Cluster Comput (2006) 9:297–311
DOI 10.1007/s10586-006-9742-7

# Improving the performance of I/O-intensive applications on clusters of workstations

**Xiao Qin · Hong Jiang · Yifeng Zhu · David R. Swanson**

Received: 1 September 2003 / Revised: 1 March 2004 / Accepted: 1 May 2004


**Abstract** Load balancing in a workstation-based cluster system has been investigated extensively, mainly focusing on the effective usage of global CPU and memory resources. However, if a significant portion of applications running in the system is I/O-intensive, traditional load balancing policies can cause system performance to decrease substantially. In this paper, two I/O-aware load-balancing schemes, referred to as IOCM and WAL-PM, are presented to improve the overall performance of a cluster system with a general and practical workload including I/O activities. The proposed schemes dynamically detect I/O load imbalance of nodes in a cluster, and determine whether to migrate some I/O load from overloaded nodes to other less- or under-loaded nodes. The current running jobs are eligible to be migrated in WAL-PM only if overall performance improves. Besides balancing I/O load, the scheme judiciously takes into account both CPU and memory load sharing in the system, thereby maintaining the same level of performance as existing schemes when I/O load is low or well balanced. Extensive trace-driven simulations for both synthetic and real I/O-intensive applications show that: (1) Compared with existing schemes that only consider CPU and memory, the proposed schemes improve the performance with respect to mean slowdown by up to a factor of 20; (2) When compared to the existing approaches that only consider I/O with non-preemptive job migrations, the proposed schemes achieve improvements in mean slowdown by up to a factor of 10; (3) Under CPU-memory intensive workloads, our scheme improves the performance over the existing approaches that only consider I/O by up to 47.5%.

**Keywords** I/O intensive · Clusters · Slowdown · Performance evaluation

X. Qin (✉)
Department of Computer Science, New Mexico Institute of Mining and Technology, 801 Leroy Place, Socorro, New Mexico 87801-4796, USA
e-mail: xqin@cs.nmt.edu

H. Jiang · Y. Zhu · D. R. Swanson
Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, Nebraska 68588-0115, USA
e-mail: jiang@cse.unl.edu

Y. Zhu
e-mail: yzhu@cse.unl.edu

D. R. Swanson
e-mail: dswanson@cse.unl.edu

## 1 Introduction

In a cluster of workstations, load balancing schemes can improve system performance by assigning work, at run time, to machines with idle or under-utilized resources. Figure 1 illustrates the architecture of a workstation-based cluster system, where each node has a combination of multiple types of resources, such as CPU, memory, network connectivity, and disks. In the recent past, some efforts have focused on storing data for I/O-intensive applications in a huge disk array or storage area network.

However, in this study we choose to utilize the commodity IDE disks that already exist as an integral part of each cluster node. This is because by August 2003, the average price of commodity IDE disks has decreased below US$0.5/GB. Our approach is a desirable way to develop cost-effective clusters in the sense that the approach provides high performance storage services without requiring any additional hardware upgrades. Furthermore, our approach can potentially sustain the high bandwidth requirements of I/O-intensive applications, thereby making clusters more scalable. In contrast, cluster-

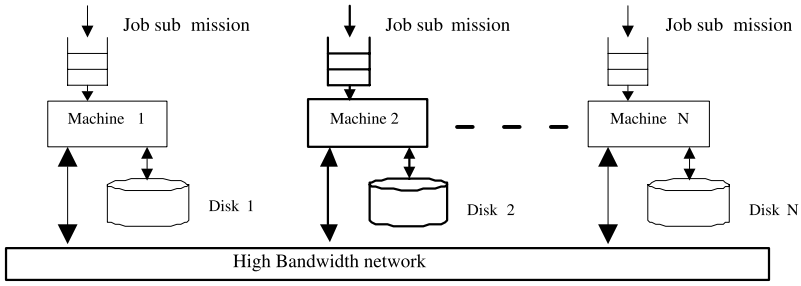



**Fig. 1** Architecture of a cluster system

attached disk arrays or storage area networks rely on single node servers, where all the disks are connected with one node. However, the snigle node servers tend to become performance bottlenecks for large-scale clusters. Although such a bottleneck can be alleviated by a variety of techniques such as prefeching and collective I/O, these techniques are beyond the scope of this paper.

Several distributed load-balancing schemes based on this architecture have been presented in the literature, considering CPU [10, 12], memory [1, 29], network [7], a combination of CPU and memory [33, 34], or a combination of CPU and network [3]. Although these policies have been effective in increasing the utilization of resources in distributed systems, they have ignored disk I/O. The impact of disk I/O on overall system performance becomes significant as more and more data-intensive and/or I/O-intensive applications are running on clusters. The speed gap between CPU and disk I/O is fast widening [35, 36], which makes storage devices a likely performance bottleneck. Therefore, dynamic load balancing schemes have to "I/O-aware" to achieve high performance in this new application environment [20, 22].

Typical examples of I/O-intensive applications include long running simulations of time-dependent phenomena that periodically generate snapshots of their state [27], archives of raw and processed remote sensing data [5], and out-of-core applications [4], to name just a few. These applications share a common feature in that their disk I/O requirements are extremely high. For example, out-of-core applications have high demands to access data sets that exceed the capacity of physical memory, and it has become conventional wisdom to develop out-of-core applications in a way to explicitly handle data movement in and out of core memory avoiding the use of virtual memory [6]. Load balancing with I/O-awareness, when appropriately designed, is potentially capable of delivering high-performance storage, in addition to the high utilization of I/O buffers.

This paper studies two schemes, referred to as IOCM (load balancing for I/O, CPU, and Memory) and WAL-PM (Weighted Average Load balancing with Preemptive Migration). Each balances load in a cluster environment in such a way that CPU, memory, and I/O resources at each node can be simultaneously well utilized. Extensive simulations for both synthetic and real I/O-intensive applications were performed to compare existing load balancing schemes with IOCM and WAL-PM. Both IOCM and WAL-PM achieve improvement in mean slowdown, informally defined to be the performance degradation of a job due to resource sharing by other jobs, by up to a factor of 20 under I/O-intensive workloads. Compared with existing approaches that consider I/O with non-preemptive migration, the WAL-PM scheme improves performance by up to a factor of 10. In addition, when the workload is CPU-memory intensive, the proposed scheme improves the performance over existing approaches that only consider I/O by 24.4–47.5%.

The rest of the paper is organized as follows. In the section that follows related work in the literature is briefly reviewed. Section 3 presents the IOCM and WAL-PM schemes, and Section 4 evaluates the performances of these schemes. Finally, Section 5 concludes the paper by summarizing the main contributions of this paper.

## 2 Related work

The issue of distributed load balancing for CPU and memory resources has been extensively studied and reported in the literature in recent years. Harchol-Balter et al. [10] proposed a CPU-based preemptive migration policy that was more effective than non-preemptive migration policies. Zhang et al. [34] focused on load sharing policies that consider both CPU and memory services among the nodes. The experimental results showed that their policies not only improve performance of memory-intensive jobs, but also maintain the same load sharing quality of CPU-based policies for CPU-intensive jobs [34].

A large body of work can be found in the literature that addresses the issue of balancing the load of disk systems [13, 16, 35]. Lee et al. [16] proposed two file assignment algorithms that balance the load across all disks. Isert and Schwan studied the runtime adaptation of data streams achieved by migrating objects across machines [13]. 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. However, not all of

them can be directly applied for a complex cluster computing environment where I/O-intensive jobs may share resources with many other memory-intensive and CPU-intensive jobs. Communication-sensitive load balancing has been proposed by Cruz and Park [7]. Although the approaches proposed in our paper take into account communication load as a measure to determine migration overheads, balancing network load is beyond the scope of this paper.

Very recently, a dynamic load-balancing scheme, tailored for the specific requirements of the Question/Answer application, was developed to consider I/O, CPU and memory resources [26]. The migration scheme studied in [26], referred to as WAL-RE (Weighted Average Load with Remote Execution), prohibits remote I/O execution. In contrast, the IOCM scheme in this study allows a job's I/O operations to be conducted by a node that is different from the one in which the job's computation is assigned, thereby permitting a job to access remote I/O.

Zhang et al. also studied the performance of WAL-RE, a non-preemptive load-balancing policy. The new WAL-PM scheme presented in this paper, however, permits a currently running job to be preempted and migrated if its migration is expected to improve overall performance.

Besides WAL-RE, Zhang et al. [32] also proposed a WAL-based preemptive migration policy, which has worse performance than that of WAL-RE under memory-intensive workload. Data migration cost in [32] only considers memory migration, thereby ignoring I/O migration cost as an important component of the migration cost in load-balancing. In contrast, WAL-PM proposed in this study considers both memory and I/O migration cost as a criterion to determine jobs that are eligible for migration. Ranganathan and Foster studied a scheduling framework to improve the performance of data-intensive applications running in Data Grid systems [23]. Our approaches are different from theirs as our performance gain relies on remote I/O access and preemptive migrations rather than data replication algorithms [23]. Complememting the replication algorithms, our load balancing schemes could achieve additional performance improvements by reducing migration cost.

Many researchers showed that I/O cache and buffer are useful mechanisms to optimize storage systems [8, 17, 19]. Ma et al. implemented active buffering to alleviate the I/O burden imposed by I/O-intensive applications by using local idle memory and overlapping I/O with computation [17]. We developed a feedback control mechanism to improve the performance of a cluster by manipulating the I/O buffer size [19]. Forney et al. investigated storage-aware caching algorithms that partition caches and explicitly account for differences in performance across devices in heterogeneous clusters [8]. Although we focus solely on balancing disk I/O load in this paper, the approach proposed here is capable of improving the buffer utilization of each node, which in turn increases the buffer hit rate and reduces disk I/O access frequency. The load-balancing schemes presented in this paper is orthogonal to the existing caching/buffering techniques and, thus, integrating our proposed schemes into the caching/buffering techniques can provide additional performance improvements.

In our previous work, we developed two I/O-aware load-balancing schemes, which consider system heterogeneity in addition to I/O load balancing [20]. However, the preemptive migration technique was not incorporated into these two schemes. Furthermore, we proposed a simple yet effective I/O-aware load-balancing scheme, which assigns I/O intensive sequential and parallel jobs to nodes with light I/O loads [22]. Although this scheme leverages the preemptive migration technique as an efficient means to improve the system performance, it is assumed that remote I/O accesses are prohibited by employing a local disk based file system. The work presented in this paper extends our previous work in load balancing strategies [20, 22] by considering both preemptive migrations and remote I/O accesses.

## 3 IO-aware load balancing policies

In this section, we discuss the problem of dynamic load balancing among a cluster of workstations(the terms workstation and node are used interchangeably), connected by a high-speed network. Each node in the cluster is able to migrate a newly arrived job or a currently running job preemptively to another node if needed, and maintain reasonably up-to-date global load information by periodically exchanging load status with other nodes [21].

We assume that in a realistic cluster structure, every job has a "home" workstation that it prefers for execution [15]. The rationale behind this home model is two-fold: (1) the input data of a job has been stored in the home node and, (2) the job was created on its home node. An implication of this model is that data initially retrieved by a task is available on the task's home node. It is assumed in many load-balancing schemes [10, 26, 34] that no two or more jobs arrive at different nodes at the same time. We also assume the network in our model is fully connected and homogenous in the sense that communication delay between any pair of nodes is the same. This simplification of the network is commonly used in many load-balancing models [10, 12, 26, 34].

### 3.1 IO-CPU-memory (IOCM) based load-balancing policy

In this section, we present IOCM, a dynamic I/O-aware load-balancing scheme. Each job is described by its requirements for CPU, memory, and I/O, which are measured by the number of jobs running in the nodes, Mbytes, and number of disk accesses per ms, respectively.

**Fig. 2** IO-CPU-Memory based load balancing (IOCM)

1: **if** I/O load on node $i$ is not overloaded **then** $M_{IO}(j) \leftarrow$ local node $i$;
**else** $IO(j) \leftarrow$ node with the minimal I/O load;
2: **if** memory in node $i$ is not overloaded **then**
**if** CPU is not overloaded **then** $M_{CM}(j) \leftarrow$ local node $i$;
**else** $M_{CM}(j) \leftarrow$ node with the minimal CPU load;
**else** $M_{CM}(j) \leftarrow$ node with the minimal memory load;
3: **if** $M_{IO}(j) \neq i$ **or** $M_{CM}(j) \neq i$ **then**
**if** $M_{IO}(j) = M_{CM}(j)$ **then** Calculate the migration cost;
**else** Calculate the migration and remote I/O access cost;
4: **if** I/O migration is worthwhile **then**
**if** $M_{IO}(j) \neq i$ **and** initial data is stored in node $i$ **then**
migrate initial data from node $i$ to node $M_{IO}(j)$;
**else** $M_{IO}(j) \leftarrow$ local node $i$;
**if** job migration is worthwhile **then** migrate job $j$ from node $i$ to $M_{CM}(j)$;
**else** $M_{CM}(j) \leftarrow$ local node $i$;
5: Update the load status in node $M_{IO}(j)$ and $M_{CM}(j)$, update network load;

For a job $j$, arriving in a local node $i$, the IOCM scheme attempts to balance three different resources simultaneously following five main steps. First, if node $i$'s I/O load is overloaded, a candidate node, $M_{IO}(j)$, that processes the I/O operations issued by the job, is chosen in accordance with the I/O load status. Node $i$ is I/O-overloaded, if: (1) its I/O load is the highest; and (2) the ratio between the I/O load and the average I/O load across the system is greater than a threshold, which is set to 1.25 in our experiments. This optimal value, which is consistent with the result reported in [31], is obtained from an experiment where the threshold is varied from 1.0 to 2.0. Second, when the memory of node $i$ is overloaded, IOCM judiciously determines another candidate node, $M_{CM}(j)$, the one with the lightest memory load, to execute the job. When the node has sufficient memory space, a CPU-base policy is applied to make the load sharing decision. Third, compute migration cost and remote I/O access cost, and finalize the load balancing decision. Fourth, data migration from node $i$ to $M_{IO}(j)$ is invoked if the migration is helpful in boosting the performance and the data accessed by job $j$ is not initially available in node $M_{IO}(j)$. Likewise, the job is migrated to node $M_{CM}(j)$ if such migration improves the expected performance. Fifth and finally, the network load and the load status in nodes $M_{IO}(j)$ and $M_{CM}(j)$ are updated. A detailed pseudo code of the IOCM scheme is presented in Fig. 2.

In this scheme, three load indices for CPU, memory and I/O resources are described below (see Table 1 for a summary of notation):

(1) The CPU load index of node $i$ is characterized by the number of jobs running on the node [33, 34], denoted as $load_{CPU}(i)$.

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

$$load_{mem}(i) = \sum_{j \in M_i} l_{mem}(j) \tag{1}$$

**Table 1** A summary of notation

| Symbol | Definition |
|---|---|
| $t_j$ | computation time of job $j$ |
| $a_j$ | age of job $j$ |
| $l_{page}(i, j)$ | implicit I/O load of job $j$ assigned to node $i$ |
| $l_{IO}$ | explicit I/O load of job $j$ assigned to node $i$ |
| $r_{mem}(j)$ | memory space requested by job $j$ |
| $n_{mem}(i)$ | available memory space to running jobs on node $i$ |
| $\mu_i$ | page fault rate of node $i$ |
| $\lambda_j$ | I/O access rate of job $j$ assigned to node $i$ |
| $d_{buf}(i, j)$ | buffer size allocated to job $j$ |
| $d_{data}(j)$ | amount of data job $j$ retrieves from or stores to the disk |
| $d_j^{RW}$ | average data size of I/O accesses of job $j$ |
| $d_j^{W}$ | amount of disk (I/O) data generated at the runtime by job $j$ |
| $w_j$ | percentage of I/O operations that store data to the local disk |

where $l_{mem}(j)$ represents the memory requirement of job $j$, and $M_i$ is a set containing all the jobs that are assigned to node $i$.

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

$$load_{IO}(i) = \sum_{j \in M_i} l_{page}(i, j) + \sum_{j \in M_i} l_{IO}(i, j) \tag{2}$$

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

The calculation of I/O load is more complicated because of the need to determine the implicit and explicit I/O loads. Let $r_{mem}(j)$ denote the memory space requested by job $j$, and $n_{mem}(i)$ represent the memory space in bytes that is available to all jobs running on node $i$. When the node's available memory space is larger than or equal to the memory demand, there is no implicit I/O load imposed on the disk. Conversely, when the memory space is unable to meet the memory requirements of the jobs, the node encounters a large number of page faults. The load of implicit I/O largely depends on programs behaviors and buffer replacement policies. The implicit I/O load can be measured by monitoring the inter-page fault interval or an analytical model. For simplicity, we choose to use the following model which has been used by other researchers [33, 34] to approximately determine the implicit I/O load of a job. Note that implicit I/O load is inversely proportional to available user memory space and proportional to the page fault rates and memory space requirements of running jobs. Thus, $l_{page}(i, j)$ is defined as follows,

$$l_{page}(i, j) = \begin{cases} 0 & \text{if } load_{mem}(i) \le n_{mem}(i), \\ \dfrac{\mu_i \sum_{k \in M_i} r_{mem}(k)}{n_{mem}(i)} & \text{otherwise.} \end{cases} \tag{3}$$

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

$$l_{IO}(i, j) = \lambda_j [1 - h(i, j)] \tag{4}$$

The buffer hit rate $h(i, j)$ can be affected by the re-access rate $r_j$ (defined to be the average number of times the same data is repeatedly accessed by job $j$), the buffer size $d_{buf}(i, j)$ allocated to job $j$, and the amount of data $d_{data}(j)$ job $j$ retrieves from or stores to the disk, given a buffer with infinite size. We will discuss the impact of re-access rate on the proposed load-balancing schemes in Section 4.7. The buffer hit rate is approximated by the following formula:

$$h(i, j) = \begin{cases} \dfrac{r_j}{r_j + 1} & \text{if } d_{buf}(i, j) \ge d_{data}(j), \\ \dfrac{r_j d_{buf}(i, j)}{(r_j + 1) d_{data}(j)} & \text{otherwise,} \end{cases} \tag{5}$$

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

$$d_{buf}(i, j) = \frac{\lambda_j d_j^{RW} d_{buf}(i)}{\sum_{k \in M_i} \left(\lambda_k d_j^{RW}\right)} \tag{6a}$$

$$d_{data}(j) = \frac{\lambda_j t_j d_j^{RW}}{r_j + 1} \tag{6b}$$

In practice, I/O access rate and I/O buffer hit rate used in Expression 4 can be dynamically obtained by maintaining a running average for $\lambda_j$ and $h(i, j)$. We now turn to the calculation of the response time of a job with local/remote I/O accesses and the migration cost, which will be utilized in Step (3) to decide if migration can improve the performance. When a job $j$ accesses I/O locally on node $i$, its expected response time can be computed as follows:

$$r(i, j) = t_j E(L_i) + t_j \lambda_j \left\{ E\left(s_{disk}^i\right) + \frac{\Lambda_{disk}^i E\left[\left(s_{disk}^i\right)^2\right]}{2\left(1 - \rho_{disk}^i\right)} \right\}, \tag{7}$$

where $\lambda_j$ is the I/O access rate of job $j$. $E(s_{disk}^i)$ and $E[(s_{disk}^i)^2]$ are the mean and mean-square I/O service time in node $i$, and $\rho_{disk}^i$ is the utilization of the disk in node $i$. $E(L_i)$ represents the mean CPU queue length $L_i$, and $\Lambda_{disk}^i$ denotes the aggregate I/O access rate in node $i$.

The two terms on the right hand side of Eq. (7) represent the CPU execution time and the I/O processing time, respectively. It is assumed in our model that all I/O operations are blocking [26]. In other words, computations and I/O operations are not overlapped. Thus, the response time of a job is the summation of CPU and I/O reponse time. This simplification is conservative in the sense that it makes the proposed I/O-aware load-balancing schemes less effective.

Round-robin scheduling (time-sharing) is employed as the CPU scheduling policy, and the disk of each node is modeled as a single M/G/1 queue [16]. The aggregate I/O access rate, $\Lambda^i_{disk}$, is defined as: $\Lambda^i_{disk} = \sum_{k\in M_i} \lambda'_k$, where $\lambda'_k = \frac{\lambda_k}{E(L_i)}$, and $\lambda'_k$ is the effective I/O access rate imposed on the disk by job $k$, taking the effect of time-sharing into account. To accurately estimate the effective I/O access rate, $\lambda_k$, measured in a non-shared environment, must be deflated by the time-sharing factor, which is $E(L_i)$. Based on $\lambda'_k$, the disk utilization can be expressed as: $\rho^i_{disk} = \sum_{k\in M_i}(\lambda_k s^k_{disk})$.

Let $p^k_{disk}$ be the probability of an I/O access being from job $k$ on node $i$, we then have $p^k_{disk} = \lambda'_k/\Lambda_i$. Therefore, the mean I/O service time, used in Eq. (7), can be calculated as follows:

$$E\left(s^i_{disk}\right) = \sum_{k\in M_i}\left(\rho^k_{disk} s^k_{disk}\right) = \frac{1}{\Lambda_i}\sum_{k\in M_i}\left(\lambda'_k s^k_{disk}\right) = \frac{\rho^i_{disk}}{\Lambda^i_{disk}} \tag{8a}$$

$$E\left[\left(s^i_{disk}\right)^2\right] = \sum_{k\in M_i}\left(\rho^k_{disk}\left(s^k_{disk}\right)^2\right) = \frac{1}{\Lambda_i}\sum_{k\in M_i}\left(\lambda'_k\left(s^k_{disk}\right)^2\right) \tag{8b}$$

Let $p^k_{CPU}$ denote the probability of a job $k$ being executed by CPU or waiting in the CPU queue, as opposed to waiting for I/O access. We have $p^k_{CPU} = t_k/(t_k + t_k\lambda_k s^k_{disk}) = 1/(1 + \lambda_k s^k_{disk})$. Thus, the mean CPU queue length, used in Eq. (7), becomes: $E(L_i) = \sum_{k\in M_i} p^k_{CPU} = \sum_{k\in M_i} \frac{1}{1+\lambda_k s^k_{disk}}$.

We now turn our attention to the response time of a job with remote I/O accesses. The network links are modeled as a single M/G/1 queue [24]. Let $r(i, k, j)$ be the response time of job $j$ on node $i$ remotely accessing data on node $k$ ($k \neq i$). Thus we have:

$$r(i,k,j) = t_j E(L_i) + t_j\lambda_j\left\{E\left(s^k_{disk}\right) + \frac{\Lambda^k_{disk}E\left[\left(s^k_{disk}\right)^2\right]}{2\left(1-\rho^k_{disk}\right)}\right\} + t_j\lambda_j\left\{E\left(s^{ik}_{net}\right) + \frac{\Lambda^{ik}_{net}E\left[\left(s^{ik}_{net}\right)^2\right]}{2\left(1-\rho^{ik}_{net}\right)}\right\}, \tag{9}$$

where $E(s^{ik}_{net})$ and $E[(s^i_{net})^2]$ are the mean and mean-square network service time, $\rho^{ik}_{net}$ denotes the utilization of the network link, and $\Lambda^{ik}_{net}$ is the aggregate communication request rate. The three terms on the right hand side of Eq. (9) rerepresent the CPU execution time, the I/O processing time, and the network communication time, respectively. For simplicity, we assume that, for a remote I/O operation, there is no overlap in time between I/O processing and communication [26]. This simplification is conservative in nature, thereby making the proposed schemes less effective.

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

$$c_j = \begin{cases} e & \text{if } M_{CM}(j) = k,\ M_{IO}(j) = i, \\ d^{INIT}_j\left(\frac{1}{b^{il}_{net}} + \frac{1}{b^i_{disk}} + \frac{1}{b^l_{disk}}\right) & \text{if } M_{CM}(j) = i,\ M_{IO}(j) = l, \\ e + d^{INIT}_j\left(\frac{1}{b^{il}_{net}} + \frac{1}{b^i_{disk}} + \frac{1}{b^l_{disk}}\right) & \text{if } M_{CM}(j) = k,\ M_{IO}(j) = l. \end{cases} \tag{10}$$

where $k \neq i, l \neq i$, and $k \neq l$. $e$ is the fixed cost of migrating the job, $b^{kl}_{net}$ is the available bandwidth of the link between node $k$ and $l$, $b^k_{disk}$ is the available disk bandwidth in node $k$. In practice, $b^{kl}_{net}$ and $b^k_{disk}$ can be measured by a performance monitor [3]. $d^{INIT}_j$ represents the amount of data initially stored on disk to be processed by job $j$, and this amount of data is referred to as initial data throughout this paper. Thus, the second line of Eq. (10) represents the migration time spent on transmitting data over the network and on accessing source and destination disks. In real world applications, there is no need to migrate all the initial data, since some data will be only read once. However, IOCM might not be able to know which portion of initial data will be read only once. We assume that the initial data of a job is transmitted if the job encounters a migration. This assumption is conservative, since IOCM can be further improved if the amount of initial data that must be migrated can be accurately predicted by a monitor at run time.

The memory pressure placed by migrating data is not considered in our model. The reason is that in many real systems, data movement can be handled by storage controllers and network interface controllers without being processed by the CPU and buffered in main memory. This technique, referred to as Off-Processor I/O, is comprehensively studied in [9].

In Step (4), IOCM guarantees that the response time of the candidate migrant is less in expectation than it would be without migration. This guarantee is implemented by checking the following criterion based on Eqs. (7), (9), and (10).

$$r(i,j) = \begin{cases} r(k,i,j) + c_j & \text{if } M_{CM}(j) = k,\ M_{IO}(j) = i, \\ r(i,l,j) + c_j & \text{if } M_{CM}(j) = i,\ M_{IO}(j) = l, \\ r(k,l,j) + c_j & \text{if } M_{CM}(j) = k,\ M_{IO}(j) = l, \\ r(k,j) + c_j & \text{if } M_{CM}(j) = k,\ M_{IO}(j) = k. \end{cases} \tag{11}$$

where $k \neq i, l \neq i$, and $k \neq l$. Four migration cases in Expression 11 are: (1) the job is migrated without its I/O portion; (2) the job is executed locally, while its I/O portion is serviced

in another node; (3) both the job and its I/O portion are migrated, but to two different nodes; (4) both the job and its I/O portion are migrated to the same node, while I/O operations can still be processed locally in another node.

### 3.2 Weighted average load-balancing with preemptive migration

We are now in a position to study WAL-PM that improves performance by considering not only incoming jobs but also currently running jobs. For a newly arrived job $j$ at a node $i$, WAL-PM balances the load in the following six steps. First, the load of node $i$ is updated by adding $j$'s load, assigning the newborn job to the local node. Second, a migration is to be initiated, if node $i$ has the highest load and the ratio between node $i$'s load and the average load across the system is greater than 1.25. Third, a candidate node $k$ with the lowest load is chosen. If a candidate node is not available, WAL-PM will be terminated and no migration will be carried out. Fourth, WAL-PM determines a set *EM* of jobs eligible for migration such that the migration of each job in *EM* is able to potentially reduce the slowdown of the job. Fifth, a job $q$ from *EM* is judiciously selected in such a way that the migration benefit is maximized. In fact, this step substantially improves the performance over the WAL-based load-balancing scheme with non-preemptive migration. Finally, job $q$ is migrated to the remote node $k$, and the load of nodes $i$ and $k$ is updated in accordance with job $q$'s load. An outline of the WAL-PM scheme is presented in Fig. 3 below.

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

$$load_{WAL}(i) = W_{CPU}\frac{load_{CPU}(i)}{MAX_{j=1}^{n} load_{CPU}(j)} + W_{mem}\frac{load_{mem}(i)}{MAX_{j=1}^{n} load_{mem}(j)} + W_{IO}\frac{load_{IO}(i)}{MAX_{j=1}^{n} load_{IO}(j)} \tag{12}$$

where $load_{CPU}(i)$ is the number of running jobs, $load_{mem}(i)$ is expressed in Eq. (1), and the derivation of $load_{IO}(i)$ can be found in Eq. (2). Each weight indicating the significance of one resource is automatically configured by the load monitor. In practice, three weights are measured by the percentage of time spent on CPU, paging, and I/O accessing, respectively.

The WAL-PM scheme judiciously selects an eligible job in *EM* from the overloaded node to migrate, and the expected response time of an eligible migrant on the source node, by design, is greater than the sum of its expected response time on the destination node and the migration cost. In what follows, the expected response time of a candidate migrant $j$ on node $i$ is given in the following equation, where $a_j$ is the age of job $j$ and other variables assume the same meanings as in Eqs. (7) and (9).

$$r_{PM}(i, j) = (t_j - a_j)E(L_i) + (t_j - a_j)\lambda_j \times \left\{ E\left(s_{disk}^i\right) + \frac{\Lambda_{disk}^i E\left[\left(s_{disk}^i\right)^2\right]}{2\left(1-\rho_{disk}^i\right)} \right\}.$$

Based on Eq. (13), the set of eligible migrant jobs becomes:

$$EM(i, k) = \{j \in M_i | r_{PM}(i, j) > r_{PM}(k, j) + c_j\}, \tag{13}$$

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

$$c_j = \begin{cases} e + d_j^{INIT}\left(\frac{1}{b_{net}^{ik}} + \frac{1}{b_{disk}^{i}} + \frac{1}{b_{disk}^{k}}\right) & \text{remote execution} \\ f + \frac{m_j}{b_{net}} + \left(d_j^{INIT} + d_j^{W}\right)\left(\frac{1}{b_{net}^{ik}} + \frac{1}{b_{disk}^{i}} + \frac{1}{b_{disk}^{k}}\right) & \text{preemptive migration} \end{cases} \tag{14}$$

**Fig. 3** Pseudo code of the weighted-average-load based policy with preemptive migration

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

where $f$ is the fixed cost for preemptive migration and $m_j$ is the memory size of the migrant job. Like Eq. (10), the last three terms of both the upper and the bottom line of Eq. (14) represent the migration time spent on transmitting data over the network and on accessing source and destination disks, respectively. The second term of the bottom line of Eq. (14) is the memory transfer cost. $d_j^W$ and $m_j$ in Eq. (14) denote the amount of disk (I/O) data and of main memory data generated at the runtime by the job, respectively. Disk data $d_j^W$ is proportional to the number of write operations that has been issued by the job at the runtime and the average amount of data $d_j^{RW}$ stored by the write operations. $d_j^W$ is inversely proportional to the data re-access rate $r_j$. Thus, $d_j^W$ is defined by:

$$d_j^W = a_j \lambda_j w_j d_j^{RW}/(r_j + 1), \tag{15}$$

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

In Step (2.4), WAL-PM chooses one job $q$ from set $EM(i, k)$ in such a way that the benefit of migration is maximized. To find a maximizing factor, we define an objective function, called migration cost-effectiveness (MCE), which measures the amount of I/O load migrated per unit migration cost. More specifically, for job $j$, $MCE(j) = a_j \lambda_j / c_j$, since the numerator represents the I/O load of job $j$, while the denominator indicates migration cost of the job. Thus, the best job in $EM$ to choose for migration is the one with the maximum MCE value, as shown in Eq. (16),

$$MCE(q) = MAX_{j \in EM(i,k)}\{MCE(j)\},$$
$$\text{where } q \in EM(i, k). \tag{16}$$

## 4 Experimental results

In this section, we compare the performance of IOCM, IO-PM, and WAL-PM with four existing schemes, namely, CM-RE, CM-PM, IO-RE and WAL-RE. In what follows, we give a brief description of these policies. (1) CPU-Memory-based load balancing schemes CM-RE (with non-preemptive migration) and CM-PM (with preemptive migration) were introduced in [34]. When a node has sufficient memory space, the CM schemes balance the system using CPU load index. When the system encounters a large number of page faults due to insufficient memory space for the running jobs, memory load index $load_{mem}(i)$ is used by CM to balance the system. (2) The IO-based load balancing with non-preemptive migration (IO-RE) [16] uses a load index that represents only I/O load. For a job arriving in node $i$, the IO-RE scheme greedily assigns the computational and I/O tasks of the job to the node that has the least accumulated I/O load. (3) The Weighted-Average-Load-based balancing with non-preemptive migration (WAL-RE), proposed in [26], assigns jobs to a node that is not overloaded. If such a node is not available, WAL-RE dispatches the job to a node with the smallest value of the load index. The performance metric used in our simulations is slowdown. Since the definition of slowdown in [10, 33, 34] does not consider time spent on I/O accesses, it has to be extended by incorporating I/O access time. The definition of slowdown for a job $j$ is given as:

$$slowdown(j) = time_{wall}(j)/(time_{CPU}(j) + time_{IO}(j)), \tag{17}$$

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

### 4.1 Simulation parameters

To study dynamic load balancing, Harchol-Balter and Downey [10] implemented a trace-driven simulator for a distributed system with six nodes in which round-robin scheduling is employed. The load balancing policy studied in that simulator is CPU-based. Zhang et al. [34] extended the simulator, incorporating memory resources into the simulation system. Based on the simulator presented in [34], our simulator incorporates three new features: (1) The IOCM, IO-RE, IO-PM, WAL-RE and WAL-PM schemes are implemented; (2) a simple disk model is added; and (3) an I/O buffer is implemented. In all experiments, the simulated system is configured with parameters listed in Table 2.

Disk I/O accesses from each job are modeled as a Poisson Process with a mean I/O access rate $\lambda$. Although the durations and memory requirements of the jobs come from trace files, the I/O access rate of each job is randomly generated according to a uniform distribution between 0 and $AR$, where $AR$ represents the maximal I/O access rate. This simplification deflates any correlations between I/O requirement and other job characteristics, but we are able to control the maximal I/O access rate as a parameter and examine its impact on system performance. Data sizes of the I/O requests are randomly generated based on a Gamma distribution with the mean size of 256KByte and the standard deviation of 128Kbyte. The

**Table 2** System parameters

| Parameters | Value (fixed) (varied) |
|---|---|
| CPU speed | (800 MIPS-Millions Instruction Per Second)- |
| RAM size | (640 MB)- |
| Buffer size | (160 MB)- |
| Page fault service time | (8.1 ms)- |
| Mean page fault rate | (0.1 No./ms)- |
| Disk seek and rotation time | (8.0 ms)- |
| Disk transfer rate | (40 MB/s)- |
| Data re-access rate, $r$ | (5)-(1, 2, 3, 4) |
| Average data size | (256 KB)-(100, 150, 200, 250, 300, 350, 400 KB) |
| Average initial data size | (60 MB)-(50, 100, 150, 200, 250, 300, 350 MB) |
| Network bandwidth | (100 Mbps)-(50, 100, 250, 750, 1000 Mbps) |

sizes chosen in this way reflect typical data characteristics for many data-intensive applications, such as a fine-grained regional atmospheric climate model [25] and an ocean global circulation model, where the vast majority of I/O requests are small [14, 18].

### 4.2 Performance on I/O-intensive workload

To stress the I/O-intensive workload in this experiment, the page fault rate is fixed at a low value of 0.5 No./ms. This workload reflects a scenario where memory-intensive jobs exhibit high temporal and spatial locality of access. A realistic system is likely to have a mixed workload, where some jobs are I/O-intensive and other jobs are either CPU or memory intensive. Therefore, we randomly choose 10% of jobs from the 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.

Figure 4(a) plots slowdown as a function of the maximal I/O access rate in the range between 2.4 No./ms and 2.9 No./ms in increments of 0.1 No./ms. The mean slowdowns of IO-RE, IO-PM, and CM-RE are almost identical to those of WAL-RE, WAL-PM, and CM-PM respectively, and therefore are omitted from Fig. 4(a). Four observations were made from this experiment.

First, Fig. 4(a) reveals that the mean slowdowns of the five policies all increase with the I/O load. This is because as CPU load and memory demands are fixed, high I/O load leads to a high utilization of disks, causing longer waiting time on I/O processing.

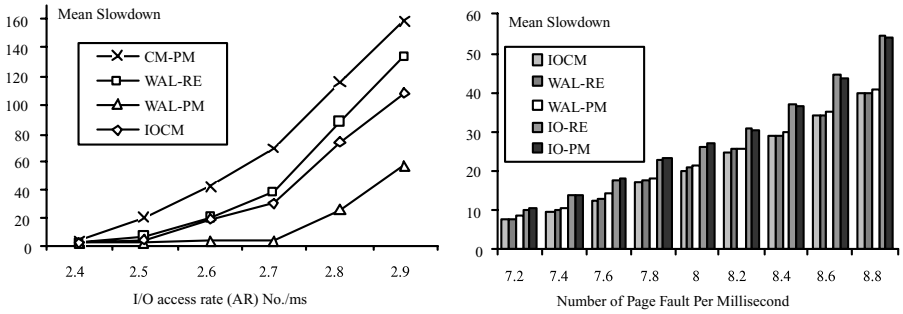
Second, the results show that the WAL-RE scheme significantly outperforms the CM-RE and CM-PM policies, suggesting that the CM-RE and CM-PM policies are not suitable for I/O intensive workload. This is because CM-RE and CM-PM only balance CPU and memory load, ignoring the imbalanced I/O load under the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

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

Finally, the proposed WAL-PM policy improves the performance even over WAL-RE by virtue of preemptive migration strategy, suggesting that preemptive migration outperforms remote execution for I/O-based schemes under I/O-intensive workload. Consequently, the slowdowns of CM-RE, CM-PM and WAL-RE are more sensitive to I/O access rate than WAL-PM. This performance improvement of WAL-PM over WAL-RE can be explained by the following reasons. First, one problem encountered in the WAL-RE policy is that the I/O demand of a newly arrived job may not be high enough to offset the migration overhead. However, WAL-PM provides better migratory opportunities by considering all existing jobs on a node, in addition to the newly arrived job. Second, unlike the preemptive scheme, in the non-preemptive scheme, once a job with high I/O demand misses the opportunity to migrate it will never have a second chance.

### 4.3 Performance on CPU-memory intensive workload

This section shows the worst-case scenario for IOCM and WAL-PM, namely, subjecting them to a highly CPU-memory-intensive workload. To simulate a memory intensive workload, the maximal I/O access rate is fixed at a low value of 0.1 No./ms, keeping the I/O demands of all jobs at a low level. Again, to simulate a mixed workload, we randomly choose 10 percent jobs to be I/O intensive by setting their I/O access rate to 2.8 No./ms. In [34], the page fault-rate is scaled according to the upper limit vertical axis presenting the mean slowdown. Similarly, we set the upper limit of the mean slowdown at 60, and scale the page fault rate from 7.2 to 8.8 No./ms in increments of 0.2 No./ms. In practice, the page fault rates of applications range from 1 to 10 No./ms [34].




a. I/O intensive workload b. CPU-memory intensive workload

**Fig. 4** (a) Impact of I/O access rate, page fault rate is 0.5 No./ms; (b) Impact of page fault rate, I/O access rate is 0.1 No./ms

The results of the mean slowdown as a function of the page fault rate are summarized in Fig. 4(b). The general observations in this experiment are in agreement with [34], where the impact of page fault rate on the mean slowdown is quantitatively evaluated. Therefore, we only present new results about the proposed schemes and their comparison with the existing schemes.

The result shows that the mean slowdowns of WAL-RE and WAL-PM are nearly identical to those of the CM-RE and CM-PM policies, because the WAL-RE and WAL-PM policies can gracefully reduce to CM-RE and CM-PM by dynamically configuring the weighted load index in accordance with the CPU-memory intensive workload. The mean slowdowns of CM-RE and CM-PM are omitted from 4(b).

As can be seen in Fig. 4(b), when page fault rate is high and I/O rate is low, IOCM, CM-RM, CM-PM, WAL-RE, and WAL-PM outperform the IO-RM and IO-PM schemes considerably. These results can be explained by the following reasons. First, IOCM, CM-RE, CM-PM, WAL-RE, and WAL-PM consider the effective usage of global memory, attempting to balance the implicit I/O load, which makes the most significant contribution to the overall system load when page fault rate is high and the explicit I/O load is low. Second, the IO-RE and IO-PM schemes improve the utilization of disks based only on explicit I/O load, ignoring the implicit I/O load resulted from page faults. Again, Fig. 4(b) illustrates that IOCM outperforms CM-RE, CM-PM, and WAL-RE, by up to 18.3%. The reason for this phenomenon is that besides balancing the memory load and the implicit I/O load generated by the page faults, IOCM further balances the explicit I/O load measured by the I/O access rate.

As expected, CM-PM and WAL-PM consistently perform slightly worse than CM-RE and WAL-RE because remote execution results in lower data movement cost during migration than that of preemptive strategy for memory-intensive jobs. This experiment is consistent with the results reported in [34]. However, the opposite is true for IO-based policies when memory demand is comparatively high. The explanation is that a high memory demand implies an I/O-intensive workload due to a large number of page faults, and a preemptive strategy is effective for an I/O-intensive workload.

### 4.4 Initial data size

The migration cost of non-preemptive and preemptive policies depend in part on the size of initial data. Figure 5(a) shows the impact of initial data size on slowdowns under I/O-intensive workload. We only present the results of WAL-RE, WAL-PM, and IOCM, since the performance patterns of other policies are similar to these three policies.

First, Fig. 5(a) shows that the IOCM and WAL-PM policies consistently outperform the WAL-RE policy. Second, the slowdowns increase with the increasing size of the initial data size. The reason is that the large initial data size results in a high migration time, which in turn reduces the benefits gained from migration. Third, it is observed that the slowdowns of WAL-RE and IOCM are much more sensitive to the initial data size than that of WAL-PM. This result indicates that the performance gain by WAL-PM over existing policies becomes more pronounced when the initial data size is large.

Figure 5(b) illustrates the impact of the initial data size on the WAL-PM slowdown performance under memory-intensive workloads. An observation is that the sensitivity of WAL-PM to the initial data size is heightened by increasing the page fault rate. This is because migrating the initial data of a job involves two disk accesses, reading initial data from the source disk and writing it to the target disk. The average response times at the source and the target disks are likely to be long when the I/O load of two disks is heavy due to the high page fault rate, thereby leading to a large overhead of migrating initial data. This result suggests that the slowdown of the WAL-PM policy does not suffer significantly from a large initial data size when the page fault rate is low. Likewise, WAL-PM can tolerate a relatively high page fault rate if the initial data size is small.

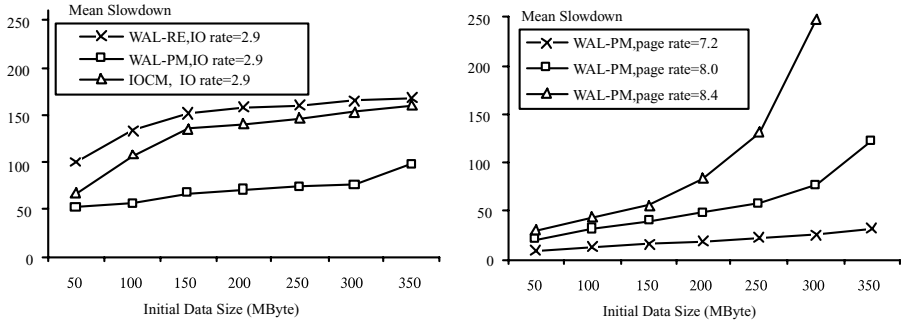



**Fig. 5** Mean slowdown as a function of the size of initial data

a. Page fault rate is 0.5 No./ms b. I/O access rate is 0.1 No./ms

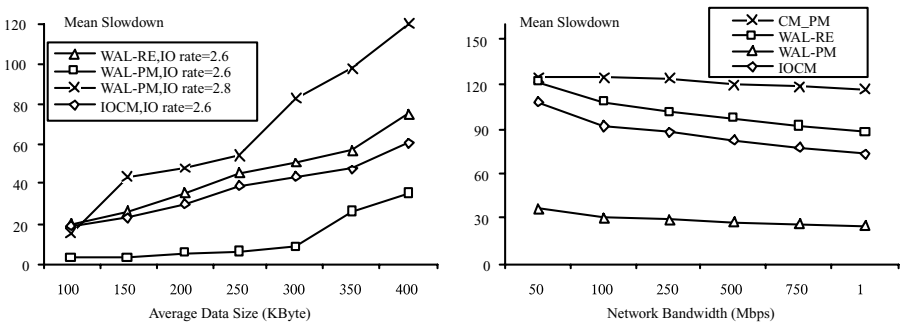



**Fig. 6** (a) Page fault rate is 0.5 No./ms, and I/O rate is 2.6 No./ms; (b) Page fault rate is 0.5 No./ms, and I/O rate is 2.8 No./ms

a. Impact of the average data size b. Impact of the network bandwidth

### 4.5 Average data size

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

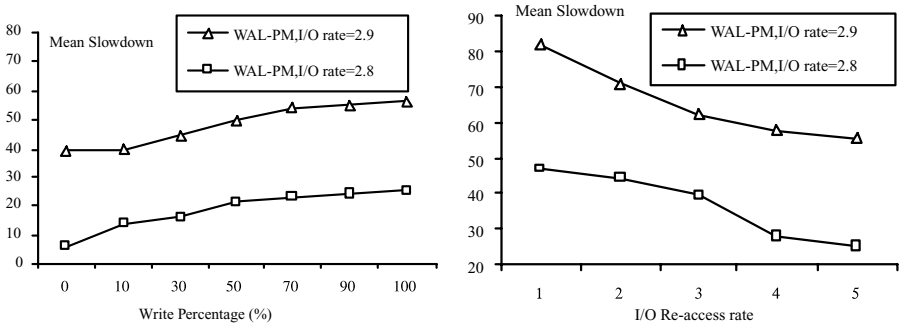
Figure 6(a) shows that the mean slowdown increases as the average data size increases. The reason is that as I/O access rate is fixed, a large average data size yields a high utilization of disks, causing longer waiting times on I/O processing. A second observation from Fig. 6(a) is that the slowdown performance of WAL-PM at a high I/O access rate is more sensitive to average data size than that at a low I/O access rate. This is because the higher the I/O access rate, the higher the disk utilization, which results in longer waiting time in disk queue.

### 4.6 Network bandwidth

Let us now consider the impact of network bandwidth on the mean slowdowns of the CM-PM, WAL-RE, IOCM, and WAL-PM policies. In order to explore this issue, we set the network bandwidth in the range between 50Mbps and 1Gbps. Since the performance of CM-RE, IO-RE, and IO-PM are almost identical to those of CM-PM, WAL-RE, and WAL-PM, respectively, Fig. 6(b) only shows the results for the CM-PM, WAL-RE, IOCM, and WAL-PM policies. As shown in Fig. 6(b), CM-PM is not sensitive to network bandwidth. Because when the workload is I/O intensive, CM-PM degrades to a non-load-balancing policy that is not affected by the network speed. The slowdowns of the other three policies share a common feature in the sense that when the network bandwidth increases, the slowdowns slightly drop. The reason is that a network with high bandwidth results in a low migration cost in these three load-balancing policies. Figure 6(b) further reveals that WAL-RE and IOCM are more sensitive to the network bandwidth than WAL-PM. This result can be explained by the fact that the preemptive scheme tends to select a job with the minimal migration cost and reduce the network traffic overhead, thereby deflecting the impact of network speed on the performance of the preemptive migration scheme.

### 4.7 Write percentage and re-access rate

Figures 7(a) and (b) illustrate the mean slowdowns of WAL-PM as functions of write percentage and re-access rate, respectively. We observe from Fig. 7(a) that the increase in the write percentage worsens the performance. The reason is that a high write percentage means a large amount of data to be migrated (see Eq. (15)), implying a high migration cost




a. Impact of the write percentage b. Impact of the re-access rate

**Fig. 7** Page fault rate is 0.5 No./ms. (a) Mean slowdown of IOCM and WAL-PM as a function of the write percentage, (b) Mean slowdown of IOCM and WAL-PM as a function of the I/O re-access rate

as given in Eq. (14). Consequently, the high migration cost yields a high slowdown.

Figure 7(b) indicates that the mean slowdowns decrease as the value of re-access rate increases. This is because if the I/O access rate of a job is fixed, increasing re-access rate implies a smaller amount of data stored in disk for the job, and thus a smaller amount of migrated data. As mentioned earlier, the migration cost is proportional to the amount of migrated data, and reducing the amount of migrated data results in a reduced cost for migration.

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

### 4.8 Real I/O-intensive applications

The experimental results reported in the previous sections are obtained from jobs with synthetic I/O requests. To validate the results based on the synthetic I/O workload, we simulate a number of real I/O-intensive applications using five sets of I/O traces collected from the University of Maryland [28]. These sets of traces reflect both scientific and non-scientific applications with diverse disk I/O demands. We evaluate the mean slowdowns of the following five applications:

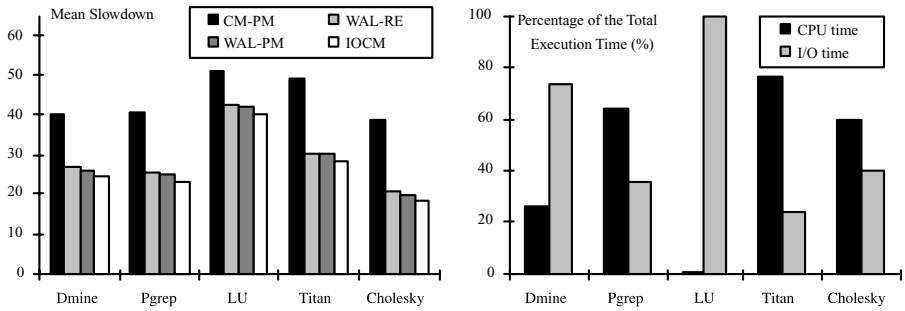
1. Data mining (Dmine): This application extracts association rules from retail data [31].
2. Parallel text search (Pgrep): This application is used for partial match and approximate searches. It is a modified parallel version of the agrep program from the University of Arizona [30].
3. LU decomposition (LU): This application computes the dense LU decomposition of an out-of-core matrix [11].
4. Titan: This is a parallel scientific database for remote-sensing data [5].
5. Sparse Cholesky (Cholesky): This application is capable of computing Cholesky decomposition for sparse, symmetric positive-definite matrices [2]. The input data of this application is a matrix that contains over 45 million double-precision nonzeros and 45,361 columns for a total of 437 MB [28].

To simulate these I/O-intensive parallel applications, we generate five job traces where the arrival patterns of jobs are extrapolated based on the job traces collected from the University of California at Berkeley [10]. The main purpose of conducting this experiment is to measure the impact of the I/O-aware load balancing schemes on a variety of real applications and, therefore, each job trace consists of one type of I/O-intensive application described above.

Figure 8(a) shows the mean slowdowns of the five job traces scheduled by four load-sharing policies. We make three observations. First, the I/O-aware load balancing schemes benefit all I/O intensive applications, and offer a 26.5–112.0% 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 uniformly low I/O demand for writes.

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

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




a. Mean slowdowns of five applications b. The execution time of CPU and I/O

**Fig. 8** (a) The mean slowdowns of the five job traces scheduled by four load-sharing policies, (b) The execution time of CPU and I/O as the components of the total execution time

Figure 8(b) illustrates the contribution of CPU and I/O execution time to the total execution time of each application in a dedicated computing environment. In particular, Fig. 8(b) shows that the total execution time of LU is dominated by I/O processing, thereby giving rise to a low utilization of CPU resources, which in turn leads to a high value of mean slowdown for the trace with LU applications (see Fig. 8(b)). Unlike the workload with the LU applications, the workloads with the Dmine, Pgrep, and Cholesky applications sustain reasonably high utilizations of CPU and disk I/O. This is because for these three applications, neither CPU time nor I/O time dominates the total execution time. Hence, the trace of the LU applications has the highest slowdown value among all application traces for the four load-balancing policies.

## 5 Conclusion

In this paper, we have proposed two dynamic load-balancing policies, referred to as IOCM (load balancing for I/O, CPU, and Memory) and WAL-PM (Weighted-Average-Load based policy with Preemptive Migration), for workstation-based cluster systems. IOCM employs remote I/O execution facilities to improve system performance, whereas WAL-PM utilizes a preemptive job migration strategy to boost performance. More specifically, IOCM allows applications and their data to be located on different nodes, if well-balanced I/O load is able to offset communication overheads imposed by remote I/O executions. WAL-PM considers not only newly arrived jobs but also older, currently running jobs as candidate migrant jobs, and migrates jobs that are the most migration cost-effective. In addition to CPU and memory utilization, both IOCM and WAL-PM consider I/O load, leading to a performance improvement over existing I/O-based and CPU-Memory-based policies under I/O-intensive workload. We compare IOCM and WAL-PM with four existing approaches, namely, (1) CPU-Memory-based policy with preemptive migration (CM-PM), (2) CPU-Memory-based policy with non-preemptive migration (CM-RE), (3) IO-based policy with non-preemptive migration (IO-RE), and (4) Weighted-Average-load based policy with non-preemptive migration (WAL-RE). For comparison purposes, IO-based policy with preemptive migration (IO-PM) is also simulated and compared with the proposed schemes. A trace-driven simulation demonstrates that applying IOCM and WAL-PM to clusters for I/O-intensive workload is highly effective.

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

A future direction of this research is to study a feedback control mechanism to dynamically configure resource weights in such a way that the weights are capable of reflecting the significance of system resources. Since data movement has a significant impact on the overall performance of load balancing polices, another extension of this work will be the study of a predictive model for moving data without compromising the performance of applications running on local nodes in a cluster.

**Acknowledgments** This work was partially supported by an NSF grant (EPS-0091900), a Nebraska University Foundation grant (26-0511-0019), and a UNL Academic Program Priorities Grant. Work was completed using the Research Computing Facility at the University of Nebraska-Lincoln. We are grateful to the anonymous referees for their insightful suggestions and comments.

## References

1. A. Acharva and S. Setia, Availability and utility of idle memory in workstation clusters, in: *Proceedings of the ACM SIGMETRICS Conf. on Measuring and Modeling of Computer Systems* (1999).
2. A. Acharya et al, Tuning the performance of I/O-intensive parallel applications, in: *Proceedings of the 4th IOPADS*, Philadelphia, PA (1996) pp. 15–27.

3. J. Basney and M. Livny, Managing network resources in condor, in: *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9)* (2000) pp. 298–299.
4. A. D. Brown, T. C. Mowry, and O. Krieger, Compiler-based I/O prefetching for out-of-core applications. *ACM Transactions on Computer Systems* 19(2) (2001) 111–170.
5. C. Chang, B. Moon, A. Acharya, C. Shock, A. Sussman, and J. Saltz, Titan: A high-performance remote-sensing database, in: *Proc. of International Conference on Data Engineering* (1997).
6. M. M. Cettei, W. B. L. III, and R. B. Ross, Support for parallel out of core applications on beowulf workstations, in: *Proceedings of the 1998 IEEE Aerospace Conference* (1998).
7. J. Cruz and K. Park, Towards communication-sensitive load balancing, in: *Proc. 21 Int'l Conf. Distributed Computing Systems (ICDCS 2001)* (2001).
8. B. Forney, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, Storage-aware caching: Revisiting caching for heterogeneous storage systems, in: *Proceedings of the 1st Symposium on File and Storage Technology* Monterey, California, USA (2002).
9. P. Geoffray, OPIOM: Off-processor I/O with Myrinet. *Future Generation Computer Systems* 18 (2002) 491–499.
10. M. Harchol-Balter and A. Downey, Exploiting process lifetime distributions for load balancing. *ACM Transactions on Computer Systems* 15(3) (1997) 253–285.
11. B. Hendrickson and D. Womble, The torus-wrap mapping for dense matrix calculations on massively parallel computers. *SIAM J. Sci. Comput.* 15(5) (1994).
12. C. Hui and S. Chanson, Improved strategies for dynamic load sharing. *IEEE Concurrency* 7(3) (1999).
13. C. Isert, and K. Schwan, ACDS: Adapting computational data streams for high performance, in: *International Parallel and Distributed Processing Symposium (IPDPS)* (2000).
14. D. Kotz and N. Nieuwejaar, Dynamic file-access characteristics of a production parallel scientific workload, in: *Proceedings of the ACM Conference on Supercomputing* (1994) pp. 640–649.
15. R. Lavi and A. Barak, The home model and competitive algorithm for load balancing in a computing cluster, in: *Proceedings of the 21st Int'l Conf. Distributed Computing Systems (ICDCS 2001)*.
16. L. Lee, P. Scheauermann, and R. Vingralek, File assignment in parallel I/O systems with minimal variance of service time. *IEEE Trans. on Computers* 49(2) (2000) 127–140.
17. X. Ma, M. Winslett, J. Lee, and S. Yu, Faster collective output through active buffering, in: *Proceedings of the International Symposium on Parallel and Distributed Processing* (2002).
18. B. Pasquale and G. Polyzos, Dynamic I/O characterization of I/O intensive scientific applications, in: *Proceedings of the Supercomputing* (1994) pp. 660–669.
19. X. Qin, H. Jiang, Y. Zhu, and D. Swanson, Dynamic load balancing for I/O- and memory-intensive workload in clusters using a feedback control mechanism, in: *Proceedings of the 9th International Euro-Par Conference on Parallel Processing (Euro-Par 2003)*, Klagenfurt, Austria (2003a).
20. X. Qin, H. Jiang, Y. Zhu, and D. Swanson, Dynamic load balancing for I/O-intensive tasks on heterogeneous clusters, in: *Proceedings of the 10th International Conference on High Performance Computing (HiPC 2003)*, India (2003b).
21. X. Qin, H. Jiang, Y. Zhu, and D. Swanson, A dynamic load balancing scheme for I/O-intensive applications in distributed systems, in: *Proceedings of the 32nd International Conference on Parallel Processing Workshops* (2003c).
22. X. Qin, H. Jiang, Y. Zhu, and D. Swanson, Towards load balancing support for I/O-intensive parallel jobs in a cluster of workstations, in: *Proceedings of the 5th IEEE International Conference on Cluster Computing (Cluster 2003)*, Hong Kong (2003d).
23. K. Ranganathan and I. Foster, Decoupling computation and data scheduling in distributed data-intensive, in: *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing* Edinburgh, Scotland, UK (2002).
24. A. Riska and E. Smirni, Exact aggregate solutions for M/G/1-type Markov processes, in: *Proceedings of ACM Sigmetircs 2002* Edinburgh, Scotland, UK (2002) pp. 86–96.
25. J. Roads et al., A preliminary description of the Western U.S. climatology, in: *Proceedings of the Ninth Annual Pacific Climate (PACLim) Workshop* (1992).
26. M. Surdeanu, D. Modovan, and S. Harabagiu, Performance analysis of a distributed question/answering system. *IEEE Trans. on Parallel and Distributed Systems* 13(6) (2002) 579–596.
27. T. Tanaka, Configurations of the solar wind flow and magnetic field around the planets with no magnetic field: Calculation by a new MHD. *Journal of Geophysical Research* (1993) pp. 17251–17262.
28. M. Uysal, A. Acharya, and J. Saltz, Requirements of I/O systems for parallel machines: An Application-driven study, in: *Technical Report, CS-TR-3802*, University of Maryland, College Park (1997).
29. G. Voelker, Managing server load in global memory systems, in: *Proceedings of the ACM SIGMETRICS Conf. on Measuring and Modeling of Computer Systems* (1997).
30. S. Wu and U. Manber, Agrep—A fast approximate pattern-matching tool, in: *the USENIX Conference Proceedings* San Francisco, CA (1992) pp. 153–162.
31. X. Wu, V. Taylor, and R. Stevens, Design and implementation of prophesy automatic instrumentation and data entry system, in: *Proc. of the 13th IASTED Int. Conf. on Parallel and Distributed Computing and Systems* CA (2001).
32. L. Xiao, S. Chen, and X. Zhang, Dynamic cluster resource allocations for jobs with known and unknown memory demands. *IEEE Trans. on Parallel and Distributed Systems* 13(3) (2002) 223–240.
33. L. Xiao, X. Zhang, and Y. Qu, Effective load sharing on heterogeneous networks of workstations, in: *Proc. of International Symposium on Parallel and Distributed Processing* (2000).
34. X. Zhang, Y. Qu, and L. Xiao, Improving distributed workload performance by sharing both cpu and memory resources, in: *Proceedings of the 20th Int'l Conf. on Distributed Computing Systems* (2000).
35. Y. Zhu, H. Jiang, X. Qin, D. Feng, and D. Swanson, Improved read performance in a cost-effective, fault-tolerant parallel virtual file system (CEFT-PVFS), in: *Proc. of the 3rd IEEE/ACM Intl. Symp. on Cluster Computing and the Grid* (2003a) pp. 730–735.
36. Y. Zhu, H. Jiang, X. Qin, and D. Swanson, A case study of parallel I/O for biological sequence analysis on linux clusters, in: *Proceedings of the 5th IEEE International Conference on Cluster Computing*, Hong Kong (2003b).

**Xiao Qin** received the BSc and MSc degrees in computer science from Huazhong University of Science and Technology in 1992 and 1999, respectively. He received the PhD degree in computer science from the University of Nebraska-Lincoln in 2004. Currently, he is an assistant professor in the department of computer science at the New Mexico Institute of Mining and Technology. His research interests include parallel and distributed systems, storage systems, real-time computing, performance evaluation, and fault-tolerance. He served on program committees of international conferences like CLUSTER, ICPP, and IPCCC. During 2000–2001, he was on the editorial board of *The IEEE Distributed System Online*. He is a member of the IEEE.

**Hong Jiang** received the B.Sc. degree in Computer Engineering in 1982 from Huazhong University of Science and Technology, Wuhan, China; the M.A.Sc. degree in Computer Engineering in 1987 from the University of Toronto, Toronto, Canada; and the PhD degree in Computer Science in 1991 from the Texas A&M University, College Station, Texas, USA. Since August 1991 he has been at the University of Nebraska-Lincoln, Lincoln, Nebraska, USA, where he is Associate Professor and Vice Chair in the Department of Computer Science and Engineering. His present research interests are computer architecture, parallel/distributed computing, computer storage systems and parallel I/O, performance evaluation, middleware, networking, and computational engineering. He has over 70 publications in major journals and international Conferences in these areas and his research has been supported by NSF, DOD and the State of Nebraska. Dr. Jiang is a Member of ACM, the IEEE Computer Society, and the ACM SIGARCH and ACM SIGCOMM.

**Yifeng Zhu** received the B.E. degree in Electrical Engineering from Huazhong University of Science and Technology in 1998 and the M.S. degree in computer science from University of Nebraska Lincoln (UNL) in 2002. Currently he is working towards his Ph.D. degree in the department of computer science and engineering at UNL. His main fields of research interests are parallel I/O, networked storage, parallel scheduling, and cluster computing. He is a student member of IEEE.

**David Swanson** received a Ph.D. in physical (computational) chemistry at the University of Nebraska-Lincoln (UNL) in 1995, after which he worked as an NSF-NATO postdoctoral fellow at the Technical University of Wroclaw, Poland, in 1996, and subsequently as a National Research Council Research Associate at the Naval Research Laboratory in Washington, DC, from 1997–1998. In early 1999 he returned to UNL where he has coordinated the Research Computing Facility and currently serves as an Assistant Research Professor in the Department of Computer Science and Engineering. The Office of Naval Research, the National Science Foundation, and the State of Nebraska have supported his research in areas such as large-scale parallel simulation and distributed systems.