O. La’adan, and A. Shiloh, “Scalable Cluster Computing with MOSIX for Linux,” *Linux Expo’99*, pp.95-100, 1999.

[Basney and Livny, 2000] J. Basney and M. Livny, “Managing Network Resources in Condor”, *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing* (HPDC9), Pittsburgh, Pennsylvania, August 2000, pp 298-299.

[Boden et al., 1995] N.J. Boden, D. Cohen, and W.K. Su., “Myrinet: A Gigabit-per-second Local Area Network,” IEEE Micro, Vo. 15, No. 1, February 1995.

[Bode, 2000] B. Bode, D. M. Halstead, R. Kendall, and Z. Lei, “The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters,” *In Proc. of the 4th Annual Linux Showcase & Conference*, 2000.

[Bordawekar et al., 1994] R. Bordawekar, R. Thakur, and A. Choudhary, “Efficient Compilation of Out-of-core Data Parallel Programs,” *Technical Report, SCCS-662*, NPAC, April 1994.

[Bovet and Cesati, 2001] D. P. Bovet, and M. Cesati. *Understanding the Linux Kernel. Sebastopol*, CA: O’Reilly 2001.

[Cap and Strumpen, 1993] C. H. Cap and V. Strumpen, “Efficient Parallel Computing in Distributed Workstation Environment,” *Parallel Computing*, Vol.19, pp.1221-1234, 1993.

[Carns et al., 2000] P.H. Carns, W.B. Ligon, R.B. Ross, and R. Thakur, "PVFS: A Parallel File System for Linux Clusters," *Proceedings of the 4th Annual Linux Showcase and Conference*, Atlanta, GA, 2000, pp.317-327, USENIX Association.

[Chang et al., 1997] C. Chang, B. Moon, A. Acharya, C. Shock, A. Sussman, and J. Saltz. "Titan: a High-Performance Remote-Sensing Database," In *Proc. of the 13th International Conference on Data Engineering*, Apr 1997.

[Chervenak et al., 2000] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets," *Proceedings of Network Storage Symposium*, 2000.

[Cho et al., 1997] Y. Cho, M. Winslett, M. Subramaniam, Y. Chen, S. Kuo, and K. E. Seamons, "Exploiting Local Data in Parallel Array I/O on a Practical Network of Workstations," *Proceedings of the Fifth Workshop on I/O in Parallel and Distributed Systems*, pp.1-13, Nov. 1997.

[Cho, 1999] Y. E. Cho, "Efficient Resource Utilization for Parallel I/O in Cluster Environments," *Ph.D. Dissertation*, Department of Computer Science, University of Illinois at Urbana-Champaign, 1999.

[Chronopoulos et al., 2001] A. Chronopoulos, D. Grosu, A. M. Wissink, M. Benche, "Static Load Balancing for CFD Simulations on a Network of Workstations," *Proceedings of the IEEE International Symposium on Network Computing and Applications, (NCA 2001)*, October 8-10, 2001, Cambridge, MA, USA, pp. 364-367, Oct. 2001.

[Cirne and Berman, 2003] W. Cirne, F. Berman, "When the herd is smart Aggregate behavior in the selection of job request," *IEEE Trans. on Parallel and Distributed Systems*, no.2, Feb. 2003.

[Colajanni et al., 1997] M. Colajanni, P. Yu, D. Dias, "Scheduling Algorithms for Distributed Web Servers," *Proceedings of the International Conference on Distributed Computing Systems*, pp. 169-176, 1997.

[Colajanni et al., 1998] M. Colajanni, P. Yu, V. Cardellini, "Dynamic Load Balancing in Geographically Distributed Web Servers," *Proceedings of the International Conference on Distributed Computing Systems*, 1998.

[Cypher et al., 1993] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina, "Architectural Requirements of Parallel Scientific Applications with Explicit Communication," *Proc. of the 20th International Symposium on Computer Architecture*, pp.2-13, 1993.

[Czajkowski et al., 1998] K. Czajkowski, I. Foster, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems," In *Proceedings of the IPPS/SPDP 98 Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[Dannenberg and Hibbard, 1985] R. Dannenberg and P. Hibbard, "A Butler Process for Resource Sharing on Spice Machines," *ACM Transactions on Office Information Systems*, 3(3), pp. 234-252, 1985.

[Douglis and Ousterhout, 1991] F. Douglis and J. Ousterhout, “Transparent Process Migration: Design Alternatives and Sprite Implementation,” Software-Practice and Experience, 21(8), pp.757-785, 1991.

[Duke et al., 1994] D. Duke, T. Green, and J. Pasko, Research Toward a Heterogeneous Networked Computing Cluster: The Distributed Queuing system Version 3.0, Florida State University, May 1994.

[Dusseau et al., 1996] A.C. Dusseau, R.H.Arpaci, and D.E. Culler, “Effective Distributed Scheduling of Parallel Workloads,” In *Proceedings of SigMetrics*, pp.25-36, May 1996.

[Eager et al., 1986] D.L. Eager, E.D. Lazowska, and J. Zahorjan, “Adaptive Load Sharing in Homogeneous Distributed Systems,” *IEEE Trans. on Software Eng.,* Vol. 12, No. 5, pp.662-675, 1986.

[Feng et al., 2003] W. Feng, G. Hurwitz, et al., “Optimizing 10-Gigabit Ethernet for Networks of Workstations, Clusters and Grids: A Case Study,” *Proc. of SC2003: High-Performance Networking and Computing Conference*, Phoenix AZ, November 2003.

[Figueira and Berman, 2001] S. Figueira and F. Berman, “A Slowdown Model for Applications Executing on Time-Shared Clusters of Workstations,” *IEEE Transactions on Parallel and Distributed Systems*, Vol. 12, No.6, pp.653-670, 2001.

[Forney et al., 2001] B. Forney, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, “Storage-Aware Caching: Revisiting Caching for Heterogeneous Storage Systems,” *Proc. of 1st File and Storage Technology*, 2001.

[Geoffray, 2002] P. Geoffray, “OPIOM: Off-Processor I/O with Myrinet”, Future Generation Comp. Sys., 2002(19).

[Grop and Lusk, 2001] W. Grop and E. Lusk, The Message Passing Interface (MPI) Standard, Argonne National Laboratories, 2001

[Harchol-Balter and Downey, 1997] M. Harchol-Balter and A. Downey, “Exploiting Process Lifetime Distributions for Load Balacing,” *ACM transaction on Computer Systems*, vol. 3, no. 31, 1997.

[Hendrickson and Womble, 1994] B. Hendrickson and D. Womble, “The torus-wrap mapping for dense matrix calculations on massively parallel computers,” *SIAM J. Sci. Comput.*, 15(5), Sept. 1994.

[Hennessy and Patterson, 2002] J. L. Hennessy, D. A. Patterson, and D. Goldberg, *Computer Architecture: A Quantitative Approach*, 3rd Edition, Morgan Kaufmann, May 2002.

[Hsu and Liu, 1986] C.-Y.H. Hsu and J.W.-S. Liu, “Dynamic Load Balancing Algorithms in Homogeneous Distributed Systems,” *IEEE Proc. 6th International Conf. Distributed Computing Systems*, pp.216-223, 1986.

[Huber et al., 1995] J. Huber, C. L. Elford, D. A. Reed, A. A. Chien, and D. S. Blume, “PPFS: A high Performance Portable Parallel File System,” *Proceedings of the 9th ACM International Conference on Supercomputing*, pp.385-394, July 1995.

[Hui and Chanson, 1999] C. Hui and S. Chanson, "Improved Strategies for Dynamic Load Sharing," *IEEE Concurrency*, vol.7, no.3, 1999.

[Kim and Kameda, 1992] C. Kim and H. Kameda, "An Algorithm for Optimal Static Load Balancing in Distributed computer Systems," IEEE Transactions on Computers, Vol.41, No.3, pp.381-284, March 1992.

[Kotz and Nieuwejaar, 1994] D. Kotz and N. Nieuwejaar, "Dynamic file-access characteristics of a production parallel scientific workload," *Proc. ACM conference on Supercomputing* 1994, pp.640-649.

[Lavi and Barak, 2001] R. Lavi and A. Barak, "The Home Model and Competitive Algorithm for Load Balancing in a Computing Cluster," *Proceedings of the 21st International Conference on Distributed Computing Systems* (ICDCS 2001), Apr. 2001.

[Lee et al., 2000] L. Lee, P. Scheauermann, and R. Vingralek, "File Assignment in Parallel I/O Systems with Minimal Variance of Service time," *IEEE Trans. on Computers*, Vol. 49, No.2, pp.127-140, 2000.

[Li and Curkendall, 1992] P. Li and D. Curkendall, "Parallel 3-D perspective rendering," *Proceedings of the 1st International Delta Applications Workshop*, Technical Report CCSF-14-92, California Institute of Technology, pp.52-58, 1992.

[Li and Kameda, 1998] J. Li and H. Kameda, "Load Balancing Problems for Multiclass jobs in Distributed Parallel Computer Systems," *IEEE Transactions on Computers*, Vol. 47, No.3, pp.322-332, March, 1998.

[Ligon and Ross, 1996] W. B. Ligon and R. B. Ross, "Implementation and Performance of a Parallel File System for High Performance Distributed Applications," *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, pp.471-480, August 1996.

[Ma et al., 2002] X. Ma, M. Winslett, J. Lee, and S. Yu, "Faster Collective Output through Active Buffering," *Proc. of International Symposium on Parallel and Distributed Processing*, 2002.

[McCann et al., 1993] C. McCann, R. Vaswani, and J. Zahorjan, "A Dynamic Processor Allocation Policy for Multiprogrammed Shared-Memory Multiprocessors," *ACM Transactions on Computer Systems*, Vol. 11, No.2, pp.146-178, May 1993.

[Menasce et al., 1995] D. Menasce, D. Saha, S. da Silva Porto, V. Almeida, S. Tripathi, "Static and Dynamic Processor Scheduling Disciplines in Heterogeneous Parallel Architecture," *Journal of Parallel and Distributed Computing*, v.28, pp.1-18, 1995.

[Moyer and Sunderam, 1995] S. Moyer and V. S. Sunderam, "Parallel I/O as a Parallel Application," *International Journal of Supercomputer Application*, Vol.9, No.2, pp.95-107, 1995.

[Mueller 1995] A. Mueller, "Fast sequential and parallel algorithms for association rule mining: A comparison," *Technical Report CS-TR-3515*, University of Maryland, College Park, August 1995.

[Mullmann et al., 2001] D. Mullmann, W. Hosckek, J. Jaen-Martinez, and B. Segal, “Models for Replica Synchronization and Consistency in Data Grid,” *Proceedings IEEE Symposium on High Performance on Distr. Computing*, 2001.

[Natarajan and Iyer, 1996] C. Natarajan and R. K. Iyer, “Measurement and Simulation Based Performance Analysis of Parallel I/O in a High-Performance Cluster System,” *Proceedings of the Eighth Symposium on Parallel and Distributed Processing*, pp.332-339, 1996.

[Paoli et al., 1996] D.De Paoli, A. Goscinski, M. Hobbs, P. Joyce, “Performance Comparison of Process Migration with Remote Process Creation Mechanisms in RHODOS,” *Proceedings of the IEEE 16th International Conference on Distributed Computing Systems*, pp.554-561, 1996.

[Pasquale and Polyzos, 1994] B.K. Pasquale and G.C. Polyzos. “Dynamic I/O Characterization of I/O Intensive Scientific applications,” *Proceedings of the Supercomputing 1994*, pages 660 - 669.

[Preslan et al., 1999] K. W. Preslan, A. P. Barry, J. E. Brassow, G. M. Erickson, E. Nygaard, C. J. Sabol, S. R. Soltis, D. C. Teigland, and M. T. O’Keefe, “64-bit, Shared Disk File System for Linux,” *Proceedings of the Seventh NASA Goddard Conference on Mass Storage Systems*, March 1999.

[Qin et al., Cluster2003] X. Qin, H. Jiang, Y. Zhu, and D. Swanson, "Towards Load Balancing Support for I/O-Intensive Parallel Jobs in a Cluster of Workstations," *Proceedings of the 5th IEEE International Conference on Cluster Computing* (Cluster 2003), Hong Kong, Dec. 1-4, 2003.

[Qin et al., HiPC2003] X. Qin, H. Jiang, Y. Zhu, and D. Swanson, "Dynamic Load Balancing for I/O-Intensive Tasks on Heterogeneous Clusters," *Proceedings of the 10th International Conference on High Performance Computing* (HiPC 2003), Dec.17-20, 2003, Hyderabad, India.

[Qin et al., Euro-Par2003] X. Qin, H. Jiang, Y. Zhu, and D. Swanson, "Dynamic Load balancing for I/O- and Memory-Intensive workload in Clusters using a Feedback Control Mechanism," *Proceedings of the 9th International Euro-Par Conference on Parallel Processing (Euro-Par 2003)*, pp. 224-229. Klagenfurt, Austria, Aug. 2003.

[Qin et al., ICPPW2003] X. Qin, H. Jiang, Y. Zhu, and D. Swanson, "A Dynamic Load Balancing Scheme for I/O-Intensive Applications in Distributed Systems," *Proceedings of the 32nd International Conference on Parallel Processing Workshops (ICPP2003 Workshops)*, Oct. 6-9, 2003. IEEE Press.

[Qin et al., SBAC-PAD2003] X. Qin, H. Jiang, Y. Zhu, and D. Swanson, "Boosting Performance for I/O-Intensive Workload by Preemptive Job Migrations in a Cluster System," *Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing*, Nov. 10-12, 2003, Brazil. IEEE Press.

[Riska and Smirni, 2002] A. Riska, and E. Smirni, “Exact Aggregate Solutions for M/G/1-type Markov Processes,” *Proc. ACM Sigmetircs 2002*, pp.86-96.

[Roads, et al., 1992] J.O. Roads, et al., “A Preliminary Description of the Western U.S. Climatology”, Proceedings of the Ninth Annual Pacific Climate (PAClim) Workshop, September 8, 1992.

[Rosario and Choudhary, 1994] J. M. Rosario and A. Choudhary, “High Performance I/O for Parallel Computers: Problems and Prospects,” IEEE Computer, Vol.27, No.3, pp.59-58, March 1994.

[Ryu and Hollingsworth, 2000] K. D. Ryu and J. K. Hollingsworth, “Exploiting Fine-Grained Idle Periods in Networks of Workstations,” *IEEE Trans. on Parallel and Distributed Systems*, No.7, Vol.11, pp. 683-698, July 2000.

[Salem and Garcia-Molina, 1986] K. Salem and H. Garcia-Molina, “Disk Striping,” *Proceedings of the 2nd international conference on Data Engineering*, pp.336-342, Feb. 1986.

[Scheuermann et al., 1998] P. Scheuermann, G. Weikum, P. Zabback, “Data Partitioning and Load Balancing in Parallel Disk Systems,” The VLDB Journal, pp. 48-66, July, 1998.

[Shin and Chang, 1989] K.G. Shin and Y.-C. Chang, “Load Sharing in Distributed Real-time Systems with State Change Broadcasts,” *IEEE Trans. on Computers*, Vol. 38, no.8, pp.1124-1142, Aug. 1989.

[Spring and Wolski, 1998] N. T. Spring, R. Wolski, “Application Level Scheduling of Gene Sequence Comparison on Metacomputers,” *Proceedings of the International Conference on Supercomputing*, pp. 141-148, 1998.

[Stankovic, 1984] J.A. Stankovic, “Simulation of Three Adaptive, Decentralized Controlled, Job Scheduling Algorithms,” *Computer Networks*, Vol.8, pp.199-217, 1984.

[Sterling et al., 1999] T. L. Sterling, J. Salmon, D. Becker, and D. Savarese, “How to Build a Beowulf. A Guide to the Implementation and Application of PC Clusters,” The MIT Press, 1999.

[Subramani et al., 2002] V. Subramani, R. Kettimuthu, S. Srinivasan, J. Johnston, P. Sadayappan, “Selective Buddy Allocation for Scheduling Parallel Jobs on Clusters,” *Proceedings of the International Conference on Cluster Computing*, 2002.

[Surdeanu et al., 2002] M. Surdeanu, D. Modovan, and S. Harabagiu, “Performance Analysis of a Distributed Question/ Answering System,”*IEEE Trans. on Parallel and Distributed Systems*, vol. 13, no. 6, pp. 579-596, 2002.

[Tantawi and Towsley, 1985] A. Tantawi, and D. Towsley, “Optimal Static Load Balancing in Distributed Computer Systems,” *Journal of ACM*, Vol.21, No.2, pp.445-465, April, 1985.

[Uysal et al., 1997] M. Uysal, A. Acharya, and J. Saltz. “Requirements of I/O Systems for Parallel Machines: An Application-driven Study,” *Technical Report, CS-TR-3802*, University of Maryland, College Park, May 1997.

[Valiant, 1990] L. Valiant, "Bridging model for parallel computation," *Comm. of ACM*, 33(8):103–111, 1990.

[Vetter and Mueller, 2002] J. S. Vetter and F. Mueller, "Communication Characteristics of Large-Scale Scientific Applications for Contemporary Cluster Architectures," *Proc. of International Parallel and Distributed Processing Symposium*, April 2002.

[Vetter and Yoo, 2002] J. S. Vetter and A. Yoo, "An Empirical Performance Evaluation of Scalable Scientific Applications," *Proceedings of the IEEE/ACM SC2002 Conference*, November 2002.

[Voelker, 1997] G. Voelker, "Managing Server Load in Global Memory Systems," *Proceedings of the ACM SIGMETRICS Conf. Measuring and Modeling of Computer Systems*, May 1997.

[Watts and Taylor, 1998] J. Watts and S. Taylor, "A Practical Approach to Dynamic Load Balancing," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No.3, pp.235-248, March, 1998.

[Weissman, 2002] J. B. Weissman, "Predicting the Cost and Benefit of Adaptive Data Parallel Applications in Clusters," Journal of Parallel and Distributed Computing, No.8, pp.1248-1271, 2002.

[Wu et al., 2004] J. Wu, P. Wyckoff, and D. K. Panda, "High Performance Implementation of MPI Datatype Communication over InfiniBand," *Proceedings of the International Parallel and Distributed Processing Symposium*, April, 2004.

[Wu et al., 2001] X. Wu, V. Taylor, and R. Stevens, "Design and Implementation of Prophesy Automatic Instrumentation and Data Entry System," *Proc. of the 13th IASTED Int. Conf. on Parallel and Distributed Computing and Systems*, CA, 2001.

[Wu and Manber, 1992] S. Wu and U. Manber, "agrep - A fast approximate pattern-matching tool," In *USENIX Conference Proceedings*, pages 153–162, San Francisco, CA, Winter 1992.

[Wyllie, 1999] J. Wyllie, "Spsort: How to Sort a Terabyte Quickly," Feb. 1999. *http://www.almadem.ibm.com/cs/gpfs-sport.html*

[Xiao et al., 2000] L. Xiao, X. Zhang, and Y. Qu, "Effective Load Sharing on Heterogeneous Networks of Workstations," *Proc. of International Symposium on Parallel and Distributed Processing*, 2000.

[Xiao et al., 2002] L. Xiao, S. Chen, and X. Zhang, "Dynamic Cluster Resource Allocations for Jobs with Known and Unknown Memory Demands", *IEEE Trans. on Parallel and Distributed Systems*, vol.13, no.3, 2002.

[Zhang et al., 2000] X. Zhang, Y. Qu, and L. Xiao, "Improving Distributed Wrokload Performance by Sharing both CPU and Memory Resources," *Proc. 20th Int'l Conf. Distributed Computing Systems (ICDCS 2000)*, Apr. 2000.

[Zhang et al., 2003] Y. Zhang, A. Yang, A. Sivasubramaniam, and J. Moreira, “Gang Scheduling Extensions for I/O Intensive Workloads”, *in Proc. of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, 2003.

[Zhou et al., 1993] S. Zhou, X. Zheng, J. Wang, and P. Delisle, “Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems,” *SPE*, Vol.23, No.12, pp. 1305-1336, 1993.

[Zhu et al., Cluster03] Y. Zhu, H. Jiang, X. Qin, and D. R. Swanson, “A Case Study of Parallel I/O for Biological Sequence Analysis on Linux Clusters”, *Proceedings of the 5th IEEE International Conference on Cluster Computing* (Cluster 2003), Hong Kong, Dec. 1-4, 2003.

[Zhu et al., SNAPI03] Y. Zhu, H. Jiang, X. Qin, and D. R. Swanson, “Design, Implementation, and Performance evalaution of a Cost-Effective Fault-Tolerant Parallel Virtual File System,” *International Workshop on Storage Network Architecture and Parallel I/Os*, in conjunction with the 12th International Conference on Parallel Architectures and Compilation Techniques (PACT03), Sept. 27 - Oct. 1, 2003, New Orleans, LA.

[Zhu et al., LCI03] Y. Zhu, H. Jiang, X. Qin, D. Feng, and D. R. Swanson, "Scheduling for improved write performance in a Cost-Effective, Fault-Tolerant Parallel Virtual File System (CEFT-PVFS)," *ClusterWorld Conference and Expo Partners with the Fourth LCI International Conference on Linux Clusters: The HPC Revolution 2003*, San Jose, California, June 24-26, 2003.

[Zhu et al., CCGrid03] Y. Zhu, H. Jiang, X. Qin, D. Feng, and David R. Swanson, "Improved Read Performance in CEFT-PVFS: Cost Effective, Fault-Tolerant Parallel Virtual File System," *Proceedings of IEEE/ACM CCGrid 2003*, pp.730-735. *Workshop on Parallel I/O in Cluster Computing and Computational Grids*, Japan, May 2003.

[CASA, 1991] Corporation for National Research Initiatives, *CASA Gigabit Testbed: 1991 Annual Report*, June 1991.

[Compaq et al, 1997] Computer Corporation, Intel Corporation, Microsoft Corporation. *Virtual Interface Architecture Specification Version 1.0*, December 1997.

[IBM, 2001] IBM Corporation. *LoadLeveler - Efficient Job Scheduling and management*. 2001.

[IBM DB2] IBM Corporation. *IBM DATABASE 2 Parallel Edition for AIX - AdministrationGuide and Reference*.