In the Proceedings of 2003 International Conference on Parallel Processing Workshops (ICPP 2003 Workshops)

A Dynamic Load Balancing Scheme for I/O-Intensive Applications in Distributed Systems

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

In this paper, a new I/O-aware load-balancing scheme is presented to improve overall performance of a distributed system with a general and practical workload including I/O activities. The proposed scheme dynamically detects I/O load imbalance on nodes of a distributed system and determines whether to migrate the I/O requests of some jobs from overloaded nodes to other less- or under-loaded nodes, depending on data migration cost and remote I/O access overhead. Besides balancing I/O load, the scheme judiciously takes into account both CPU and memory load sharing in distributed systems, thereby maintaining the same level of performance as the existing schemes when I/O load is low or well balanced. Results from a trace-driven simulation study show that, compared with the existing schemes that only consider CPU and memory, the proposed scheme reduces the mean slowdown by up to 54.5% (with an average of 39.9%). On the other hand, when compared to the existing approaches that only consider I/O, the proposed scheme reduces the mean slowdown by up to 57.2% (with an average of 31.6%). More importantly, the new scheme improves over a very recent algorithm found in the literature that considers all the three resources by up to 49.6% (with an average of up to 41.9%).

1. Introduction

In distributed environments, such as a network of workstations and clusters of SMPs, dynamic load balancing schemes can improve system performance by attempting to assign work, at run time, to machines with idle or under-utilized resources. Figure 1 illustrates the architecture of a distributed system considered in this study, in which each node has a combination of multiple types of resources, such as CPU, memory, network connectivity and disks. In this architecture, nodes may or may not be homogeneous and each node is assumed to be capable of migrating a newly arrived job to another node if needed, and maintains a reasonably up-to-date global load information by periodically exchanging load status with other nodes.

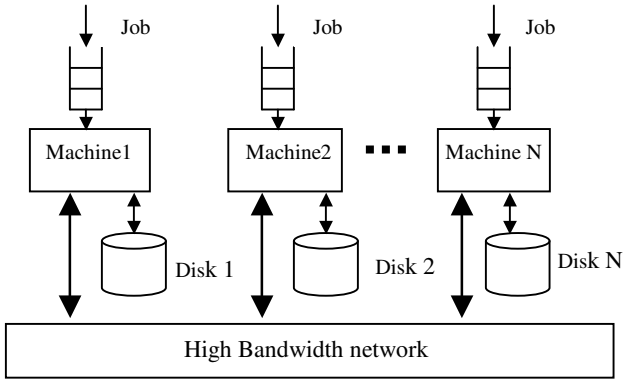

Figure 1. Architecture of a distributed system

Several distributed load-balancing schemes, based on the above architecture, have been presented in the literature, mainly considering CPU [11][12], memory [1][17], or a combination of CPU and memory [6][18] [19]. While these load-balancing policies have been by and large very effective in increasing the utilization of resources in distributed systems, they have ignored one type of resource, namely disk (and disk I/O). The impact of disk I/O on overall system performance is becoming increasingly significant as more and more data-intensive and/or I/O-intensive applications are running on distributed systems. This makes storage devices a likely performance bottleneck. 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 [16], archiving of raw and processed remote sensing data [4][8], 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 depends heavily on the effective usage of global storage, in

addition to that of CPU and memory. Compounding the performance impact of I/O in general, and disk I/O in particular, there is the steadily widening of speed gap between CPU and I/O, making the load imbalance in I/O increasingly more sensitive 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 attempts to comprehensively study an approach, referred to as *IOCM* (load balancing for I/O, CPU, and Memory), to balance a distributed environment in such a way that CPU, memory, and I/O resources at each node can be simultaneously well utilized. The experimental results indicate that, compared with existing load balancing schemes that only consider CPU and memory, IOCM reduces the mean slowdown, informally defined to be the performance degradation of a job due to resource sharing by other jobs [18][19], by up to 54.5%. Compared with existing approach that solely considers I/O, the IOCM scheme reduces the mean slowdown by up to 57.2%. More importantly, IOCM improves the slowdown performance over the scheme found in the literature that also considers all three resources by an average of up to 49.6%.

The rest of the paper is organized as follows. In the section that follows, related work in the literature is briefly reviewed. In Section 3, we describe the IOCM scheme. Section 4 evaluates the performance of the IOCM scheme, and compares it with that of other existing solutions. 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 in the literature in recent years. Harchol-Balter et al. [11] address the question of whether preemptive migration is necessary for CPU-based load balancing in networks of workstations. Zhang et al. [6][19] focus on load sharing policies that consider both CPU and memory services among the nodes. In what follows, the CPU-memory-based load-balancing policy presented in [19] will be referred to as the CM policy. The experimental results show that CM not only improves performance of memory-intensive jobs, but also maintains the same load sharing quality of the CPU-based policies for CPU-intensive jobs [6][19].

A large body of work can be found in the literature that addresses the issue of balancing the load of disk systems [2][13][14]. Scheuermann et al. [14] study two issues in parallel disk systems, namely striping and load balancing, and show their relationship to response time and throughput. Lee et al. [13] propose two file assignment algorithms that minimize the variance of the service time at each disk, in addition to balancing the load across all disks. Aerts et al. [2] use randomization and data redundancy to enable effective load balancing. Since the problem of balancing the utilizations across all disks is isomorphic to multiprocessor scheduling problem [9], a greedy multiprocessor-scheduling algorithm, LPT [10], can be applied to disk load balancing [13]. Thus, LPT greedily assigns a process to the processor with the lightest I/O load [13]. Throughout this paper, we refer the approaches that directly apply LPT to the I/O load balancing as the IO (IO-based) policy. 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.

Communication-sensitive load balancing, a kind of I/O-aware load balancing, has been proposed by Cruz and Park [7]. Compared with their work, the proposed approach in this study focuses on another kind of I/O, namely disk I/O. While the IOCM approach takes into account the communication load as a measure to determine the migration cost, balancing the network load, however, is beyond the scope of this paper.

Very recently, two load balancing models, which consider I/O, CPU and memory resources, have been presented [15][18]. In [15], 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. One of the load-balancing policies presented in [18] considers the three types of resources, and results show that the policy improves overall job execution performance. The two policies proposed in [15][18] are similar in the sense that, for every node, the average load is defined as the weighted average of the required resource load. Throughout this paper, these two load-balancing policies are referred to as the WAL (Weighted Average Load) policy. In WAL, remote I/O accesses are prohibited, thus computation and I/O portions of a job have to be always allocated to the same node. 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. The trace-driven simulations show that, compared with the IO, CM, and WAL policies, our IOCM scheme significantly enhances the overall performance of a distributed system under workload with

a mixture of CPU-memory-intensive and I/O-intensive jobs.

3. IO-CPU-Memory based load balancing

In this section, we present an IO-CPU-Memory based load balancing (IOCM) for a distributed system. Each job is described by its requirements for I/O, CPU and memory. Jobs with intensive I/O requests can be regarded as having two sub-tasks, namely, computaiontal task along with the CPU and memory demands, and I/O task associated with I/O requirement. Due to the partitioning of computational and I/O tasks in each job, the IOCM scheme allows the two tasks to be assigned to different nodes in order to balance load. Consequently, the computational tasks are assigned to nodes based on the jobs' CPU/memory load status, whereas I/O tasks are assigned according to the jobs' I/O load status.

The IOCM scheme attempts to balance the system in such a way that:

(1) I/O usages of all nodes in the system are balanced with best effort;

(2) CPU and memory resources are balanced with best effort; and

(3) Cost in network traffic due to remote I/O access is maintained under a certain level (threshold).

It has been observed that finding the optimal solution, even for relatively simple formulations of this problem, is an NP-hard problem [5]. Consequently, our approach to solve the dynamic load-balancing problem is heuristic and greedy in nature.

Throughout this paper, let i represent node i, and let j denote job j. For a job j, arriving in a local node i, the IOCM scheme attempts to balance three different resources simultaneously following four main steps. First, the 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. Second, IOCM judiciously determines another candidate node, $M_{CM}(j)$, the one with the lightest CPU/memory load, to execute the job. Third, if the network load between nodes $M_{IO}(j)$ and $M_{CM}(j)$ is overloaded, IOCM avoids the job's remote I/O accesses by assigning its computational task to the same node as the I/O task, thereby making the I/O accesses local. Fourth, data migration from node i to $M_{IO}(j)$ is invoked if the data that will be accessed by job j is not initially available in node $M_{IO}(j)$. Finally, the network load and the load status in nodes $M_{IO}(j)$ and $M_{CM}(j)$ are updated.

The detailed pseudo code of the IOCM scheme is presented in Figure 2.

In this scheme, three load indices are applied to measure the workload of CPU, memory and I/O, which are described below:

(1) The CPU load index of node i is characterized by the length of the CPU waiting queue [18][19], denoted as $load_{CPU}(i)$. To identify whether node i's CPU is overloaded, as in step 2.2 of Figure 2, a CPU threshold, denoted as $CPU_threshold(i)$, is assigned to node i. $CPU_threshold(i)$ is defined in accordance with node i's CPU capability. Node i's CPU is considered overloaded, if $load_{CPU}(i) \geq CPU_threshold(i)$. In the experiments reported in Section 4, the value of CPU threshold is kept to four.

Algorithm: IO-CPU-Memory based load balancing (IOCM)
Input: Job j, node i, **Output:** $M_{IO}(j)$, $M_{CM}(j)$.
1 /* *Balance I/O load* */
  if *I/O load on node* i *is not the maximum among all nodes*
  then $M_{IO}(j) \leftarrow$ *local node i;*
  else $M_{IO}(j) \leftarrow$ node with the minimal I/O load;
2 /* Balance CPU and memory load */
 2.1 **if** memory in node i is not overloaded **then**
 2.2 **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 /* Evaluate migration cost and remote I/O access cost */
 3.1 **if** $M_{IO}(j) \neq i$ and migration cost > threshold
  then $M_{IO}(j) \leftarrow i$;
 3.2 **if** $M_{IO}(j) \neq M_{CM}(j)$ and network load between nodes $M_{IO}(j)$ and $M_{CM}(j)$ is overloaded
  then $M_{CM}(j) \leftarrow M_{IO}(j)$ /*Avoided remote I/O accesses*/
4 **if** $M_{IO}(j) \neq i$ and initial data is stored in node i **then** migrate initial data from node i to node $M_{IO}(j)$;
5 Update the load status in Nodes $M_{IO}(j)$ and $M_{CM}(j)$;
6 Update network load;

Figure 2. Pseudo code of the IOCM-based load balancing

(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 $mem_load(j)$ represent the memory load (requirement) of job j, then we have

$$load_{mem}(i) = \sum_{j \in C(i)} mem_load(j), \qquad (1)$$

where $C(i)$ is a set of jobs whose computational tasks are assigned to node i.

(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 issued from jobs. Let $page_load(i, j)$ denote the implicit I/O load, and $IO_load(i, j)$ the explicit I/O load, then, the I/O load index of node i can be defined as:

$$load_{IO}(i) = \sum_{j \in C(i)} page_load(i, j) + \sum_{j \in L(i) \cup R(i)} IO_load(i, j), \qquad (2)$$

where $L(i)$ is a set of jobs whose computational and I/O tasks are both assigned to node i. $R(i)$ is a set of jobs

whose I/O tasks are assigned to node *i* while their computational tasks are assigned to other nodes. It is noted that *L(i)* is a subset of *C(i)*, thus, $L(i) \subseteq C(i)$.

IO_load(i, j) is proportional to I/O access rate and inversely proportional to I/O buffer hit rate *hit_rate(i, j),* which is approximated by the following expression:

$$hit_rate(I, j) = \begin{cases} r/(r+1) & \text{if } buf_size(i, j) \geq data_size(j), \\ \frac{r}{r+1} \times \frac{buf_size(i, j)}{data_size(j)} & \text{otherwise,} \end{cases} \quad (3)$$

where *r*, the data re-access rate, is defined to be the number of times the same data is accessed by a job, *buf_size(i, j)* is the buffer size allocated to job *j*, and *data_size(j)* is the amount of data job *j* retrieves from or stored to the disk, given a buffer with infinite size.

Besides measuring the load of three resources, IOCM estimates the data migration cost in step 3.1 and the network load in step 3.2. In case the data migration cost is too high, migration will not be invoked. Similarly, if the network load is higher than a given threshold, implying a high remote I/O access cost, remote I/O accesses will be avoided by assigning the computational and I/O tasks into the same node.

One efficient way to reduce data migration costs is to duplicate data to be initially accessed by jobs across all disks, if such data is known and available and disk capacity is sufficient. In practice, initial data may not be duplicated in every node, giving rise to the following expression for data migration cost,

$$T_{data_mig}(j) = \begin{cases} D_{init}(j)/B_{net} & \text{Initial data has no replicated copy on the remote node,} \\ 0 & \text{Otherwise,} \end{cases} \quad (4)$$

where $D_{init}(j)$ is the initial data size of job *j*, and b_{net} is the available network bandwidth. It is noted that b_{net} is not a fixed value, and it is dynamically updated in accordance with the network load.

In our model, if a job's computation and I/O tasks are allocated to nodes *i* and *k*, respectively, the job is said to have reserved one unit of usage of the link connecting nodes *i* and *k*. Thus, the network load for any node pair can be approximated by the total units of link usage between the two nodes. In IOCM, step3.2 guarantees that the total link usage units will not exceed to a given threshold, denoted as *net_threshold(i, k)*. It is noted that the value of *net_threshold(i, k)* can be set based on the bandwidth of the network link connecting with two nodes.

4. Performance evaluation

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

(1) IO-based load balancing (IO). The load index in this policy represents only the I/O load, given in expression (2). 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 (CM) [19]. 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 3. 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 (1), is used by CM to balance the system.

(3) Weighted-Average-Load-based balancing (WAL) [15]. For every node *i*, the load index defined in WAL is the weighted average of the required resource load:

$$load(i) = W_{IO} \times load_{IO}(i) + W_{CPU} \times load_{CPU}(i). \quad (5)$$

For a new coming job *j*, WAL assigns it to a node that is not overloaded. If such 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* [11][19], since jobs may be delayed because of waiting in queues or being migrated to remote nodes. Since the definition of slowdown in [11][19] 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:

$$slowdown(j) = \frac{wall_time(j)}{CPU_time(j) + IO_time(j)}, \quad (6)$$

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

4.1 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 [11] implemented a simulator of a distributed system with six nodes, in which round-robin scheduling is employed. The load balancing policy studied in this simulator is CPU-based. Zhang et. al [19] extended the simulator, incorporating memory recourses into the simulation system. Based on the simulator, presented in

[19], our simulator incorporates the following four new features:

(1) The IOCM, IO and WAL 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	800MIPS (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 time sharing	10 ms
Context switch time	0.1 ms
Disk seek time and rotation time	8.0 ms
Disk transfer rate	40 MB/s
I/O access rate	Uniformly distributed between 0 and AR
AR (Maximal I/O access rate)	0.1, 0.2, …, 2.9
Re-access rate, *r*	5

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:

$$I/O_Service_time = Seek_time + Rotational_delay + Ttransfer_time, \quad (7)$$

$$Transfer_time = \frac{Data_size}{Transfer_rate}, \quad (8)$$

where *Seek_time* is the disk arm positioning time for 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 (8). 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 Gamma Distribution	Mean	100 KByte
	Standard Deviation	50 KByte

We modified the traces used in [11][19], 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 [19]. The I/O access rate for each job is generated from a uniform distribution between 0 to AR.

4.2 Overall Performance Comparison

In our first experiment, s*lowdown* is measured as a function of I/O access rate in the range between 1.0 and 1.9 No./ms with increments of 0.1 No./ms, as shown in Figure 3, and as a function of page fault rate in the range between 9.4 and 10.0 No./ms with increments of 0.1 No./ms, as shown in Figure 4. To show that our approach is able to simultaneously balance three resources, the traces in this experiment are generated with a "good-mix" of CPU-memory-intensive and I/O-intensive jobs. In Figure 3, the mean slowdowns of WAL are almost identical to those of IO, and thus are omitted from the figure.

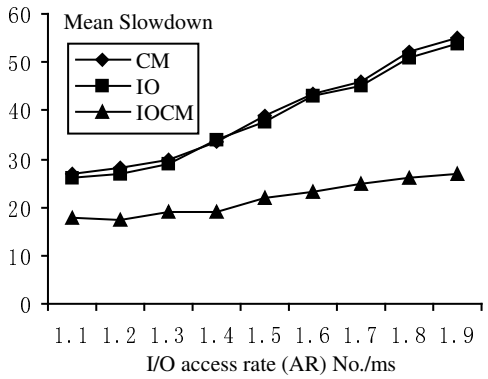

Figure 3. Mean slowdowns as a function of I/O access rate, for a trace with a page fault rate of 8.125 No./ms

First, both Figure 3 and Figure 4 show that mean slowdowns of the four policies all increase with the explicit (Fig.3) or implicit (Fig.4) 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 times on I/O processing.

Second, the results further reveal that the IOCM scheme significantly outperforms IO, CM, and WAL, suggesting

that IO, CM, and WAL are not suitable for the workload with mixture of CPU-memory-intensive and I/O-intensive jobs. For example, as shown in Figure 3, IOCM reduces the mean slowdown by up to 54.5% (with an average of 39.9%). In Figure 4, IOCM reduces the slowdown of CM by up to 50.4% (with an average of 34.9%), and IOCM experiences an average decrease of 27.3% over both IO and WAL. 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.

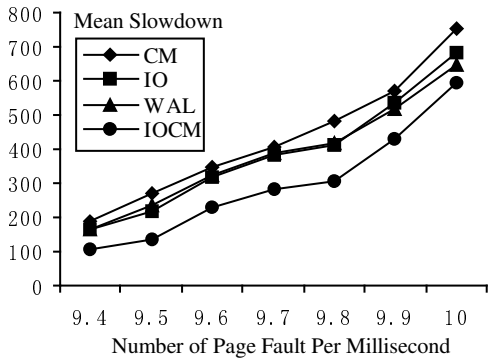

Figure 4. Mean slowdowns as a function of page fault rate for a trace with a maximal I/O access rate of 2.0 No./ms.

4.3 Stress Tests for I/O load

To stress the I/O workload, the page fault rate, in this experiment, 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 may happen when memory-intensive jobs exhibit high temporal and spatial locality of access. In Figure 5, we plot the slowdowns as a function of I/O access rate for four policies, including a policy that does not apply any load-balancing scheme, denoted as ***NL*** (**N**o **L**oad balancing). The I/O access rate is chosen in the range between 2.1 No./ms and 2.9 No./ms with increments of 0.1 No./ms. Since the mean slowdowns of WAL and IO are approximately identical, the data for WAL is omitted from Figure 5.

As noted earlier, the mean slowdown increases with the increase in I/O access rate. The slowdowns of NL and CM are more sensitive to I/O access rate than IOCM, IO, and WAL do. It is also observed from Figure 5 that, when I/O access rate is higher than 2.3 No./ms, IOCM, IO, and WAL consistently outperform perform CM and NL. This is mainly because IOCM, IO, and WAL improve the utilization of disks, which dominate the overall performance when the explicit I/O access rate is high. Figure 5 shows that the slowdowns of CM and NL are very similar to each other. This result indicates that, when the page fault rate is low, the CM scheme is unable to improve the overall system performance any further. More interestingly, though the slowdowns of IO and IOCM appear to be close to each other, IOCM outperforms IO by 9.4% on average. This is because IOCM further reduces implicit I/O access rate by improving the utilization of global memory, whereas IO simply balances the I/O load without considering the memory resources and, as a result, the implicit I/O load is increased due to a large number of page faults.

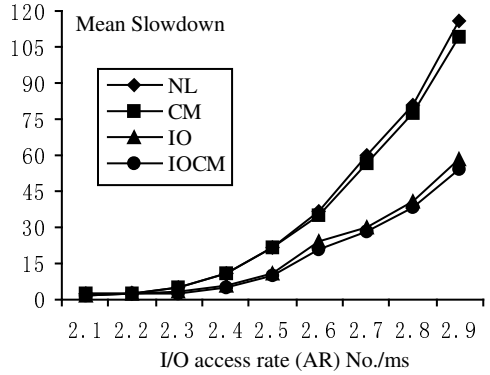

Figure 5. Mean slowdowns as the I/O access rate increases on the trace with page fault rate of 0.5 No./ms

4.4 Stress Tests for Page Fault Rate

To stress the page fault rate, the I/O access rate is fixed at a low value of 0.1 No./ms, keeping the I/O demands at a very low level. This workload represents the scenario where a significant portion of applications running in a distributed system is CPU-Memory-intensive, and there are only a small number of I/O-intensive jobs in the system. The results of the mean slowdown as a function of the page fault rate are summarized in Figure 6. The page fault rate is set from 7.2 No./ms to 8.8 No./ms with increments of 0.2 No./ms. Since the mean slowdowns of WAL and CM are nearly identical, the data for WAL is omitted from Figure 6.

As can be seen in Figure 6, when page fault rate is higher and I/O rate is very low, IOCM, CM, and WAL outperform the IO scheme considerably, with IOCM improving over IO by up to 32.2% (with an average of 24.8%). These results can be explained by the following reasons. First, IOCM, CM, and WAL 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 scheme improves the utilization of disks based only on I/O load, ignoring the imbalanced memory load. Again, Figure 6 illustrates that IOCM consistently outperforms

WAL and CM, by up to 7.0% (with an average of 3.4%). 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.

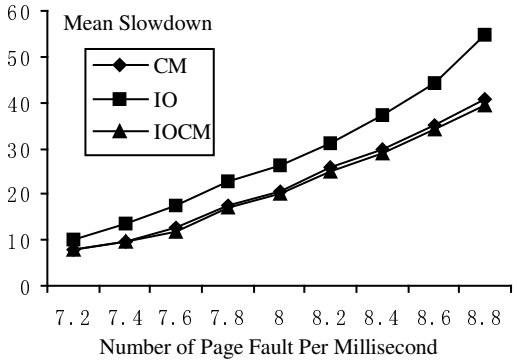

Figure 6. Mean slowdowns as the page fault rate increases on the trace with the maximal I/O access rate of 0.1No./ms

4.5 Data Re-access Rate

As mentioned earlier, data re-access rate, which largely depends on I/O access patterns, affects the overall performance of a distributed system. Figure 7 shows the impact of data re-access rate on the mean slowdown for four policies. In this experiment, the page fault rate and the maximal I/O access rate are fixed to 9.7 No./ms and 2.0 No./ms, respectively. IO and WAL are omitted from Figure 7, since their performances are almost identical to that of CM.

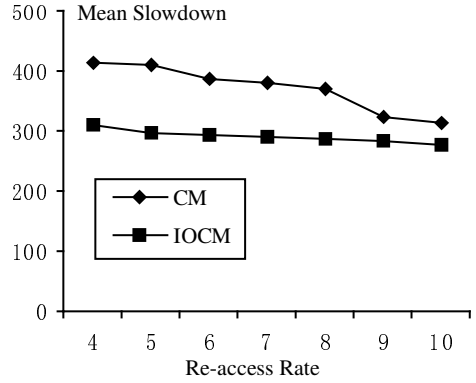

Figure 7. Mean slowdowns as a function of the re-access rate for traces with a maximal I/O access rate of 2.0 No./ms and page fault rate of 9.7No./ms

As shown in Figure 7, the mean slowdowns of all policies decrease as the re-access rate increases. The reason is that the high re-access rate yields a high buffer hit rate, given in expression (3) in Section 3. Consequently, the high buffer hit rate in a node leads to a short I/O service time, since data is more likely to be read from or written to the buffer instead of the disk in the node. A second observation is that the slowdowns of CM, IO, and WAL are much sensitive to re-access rate than that of IOCM, and the improvement of IOCM over three policies decreases with the increasing value of re-access rate. This result suggests that the performance gain by IOCM over three existing policies becomes more pronounced when the re-access rate is low.

4.6 Network Bandwidth

It is expected that network bandwidth is one of the main factors that affect the overall performance of a distributed system. Figure 8 shows the impact of network bandwidth on the mean slowdowns of the four policies. Again, the page fault rate and the maximal I/O access rate in this experiment are fixed at 9.7 No./ms and 2.0 No./ms, respectively. Since the slowdowns of IO and WAL are almost identical, the performance of WAL is not depicted in Figure 8.

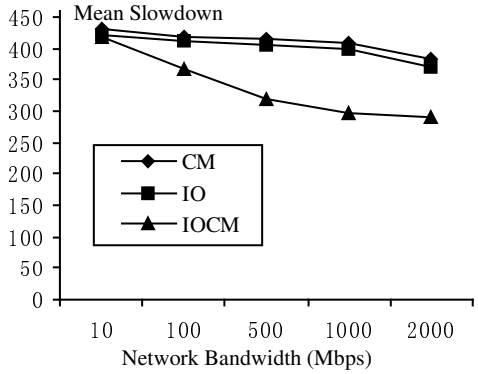

Figure 8. Mean slowdowns as a function of the network band-width for traces with a I/O access rate of 2.0 No./ms and page fault rate of 9.7No./ms

Four polices share a common feature in the sense that, when the network bandwidth increases, the slowdown drops. The reason is that a network with high bandwidth results in a low migration cost in four load-balancing policies. Fig. 8 also reveals that IOCM is much sensitive to the network bandwidth than the other three policies. This result can be explained by the fact that, in addition to decreasing migration cost, the high bandwidth network in IOCM also helps in reducing remote I/O access cost, which may dominate the communication cost when the explicit I/O load in distributed system is high. It is suggested from this experiment that a fast network, such as Myrinet, is strongly recommended for the distributed system that applies the IOCM scheme as a load-balancing policy.

5. Conclusions

In this paper, we have studied a dynamic load balancing policy, referred to as *IOCM* (load balancing for I/O, CPU, and Memory), for distributed systems executing applications that represent general and practical workload including intensive I/O activities. IOCM considers I/O load, in addition to CPU and memory utilizations. To evaluate the performance of IOCM, we compare it with three existing approaches, namely, (1) CPU-Memory-based policy *(CM)*, (2) IO-based policy *(IO)*, and (3) Weighted-Average-load based policy *(WAL)*. IOCM is more general than the existing approaches, and able to maintain a high performance under a diversity of workload conditions. A trace-driven simulation provides us with extensive empirical results to draw several conclusions:

(1) When both memory and I/O demands are high, the performance of IOCM is significantly superior to that of any of the three existing approaches;

(2) Under workload conditions where the I/O load is high and the memory load is low, IO and WAL outperform CM, while IOCM further improves over WAL and IO by up to 14.7%;

(3) When the I/O load is low and the memory load is high, CM and WAL are better than IO, while CM and WAL are outperformed by IOCM;

(4) Data re-access rate affects the performance. A high re-access rate yields a high buffer hit rate, which in turn implies a short I/O service time. Interestingly, the improvement of IOCM over other policies is more significant when the data re-access rate is relatively low.

(5) Network bandwidth plays an important role in the overall performance, since high bandwidth leads to a low migration cost and a low remote I/O cost. Therefore, IOCM gains more benefits from a typical cluster architecture in which the nodes are connected by a high-bandwidth interconnect (e.g., Gigabit Ethernet or Myrinet).

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 *International Conference 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] S. Chen, L. Xiao, and X. Zhang, "Dynamic Load Sharing with Unknown Memory Demands of Jobs in Clusters," *Proc. 21 Int'l Conf. Distributed Computing Systems (ICDCS 2001)*, Apr. 2001.

[7] J. Cruz and Kihong Park, "Towards Communication-Sensitive Load Balancing," *Proc. 21 Int'l Conf. Distributed Computing Systems (ICDCS 2001)*, Apr. 2001.

[8] 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.

[9] M. Garey and D. Johnson, Computers and Intractability: A Guide to the theory of NP-Completeness. W.H. Freeman, 1979.

[10] R.L. Graham, "Bounds on Multiprocessing Timing Anomalies," *SIAM J. Applied Math.*, Vol.17, No.2, pp.416-429, 1969.

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

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

[13] 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.

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

[15] 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.

[16] 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.

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

[18] 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.

[19] 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.