Dynamic Load Balancing for I/O-Intensive Applications on Clusters

XIAO QIN
Auburn University
HONG JIANG
University of Nebraska, Lincoln
and
ADAM MANZANARES, XIAOJUN RUAN, and SHU YIN
Auburn University

9

Load balancing for clusters has been investigated extensively, mainly focusing on the effective usage of global CPU and memory resources. However, previous CPU- or memory-centric load balancing schemes suffer significant performance drop under I/O-intensive workloads due to the imbalance of I/O load. To solve this problem, we propose two simple yet effective I/O-aware load-balancing schemes for two types of clusters: (1) homogeneous clusters where nodes are identical and (2) heterogeneous clusters, which are comprised of a variety of nodes with different performance characteristics in computing power, memory capacity, and disk speed. In addition to assigning I/O-intensive sequential and parallel jobs to nodes with light I/O loads, the proposed schemes judiciously take into account both CPU and memory load sharing in the system. Therefore, our schemes are able to maintain high performance for a wide spectrum of workloads. We develop analytic models to study mean slowdowns, task arrival, and transfer processes in system levels. Using a set of real I/O-intensive parallel applications and synthetic parallel jobs with various I/O characteristics, we show that our proposed schemes consistently improve the performance over existing non-I/O-aware load-balancing schemes, including CPU- and Memory-aware schemes and a PBS-like batch scheduler for parallel and sequential jobs, for a diverse set of workload conditions. Importantly, this performance improvement becomes much more pronounced when the applications are I/O-intensive. For example, the proposed approaches deliver 23.6–88.0 % performance improvements for I/O-intensive applications such as LU decomposition, Sparse Cholesky, Titan, Parallel text searching, and Data Mining. When I/O load is low or well balanced, the proposed schemes are capable of maintaining the same level of performance as the existing non-I/O-aware schemes.

The work reported in this article was supported by the US National Science Foundation under Grants CCF-0845257 (CAREER), CNS-0757778 (CSR), CCF-0742187 (CPA), CNS-0917137 (CSR), CNS-0831502 (CyberTrust), CNS-0855251 (CRI), OCI-0753305 (CI-TEAM), DUE-0837341 (CCLI), and DUE-0830831 (SFS), as well as Auburn University under a startup grant and a gift (Number 2005-04-070) from the Intel Corporation.
Author's address: X. Qin, Department of Computer Science and Software Engineering, Auburn University, AL 36849; email: Xzq0001@auburn.edu.

DOI 10.1145/1629075.1629078 http://doi.acm.org/10.1145/1629075.1629078

Categories and Subject Descriptors: D.4.1 [**Operating Systems**]: Process Management—*Scheduling*; D.4.8 [**Operating Systems**]: Performance—*Simulation*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Load balancing, I/O-intensive applications, storage systems, clusters, heterogeneity

ACM Reference Format:

Qin, X., Jiang, H., Manzanares, A., Ruan, X., and Yin, S. 2009. Dynamic load balancing for I/O-intensive applications on clusters. ACM Trans. Storage 5, 3, Article 9 (November 2009), 38 pages. DOI = 10.1145/1629075.1629078 http://doi.acm.org/10.1145/1629075.1629078

1. INTRODUCTION

In the last decade, clusters have become increasingly popular as powerful and cost-effective platforms for executing parallel applications [Zhu et al. 2004]. In such systems, load-balancing schemes can improve system performance by attempting to assign work, at run time, to machines with idle or underutilized resources. Several distributed load-balancing schemes, based on this architecture, have been presented in the literature, primarily considering CPU [Harchol-Balter and Downey 1996; Hui and Chanson 1999], memory [Acharya and Setia 1999; Voelker et al. 1997], network [Cruz and Park 2001], a combination of CPU and memory [Zhang et al. 2000], or a combination of CPU and network resources [Basney and Livny 2000]. For example, Harchol-Balter and Downey [1996] proposed a CPU-based preemptive migration policy that was more effective than non-preemptive migration policies. Zhang et al. [2000] focused on load sharing policies that consider both CPU and memory services among the nodes. Although these policies achieve high system performance by increasing the utilization of resources in distributed systems, they are less effective when the workload comprises a large number of I/O-intensive jobs and I/O resources exhibit an imbalanced load.

Typical examples of I/O-intensive applications include long running simulations of time-dependent phenomena that periodically generate snapshots of their state [Tanaka 1993], remote-sensing database systems that process remote sensing data [Chang et al. 1997], and biological sequence analysis [Zhu et al. 2004], to name just a few. The high performance of I/O-intensive applications heavily depends on the effective usage of global storage, because the impact of disk I/O on overall system performance is becoming increasingly significant due to the rapidly widening gap between CPU and disk I/O speeds. To alleviate the I/O bottleneck in clusters, load balancing policies have to achieve high utilization of disk I/O resources by being I/O-aware, which in turn improves the overall performance of cluster systems under I/O-intensive workloads.

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. [2000] proposed two file assignment algorithms that balance the load across all disks. The I/O load balancing policies in these studies have been shown to be effective in improving overall system performance by fully utilizing the available hard drives. Zhang

et al. [1993] proposed three I/O-aware scheduling schemes that are aware of the job's spatial preferences. However, because these techniques are developed to balance explicit I/O load, these approaches become less effective under a complex workload where I/O-intensive tasks share resources with many memory- and CPU-intensive tasks. The main distinction between the existing I/O-aware load balancing schemes and our approaches is fourfold. First, our schemes consider both explicit I/O invoked by application programs and implicit I/O induced by page faults. Second, while these approaches address the issue of load balancing at the storage level, our technique tackles the problem of load balancing at the application level. Third, one of our schemes considers heterogeneities in CPU, memory, and disk resources. Fourth, our schemes can handle imbalanced loads in three difference types of resources under a diverse set of workload conditions, whereas the existing ones can only deal with an imbalance in disk resources.

Communication-aware load balancing has been studied in Cruz and Park [2001] and Keren and Barak [2003]. Our approach takes into account the communication load as a measure to determine the migration cost, but balancing the network load is beyond the scope of this paper.

Many researchers have shown that I/O cache and buffer are useful mechanisms to optimize storage systems. Ma et al. [2002] implemented an active buffering scheme to alleviate I/O burdens by using local idle memory and overlapping I/O with computation. We developed a feedback control mechanism to improve performance of clusters by adaptively manipulating the I/O buffer size [Qin et al. 2003a]. Forney et al. [2002] investigated storage-aware caching algorithms for heterogeneous clusters. Although we focus solely on balancing disk I/O load in this article, the approach proposed here is also capable of improving the buffer utilization of each node.

The work presented in this article extends our previous work in load balancing strategies for sequential I/O-intensive jobs [Qin et al. 2003b, 2003c]. In the first part of this study, we develop two simple yet effective I/O-aware load-balancing schemes for parallel jobs in homogeneous clusters. 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. It is assumed in this part that the clusters are homogeneous in nature.

Although this assumption is reasonable for a new and stand-alone cluster system, upgraded clusters or networked clusters are likely to be heterogeneous in practice. 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, we have addressed the issue of heterogeneity in the second part of this study. A load balancing scheme is proposed to

hide 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.

The main contributions of this article are summarized as follows. (1) a disk I/O model is proposed to efficiently estimate I/O load levels in the long term; (2) an analytical model is built to approximate mean slowdown of all jobs running on a cluster; (3) two I/O-aware load balancing schemes are developed for homogeneous clusters; (4) an I/O-aware load balancing scheme is designed for heterogeneous clusters; (5) a simulated cluster is implemented to verify the proposed load balancing schemes; (6) a detailed comparison with the performance of three other load balancing policies is provided; and (7) six real-world applications are used to demonstrate the strengths of our I/O-aware load balancing approaches.

The rest of the article is organized as follows. Section 2 introduces system models. Section 3 describes two I/O-aware load-balancing policies for parallel jobs running on homogeneous clusters. In Section 4, I/O-aware load balancing policies for heterogeneous clusters are studied. Finally, Section 5 summarizes the paper and comments on future directions for this work.

2. SYSTEM MODELS

A head node in a cluster could apply a broadcast mechanism (e.g., gather and scatter like operations) to handle load distribution in a dedicated computing cluster. The head node increasingly becomes a severe bottleneck when the cluster scales up. In this study, we propose a scalable infrastructure, where each node maintains a load manager that is responsible for controlling, monitoring, and load balancing the available resources, in addition to handling the execution of processes generated from the local node. In this infrastructure, every job has a "home" node that it prefers for execution [Lavi and Barak 2001]. The home model has two features: (1) the input data of a job has been stored in the home node, and (2) the job was created on its home node. The network considered in this study is full connectivity in the sense that any two nodes are connected through either a physical link or a virtual link. The memory burden placed by migrating data is ignored, because data movement can be handled by interface controllers without the local CPU's intervention [Geoffray 2002].

To improve performance of disk subsystems, disk arrays may be attached to nodes in a cluster. For simplicity and without loss of generality, we assume that each node as a single disk subsystem. This model captures the key aspects of nodes with disk arrays, since in the model we are able to treat an disk array in each node as a single disk by readily configuring disk mechanical delay parameters (e.g., device rotation, arm positioning, and data transfer). Our model is also valid for clusters equipped with networked storage systems. The reason is twofold. First, in most clusters a local disk is attached to each computing node

to cache the most popular data sets. Second, some network storage subsystems have computing capacities.

The collective I/O technique is widely employed in clusters, where large files are stored over a number of disks in different nodes in a RAID-like fashion (see, for example, PVFS: a Parallel File System [Carns et al. 2000]). Our proposed I/O-aware load balancing mechanism can be readily integrated with PVFS, because large datasets can be distributed in a RAID-like fashion across multiple disks in different nodes. In addition, we consider disk parallel I/O processes, where I/O processes communicate with one another through the message-passing interface or MPI. Thus, our approach is adequate for a variety of parallel I/O patterns.

3. LOAD BALANCING FOR I/O-INTENSIVE JOBS ON HOMOGENEOUS CLUSTERS

In this part of the study, we consider the problem of dynamic load balancing among a cluster of homogeneous nodes connected by a high-speed network, where tasks arrive at each node dynamically and independently, and share resources available there. Some preliminary results of this part of study have been discussed in Qin [2008].

3.1 I/O Aware Load Balancing Schemes

3.1.1 *I/O-Aware Load Balancing with Remote Execution (IOCM-RE).* In this section, we present IOCM-RE, a simple yet effective dynamic I/O-aware load-balancing scheme. For a parallel job, arriving in its home node 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 running on the node, especially those with I/O- and memory-intensive workloads, are likely to experience waiting time for I/O processing. To alleviate the problem of unevenly distributed I/O load, IOCM-RE selects a group of nodes with a lighter 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: the implicit I/O load induced by page faults and the explicit I/O requests resulting from tasks accessing disks. Let page(i, j) denote the implicit I/O load of task j running on node i, and IO(j) denote the explicit I/O requirement of task j. Thus, node i's I/O load index is expressed by Equation (1).

$$L_{IO}(i) = \sum_{j \in N_i} page(i, j) + \sum_{j \in N_i} IO(j). \quad (1)$$

Many existing load-balancing schemes use load levels of current time as a means of defining load indices; therefore, load-balancing mechanisms may overreact to temporary load fluctuations. To remedy this limitation, we propose a new way to efficiently estimate future load levels, which capture CPU, memory

and disk I/O requirements of tasks running on each node. The load indices used in this study can closely approximate load levels in the long term, thereby helping to solve the load fluctuation problem. 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. (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. When the memory load exceeds the amount of available memory space, the IOCM-RE policy transfers the tasks of the newly arriving 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 arriving 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.

3.1.2 *IO-Aware Load Balancing with Preemptive Migration (IOCM-PM).* 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.

For a newly arriving 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 arriving 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 nonpreemptive migration. The set of eligible migrant tasks is: $EM(i, k) = \{j \in M_i \mid r_{PM}(i, j) > r_{PM}(k, j) + c_j(i, k)\}$, 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, respectively, and $c_j(i, k)$ is the migration cost of task 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. 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.

We can employ a task migration mechanism (see, for example, M-JavaMPI and LAM/MPI) to migrate tasks that are part of a parallel job (e.g., MPI jobs). The task migration mechanism is able to migrate tasks from one node to another in a cluster by virtue of saving tasks' images. To migrate a task of a parallel application, the migration mechanism transfers the task's image from its local

node to a remote node, where the task resumes its execution without having to restart the entire parallel application.

In an effort to integrate an existing task migration mechanism with our load-balancing schemes, we have to distinguish two types of parallel applications: embarrassingly parallel applications and parallel applications with communications. Tasks in an embarrassingly parallel application are executed independently; thus, the tasks can be readily migrated without dealing with the issue of synchronizations. In contrast, many parallel applications with communications typically have barrier synchronization events from time to time. It is imperative that migrations of a task in a parallel application do not intervene in synchronizations of the task with other tasks in the application. Therefore, the task migration mechanism must bring the parallel application into a consistent state after each task migration in the application is successfully completed.

3.2 Analytic Model

In this subsection, we develop models to help in evaluating the performance of the proposed schemes. With the analytic models in place, we study task arrival and transfer processes in the level of cluster computing systems. First and foremost, let us estimate the mean slowdown (see Equations (5) and (6)) of all jobs running on a cluster. Note that Equations (5) and (6) can be derived from Equations (2)–(4).

Let R be the cumulative execution time of a node, and $p_R(i)$ be the probability that the cumulative execution time is i (i.e., $p_R(i) = \Pr(R = i)$). Since the cumulative execution time consists of both I/O execution time (denoted by R_{IO}) and CPU execution time (denoted by R_{CPU}), $p_R(i)$ is expressed as

$$\begin{aligned} p_R(i) = \Pr(R = i) &= \Pr(R_{IO} + R_{CPU} = i) \\ &= \sum_{j=0}^{i-1} \big(\Pr(R_{IO} = j)\Pr(R_{CPU} = i - j)\big), \end{aligned} \tag{2}$$

where $\Pr(R_{IO} = j)$ is the probability that the I/O cumulative execution time equals to j, and $\Pr(R_{CPU} = i - j)$ represents the probability that the CPU cumulative execution time is $i - j$. Thus, the expected cumulative execution time of a node is obtained from

$$E(R) = \sum_{i=0}^{\infty} i p_R(i) = \sum_{i=0}^{\infty} \left(i \sum_{j=0}^{i-1} p_{IO}(j) p_{CPU}(i - j) \right), \tag{3}$$

where $p_{IO}(j) = \Pr(R_{IO} = j)$ and $p_{CPU}(i - j) = \Pr(R_{CPU} = i - j)$.

Let T, T_{CPU}, and T_{IO} denote the execution time, I/O time, and CPU time of a task. Then, we have $T = T_{CPU} + T_{IO}$. The execution of a task requires j time units in I/O processing with probability $\alpha_j (1 \le j \le l)$ and k time units on CPU with probability $\beta_k (1 \le k \le m)$. The expected execution time $E(T)$ is computed by Equation (4).

$$E(T) = \sum_{i=1}^{l+m} i p_T(i) = \sum_{i=1}^{l+m} \left(i \sum_{j=0}^{i} \alpha_j \beta_{i-j} \right), \tag{4}$$

The performance metric used in our experiments is the mean *slowdown* [Harchol-Balter and Downey 1996; Zhang et al. 2000] of all jobs in a trace. 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. Let S be the slowdown of a job, and the mean slowdown can be expressed as

$$E(S) = \frac{E(R) + E(T)}{E(T)}. \tag{5}$$

Substituting Equations (3) and (4) into (5), we get

$$E(S) = \frac{\sum_{i=0}^{\infty} \left(i \sum_{j=0}^{i-1} p_{IO}(j) p_{CPU}(i-j) \right)}{\sum_{i=1}^{l+m} \left(i \sum_{j=0}^{i} \alpha_j \beta_{i-j} \right)} + 1. \tag{6}$$

Now we derive the composite job arrival rate $\lambda_i(L_{IO}, L_{MEM}, L_{CPU})$ at node i (see Equation (7)), where L_{IO}, L_{MEM}, and L_{CPU} are the I/O, memory, and CPU load of a node. The composite job arrival rate of each node is used in our empirical experiments to estimate the load of each node. Note that the composite job arrival rate, which depends on L_{IO}, L_{MEM}, and L_{CPU}, is a summation of external arrival rate i and transferred rate denoted by $\rho_i(L_{IO}, L_{MEM}, L_{CPU})$. Let $\varphi_i(L_{IO}, L_{MEM}, L_{CPU})$ be the rate of transferring jobs out of node i when the node's I/O, memory, and CPU loads are L_{IO}, L_{MEM}, and L_{CPU}, respectively. Consequently, we have

$$\lambda_i(L_{IO}, L_{MEM}, L_{CPU}) = \lambda_i + \rho_i(L_{IO}, L_{MEM}, L_{CPU}) - \varphi_i(L_{IO}, L_{MEM}, L_{CPU}). \tag{7}$$

The composite job arrival rate computed by Equation (7) can be derived from $\varphi_i(L_{IO}, L_{MEM}, L_{CPU})$ and $\rho_i(L_{IO}, L_{MEM}, L_{CPU})$, which are expressed as Equations (8) and (12), respectively. In what follows, we approximate these two important parameters used to model transfer and location policies. The parameter $\varphi_i(L_{IO}, L_{MEM}, L_{CPU})$ corresponds to the transfer policy used in a load-balancing scheme, because the transfer policy determines if a job has to be executed remotely. A job is transferred to other nodes if L_{IO} is greater than zero and performance gains are not offset by migration cost (see Euqation (1)). If no I/O load is imposed on the node, the memory and CPU load will be balanced (see Equations (2) and (3)). Thus, we have Equation (8), which can be derived from Equations (9)–(11).

$$\begin{aligned} \varphi_i(L_{IO}, L_{MEM}, L_{CPU}) = {} & \lambda_i u^i_{IO}(L_{IO}) v^i_{IO}(L_{IO}) \\ & + \lambda_i (1 - u^i_{IO}(L_{IO})) \cdot u^i_{MEM}(L_{MEM}) v^i_{MEM}(L_{MEM}) \\ & + \lambda_i (1 - u^i_{IO}(L_{IO})) \cdot (1 - u^i_{MEM}(L_{MEM})) u^i_{CPU}(L_{CPU}) v^i_{CPU}(L_{CPU}), \end{aligned} \tag{8}$$

where $u^i_{IO}, u^i_{MEM}, u^i_{CPU}$ are the probabilities that I/O, memory, and CPU loads need to be balanced given that node $i's$ load indices are $L_{IO}, L_{MEM}, and\, L_{CPU}$. $v^i_{IO}, v^i_{MEM}, v^i_{CPU}$ are the probabilities that balancing I/O, memory, and CPU loads can ultimately result in performance gains. For node i in a cluster with n nodes, we can obtain $u^i_{IO}(L_{IO})$ from Equation (9), where $q^j_{IO}(k)$ is the probability that

node $j's$ I/O load equals to k, and $\prod_{j=1,j\neq i}^{n}\sum_{k=L_{IO}+1}^{\infty} q_{IO}^{j}(k)$ is the probability that the I/O load of node i is the lowest among all the nodes in the cluster.

$$u_{IO}^{i}(L_{IO}) = 1 - \prod_{j=1,j\neq i}^{n}\sum_{k=L_{IO}+1}^{\infty} q_{IO}^{j}(k) \tag{9}$$

Suppose the workload conditions of all the nodes are identical, that is, $\forall 1 \leq j \leq n : q_{IO}^{j}(k) = q_{IO}(k)$, Equation (9) can be rewritten as $u_{IO}^{i}(L_{IO}) = 1 - (\sum_{k=L_{IO}+1}^{\infty} q_{IO}(k))^{n-1}$. Similarly, $u_{MEM}^{i}(L_{MEM})$ and $u_{CPU}^{i}(L_{CPU})$ in Equation (8) can be expressed as follows, where $q_{MEM}^{j}(k)$ and $q_{CPU}^{j}(k)$ are the probabilities that node $j's$ memory and CPU loads equal to k, respectively.

$$u_{MEM}^{i}(L_{MEM}) = 1 - \prod_{j=1,j\neq i}^{n}\sum_{k=L_{IO}+1}^{\infty} q_{MEM}^{j}(k) \tag{10}$$

$$u_{CPU}^{i}(L_{CPU}) = 1 - \prod_{j=1,j\neq i}^{n}\sum_{k=L_{IO}+1}^{\infty} q_{CPU}^{j}(k) \tag{11}$$

Now we are positioned to derive parameter $\rho_i(L_{IO}, L_{MEM}, L_{CPU})$ characterizing the location policy determining to which remote node a job has to be migrated. The proposed schemes choose the best candidate remote node to which jobs submitted to overloaded nodes will be transferred. Without loss of generality, we assume that the external arrival rates of all the nodes are identical, that is, $\forall 1 \leq j \leq n : \lambda_i = \lambda$ and, thus, we express $\rho_i(L_{IO}, L_{MEM}, L_{CPU})$ in terms of L_{IO}, L_{MEM}, and $_{CPU}$ as

$$\begin{aligned}\rho_i(L_{IO}, L_{MEM}, L_{CPU}) &= \lambda\mu_{IO}\pi_{IO}^{i}(L_{IO})\theta_{IO}^{i}(L_{IO}) \\ &\quad + \lambda(1-\mu_{IO})\cdot\mu_{MEM}\pi_{MEM}^{i}(L_{MEM})\theta_{MEM}^{i}(L_{MEM}) \\ &\quad + \lambda(1-\mu_{IO})(1-\mu_{MEM})\cdot\mu_{CPU}\pi_{CPU}^{i}(L_{CPU})\theta_{CPU}^{i}(L_{CPU}),\end{aligned} \tag{12}$$

where $\mu_{IO}^{i}, \mu_{MEM}^{i}, \mu_{CPU}^{i}$ are the probabilities that I/O, memory, and CPU loads are unbalanced, respectively. $\pi_{IO}^{i}, \pi_{MEM}^{i}, \pi_{CPU}^{i}$ are the probabilities that node i has the lightest load with respect to I/O, memory, and CPU resources. $\theta_{IO}^{i}, \theta_{MEM}^{i}, \theta_{CPU}^{i}$ are the probabilities that transferring jobs from other nodes to node i balancing I/O, memory, and CPU loads can improve system performance. $\pi_{IO}^{i}(L_{IO})$, $\mu_{MEM}^{i}(L_{MEM})$ and $\mu_{CPU}^{i}(L_{CPU})$ in Equation (12) can be obtained by the following equations, where $q_{IO}^{j}(k)$, $q_{MEM}^{j}(k)$, and $q_{CPU}^{j}(k)$ are the probabilities that node $j's$ I/O, memory, and CPU indices respectively equal k.

$$\pi_{IO}^{i}(L_{IO}) = \prod_{j=1,j\neq i}^{n}\sum_{k=L_{IO}+1}^{\infty} q_{IO}^{j}(k) \tag{13}$$

$$\pi_{MEM}^{i}(L_{MEM}) = \prod_{j=1,j\neq i}^{n}\sum_{k=L_{IO}+1}^{\infty} q_{MEM}^{j}(k) \tag{14}$$

$$\pi_{CPU}^{i}(L_{CPU}) = \prod_{j=1,j\neq i}^{n}\sum_{k=L_{IO}+1}^{\infty} q_{CPU}^{j}(k). \tag{15}$$

3.3 Performance Evaluation

3.3.1 *Evaluation of the IOCM-RE Scheme.* To evaluate the performance of our I/O-aware load balancing schemes, we performed a large number of trace-driven simulations. We have extended the simulator implemented by Harchol-Balter and Downey [1996]. Zhang et al. [2000] also upgraded this simulator by incorporating memory recourses. Compared to the earlier versions of this simulator, ours embraces the following four new features. First, the new load balancing schemes were implemented. Second, a fully connected network was simulated. Third, our simulator was integrated with a simple disk model. Last, an I/O buffer was implemented in a simulated disk in each node. We simulated a cluster with 32 nodes. The workload we used is represented by trace files extrapolated from those reported in Harchol-Balter and Downey [1996] and [Zhang et al. 2000]. To simulate a multiuser 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 simulated 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]. In our simulation, the time interval between two consecutive synchronization phases is 100 ms. A realistic cluster is likely to have a mixed workload, where some jobs are I/O-intensive and other jobs are either CPU or memory intensive. Thus, we randomly choose 10% of jobs from the trace to be non-I/O-intensive by setting their I/O access rate to be 0. 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 8 No./ms.

Disk I/O operations issued by each task are modeled as a Poisson Process with a mean arrival rate. Although the durations and memory requirements of the jobs are specified in trace data, the I/O access rate of each job is randomly generated according to a uniform distribution. This simplification deflates any correlations between I/O requirement and other job characteristics, but we are able to control the mean 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, which reflects typical data characteristics for many data-intensive applications [Pasquale and Polyzos 1994; Roads and et al. 1992]. The parameters for disk subsystems are listed in Table I.

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

To compare IOCM-RE and BS under I/O-intensive workload conditions, we set the page fault rate to a low value of 0.5 No./ms. This type of workload

Table I. Disk Subsystem Characteristics

Description	Value
Disk Model	Seagate Cheetah STs9205LC
Standard Interface	SCSI
Storage Capacity	9.17 GBytes
Number of Platters	1
Rotational Speed	10,000 RPM
Average Seek Time	5.4 msec
Average Rotation Time	3 msec
Transfer Rate	31 MB/Sec

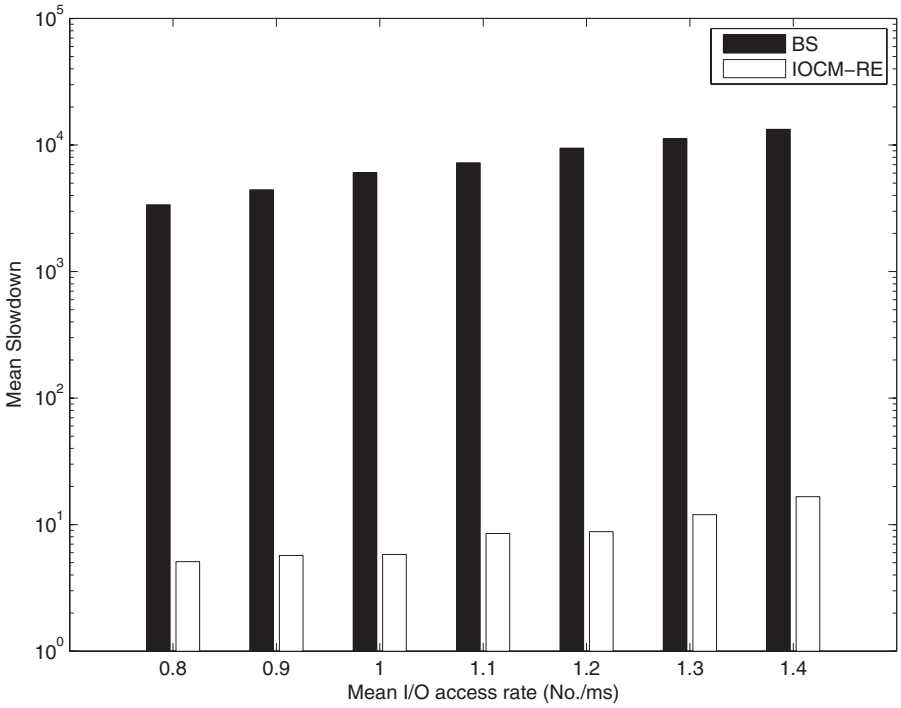

Fig. 1. Mean slowdown as a function of I/O access rate. Page fault rate is 0.5 No./ms. The traces only contain sequential jobs.

reflects a scenario where memory-intensive jobs exhibit high temporal and spatial locality of accesses. We randomly choose 10% of jobs from the trace to be non-I/O-intensive by setting their I/O access rate to 0.

Figures 1 and 2 plot slowdown as a function of the mean I/O access rate. While Figure 1 reports the results for seven traces with sequential jobs, Figure 3 illustrates the mean slowdown of another seven traces with 30% jobs being parallel. Figures 2 and 3 indicate that both IOCM-RE and BS experience an increase in the mean slowdown when the I/O access rate increases. This is because high I/O load leads to high utilization of disks, causing longer waiting time on I/O processing.

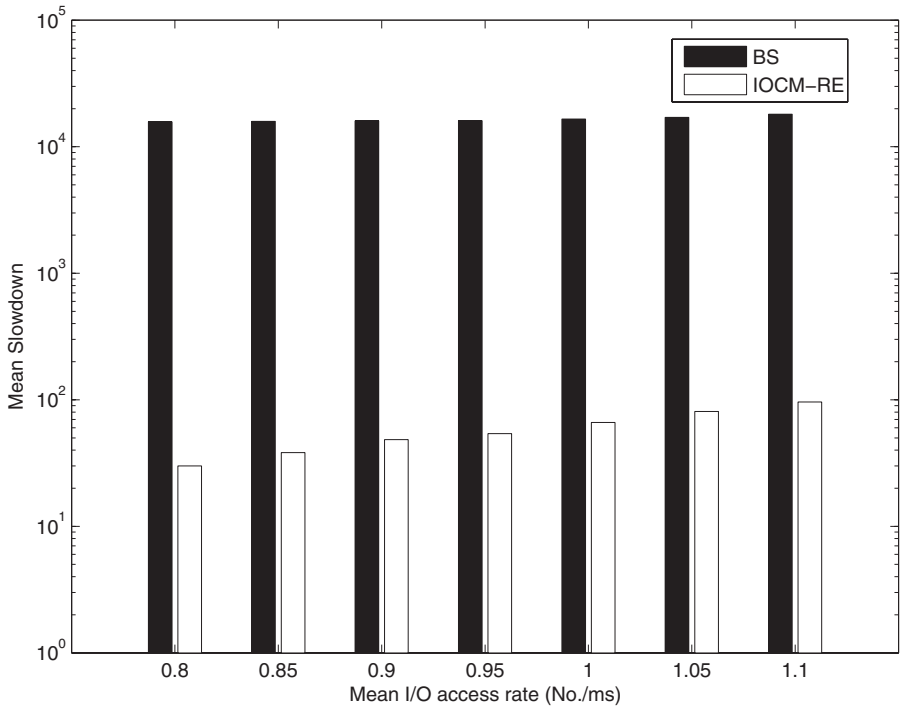

Fig. 2. Mean slowdown as a function of I/O access rate on the traces with 30% parallel jobs. Page fault rate of 0.5 No./ms.

We observe from Figures 1 and 2 that under the I/O-intensive workload, IOCM-RE is significantly better than BS. The results suggest that it is more difficult to utilize dedicated clusters as efficient, multiuser time-sharing platforms to execute I/O-intensive jobs. Figure 2 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, because the synchronizations among processes of parallel jobs further decrease the utilization of resources.

Now, we compare the performance of IOCM-RE with two existing schemes: CPU-based (CPU) [Eager et al. 1986; Harchol-Balter and Downey 1996] and memory-based (MEM) [Zhang et al. 2000] policies. We also simulated a policy (called NLB) that makes no effort to alleviate the problem of imbalanced load in any resource. Figures 3 and 4 plot slowdown as a function of I/O access rate in the range between 0.45 and 0.8 No./ms. Figures 3 and 4 show that IOCM-RE significantly outperforms the CPU-based and memory-based policies. The results suggest that the existing policies are inadequate for I/O-intensive workloads because these two policies ignore imbalanced I/O loads.

After comparing Figure 3 with Figure 4, we realize that if the mean I/O access rate is unchanged, the mean slowdowns of the four policies all increase with the percentage of parallel jobs. The results are expected because a higher percentage of parallel jobs leads to more tasks being concurrently executed, causing more synchronization overhead and longer waiting time on both CPU

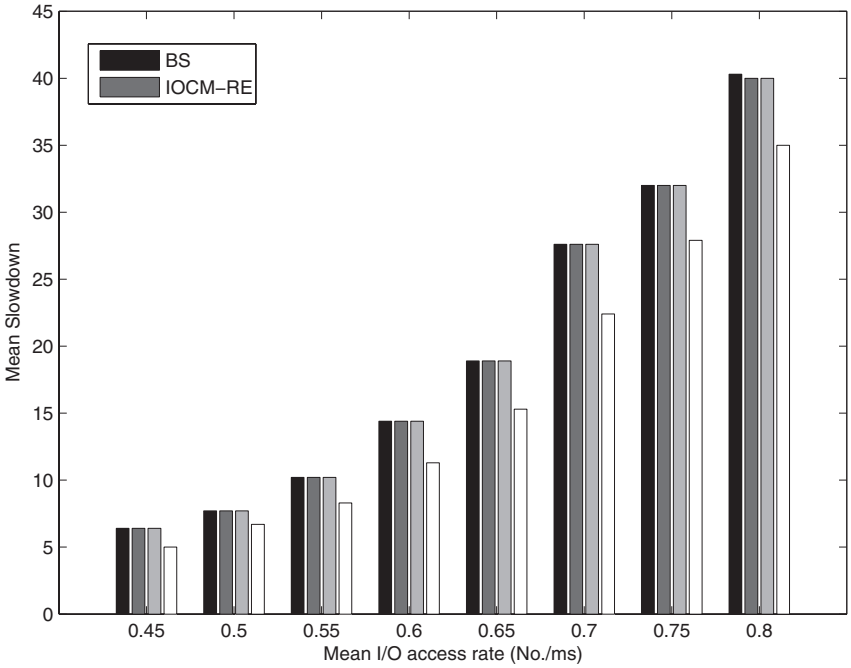

Fig. 3. Mean slowdown as a function of I/O access rate on the traces with 30% parallel jobs. Page fault rate is 0.5 No./ms.

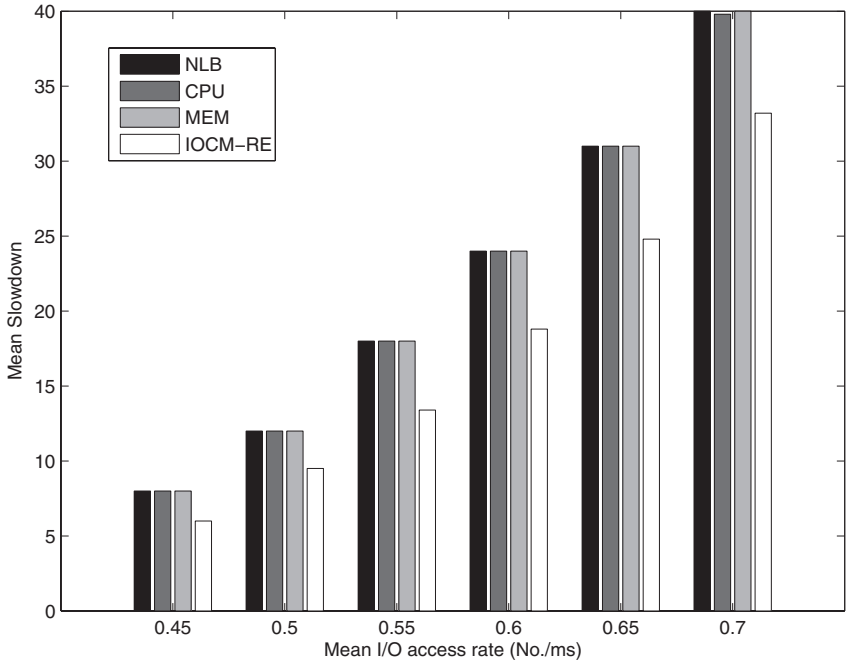

Fig. 4. Mean slowdown as a function of I/O access rate on the traces with 60% parallel jobs. Page fault rate is 0.5 No./ms.

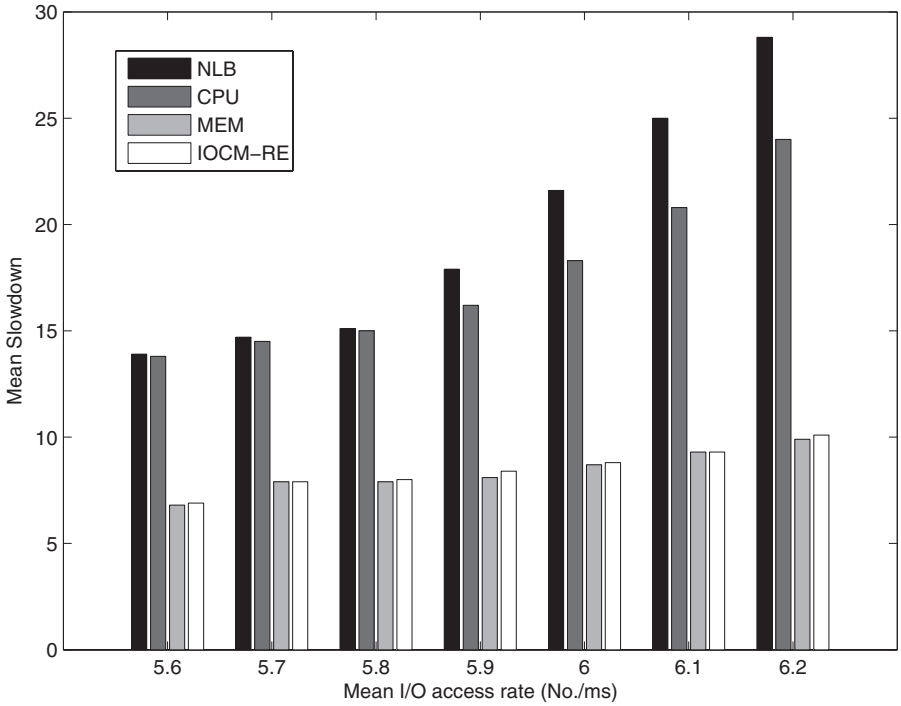

Fig. 5. Mean slowdown as a function of page fault rate on the traces with 30% parallel jobs. Mean I/O access rate is 0.01 No./ms.

and disks. Interestingly, Figure 3 shows that a small percentage of parallel jobs in most cases makes the memory-based policy outperform the CPU-based policy. This is mainly because when I/O and memory demands are higher than that of CPU demands, the CPU-based scheme is unable to significantly reduce the mean slowdown of the cluster. In contrast, Figure 4 indicates that in case of a large percentage of parallel jobs, the CPU-based policy is superior to or at least as good as the memory-based policy. We attribute this trend to increased CPU demands caused by a high percentage of parallel jobs, which result in long waiting time on CPU resources.

We now turn our attention to memory-intensive workloads. The mean I/O access rate is fixed at a low value of 0.01 No./ms. In practice, the page fault rates of applications range from 1 to 10 [Zhang et al. 2000]. Figures 5 and 6 show that when page fault rate is high and I/O rate is very low, IOCM-RE gracefully degrades to the memory-based load-balancing scheme. Furthermore, IOCM-RE and MEM are superior to the CPU-based and NLB schemes considerably under memory-intensive workload conditions. The reason is twofold. First, IOCM-RE and MEM balance implicit I/O loads, which make the most significant contribution to the overall system loads when page fault rate is high. Second, the CPU-based scheme improves the utilization of CPU, ignoring implicit I/O loads resulting from page faults.

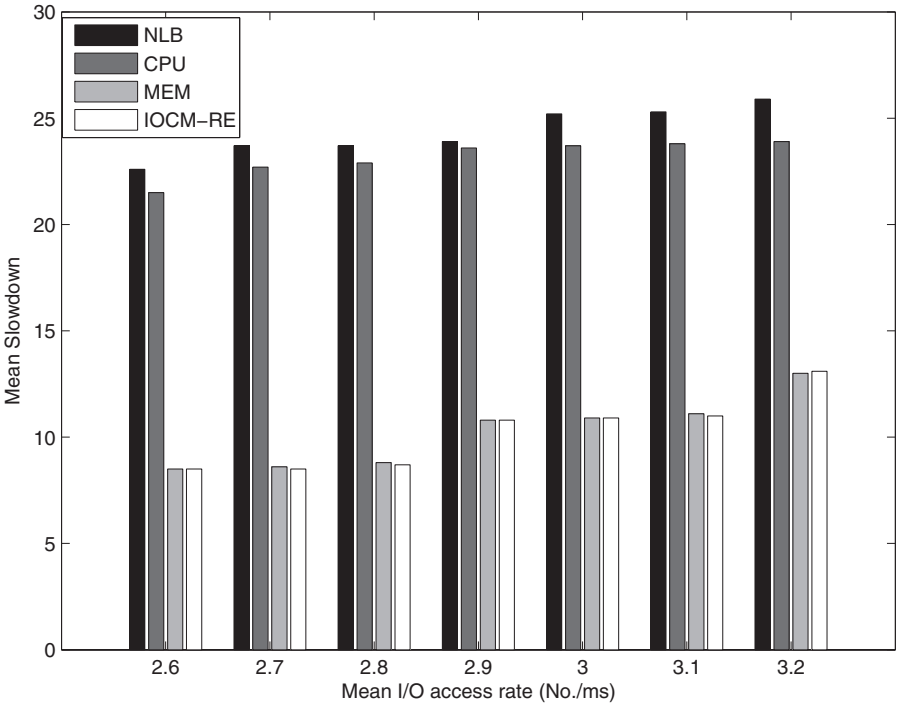

Fig. 6. Mean slowdown as a function of page fault rate on the traces with 60% parallel jobs. Mean I/O access rate is 0.01 No./ms.

We also conducted a set of experiments under CPU-intensive workloads, where most jobs are either CPU-intensive or memory-intensive (but in-core). Experimental results show that both the IOCM-RE and MEM schemes gracefully degrade to the CPU-based scheme. We do not present these results here due to the space limitations.

3.3.2 *Evaluation of the IOCM-PM Scheme.* We compare IOCM-PM with NLB, CPU, MEM, and IOCM-RE under 20 synthetic I/O-intensive traces, which use the same configurations given in the previous section. It is observed from Figures 7 and 8 that IOCM-PM consistently performs the best among all the schemes. These results indicate that load-balancing schemes with preemptive migrations outperform those scheme without using preemptive migrations under I/O-intensive workloads. In addition, the slowdowns of the CPU-based, memory-based, and IOCM-RE are more sensitive to I/O access rate than IOCM-PM. The performance improvement gained by IOCM-PM can be explained by the following reasons. First, IOCM-RE only considers newly arriving jobs for migrations, completely ignoring running tasks that might take advantages of migrations. In the non-preemptive schemes if a task with high I/O demand misses the opportunity to migrate, it will never have a second chance. Second, I/O demand of tasks in a newly arriving job may not be high enough to offset migration overhead. Third, IOCM-PM provides better migratory opportunities by considering all running tasks on a node, in addition to newly arriving tasks.

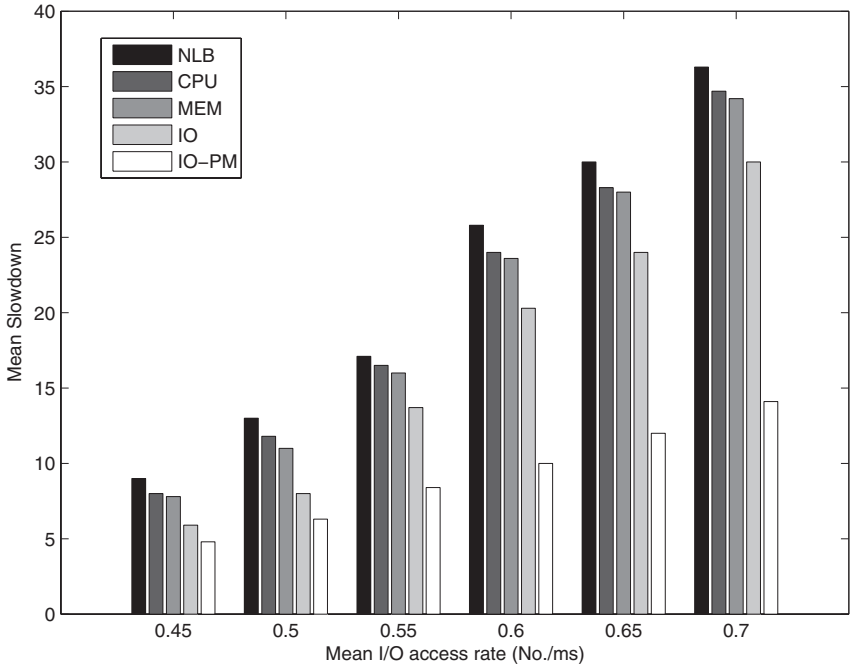

Fig. 7. Mean slowdown as a function of I/O access rate on the traces with 30% parallel jobs. Page fault rate is 0.5 No./ms.

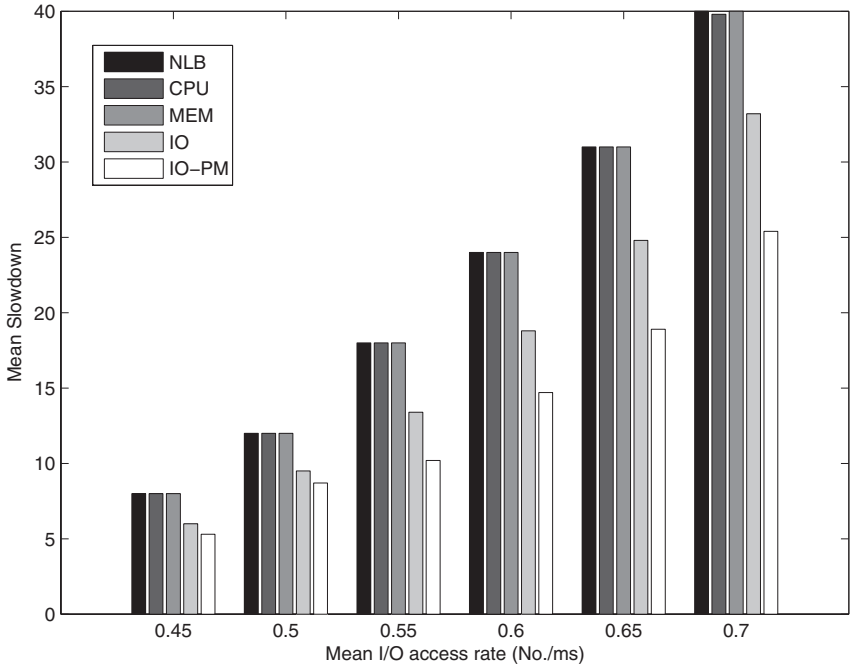

Fig. 8. Mean slowdown as a function of I/O access rate on the traces with 60% parallel jobs. Page fault rate is 0.5 No./ms.

Table II. Descriptions of Real I/O-Intensive Applications

Application	Description
Data mining (Dmine)	This application extracts association rules from retail data [Mueller 1995].
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].
Titan	This is a parallel scientific database for remote-sensing data [Chang et al. 1997].
DB2	This is a parallel relational database management system from IBM [1995]. Due to long run times, only a part of the traces were executed.
LU decomposition (LU)	This application tries to compute the dense LU decomposition of an out-of-core matrix [Hendrickson and Womble 1994].
Sparse Cholesky (Cholesky)	This application is capable of computing Cholesky decomposition for sparse, symmetric positive-definite matrices [Acharya et al. 1996].

3.3.3 *Real I/O-Intensive Parallel Applications.* To validate the results based on the synthetic I/O workloads, we simulate a number of real I/O-intensive parallel applications using six 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 (see Table II) with diverse disk I/O demands. 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 1996]. We measure the impact of the I/O-aware load balancing schemes on a variety of real applications; thus, each job trace consists of one type of I/O-intensive parallel application described above. A 64-node cluster is simulated to run the applications with different I/O demands in each trace.

Figure 9 shows the mean slowdowns of the six job traces under the four load-balancing 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 have approximately identical performance. This is because all jobs running on the cluster belong to the same application and have nearly identical CPU and I/O requirements, tasks of a newly arriving parallel job are most likely to become the most suitable tasks for migrations because of low migration costs. Thus, IOCM-RE and IOCM-PM attempt to migrate the tasks of newly arriving jobs when a local node is overloaded; therefore, IOCM-PM reduces to IOCM-RE when the variance in CPU and I/O demand is minimum.

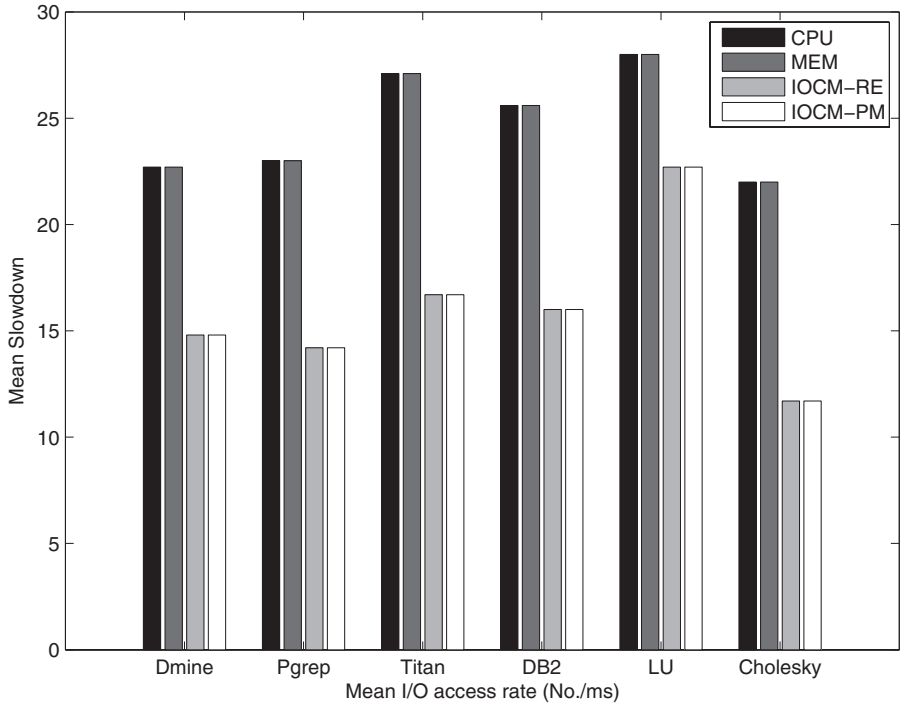

Fig. 9. Mean slowdowns of four policies on six applications.

Third, the LU application exhibits a larger mean slowdown than the other five applications. The slowdowns of the applications depend partially on applications' total execution time, which in turn is affected by the CPU and I/O execution times of jobs running on a cluster. Figure 10 plots the total execution time of the six applications. Figure 10 shows that the total execution time of LU is considerably longer than the other applications, indicating that LU is expected to spend more time-sharing resources with other running jobs. Consequently, there is a strong likelihood that each LU job experience higher slowdowns.

Before comparing the performance improvement of our approaches in slowdown, we illustrate the contribution of CPU and I/O execution time to the total execution time of the real-world applications in a dedicated computing environment. Figure 11 shows that the total execution time of LU is dominated by I/O processing, which gives rise to a low utilization of CPU resources. Unlike 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 five applications, neither CPU time nor I/O time dominates total execution times. Hence, LU has the highest slowdown value among all application traces for the four load-balancing policies (see Figure 11).

Figure 12 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 I/O-aware load balancing that dynamically distributes I/O load among all nodes

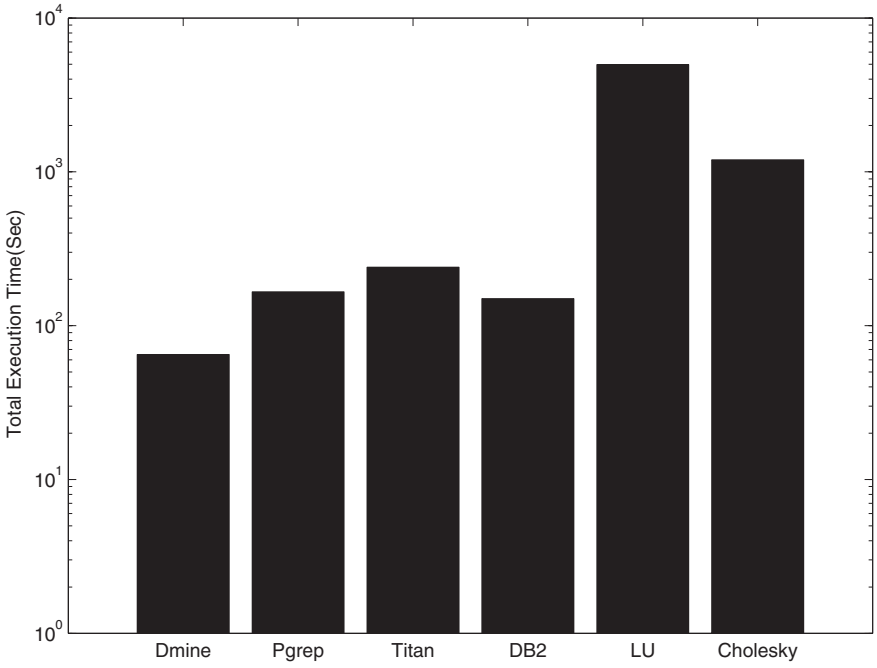

Fig. 10. The total execution times of six applications on a dedicated cluster.

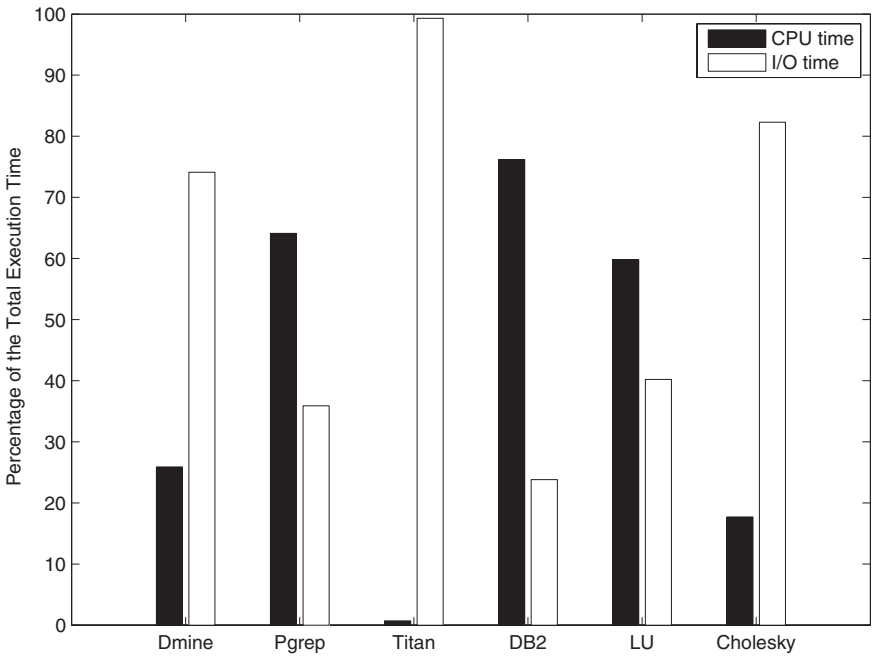

Fig. 11. CPU and I/O times as the components of total execution times.

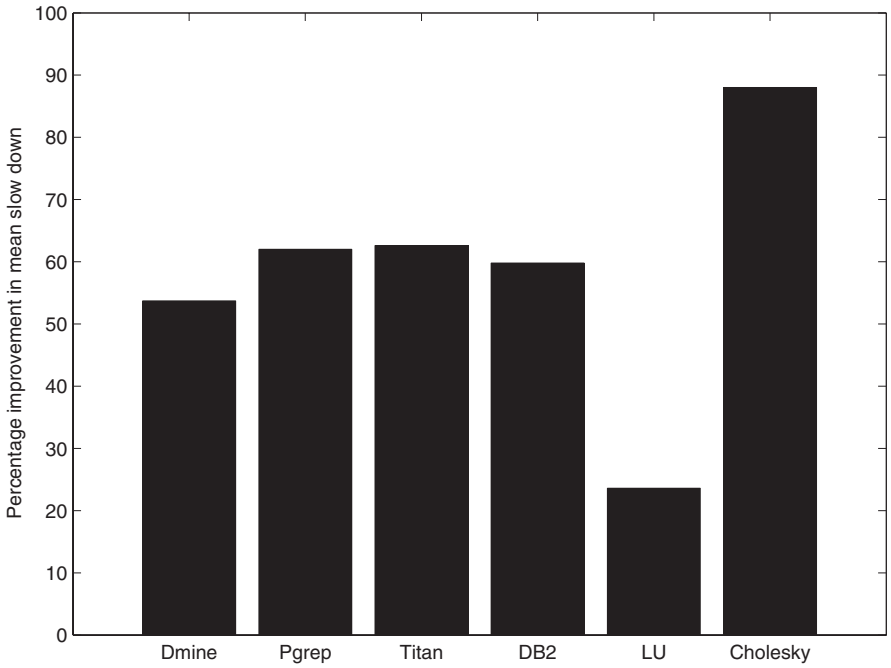

Fig. 12. Comparison of performance improvement in mean slowdown on six traces.

in the cluster. The benefits are pronounced for Cholesky, Titan, Pgrep, DB2, and Dmine; 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 LU application, because most of the running LU jobs compete for disk I/O resources.

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 six 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. Figures 13 and 14 plot 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. To keep slowdown values in a realistic range, we increase the number of nodes in the cluster from 32 to 512. We make the following three observations from Figures 13 and 14. First, a high percentage of I/O-intensive parallel jobs leads to a high slowdown due to high I/O intensity. Second, 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. Third, the IOCM-PM scheme is substantially better than the other policies. For most cases, the IOCM-RE scheme is the second best load-balancing policy. Interestingly, when the I/O access rate is as low as 0.3 No./ms for the

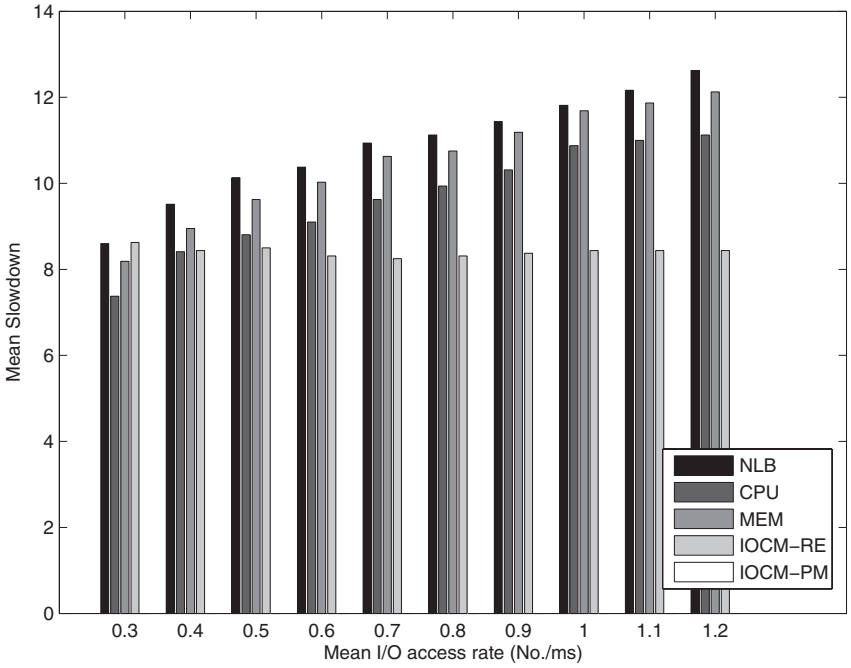

Fig. 13. Mean slowdown of all jobs. Page fault rate of 0.5 No./ms. In each trace, 30% jobs are parallel.

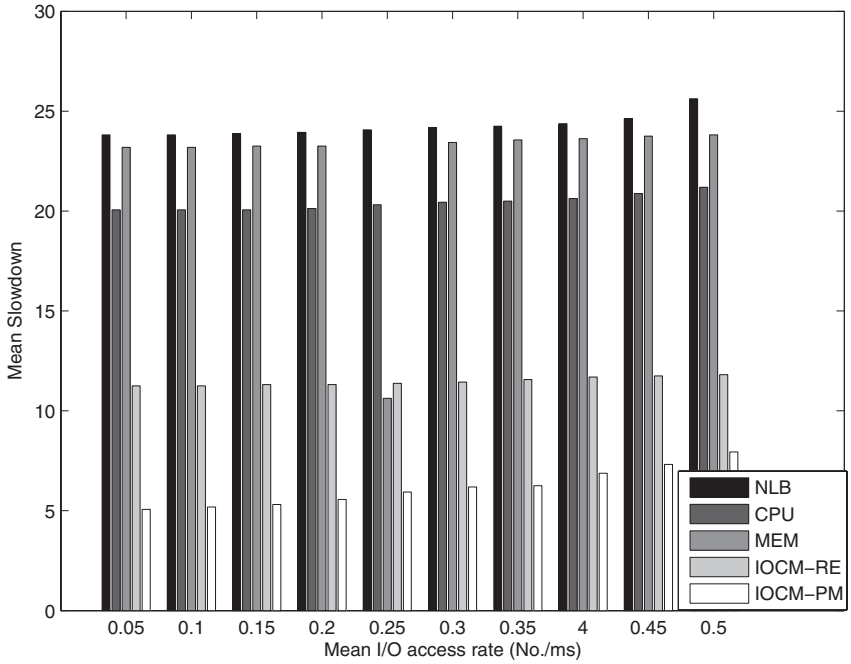

Fig. 14. Mean slowdown of all jobs. Page fault rate of 0.5 No./ms. In each trace, 60% jobs are parallel.

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 I/O-aware load-balancing schemes are less sensitive to I/O intensity than the other policies.

3.4 Summary

In this part of study, we proposed two I/O-aware load-balancing policies, referred to as IOCM-RE (with remote execution) and 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. In addition to CPU and memory utilization, both IOCM-RE and IOCM-PM consider both explicit and implicit I/O load, leading to a performance improvement over the existing CPU- and Memory-based policies under I/O-intensive workload. 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 for I/O-intensive workload is not only necessary but also highly effective. Our proposed schemes offer 23.6–88.0% performance improvements in mean slowdown for I/O-intensive applications. In case that I/O load is low or well balanced, our schemes can maintain the same level of performance as that of the existing non-I/O-aware schemes.

4. BALANCING LOAD ON HETEROGENEOUS CLUSTERS

In the previous section, we have developed two load balancing schemes for homogeneous clusters, which comprise a set of nodes with a given set of performance characteristics in computing power, memory capacity, and disk speed. It is common that a new and stand-alone cluster system is homogeneous in nature, whereas upgraded clusters or networked clusters are likely to be heterogeneous in practice. In other words, heterogeneities of a variety of resources like CPU, memory, and disk I/O may exist in cluster systems. The heterogeneity of disks, compared to that in other resources, results in more significant performance degradation when coupled with imbalanced load of memory and I/O resources. To solve this problem, we develop a load-balancing scheme that is able to sustain high performance for a wide spectrum of workload conditions on clusters with heterogeneous resources.

4.1 Heterogeneity Level

In this part of 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_i^{disk}, 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: $D_i = 1/(S_i + R_i + d/B_i^{disk})$ B, where d is the average data size of I/O requests. The weight of a disk performance W_i^{disk} is defined as a ratio between its performance and that of the fastest disk in the cluster. Thus, we have $W_i^{disk} = D_i/MAX_{j=1}^{n}(D_j)$. 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 $H_D = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(W_{avg}^{disk} - W_i^{disk})^2}$, where W_{avg}^{disk} is the average disk weight. Likewise, the CPU and memory heterogeneity levels are defined as follows: $H_C = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(W_{avg}^{CPU} - W_i^{CPU})^2}$, $H_M = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(W_{avg}^{mem} - W_i^{mem})^2}$, 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].

4.2 IO-Aware Load Balancing in Heterogeneous Clusters

We now turn our attention to a heterogeneity-aware load balancing policy 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 (1)) to measure system workload 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. 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:

$$threshold_{IO}(i) = \left(\sum_{j=1}^{n} L_{IO}(j)\right) \times \left(D_i \Big/ \sum_{j=1}^{n} D_j\right), \tag{16}$$

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 (1) and (16) 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 (17). 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 judiciously 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

Table III. Characteristics of System Parameters. CPU speed and page fault rate are measured by Millions Instruction Per Second (MIPS) and No./ms, respectively.

Parameter	Values assumed	Parameter	Values assumed
CPU Speed	100–400 MIPS	Mean page fault rate	0.01No./ms
RAM Size	32–256 Mbytes	Time slice of CPU time sharing	10 ms
Buffer Size	64 Mbytes	Context switch time	0.1 ms
Network Bandwidth	100 Mbps	Data re-access rate, r	5
Page fault service time	8.1 ms	CPU Threshold	4

Table IV. Characteristics of Disk Systems

Age (years)	Avg. Seek Time (ms)	Avg. Rotation Time (ms)	Bandwidth (MB/s)	Age (years)	Avg. Seek Time (ms)	Avg. Rotation Time (ms)	Bandwidth (MB/s)
1	5.3	3	20	4	7.29	4.11	7.29
2	5.89	3.33	14.3	5	5.21	4.56	5.21
3	6.54	3.69	10.2	6	3.72	5.07	3.72

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 and finally, task j is transferred to the remote node k, and the I/O load of nodes i and k is updated in accordance with $j's$ load. Since the main target of the IO-RE policy is exclusively I/O-intensive workload, IO-RE is unable to maintain a high performance when the workload tends 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 in Section 2.2, 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.

4.3 Performance Evaluation

In this section, we experimentally compare IOCM-RE and IO-RE with the other schemes including the CPU-RE [Eager et al. 1986], MEM-RE [Zhang et al. 2000], and NLB policies.

4.3.1 *Simulation Parameters.* In this part of study, we simulated a cluster that comprises sixty nodes with the configuration parameters listed in Table III. The parameters resemble some workstations like Sun SPARC-20 and Sun Ultra 10. The configuration of disks used in our simulated environment is based on the assumption of device aging and performance-fault injection. Specifically, we chose IBM 9LZX as a base disk whose performance is aged over years to generate a variety of disk characteristics [Forney et al. 2002] shown in Table IV.

We use the same method delineated in Section 3.3.1 to generate a set of traces. To evaluate the performance of our approach under a diversity of workloads, we used the following four traces (see Table V) with a mix of CPU-, memory-,

Table V. Characteristics of Traces

	I/O-intensive	I/O-intensive	I/O-intensive	Memory-intensive
Trace Type	Trace 1	Trace 2	Trace 3	Trace 4
Mean I/O request size	256 Kbyte	256 Kbyte	1 Mbyte	64 Kbyte
I/O request size distribution	Gamma	Gamma	Gamma	Uniform
Mean I/O access rate	2.0 No./ms	2.0 No./ms	2.0 No./ms	0.01 No./ms
I/O request distribution	Exponential	Exponential	Exponential	Uniform
Mean initial data size	50 Mbyte	0 Kbyte	0 Kbyte	100 KByte

Table VI. Characteristics of Five Heterogeneous Clusters. CPU and memory are measured by MIPS and MByte. Disk bandwidth is measured in MByte/S. HL-Heterogeneity Level

	System A			System B			System C			System D			System E		
Node	Cpu	Mem	Disk	Cpu	Mem	Disk	Cpu	Mem	Disk	Cpu	Mem	Disk	Cpu	Mem	Disk
1–10	100	480	20	100	480	20	100	480	10.2	50	320	10.2	50	320	5.21
11–20	100	480	20	150	640	20	150	640	20	200	800	20	200	800	14.3
21–30	100	480	20	150	640	20	150	640	20	200	800	20	200	800	20
31–40	100	480	20	50	320	20	50	320	10.2	50	320	14.3	50	320	5.21
41–50	100	480	20	100	480	20	100	480	20	50	320	14.3	50	320	7.29
51–60	100	480	20	50	320	20	50	320	10.2	50	320	10.2	50	320	3.72
HL	0	0	0	0.27	0.2	0	0.27	0.2	0.25	0.35	0.28	0.2	0.35	0.28	0.3

and I/O-intensive jobs. 10% jobs in Traces 1–3 are either CPU-intensive or memory-intensive, whereas 10% jobs in Trace 4 are I/O-intensive in nature. Data sizes in Traces 1–3 reflect typical data characteristics for many data-intensive applications, where the vast majority of I/O requests are small [Kotz and Nieuwejaar 1994; Pasquale and Polyzos 1994].

4.3.2 *Impact of Heterogeneity on the Performance of Load-Balancing Policies.* First, we evaluate the performance improvement of the proposed load-balancing policies over the existing schemes while understanding the sensitivity of the policies to heterogeneity levels. The configurations of five clusters are summarized in Table VI. For comparison purpose, system A is homogenous, and system B is homogenous in terms of disk I/O. Figure 15 reveals that IO-RE and IOCM-RE significantly outperform the other three policies. Specifically, IO-RE and IOCM-RE improves the performance over CPU-RE and MEM-RE by up to a factor of 5 and 3, respectively. Figure 15 shows that for all the policies, the mean slowdowns of increase consistently as the system heterogeneity increases.

Interestingly, 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 performance. Comparing systems A and B, we realize that 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 by 34.7% and 47.9%. The results are reasonable because the I/O-aware policies ignore the heterogeneity in CPU resources. Furthermore, when the heterogeneities 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, let us compare the slowdowns of systems D and E. We observe that the mean

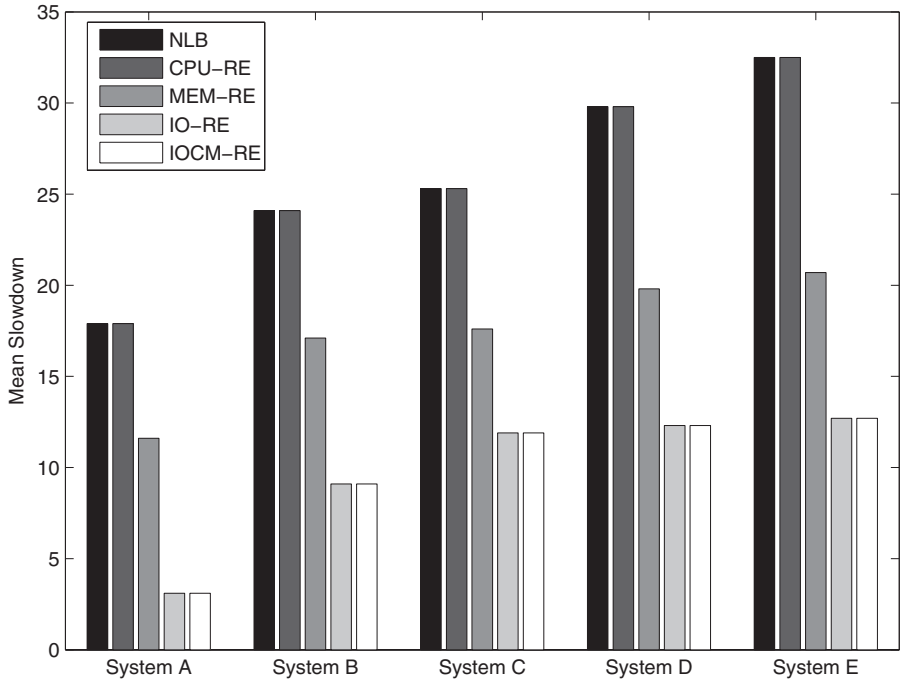

Fig. 15. Mean slowdown when trace 1 is used on five heterogeneous systems.

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 and NLB increase by nearly 9%. This is because both IO-RE and IOCM-RE address the issue of disk heterogeneity in addition to the effective usage of I/O resources.

4.3.3 *Effect of Data Replications.* Now, we investigate the effects of data replications on the performance of heterogeneous clusters. Data replication strategies greatly affect initial data sizes, which in turn determine migration overheads. Figure 16 plots the mean slowdowns of CPU-RE, MEM-RE, and IOCM-RE for Traces 1 and 2, as the heterogeneity levels are increased. Trace 1 represents a workload where the initial data of each remotely executed task is not available at the remote node (i.e., no data replication is provided), whereas Trace 2 illustrates a scenario where migration overheads are considerably reduced by replicating initial data across all the nodes. The experimental data for NLB and IO-RE is omitted from Figure 16 because the slowdowns of NLB and IO-RE are similar to those of CPU-RE and IOCM-RE. Figure 16 shows that IOCM-RE consistently improves the performance of the CPU-RE and MEM-RE policies. These results are consistent with those reported in Section 3.3. Moreover, Figure 16 indicates that the slowdowns of the CPU-RE policy for two traces are roughly identical, implying that the sensitivity of CPU-RE to initial data size is not noticeable under I/O-intensive workloads. This is because CPU-RE makes no effort to balance disk resources; thus, very few remote executions occur when workloads are I/O-intensive. We observe from Figure 16

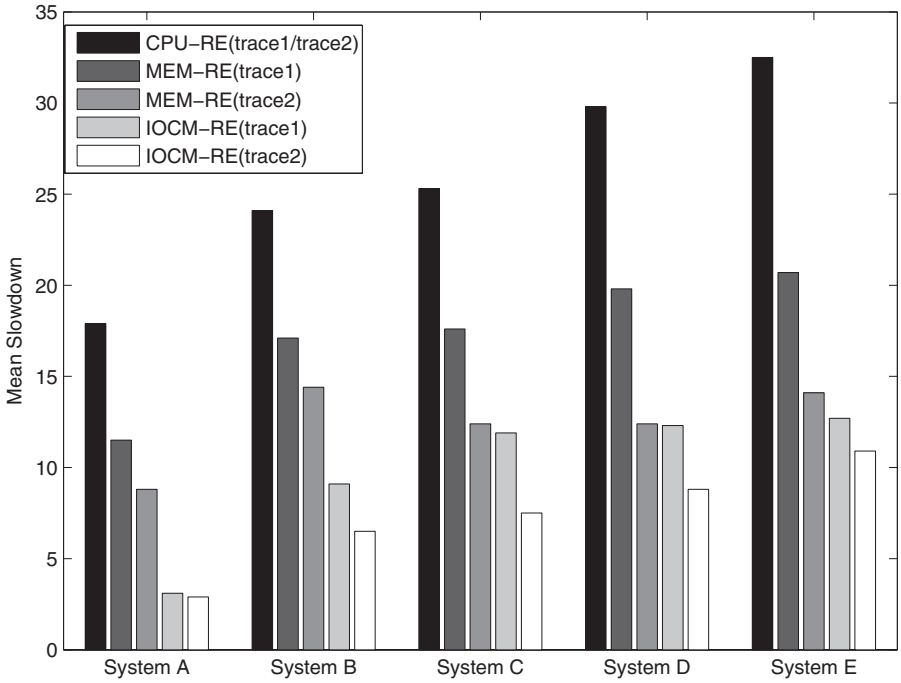

Fig. 16. Mean slowdown when trace 1 and 2 are used on five heterogeneous systems.

that for MEM-RE and IOCM-RE, the slowdowns of Trace 2 are noticeably lower than those of Trace 1. These results suggest that the system performance improves dramatically as the initial data size is decreased, and the reason can be explained as follows. The initial data of a remotely executed task 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 the load-balancing schemes. Thus, small amount of migrated data results in low remote execution overheads, which in turn help to alleviate network burdens. The indication of these results is that our approach can achieve additional performance improvements by reducing data migration overheads.

4.3.4 *Sensitivity to I/O Demands.* Figure 17 plots the performance of CPU-RE, MEM-RE, and IOCM-RE for Traces 2 and 3. Figure 17 shows that for all policies on each cluster, the mean slowdown of Trace 2 is significantly lower than that of 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 18 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. The workload parameters of the modified trace are identical to those of Trace 2, except that the average I/O access rate is gradually increased. Figure 19 reveals that slowdowns of the three policies increase with the increasing I/O access rate,

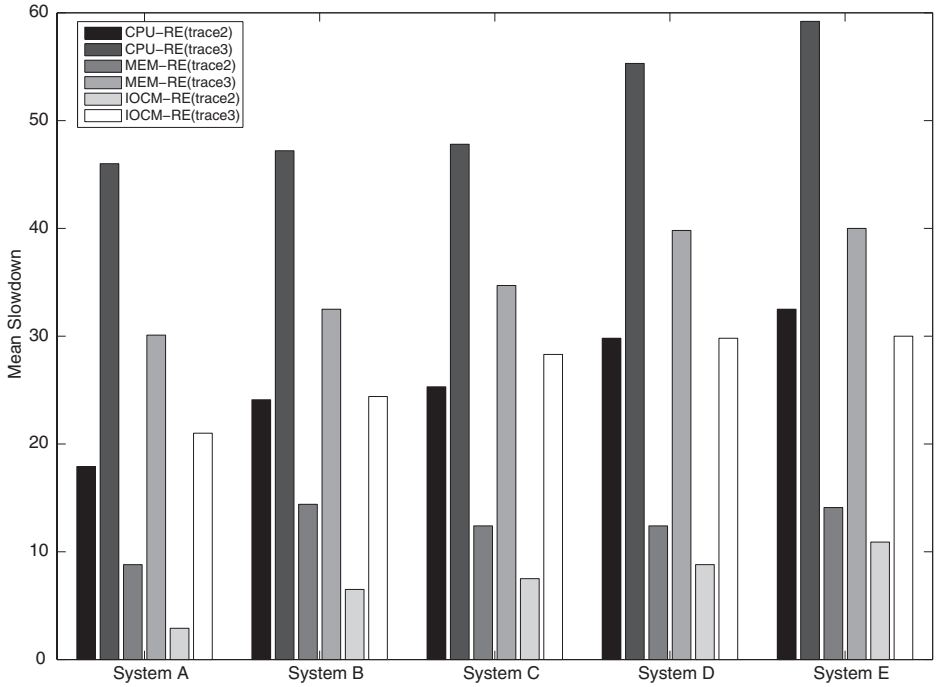

Fig. 17. Mean slowdown when trace 2 and 3 are used on five heterogeneous systems.

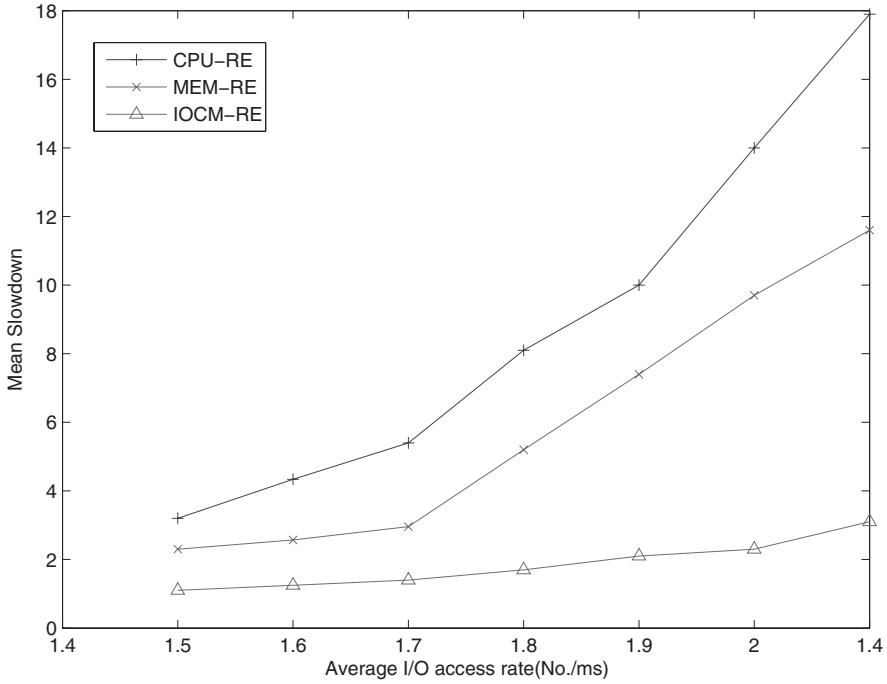

Fig. 18. Mean slowdown under a modified trace 2 on System A.

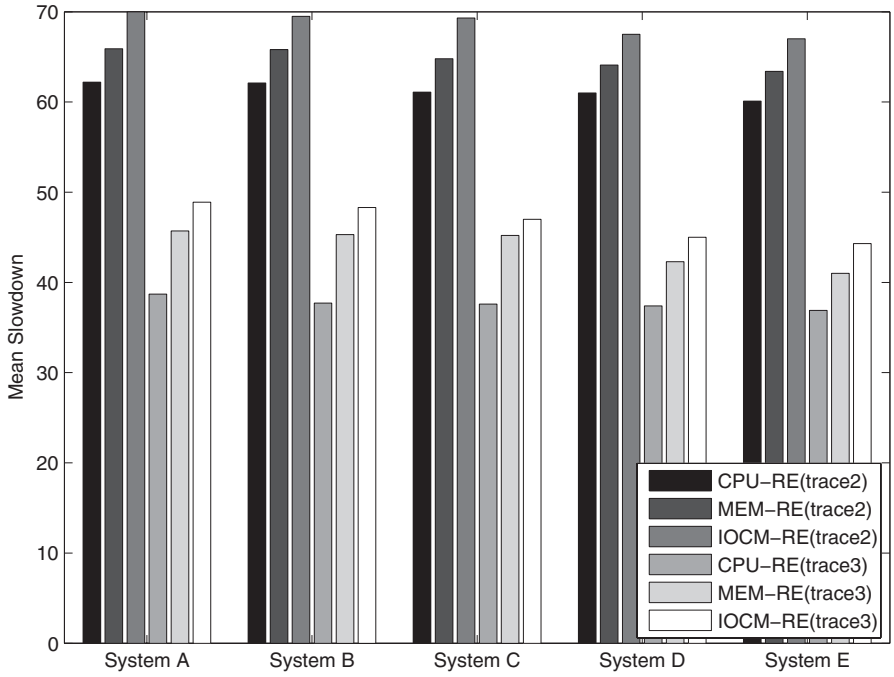

Fig. 19. Buffer hit ratios of five systems when trace 2 and trace 3 are used.

because high I/O access rate leads to heavy disk I/O loads, which in turn give rise to long I/O waiting times. The slowdowns of CPU-RE and MEM-RE increase more quickly than that of IOCM when the I/O access rate is increased, indicating that CPU-RE and MEM-RE are more sensitive to changes in I/O access rate than IOCM-RE under I/O-intensive workload conditions. The explanation is that IOCM-RE achieves a highly effective usage of global disk I/O resources, which dominate the overall performance under I/O-intensive workloads.

4.3.5 *Effectiveness of Improving I/O Buffer Hit Rate.* The IOCM-RE policy is capable of boosting the utilization of I/O buffers, which decreases I/O access frequencies. Figure 19 depicts the I/O buffer hit rates for Traces 2 and 3 using CPU-RE, MEM-RE, and IOCM-RE. Three observations can be made from Figure 19. First, the buffer hit rate of IOCM-RE is consistently higher than those of CPU-RE and MEM-RE. 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. The improvements in turn enable the overall performance to be increased by 93.4% and 33.2% (see Figure 17), respectively. The overall performance gains can be attributed to the high buffer hit rates that help reduce both paging times and I/O processing times. Second, increasing the system heterogeneity results in a slight reduction in the buffer hit ratio, thereby worsening the overall performance in terms of slowdowns. Third, Figure 19 shows that 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.

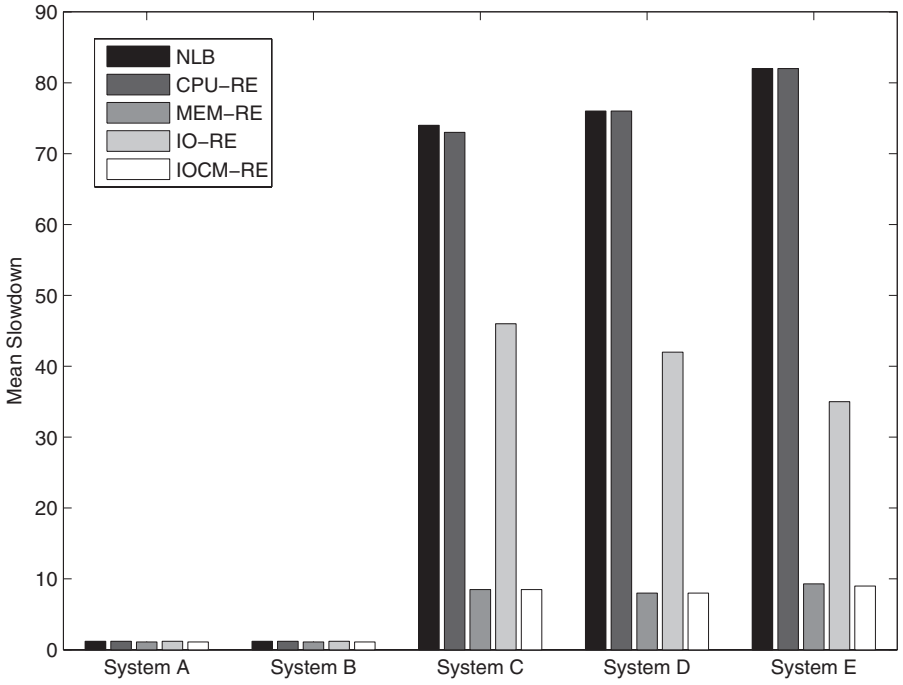

Fig. 20. Mean slowdowns of five systems under memory-intensive workload. Trace 4 is used.

4.3.6 *Memory-Intensive Workloads.* Figure 20 shows that IOCM-RE gracefully degrades to the MEM-RE scheme under memory-intensive workloads. A second interesting observation from Figure 20 is that MEM-RE and IOCM-RE are less sensitive to the heterogeneity levels than the other three policies. For example, the slowdown of CPU-RE in System E is more than 223 times higher than that of CPU-RE in System A, whereas the IOCM-RE's slowdown for System E is only 29 times higher than that for System A. The reason for these results is that MEM-RE and IOCM-RE can mask heterogeneity effects by migrating tasks with high memory demands to the nodes with adequate free memory resources in the clusters.

4.3.7 *Real I/O-Intensive Applications.* We now use the same method described in Section 3.3.3 to simulate six traces with real I/O-intensive applications. In this set of experiments 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 21 denotes the age of the slow disk in the cluster, whereas the y-axis represents the mean slowdown of the real-world applications. Figure 22 shows that IOCM-RE achieves performance improvements over CPU-RE and MEM-RE ranging from 40% to 129%. For all the six applications the mean slowdowns increase as the slow disk ages, because aging a disk results in a higher level of disk I/O heterogeneity, which makes long I/O processing times. Figure 21 illustrates that the performance of IOCM-RE is less sensitive to the change in the age of the slow disk than CPU-RE and MEM-RE. For example, when the trace with DB2 applications

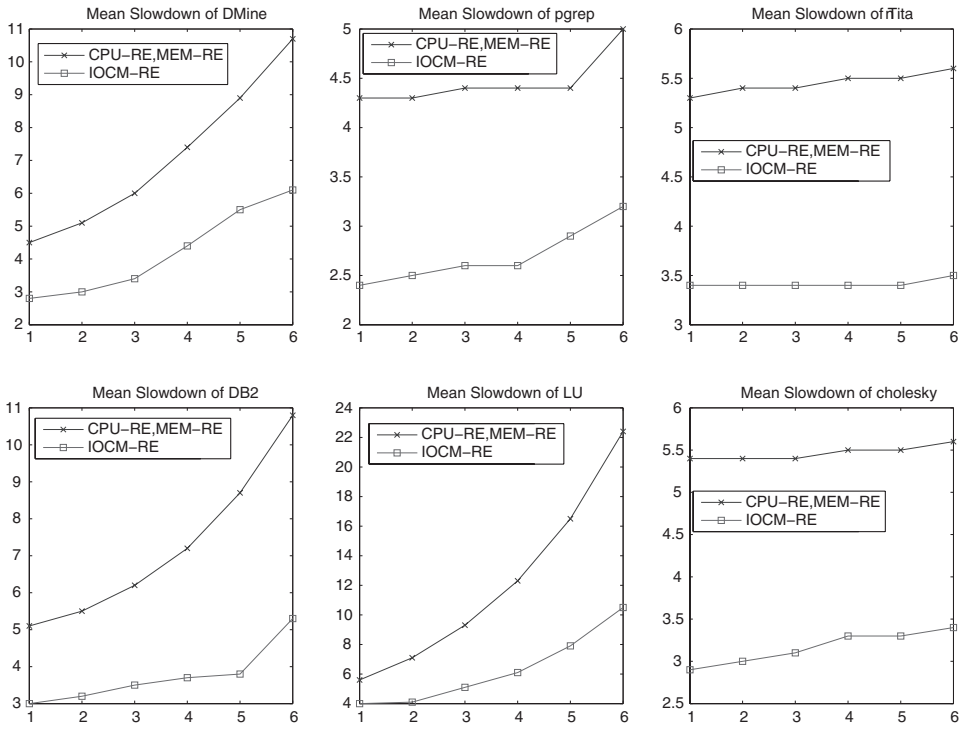

Fig. 21. Mean slowdowns as a function of the age of a single disk.

is evaluated, 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 the traces with pgrep, Titan, and Cholesky. The sensitivity to disk heterogeneity levels partially depends on the ratio of a job's I/O processing time to its total execution time, which can be used to quantify I/O-intensive levels of applications. Recall that for Dmine, DB2, and LU the percentages of total execution time spent in performing I/O operations are 74%, 82%, and 99%, respectively (see Figure 12). In contrast, such percentages for the other three applications are as relatively low as 36%, 24%, and 40%, respectively. Thus, Dmine, DB2, and LU are more I/O-intensive than the other three applications, meaning that Dmine, DB2, and LU are expected to spend more time in sharing disk I/O resources with other running jobs. Consequently, the traces with Dmine, DB2, and LU are more sensitive to disk heterogeneity levels.

4.4 Summary

In the second part of the study, we addressed the issue of balancing load under I/O- and memory-intensive workload for heterogeneous clusters. In particular, we developed two I/O-aware load balancing policies, namely IO-RE

(I/O-based policy) and IOCM-RE (load balancing for I/O, CPU, and Memory). We observed from our experimental results that for almost all the policies, slowdowns increase consistently with 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.

5. IMPLEMENTATION CONSIDERATIONS

5.1 Measuring I/O Load

To implement our proposed I/O-aware load balancing schemes, we have to measure the implicit and explicit I/O load for each running task. In our load balancing mechanism, the memory space requested by task j is specified by application developers. The mechanism keep track of the available memory space (i.e., M_i) on node i. When M_i can not satisfy the accumulative memory requirements of running tasks, the node encounters page faults, leading to implicit I/O load, which depends on three factors: the available user memory space, the page fault rate, and the memory space requested by running tasks. Similar measurement for implicit I/O load can be readily found in the literature (see, for example, Xiao et al. [2000] and Zhang et al. [2000]). In a typical real-world setting, it may be challenging to accurately estimate a task's memory needs. However, the implicit I/O load of a job can be explicitly measured by monitoring its auxiliary memory traffic. This argument is supported by previous results from Carr and Hennessy [1981]. Explicit I/O loads can be efficiently measured using disk throughput data [Vazhkudai and Schopf 2002]. Reasonable estimations for explicit I/O loads can be obtained by profiling the cluster (see, for example, Chow and Kwok [2002]). In our implementation, we measure explicit I/O load using I/O access patterns and buffer hit rates, which are monitored on-the-fly (see Varman and Verma [1999] for an analysis of buffer management). The access pattern of tasks can be characterized by I/O access rates and data sizes.

5.2 Reducing Remote Execution Cost

The remote execution cost of a task depends on a fixed cost of migrating the job and its initial data that is measured by a profiling tool or by a code analysis tool. Similar ways of measuring remote execution cost can be found in the literature (see, for example, Harchol-Balter and Downey [1996] and Zhang et al. [2000]). In practice, data migration overhead can be measured by a performance monitor [Agarwala et al. 2003; Basney and Livny 2000], which stores the most recent values of the disk and network workload. For read-intensive applications, there is unnecessary to migrate all initial data sets, since only popular data sets that are frequently accessed need to be migrated. Popular data sets can be identified by either profiling or by analyzing the codes of read-intensive applications. To reduce remote execution and migration overheads, we can dynamically predict and replicate popular data sets. Thus, the performance of the I/O-aware load balancing schemes can be improved if the amount of initial data that must be migrated can be accurately predicted and replicated at run

time. For write-intensive applications, remote execution cost does not dominate the performance of applications because most files in write-intensive applications are write once. When it comes to read-intensive applications, existing caching/buffering techniques (e.g., active buffering and storage-aware caching mechanisms) can efficiently reduce remote execution and migration costs by migrating a smaller amount of initial data.

5.3 Predicting Response Time

The response time of a job is utilized to decide if the job's remote execution can improve the performance. Therefore, a response time predictor must be implemented in the I/O-aware load balancing mechanism. The response time predicting module is developed based on experimental and theoretical considerations. For example, we consider the round-robin scheme (time-sharing) employed as the CPU scheduling policy. We characterize each disk as a single M/G/1 queue. Similar approximations for CPU and I/O processing times can be found in the literature (see, for example, Lee et al. [2000], Brown [1979], and Kim [1986]). I/O operations in real systems can be either synchronous or asynchronous. It is assumed in our mechanism that all I/O operations are synchronous, because many I/O-intensive parallel applications issue synchronous read/write operations [Surdeanu et al. 2002; Uysal et al. 1997]. This assumption is conservative in the sense that it underestimates load balancing benefits (i.e., this assumption causes a number of undesired migrations with negative impact). To implement the IOCM-PM scheme that judiciously selects an eligible task in EM (see Section 3.1.2) from the overloaded node to migrate, we extend the response time predicting module to estimate the expected response time of a candidate migrant. In doing so, IOCM-PM can ensure that the expected response time of an eligible migrant on the source node is greater than the sum of its expected response time on the destination node and the migration cost.

5.4 Measuring Migration Cost

The migration cost for preemptive migration includes the fixed cost and the time spent on transmitting migrated data over the network and on accessing source and destination disks. Note that migrated data sets are obtained by a performance monitor in a typical real-world setting [Agarwala et al. 2003; Basney and Livny 2000]. In some write-intensive applications like long running simulations, a large number of snapshots and checking points are spawned by write-only operations. Since snapshots and checking points are unlikely to be frequently retrieved again by the same job, there is no need to move such write-only data when the job is migrated. Therefore, migration overhead for such I/O-intensive applications is usually very low. To maximize the migration benefit gained by our I/O-aware load-balancing scheme, we implement an objective function called *migration cost-effectiveness*, which measures the amount of I/O load migrated per unit migration cost. The best task to be migrated is the one with the maximum migration cost-effectiveness value.

5.5 Implementing Load Managers

When a job is submitted to its home node, a load manager assigns the job to a node (for the sequential job) or a group of nodes (for the parallel job) with the least load. The load manager continues to receive reasonably up-to-date global load information from the head node, which monitors resource utilization of the cluster and periodically broadcasts global load information to other nodes of the cluster. If the load manager detects that the local node is heavily loaded, a migration will be carried out to transfer an eligible process to a node with the lightest load. Each parallel job consists of a number of tasks, the tasks of a parallel job are assumed to synchronize with one another [Dusseau et al. 1996]. 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. It is worth noting that the resource requirements of tasks can be estimated by code profiling and statistical prediction [Braun et al. 1999]. Each node serves several tasks in a time-sharing fashion so that the tasks can dynamically share the cluster resources.

5.6 Integration with MPI and File Systems

We need to decide which MPI (Message Passing Interface) implementation to use in our proposed load balancing mechanism. The MPI management not only has to support efficient process migration but also has facilities for monitoring CPU, Memory, and I/O usage of individual nodes to make load balancing feasible. We chose the LAM/MPI implementation because it met our requirements. There has been research on how to extend LAM/MPI base capabilities to support efficient process migration [Cao et al. 2005; Singh and Graham 2008]. Our work is heavily dependent on the home node concept, which assumes the node where a process is executed on also has the data the process requires. MapReduction is a technique that improves the performance of parallel applications and leverages the home node assumption [Dean and Ghemawat 2008]. In addition, LAM/MPI was designed with the assumption that there would be heterogeneous nodes in the cluster computer and this matches our assumption of our load balancing schemes working on either homogenous or heterogeneous cluster computing systems. LAM/MPI also has the additional benefit of being run in user space, which should allow easy integration with existing cluster computers. Another major decision we had to make was the choice of the file system we would be integrated in our load balancing mechanism for parallel clusters. An ideal file system supporting our mechanism has to be designed for parallel applications. The ideal file system also needs to be implemented in user space to allow the easy integration with our load balancing mechanism. The Google file system seemed like a good choice because it was designed for parallel applications and was written in user space [Ghemawat et al. 2003]. A main problem of using the Google file system in our case is that it was written with particular constraints that may not be necessary for various cluster

computing systems. Therefore we decided that PVFS would be a better choice for the file system for the cluster system that would support our research. It has been demonstrated that PVFS can be used for load balancing for certain applications and PVFS is also written in user space [Vydyanathan et al. 2004]. To implement our I/O-aware load balancing mechanism in a cluster computing system, we must integrate CPU, Memory, and I/O usage of nodes into a scheduler for the cluster system. The open source nature of LAM/MPI and PVFS allows for modifications to LAM/MPI that can support our proposed load balancer. Load balancing has been implemented using monitors in the LAM/MPI platform [Deng et al. 2005]. The I/O monitor integrated with LAM/MPI allows for the tuning of PVFS to support process migration and load balancing in an efficient manner.

6. CONCLUSIONS

In this article, we have addressed the issue of balancing the load for I/O- and/or memory-intensive applications running on clusters. In the first part of the study, we investigate our schemes on a homogeneous cluster that consists of a group of identical nodes. The proposed schemes achieve a significant performance improvement over the existing CPU- and Memory-based policies by considering both explicit and implicit I/O load. The empirical results show that the proposed schemes are more general than the existing approaches in the sense that ours can maintain high performance under a wide variety of workload conditions including: CPU-, memory-, and I/O-intensive workload.

To fit the needs of a large fraction of clusters that are heterogeneous in practice, we develop two load balancing schemes for heterogeneous clusters where computational nodes have different performance characteristics in computing power, memory capacity, and disk speed. We develop analytic models to study mean slowdowns, task arrival, and transfer processes in system levels. Using a group of job traces with both synthetic and real application I/O demands, we show that, compared with the existing load balancing approaches, ours are not only effective in improving performance, but also less sensitive to changes in heterogeneity level.

The proposed approaches can be considered complementary to the previous techniques such as active buffering, storage-aware caching, date replication algorithms and a feedback control mechanism. In other words, by combining the above techniques, more intelligent load-balancing decisions can be made to provide additional performance improvement.

Due to long runtimes, we have studied the performance of a cluster with 6 and 32 nodes, respectively. Therefore, future research will deal with a rigorous testing experiment where the performance of a cluster with more than 1000 nodes will be evaluated. Since conducting this experiment largely depends on the simulator, we will make an effort to develop a parallel simulator to efficiently investigate the performance of the I/O-aware load balancing schemes for large-scale clusters.

The corresponding source code for the simulator used in this work can be found at http://www.eng.auburn.edu/ xqin/software/ioBalanceSim/.

REFERENCES

ACHARYA, A. AND SETIA, S. 1999. Availability and utility of idle memory in workstation clusters. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'99)*. ACM, New York, NY, 35–46.

AGARWALA, S., AGARWALA, I., POELLABAUER, C., KONG, J., SCHWAN, K., AND WOLF, M. 2003. Resource-aware stream management with the customizable dproc distributed monitoring mechanisms. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*. 250–259.

BASNEY, J. AND LIVNY, M. 2000. Managing network resources in condor. In *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing (HPDC'00)*. IEEE Computer Society, Los Alamitos, CA, 298.

BODE, B., HALSTEAD, D. M., KENDALL, R., LEI, Z., AND JACKSON, D. 2000. The portable batch scheduler and the maui scheduler on linux clusters. In *Proceedings of the 4th Annual Linux Showcase & Conference (ALS'00)*. USENIX Association, Berkeley, CA, 27–27.

BRAUN, T. D., SIEGAL, H. J., AND BECK, ET AL. 1999. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. In *Proceedings of the 8th Heterogeneous Computing Workshop (HCW'99)*. IEEE Computer Society, Los Alamitos, CA, 15.

BROWN, T. 1979. M/g/1 round robin discipline. *Computing 22*, 3, 225–241.

CAO, J., LI, Y., AND GUO, M. 2005. Process migration for mpi applications based on coordinated checkpoint. In *Proceedings of the 11th International Conference on Parallel and Distributed Systems (ICPADS'05)*. IEEE Computer Society, Los Alamitos, CA, 306–312.

CARNS, P. H., LIGON, W. B., III, ROSS, R. B., AND THAKUR, R. 2000. Pvfs: A parallel file system for linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*. USENIX Association, 317–327.

CARR, R. W. AND HENNESSY, J. L. 1981. Wsclock—a simple and effective algorithm for virtual memory management. *SIGOPS Oper. Syst. Rev. 15*, 5, 87–95.

CHANG, C., MOON, B., ACHARYA, A., SHOCK, C., SUSSMAN, A., AND SALTZ, J. 1997. Titan: a high-performance remote-sensing database. In *Proceedings of the International Conference on Data Engineering*. IEEE Computer Society Press, 375–384.

CHOW, K.-P. AND KWOK, Y.-K. 2002. On load balancing for distributed multiagent computing. *IEEE Trans. Parall. Distrib. Syst. 13*, 8, 787–801.

CRUZ, J. AND PARK, K. 2001. Towards communication-sensitive load balancing. In *Proceedings of the 21st International Conference on Distributed Computing Systems,* 731–734.

DEAN, J. AND GHEMAWAT, S. 2008. Mapreduce: Simplified data processing on large clusters. *Comm. ACM 51*, 1, 107–113.

DENG, Q., WANG, X., AND ZANG, D. 2005. Monitoring MPI running nodes status for load balance. In *Proceedings of the 4th International Conference on Grid and Cooperative Computing*. Lecture Notes in Computer Science, vol. 3795, Springer, Berlin, 467–473.

DUSSEAU, A. C., ARPACI, R. H., AND CULLER, D. E. 1996. Effective distributed scheduling of parallel workloads. *SIGMETRICS Perform. Eval. Rev. 24*, 1, 25–36.

EAGER, D. L., LAZOWSKA, E. D., AND ZAHORJAN, J. 1986. Adaptive load sharing in homogeneous distributed systems. *IEEE Trans. Softw. Engin. 12*, 5, 662–675.

FORNEY, B. C., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. 2002. Storage-aware caching: Revisiting caching for heterogeneous storage systems. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST'02)*. USENIX Association, Berkeley, CA, 5.

GEOFFRAY, P. 2002. Opiom: off-processor i/o with myrinet. *Future Gener. Comput. Syst. 18*, 4, 491–499.

GHEMAWAT, S., GOBIOFF, H., AND LEUNG, S.-T. 2003. The google file system. *SIGOPS Oper. Syst. Rev. 37*, 5, 29–43.

HARCHOL-BALTER, M. AND DOWNEY, A. B. 1996. Exploiting process lifetime distributions for dynamic load balancing. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'96)*. ACM, New York, NY, 13–24.

HUI, C.-C. AND CHANSON, S. T. 1999. Improved strategies for dynamic load balancing. *IEEE Concurrency 7*, 3, 58–67.

KANNAN, S., ROBERTS, M., MAYES, P., BRELSFORD, D., AND SKOVIRA, J. F. 2001. *Workload Management with LoadLeveler*. IBM.

KEREN, A. AND BARAK, A. 2003. Opportunity cost algorithms for reduction of i/o and interprocess communication overhead in a computing cluster. *IEEE Trans. Parall. Distrib. Syst. 14*, 1, 39–50.

KIM, M. Y. 1986. Synchronized disk interleaving. *IEEE Trans. Comput. 35*, 11, 978–988.

KOTZ, D. AND NIEUWEJAAR, N. 1994. Dynamic file-access characteristics of a production parallel scientific workload. In *Proceedings of the Conference on Supercomputing (Supercomputing'94)*. IEEE Computer Society Press, Los Alamitos, CA, 640–649.

LAVI, R. AND BARAK, A. 2001. The home model and competitive algorithms for load balancing in a computing cluster. In *Proceedings of the The 21st International Conference on Distributed Computing Systems (ICDCS'01)*. IEEE Computer Society, Los Alamitos, CA, 127.

LEE, L.-W., SCHEUERMANN, P., AND VINGRALEK, R. 2000. File assignment in parallel i/o systems with minimal variance of service time. *IEEE Trans. Comput. 49*, 2, 127–140.

MA, X., WINSLETT, M., LEE, J., AND YU, S. 2002. Faster collective output through active buffering. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS'02)*. IEEE Computer Society, Los Alamitos, CA, 151.

PASQUALE, B. K. AND POLYZOS, G. C. 1994. Dynamic i/o characterization of i/o intensive scientific applications. In *Proceedings of the Conference on Supercomputing (Supercomputing'94)*. IEEE Computer Society Press, Los Alamitos, CA, 660–669.

QIN, X. 2008. Performance comparisons of load balancing algorithms for i/o-intensive workloads on clusters. *J. Netw. Comput. Appl. 31*, 1, 32–46.

QIN, X., JIANG, H., ZHU, Y., AND SWANSON, D. R. 2003a. Dynamic load balancing for i/o- and memory-intensive workload in clusters using a feedback control mechanism. In *Proceedings of the 9th International Euro-Par Conference on Parallel Processing (Euro-Par'03*). 224–229.

QIN, X., JIANG, H., ZHU, Y., AND SWANSON, D. R. 2003b. Dynamic load balancing for i/o-intensive tasks on heterogeneous clusters. In *Proceedings of the 10th International Conference on High Performance Computing (HiPC'03)*. 300–309.

QIN, X., JIANG, H., ZHU, Y., AND SWANSON, D. R. 2003c. A dynamic load balancing scheme for i/o-intensive applications in distributed systems. *Proceedings of the International Conference on Parallel Processing Workshops*. 79.

ROADS, J. AND ET AL. 1992. A preliminary description of the western u.s. climatology. In *Proceedings of the Annual Pacific Climate Workshop*.

SINGH, R. AND GRAHAM, P. 2008. Performance driven partial checkpoint/migrate for lam-mpi. In *Proceedings of the 22nd International Symposium on High Performance Computing Systems and Applications (HPCS'08)*. IEEE Computer Society, Los Alamitos, 110–116.

SURDEANU, M., MOLDOVAN, D. I., AND HARABAGIU, S. M. 2002. Performance analysis of a distributed question/answering system. *IEEE Trans. Parall. Distrib. Syst. 13*, 6, 579–596.

TANAKA, T. 1993. Configurations of the solar wind flow and magnetic field around the planets with no magnetic field: Calculation by a new mhd. *Geophys. Res.*, 17251–17262.

UYSAL, M., ACHARYA, A., AND SALTZ, J. 1997. Requirements of i/o systems for parallel machines: an application-driven study. Tech. rep., College Park, MD.

VARMAN, P. J. AND VERMA, R. M. 1999. Tight bounds for prefetching and buffer management algorithms for parallel i/o systems. *IEEE Trans. Parall. Distrib. Syst. 10*, 12, 1262–1275.

VAZHKUDAI, S. AND SCHOPF, J. M. 2002. Using disk throughput data in predictions of end-to-end grid data transfers. In *Proceedings of the 3rd International Workshop on Grid Computing (GRID'02)*. Springer-Verlag, Berlin, 291–304.

VOELKER, G. M., JAMROZIK, H. A., VERNON, M. K., LEVY, H. M., AND LAZOWSKA, E. D. 1997. Managing server load in global memory systems. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'97)*. ACM, New York, NY, 127–138.

VYDYANATHAN, N., KHANNA, G., KURC, T., CATALYUREK, U., WYCKOFF, P., SALTZ, J., AND SADAYAPPAN, P. 2004. Use of pvfs for efficient execution of jobs with pipeline-shared i/o. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID'04)*. IEEE Computer Society, Los Alamitos, CA, 235–242.

XIAO, L., ZHANG, X., AND QU, Y. 2000. Effective load sharing on heterogeneous networks of workstations. In *Proceedings of the of International Symposium on Parallel and Distributed Processing*. IEEE Computer Society Press, 431–438.

ZHANG, X., XIAO, L., AND QU, Y. 2000. Improving distributed workload performance by sharing both cpu and memory resources. In *Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS'00)*. IEEE Computer Society, Los Alamitos, CA, 233.

ZHANG, Y., YANG, A., SIVASUBRAMANIAM, A., AND MOREIRA, J. 1993. Gang scheduling extensions for i/o intensive workloads. In *Proceedings of the Job Scheduling Strategies for Parallel Processing Workshop*.

ZHU, Y., JIANG, H., QIN, X., AND SWANSON, D. 2004. A case study of parallel i/o for biological sequence search on linux clusters. *Int. J. High Perform. Comput. Netw. 1*, 4, 214–222.

Received October 2007; revised March 2009; accepted August 2009