In the Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing, Nov.10-12, 2003.

Boosting Performance for I/O-Intensive Workload by Preemptive Job Migrations in a Cluster System

Xiao Qin Hong Jiang Yifeng Zhu David R. Swanson
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, NE 68588-0115, {xqin, jiang, yzhu, dswanson}@cse.unl.edu

Abstract

Load balancing in a 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 that focus on CPU and memory usage may cause the system performance to decrease substantially. To solve this problem, a new I/O-aware load-balancing scheme with preemptive job migration is presented to sustain the high performance of a cluster with a diverse set of workload conditions. The proposed scheme dynamically detects I/O load imbalance on nodes of a cluster, and determines whether to preempt some running jobs on overloaded nodes and migrate them to other less- or under-loaded nodes. Besides balancing I/O load, the scheme takes into account both CPU and memory load sharing in clusters, thereby maintaining the same level of performance as existing schemes when I/O load is low or well balanced. Results from a trace-driven simulation show that, compared to the existing approaches that only consider I/O with non-preemptive job migrations, the proposed schemes achieve the improvement in mean slowdown by up to a factor of 10.

1. Introduction

A cluster consists of a number of nodes, and each node has a combination of multiple types of resources, such as CPU, memory, network connectivity and disks. In a cluster system, dynamic load balancing schemes can improve system performance by attempting to assign work, at run time, to machines with idle or under-utilized resources.

Several distributed load-balancing schemes have been presented in the literature, primarily considering CPU [7][8], memory [1][15], or a combination of CPU and memory [16][17]. Although these load-balancing policies have been very effective in increasing the utilization of resources in distributed systems, they have ignored disk I/O, which is a likely performance bottleneck when a large number of applications running on clusters are data-intensive and/or I/O-intensive. Therefore, we believe that for any dynamic load balancing scheme to be effective in this new application environment, it must be made "I/O-aware". Typical examples of I/O-intensive applications include long running simulations of time-dependent phenomena that periodically generate snapshots of their state [14], archiving of raw and processed remote sensing data [4][6], multimedia and web-based applications, to name just a few. These applications share a common feature in that their storage and computational requirements are extremely high. Therefore, the high performance of I/O-intensive applications heavily depends on the effective usage of storage, in addition to that of CPU and memory. Compounding the performance impact of I/O in general, and disk I/O in particular, is the steadily widening gap between CPU and I/O speed, making the load imbalance in I/O increasingly more crucial to overall system performance. To bridge this gap, I/O buffers allocated in the main memory have been successfully used to reduce disk I/O costs, thus improving the throughput of I/O systems. In this regard, load balancing with I/O-awareness, when appropriately designed, is potentially capable of boosting the utilization of the I/O buffer in each node, which in turn increases the buffer hit rate and decreases disk I/O access frequency.

This paper evaluates a comprehensive approach, called Weighted Average Load balancing with Preemptive Migration, or WAL-PM, to balance a cluster in such a way that CPU, memory, and I/O resources at each node can be well utilized. 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 system considerations and an example to motivate the idea behind the proposed approach. In Section 4, we describe the WAL-PM scheme.

Section 5 evaluates the performance of the WAL-PM scheme, and compares it with that of other existing solutions. Finally, Section 6 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. [7] proposed a CPU-based preemptive migration policy that was more effective than non-preemptive migration policies. Zhang et al. [17] focused on load sharing policies that consider both CPU and memory services among the nodes. Throughout this paper, the CPU-memory-based load-balancing policy presented in [17] will be referred to as ***CM-based*** policies. The experimental results show that CM-based policies not only improve performance of memory-intensive jobs, but also maintain the same load sharing quality of the CPU-based policies for CPU-intensive jobs [17].

A large body of work can be found in the literature that addresses the issue of balancing the load of disk systems [2][9][12][18][19]. Scheuermann et al. [12] studied the issues of striping and load balancing in parallel disk systems. Lee et al. [9] proposed two file assignment algorithms that minimize the variance of the service time at each disk. Aerts et al. [2] used randomization and data redundancy to enable effective load balancing. Our previous work [18][19] focused on two dynamic scheduling algorithms to improve the read and write performance of a parallel file system by balancing the global workload. 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 distributed environment where I/O-intensive jobs may share resources with many other memory-intensive and CPU-intensive jobs.

We have developed a feedback control mechanism to improve the performance of a cluster by manipulating the I/O buffer size [11]. The scheme presented in this paper is complementary to the existing buffering techniques, thereby providing additional performance improvement when combined with a feedback control mechanism.

Very recently, three load balancing models, which consider I/O, CPU and memory resources, have been presented [10][13][16]. In [13], a dynamic load-balancing scheme, tailored for the specific requirements of the Question/Answer application, is proposed along with a performance analysis of the approach. The migration scheme studied in [13] is non-preemptive, therefore being referred to as *WAL-RE* (**W**eighted **A**verage **L**oad with **R**emote **E**xecution). In the authors' previous work, another I/O-aware load-balancing scheme, referred to as IOCM, is studied [10]. IOCM 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. The result shows that IOCM significantly improves the slowdown performance over those load-balancing schemes that do not allow remote I/O access. The two policies proposed in [10][13] are similar in the sense that the policies are non-preemptive. The new WAL-PM scheme, however, permits a running job to be preempted and migrated if its migration is expected to improve the performance.

One of the load-balancing policies presented in [16] considers the three types of resources in a similar way as WAL-RE, and results show that the policy improves overall job execution performance. Besides WAL-RE, Zhang et al. [16] also proposed a WAL-based preemptive migration policy, which has worse performance than that of WAL-RE under memory-intensive workload. The reasons for this result are two-fold. First, remote execution has a significantly lower data movement cost than that of preemptive migration for jobs with high memory demand [16]. Second, the migration cost might not always be considered as a criterion to choose the eligible jobs for migration. Since the preemptive migrations in [16] are proposed for memory-intensive workload, data migration cost only take memory migration into account, thereby ignoring I/O migration cost as an important component of the migration cost in load-balancing. In contrast, the WAL-PM scheme proposed in this study considers both memory and I/O migration cost as a criterion to determine jobs that are eligible for migration. Trace-driven simulations show that, compared with the CM-based and WAL-RE policies, the proposed WAL-PM scheme significantly enhances the overall performance of a cluster system under I/O intensive workload. The results also show that, under CPU-memory-intensive workload, WAL-PM is more effective than IO-based polices, and sustains the same level of performance as the CM-based policy.

3. Problem Description

We consider the problem of distributed dynamic load balancing among a cluster of nodes connected by a high-speed network, where each node maintains reasonably up-to-date global load information by periodically exchanging load status with other nodes. Jobs arrive at each node dynamically and independently, and share resources available there. The nodes are assumed to be capable of migrating a newly arrived job or a running job preemptively to another node if needed.

In this study, we consider the sharing of, and scheduling for, three main resources, namely, CPU, main memory, and disk I/O. For simplicity, we assume that all nodes are homogeneous, and the proposed scheme may be extended

to handle heterogeneous systems by incorporating a conversion mechanism for relative load. The network in our model is fully connected and homogenous in the sense that communication delay between any pair of nodes is the same.

To help describe the problem of a dynamic load-balancing scheme in a cluster and motivate the proposed solution that improves on previous ones, we first present the following example:

Assume a cluster with two identical nodes, where two jobs have been assigned to node *1* and one job has been assigned to node *2*. The CPU, memory, and I/O resource requirements and execution status of each job are listed in Table 1. When an I/O-intensive job is migrated, load-balancing schemes are required to move the data that resides in the local disk along with the migrated job, making the data available locally to the migrated job. Mig_data in Table 1 denotes the amount of data stored in the local disk that has to be migrated together with its job once the migration is initiated. The discrepancy in Mig_data size among the jobs might be due either to the variety of I/O access patterns or to the various initial data sizes. The I/O access rate is measured by the number of disk I/O accesses per unit time, assuming that each access involves a fixed amount of data (say, one block). This I/O access rate for a given job can be viewed as a measure of the average number of instructions between two consecutive disk I/O accesses. For example, if the average I/O service time of each I/O access is 8.0ms, then the total I/O service time required by job *3* will be 3200ms, because there will be 400 I/O accesses in job 3, which is significantly larger than its CPU time. This workload suggests that balancing I/O load is a more effective means of improving the system performance than balancing CPU or memory. Therefore, for illustrative purposes, we will only consider how to dynamically balance the I/O load in this example.

Job	Node	CPU	Age	Memory	I/O rate	Mig_data
1	1	300ms	5ms	200KB	4No./ms	30 MB
2	1	800ms	2ms	300KB	5No./ms	5 MB
3	2	400ms	3ms	400KB	1No./ms	15 MB
4(New)	?	700ms	0ms	500KB	2No./ms	120 MB

Table 1. Resource requirements of the jobs and their assignment

Now consider job *4* arriving at node *1* with the resource demand shown in Table 1. The total I/O load in node *1* is 9 No./ms, whereas the I/O load in node *2* is only 1 No./ms, indicating that the system is I/O load imbalanced. The current imbalance might be caused by the fact that some jobs with high I/O demand have just been completed and left node *2*. Since job *4* will worsen the imbalance of I/O load between the two nodes if it is assigned directly to node *1*, efforts must be made to counterbalance the I/O load. There are two approaches to balance the system:

Approach 1: Job *4* is executed on remote node *2*. This approach is referred to as non-preemptive migration, or remote execution.

Approach 2: Job *4* is executed locally at node *1*, and job *2* at node *1* is preempted and migrated to node *2*. This approach is called preemptive migration.

The performance of the two approaches is compared with respect to the following two aspects, namely, load-balancing effect and migration cost.

(1) Approach *1* reduces the discrepancy in I/O load between the two nodes from 8 No./ms to 6 No./ms, by adding 2 No./ms of I/O load to node 2. This leaves the system still imbalanced in I/O load. Approach *2*, however, leads to a perfect I/O load balancing between the two nodes. This result illustrates that, in the presence of a variety of I/O load among jobs under I/O intensive workload, preemptive migration has a better ability to balance the I/O load than non-preemptive migration does.

(2) In both approaches data resident in the local disk must be migrated to the remote disk, by first reading out of the local disk, then transferring through the network, before storing in the remote disk, thus incurring a cost of two disk accesses and one network transaction. Assume that the network bandwidth is 1Gbps, and the disk bandwidth is 40Mbyte/Sec. Thus, network transaction and disk access times for migrating data for job *4* are 937.5ms and 6000ms, respectively. However, the data migration cost for job *2* is only 39ms for network transfer and 125ms for disk accesses. This example shows that Approach *2* can explore far more opportunities than Approach *1* for reducing I/O load imbalance at low migration costs by considering not only incoming jobs but also currently running jobs.

4. Weighted Average Load-balancing with Preemptive Migration

It has been observed that finding an optimal solution for the general problem of load-balancing in a distributed system, even for relatively simple formulations of this problem, is an NP-hard problem [5]. Consequently, the WAL-PM scheme, an I/O-aware load-balancing scheme with preemptive migration, presented here is heuristic in nature. The main goals of the WAL-PM scheme are to:

(1) balance I/O usage of all nodes in the system with best effort;

(2) balance CPU and memory resources with best effort; and

(3) optimize (1) and (2) by judiciously preempting running jobs to migrate with minimized migration costs.

For a newly arrived job *j* at a node *i*, the WAL-PM scheme attempts to balance the system load in the

following steps. First, the load of node *i* is updated by adding job *j*'s load, assigning the newborn job to the local node. Second, if the load of node *i* is the maximum among all nodes, meaning that the node is overloaded, a migration is to be initiated. Third, a candidate node *k*, that has the lowest load, is chosen. The load of the candidate must be less than the global average load. 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 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 Figure 1 below.

WAL-PM(**Input:** Job *j*, Node *i*)
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**
3. Select a node *k* with the smallest value of load;
   if a candidate node is not available **then**
   Preemptive migration is terminated
4. Determine a set of jobs *EM(i, k)*, in which jobs have been assigned to node *i* and are eligible for migration;
5. **if** the set *EM* is not empty **then**
   Select a job *q* in *EM(i, k)* that gains a maximal benefit from migration;
   Migrate job *q* from node *i* to node *k*;

Figure 1. Pseudo code of the Weighted-Average-Load based policy with Preemptive Migration.

The WAL-PM scheme answers three basic questions, namely, (1) when to migrate a job, (2) which job to migrate, and (3) to which node to migrate the selected job. The WAL-PM scheme deals with the first and the third questions in a similar approach as the WAL-RE scheme [10][13], as described in Steps 2 and 3. Therefore, the rest of the paper will focus on deciding which job to migrate, that is, to judiciously select an eligible job in *EM* from the overloaded node to migrate.

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 time (cost). In what follows, the expected response time of a candidate migrant *j* on node *i* is given in the following equation:

$$r(i,j) = (t_j - a_j)E(L_i) + (t_j - a_j)\lambda_j\left[E(s_i) + \frac{\Lambda_i \times E(s_i^2)}{2(1-\rho_i)}\right], \quad (1)$$

where a_j, t_j, and λ_j are the age, computation time, and I/O access rate of job *j*, respectively s_i is the I/O service time, and ρ_i is the utilization of the disk in node *i*. $E(L_i)$ represents the mean CPU queue length L_i, and Λ_i denotes the aggregate I/O access rate in node *i*.

The two terms on the right hand side of Equation (1) represent the CPU exectuion time and the I/O processing time, respectively. It is assumed that I/O access is synchronized with its CPU processing, thus, the response time of a job is the summation of CPU response time and I/O reponse time. 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 [9]. The aggregate I/O access rate, Λ_i, is defined as:

$$\Lambda_i = \sum_{k \in M_i} \lambda_k', \quad \text{where } \lambda_k' = \frac{\lambda_k}{E(L_i)}. \quad (2)$$

In Equation (2), M_i is a set containing all the jobs that are assigned to node *i*, and λ_k' is the effective I/O access rate imposed on the disk by job *k*, taking the effect of time-sharing into account. To accurately estimate the effective I/O access rate, λ_k, measured in a non-shared environment, must be deflated by the time-sharing factor, which is $E(L_i)$. Based on λ_k', the disk utilization can be expressed as: $\rho_i = \sum_{k \in M_i} \lambda_k' s_k$.

Let p_k^{IO} be the probability of an I/O access being from job *k* on node *i*, we then have $p_k^{IO} = \lambda_k' / \Lambda_i$. Therefore, the mean I/O service time, used in Equation (1), can be calculated as follows:

$$E(s_i) = \sum_{k \in M_i} \left(p_k^{IO} s_i\right) = \frac{1}{\Lambda_i} \sum_{k \in M_i} \left(\lambda_k' s_k\right) = \frac{\rho_i}{\Lambda_i}, \text{ and}$$

$$E(s_i^2) = \sum_{k \in M_i} \left(p_k^{IO} s_i^2\right) = \frac{1}{\Lambda_i} \sum_{k \in M_i} \left(\lambda' s_k^2\right). \quad (3)$$

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 $p_k^{CPU} = t_k / (t_k + t_k \lambda_k s_k) = 1/(1+\lambda_k s_k)$. Thus, the mean CPU queue length, used in Equation (1) and (2), becomes:

$$E(L_i) = \sum_{k \in M_i} p_k^{CPU} = \sum_{k \in M_i} \frac{1}{1 + \lambda_k s_k}. \quad (4)$$

Based on Equation (1), the set of eligible migrant jobs becomes:

$$EM(i,k) = \left\{ j \in M_i \,\middle|\, r(i,j) > r(k,j) + c_j \right\}, \qquad (5)$$

where k represents a destination node, and c_j is the migration 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, which is modeled as follows,

$$c_j = \begin{cases} e + \dfrac{d_j^{INIT}}{b_{net}} + 2 \times \dfrac{d_j^{INIT}}{b_{disk}} & \text{for remote execution,} \\ f + \dfrac{m_j}{b_{net}} + \dfrac{d_j^{INIT} + d_j^W}{b_{net}} + 2 \times \dfrac{d_j^{INIT} + d_j^W}{b_{disk}} & \text{for migration,} \end{cases} \qquad (6)$$

where e and f are assumed to be the fixed costs for remote execution and preemptive migration, respectively. b_{net} and b_{disk} denote the network bandwidth and disk bandwidth, respectively. d_j^{INIT} represents the amount of data that the job initially accessed, and this amount of data is referred to as *initial data* throughout this paper. Thus the last two terms of the upper line of Equation (6) represent the migration time spent on transmitting data over the network and on accessing source and destination disks, respectively. d_j^W and m_j denote, respectively, the amount of disk (I/O) data and of main memory data generated at the runtime by the job. Similar to the upper line of Equation (6), the last three terms of the lower line of Equation (6) represent the migration time spent over the network on transmitting memory data and disk data, and on accessing the source and destination disks for the migrated disk data. Disk data d_j^W is proportional to the number of write operations that have 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, defined to be the number of times the same data is accessed by the job. Thus, d_j^W is defined by the following equation,

$$d_j^W = \frac{a_j \times \lambda_j \times w_j \times d_j^{RW}}{r_j + 1}, \qquad (7)$$

where w_j is the percentage of I/O operations that store data to the local disk, and the number of write operations is a product of a_j, λ_j, and w_j in the numerator.

In Step (5), WAL-PM chooses one job j from set *EM(i, k)* (Equation 5) in such a way that the benefit of migration is maximized. To find a maximizing factor, we define an objective function, called the *migration cost-effectiveness (MCE)*, which measures the amount of I/O load migrated per unit migration cost. More specifically, for job j, $MCE(j) = (a_j \times \lambda_j) / c_j$, 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 Equation (8),

$$MCE(j) = \underset{p \in EM(i,k)}{MAX} \{MCE(p)\}, \text{ where } j \in EM(i,k) \qquad (8)$$

Besides selecting an appropriate migrant in Step (5), WAL-PM estimates the weighted average load index in Step (1). Since there are three primary resources considered in a cluster, the load index of each node i is the weighted average of CPU, memory and I/O load, thus:

$$load(i) = W_{CPU} \times load_{CPU}(i) + W_{MEM} \times load_{MEM}(i) + W_{IO} \times load_{IO}(i), \qquad (9)$$

where $load_{CPU}(i)$, $load_{MEM}(i)$, and $load_{IO}(i)$ are individual load indices for CPU, memory and I/O resources, respectively. The weight of each resource implies the significance of the resource and the feature of workload. For example, in an I/O-intensive workload where disk I/O processing dominates the overall performance of a cluster, W_{IO}, W_{CPU}, and W_{MEM}, can be configured to 1, 0 and 0, respectively. Therefore, WAL-PM, under the I/O-intensive workload, only attempts to balance I/O resources, ignoring CPU and memory resources.

The three load indices for the workload of CPU, memory and I/O are described below:

(1) The CPU load index of node i is characterized by the length of the CPU waiting queue [16][17], denoted as $load_{CPU}(i)$.

(2) The memory load index of node i, denoted as $load_{mem}(i)$, is the sum of the memory space allocated to those jobs with their computational tasks assigned to node i. More precisely, let $l_{mem}(j)$ represent the memory load (requirement) of job j, then

$$load_{mem}(i) = \sum_{j \in M_i} l_{mem}(j), \qquad (10)$$

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

$$load_{IO}(i) = \sum_{j \in M_i} l_{page}(i,j) + \sum_{j \in M_i} l_{IO}(i,j). \qquad (11)$$

Although it is straightforward to compute CPU load and memory load, the calculation of I/O load is more complicated because of the need to determine the implicit and the explicit I/O load.

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 the job 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 of a node is unable to meet the memory requirements of the jobs, the node encounters a large number of page faults, leading to a high implicit I/O load. Implicit I/O load depends on three factors, namely, the available user memory space, the page fault rate, and the memory space requested by the jobs assigned to node *i*. More precisely, $l_{page}(i, j)$ can be defined as follows [16][17], where μ_i denotes the page fault rate of the node.

$$l_{page}(i, j) = \begin{cases} 0 & \text{if } load_{mem}(i) \le n_{mem}(i), \\ \mu_i \times \dfrac{\sum_{k \in M_i} r_{mem}(k)}{n_{mem}(i)} & \text{otherwise.} \end{cases} \quad (12)$$

$l_{IO}(i, j)$ is proportional to I/O access rate and inversely proportional to I/O buffer hit rate $h(i, j)$. Therefore, $l_{IO}(i, j)$ is approximated by the following expression:

$$l_{IO}(i, j) = \lambda_j \times [1 - h(i, j)]. \quad (13)$$

The hit rate of I/O access for job *j* running on node *i* is approximated by the following formula:

$$h(i, j) = \begin{cases} \dfrac{r_j}{r_j + 1} & \text{if } d_{buf}(i, j) \ge d_{data}(j), \\ \dfrac{r_j}{r_j + 1} \times \dfrac{d_{buf}(i, j)}{d_{data}(j)} & \text{otherwise,} \end{cases} \quad (14)$$

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

$$d_{buf}(i, j) = \frac{\lambda_j \times d_j^{RW}}{\sum_{k \in M_i} \{\lambda_k \times d_j^{RW}\}} \times d_{buf}(i) \quad (15)$$

$$d_{data}(j) = \frac{\lambda_j \times t_j \times d_j^{RW}}{r_j + 1}. \quad (16)$$

5. Performance evaluation

To study the performance of the I/O-aware dynamic load-balancing scheme presented above, we have performed a large number of trace-driven simulations. In this section, we compare the performance of WAL-PM with the existing schemes, namely, IO-based, CM-based, and WAL-RE. In what follows we give a brief description of these three policies.

(1) IO-based load balancing. The load index in this policy represents only the I/O load, given in expression (11). For a job arriving in node *i*, the IO scheme greedily assigns the computational and I/O tasks of the job to the node that has the least accumulated I/O load.

(2) CPU-Memory-based load balancing [16]. When a node *i* has sufficient memory space, the CM scheme balances the system using CPU load index, $load_{CPU}(i)$, as defined in Section 4. When the system encounters a large number of page faults due to insufficient memory space for the running jobs, memory load index, $load_{mem}(i)$, given in expression (10), is used by CM to balance the system.

(3) Weighted-Average-Load-based balancing (WAL-RE) [12]. For every node *I*, the load index defined in WAL is the weighted average of the required resource load:

$$load(i) = W_{IO} \times load_{IO}(i) + W_{CPU} \times load_{CPU}(i). \quad (17)$$

For a new coming job *j*, WAL assigns it to a node that is not overloaded. If such a node is not available, WAL dispatches the job to a node with the smallest value of the load index. In our experiments, both W_{IO} and W_{CPU} are set to 0.5, assuming that I/O and CPU are equally important in the workload.

The performance metric used in our simulations is *slowdown* [7][16], since jobs may be delayed because of waiting in queues or being migrated to remote nodes. Since the definition of slowdown in [7][16] does not consider time spent on I/O access, we extend the definition by incorporating I/O access time. The extended definition of slowdown for a job *j* is given as:

$$slowdown(j) = \frac{wall_time(j)}{CPU_time(j) + IO_time(j)}, \quad (18)$$

where wall_time(j) is the total time the job spends running, accessing I/O, waiting, or migrating.

5.1 Simulator and Simulation Parameters

Before presenting the empirical results, the simulation model and the workload are discussed.

To study dynamic load balancing, Harchol-Balter and Downey [7] implemented a simulator of a distributed system with 6 nodes, in which round-robin scheduling is employed. The load balancing policy studied in this simulator is CPU-based. Zhang et. al [16] extended the simulator, incorporating memory resources into the

simulation system. Based on the simulator, presented in [16], our simulator incorporates the following four new features:

(1) The WAL-PM, WAL-RE, and IO-based schemes are implemented in the simulator;

(2) A fully connected network is simulated;

(3) A simple disk model is added into the simulator;

(4) I/O buffer, used to reduce the disk I/O access frequency, is implemented on top of the disk model.

In all experiments, we used the simulated system with the configuration parameters listed in Table 1. The parameters for CPU, memory, disks, and network are chosen in such a way that they resemble a typical cluster of the current day.

Table 1. Data Characteristics

Parameter	Values assumed
CPU Speed	800(million instructions/second)
RAM Size	640Mbytes
Buffer Size	160Mbytes
Network Bandwidth	1Gbps, 100Mbps,10Mbps
Page fault service time	8.1 ms
Page fault rate	0.1, 1.0, 2.0 per ms
Time slice of CPU	10 ms
Context switch time	0.1 ms
seek and rotation time	8.0 ms
Disk transfer rate	40 MB/s
I/O access rate	0.1, 0.2, …, 2.9

Disk accesses from each job are modeled as a Poisson process with a mean arrival rate λ. The service time of each I/O access is modeled as below:

$$I/O_Service_time = Seek_time + Rotational_delay + Ttransfer_time, \quad (19)$$

$$Transfer_time = \frac{Data_size}{Transfer_rate}, \quad (20)$$

where *Seek_time* is the disk arm positioning time for a disk head move to the desired cylinder, *Rotational_delay* is the time for the desired block to rotate under the disk head, and *Transfer_time* is the time to read/write data in the block. *Transfer_time* equals the amount of data retrieved from or stored to the disk divided by the transfer rate. We assume that both *Seek_time* and *Rotational_delay* are fixed, and the transfer time for each I/O access is computed by expression (20). Data sizes of the I/O requests are randomly generated based on a Gamma distribution, since the sizes chosen in this way reflect typical data characteristics for MPEG-1 data [3], which is retrieved by many multimedia applications. The data characteristic for the I/O requests in our simulation is given in Table 2.

Table 2. Data Characteristics

Data Size	Mean	100 Kbyte
Gamma Distribution	Standard Deviation	50 Kbyte

We modified the traces used in [7][16], adding a randomly generated I/O access rate to each job. In the traces used in our experiments, the CPU and memory demands remain unchanged, and the memory demand of each job is chosen based on a Pareto distribution with the mean size of 4Mbytes [16].

5.2 I/O-Intensive Workload

To stress the I/O-intensive workload in this experiment, the page fault rate is fixed at a very low value of 0.5No./ms, implying that, even when the requested memory space is larger than the allocated memory space, page faults do not occur frequently. This workload reflects a scenario where memory-intensive jobs exhibit high temporal and spatial locality of access. Since the workload of this experiment is highly I/O-intensive and thus heavily biased to I/O resources, the weights of the three load indices, W_{CPU}, W_{MEM}, and W_{IO}, are fixed to 0, 0 and 1, respectively. This configuration assumes that I/O resources are more important than CPU and memory in an I/O intensive workload. Figure 2 plots s*lowdown* as a function of the maximal I/O access rate in the range between 2.4 No./ms and 2.9 No./ms with increments of 0.1 No./ms. The mean slowdowns of IO_RE and IO_PM are almost identical to those of WAL-RE and WAL-PM, respectively, and therefore are omitted from Figure 2.

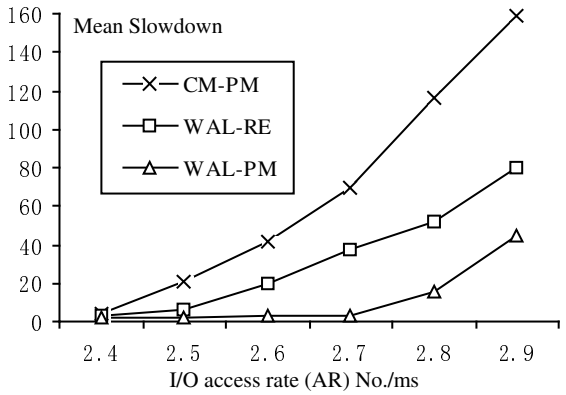

Figure 2. Mean slowdown as a function of the I/O access rate, on a trace with a page fault rate of 0.5 No./ms

The slowdown of CM-RE is also omitted from Figure 2, since its performance is almost the same as that of CM-PM. This is because, when the page fault rate is low, there is little incentive for CPU-Memory-based (*CM-based*) policies have to migrate jobs, whether preemptive or not, making CM-RE and CM-PM equally unlikely to improve the overall system performance any further.

Figure 2 reveals that the mean slowdowns of the three 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.

The results further show that the WAL-RE scheme significantly outperforms the CM-RE and CM-PM policies, suggesting that the CM-based policies are not suitable for I/O intensive workload. For example, as shown in Figure 2, WAL-RE reduces the mean slowdown by up to a factor of 2 (with an average of 112%). This is because CM-based policies only balance CPU and memory load, ignoring the imbalanced I/O load of clusters under the I/O intensive workload.

More interestingly, the proposed WAL-PM policy improves the performance even over WAL-RE by virtue of preemptive job migrations. For example, WAL-PM reduces the slowdown of WAL-RE by up to a factor of 10 (with an average of 353%), and WAL-PM improves the performance in terms of slowdown over both CM-RE and CM-PM by up to a factor of 20. Consequently, the slowdowns of CM-based policies and WAL-RE are more sensitive to I/O access rate than WAL-PM does. 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, leading the node's I/O load to accumulate. However, WAL-PM considers all existing jobs on a node, in addition to the newly arrived job. Therefore, WAL-PM can find more ("optimal") migration opportunities than WAL-RE. Second, 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 even if it soon becomes one of the best candidate migrants due to the load dynamics. Third, even when the net performance gain is insignificant and such migration consumes network resources, the non-preemptive migration policies might still have a newly arrived job with low I/O requirement migrated to a remote node.

5.3 CPU-Memory Intensive Workload

If Section 5.2 presented a best case scenario for the proposed WAL-PM scheme since the workload there was highly I/O-intensive, then this section shows the opposite, a worst case scenario for WAL-PM, namely, subjecting it to a highly CPU-memory-intensive workload. To simulate a memory intensive workload, the 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. The results of the mean slowdown as a function of the page fault rate are summarized in Figure 3. The page fault rate is set from 7.2No./ms to 8.8No./ms with increments of 0.2No./ms. For the WAL-based policies, W_{CPU}, W_{MEM}, and W_{IO} are fixed to 0.1, 0.9 and 0, respectively.

The slowdowns of the CM-based schemes are omitted from Figure 3, since their performance of is nearly identical to that of the WAL-based schemes. The reason for this is that WAL-RE and WAL-PM judiciously adjust the weighted load index to meet the demands of CPU-memory intensive workload. If the weighted load index is wisely configured in accordance with the CPU-memory intensive workload, the WAL-RE and WAL-PM policies gracefully reduce to CM-RE and CM-PM, respectively.

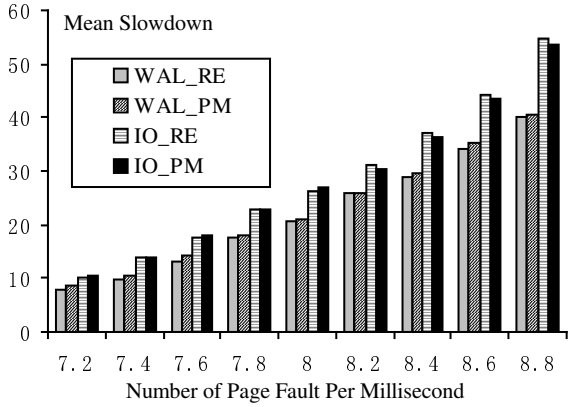

Figure 3. Mean slowdown as a function of the page fault rate, on the trace with an I/O access rate of 0.1 No./ms

When the page fault rate is higher and the I/O rate is low, WAL-RE and WAL-PM outperform the IO_RE and IO_PM considerably. For example, the WAL-based policies reduce the slowdowns over the IO-based policies by up to 40.4% (with an average of 30.4%). The reason for this result is that IO-based policies only attempt to balance explicit I/O load, ignoring the implicit I/O load resulted from page faults. When the page fault rate is high and the explicit I/O load is low, balancing explicit I/O load does not make a significant contribution to balancing the overall system load.

6. Conclusions

In this paper, we have proposed a dynamic load balancing policy, referred to as *WAL-PM* (Weighted-Average-Load based policy with Preemptive Migration), for cluster systems executing applications that represent general and practical workload including intensive I/O activities. WAL-PM considers I/O load, in addition to CPU and memory utilizations. It boosts the performance of both I/O-aware and CPU-Memory-aware load-balancing schemes under I/O-intensive workload, by

considering not only the newly arrived jobs but also older, running jobs as candidate migrant jobs, and by migrating jobs that are the most migration cost-effective.

To evaluate the performance of WAL-PM, we compare it 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 WAL-PM. WAL-PM is more general than the other five approaches, and is able to maintain a high performance under a diverse range of workload conditions. A trace-driven simulation provides extensive empirical results demonstrating that dynamic load balancing with preemptive job migrations under I/O-intensive workload is not only necessary but also highly effective. In particular, the proposed scheme improves the performance over the existing non-preemptive I/O-aware schemes by up to a factor of 10 (with an average of 353%). On the other hand, it outperforms the existing CPU-Memory-based schemes by up to a factor of 20 (with an average of 398%) when the workload is I/O intensive.

6. Acknowledgements

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 University of Nebraska-Lincoln. We are grateful to the anonymous referees for their insightful suggestions and comments.

References

[1] A. Acharya and S. Setia, "Availability and Utility of Idle Memory in Workstation Clusters," *Proc. ACM SIGMETRICS Conf. Measuring and Modeling of Computer Systems*, May 1999.

[2] J. Aerts, J. Korst, and S. Egner, "Random duplicate storage for load balancing in multimedia servers," *Information Processing Letters,* Vol. 76/1-2, pp. 51-59, 2000.

[3] E. Balafoutis, G. Nerjes, P. Muth, M. Paterakis, P. Triantafillou, and G. Weikum, "Clustered Scheduling Algorithms for Mixed-Media Disk Workloads", *Proc. Int'l Conf. on Cluster Computing (CLUSTER 2002),* 2002.

[4] C.Chang, B. Moon, A. Acharya, C. Shock, A. Sussman, J. Saltz, "Titan: A High-Performance Remote-sensing Database," *Proc. of Int'l Conf. on Data Engineering*, 1997.

[5] L. Chen and H. Choi, "Approximation Algorithm for Data Distribution with Load Balancing of Web Servers," *Proc. Int'l Conf. on Cluster Computing (CLUSTER 2001)*, 2001.

[6] R. Ferreira, B. Moon, J. Humphries, A. Sussman, J. Saltz, R. Miller, and A. Demarzo, "the Virtual Microscope," *Proc. of the 1997 AMIA Annual Fall Symposium*, pp. 449-453, Oct. 1997.

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

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

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

[10] X. Qin, H. Jiang, Y. Zhu, and D. Swanson, "A Dynamic Load Balancing Scheme for I/O-Intensive Applications in Distributed Systems*," Proc. of the 32nd International Conference on Parallel Processing Workshops (ICPP 2003 Workshops)*, Taiwan, October 6-9, 2003.

[11] 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," *Proc. of the 9th International Euro-Par Conference on Parallel Processing (Euro-Par 2003),* Klagenfurt, Austria, August 26- 29, 2003.

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

[13] M. Surdeanu, D. Modovan, and S. Harabagiu, "Performance Analysis of a Distributed Question/Answering System,"*IEEE Trans. on Parallel and Distributed Systems*, Vol. 13, No. 6, pp. 579-596, 2002.

[14] 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*, pp. 17251-17262, Oct. 1993.

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

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

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

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

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