Scalable Computing: Practice and Experience
Volume 6, Number 4, pp. 95–107. http://www.scpe.org
ISSN 1895-1767

A FEEDBACK CONTROL MECHANISM FOR BALANCING I/O- AND MEMORY-INTENSIVE APPLICATIONS ON CLUSTERS

XIAO QIN* , HONG JIANG† , YIFENG ZHU† , AND DAVID R. SWANSON†

Abstract. One common assumption of existing models of load balancing is that the weights of resources and I/O buffer size are statically configured and cannot be adjusted based on a dynamic workload. Though the static configuration of these parameters performs well in a cluster where the workload can be modeled and predicted, its performance is poor in dynamic systems in which the workload is unknown. In this paper, a new feedback control mechanism is proposed to improve overall performance of a cluster with a general and practical workload including I/O-intensive and memory-intensive load. This mechanism is also shown to be effective in complementing and enhancing the performance of a number of existing dynamic load-balancing schemes. To capture the current and past workload characteristics, the primary objectives of the feedback mechanism are: (1) dynamically adjusting the resource weights, which indicate the significance of the resources, and (2) minimizing the number of page faults for memory-intensive jobs while increasing the utilization of the I/O buffers for I/O-intensive jobs by manipulating the I/O buffer size. Results from extensive trace-driven simulation experiments show that compared with a number of schemes with fixed resource weights and buffer sizes, the feedback control mechanism delivers a performance improvement in terms of the mean slowdown by up to 282% (with an average of 125%).

Key words. Feedback control, I/O-intensive applications, cluster, load balancing

1. Introduction. Scheduling [16, 19] and load balancing [1, 10] techniques in parallel and distributed systems have been investigated to improve system performance with respect to throughput and/or individual response time. Scheduling schemes assign work to machines to achieve better resource utilization, whereas load-balancing policies can migrate a newly arrived job or a running job preemptively to another machines if needed.

Since clusters-a type of loosely coupled parallel system-have become widely used for scientific and commercial applications, several distributed load-balancing schemes in clusters have been presented in the literature, primarily considering CPU [9, 10], memory [1, 23], or a combination of CPU and memory [26, 27]. Although these load-balancing policies have been very effective in increasing the utilization of resources in distributed systems (and thus improving system performance), 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 significant as more and more jobs with high I/O demand are running on clusters. 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 [22], archiving of raw and processed remote sensing data [4], multimedia and web-based applications. 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, the steady widening gap between CPU and I/O speed makes 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.

This paper proposes a feedback control mechanism to dynamically configure resource weights and I/O buffers in such a way that the weights are capable of reflecting the significance of system resources, and the memory utilization is improved for I/O- and memory-intensive workload.

The rest of the paper is organized as follows. Related work in the literature is reviewed in Section 2. Section 3 describes system model, and Section 4 proposes the feedback control mechanism. Section 5 evaluates the performance of the mechanism. Finally, Section 6 concludes the paper by summarizing the main contributions and commenting on future directions of this work.

*Department of Computer Science, New Mexico Institute of Mining and Technology, Socorro, New Mexico 87801. http://www.cs.nmt.edu/∼xqin (xqin@cs.nmt.edu). Questions, comments, or corrections to this document may be directed to that email address.

†Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588-0115.

2. Related Work. There exists a large base of excellent research related to distributed load balancing models, and to name just a few: sender or receiver-initiated diffusion [5, 24], the gradient model [6, 13, 14], and the hierarchical balancing model Pollak [24]. Eager et al. studied both receiver and sender initiated diffusion, and the results of their study showed that receiver-initiated policies are preferable at high system loads if the overheads of task transfer under the two policies are comparable [5]. The gradient model makes use of a gradient proximity map of underloaded processors to guide the migration of tasks from overloaded to underloaded processors [6, 13, 14]. Underloaded nodes dynamically update the gradient proximity map, whereas overloaded nodes initiate task migrations. Pollark proposed a scalable approach for dynamic load balancing in large parallel and distributed systems on a multi-level control hierarchy [15]. The hierarchical scheme achieve significant performance gain due to the parallelism in the low level of the hierarchy and the possibility to aggregate information in the higher level of the control tree [15].

The issue of distributed load balancing for CPU and memory resources has been extensively studied and reported in the literature. For example, Harchol-Balter et al. [9] proposed a CPU-based preemptive migration policy that was more effective than non-preemptive migration policies. Zhang et al. [27] focused on load sharing policies that consider both CPU and memory services among the nodes of a cluster. Throughout this paper, the CPU-memory-based load balancing policy presented in [27] will be referred to as CM. The simulation results show that the CM policy 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 [27].

A large body of work can be found in the literature that addresses the issue of balancing the load of disk systems [11, 18]. Scheuermann et al. [18] studied two issues in parallel disk systems, namely striping and load balancing, and showed their relationship to response time and throughput. Lee et al. [11] proposed two file assignment algorithms that minimize the variance of the service time at each disk, in addition to balancing the load across all disks. Since the problem of balancing the utilizations across all disks is isomorphic to the multiprocessor scheduling problem [7], a greedy multiprocessor-scheduling algorithm, called LPT [8], can be applied to disk load balancing [11]. Thus, LPT greedily assigns a process to the processor with the lightest I/O load [11]. Throughout this paper, we refer to the approaches that directly apply LPT to I/O load balancing as the IO 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.

Very recently, three load balancing models, which consider I/O, CPU and memory resources simultaneously, were presented [21, 26]. In [21], a dynamic load-balancing scheme, tailored for the specific requirements of the Question/Answer application, was proposed along with a performance analysis of the approach. Xiao et al. proposed effective load sharing strategies by minimizing both CPU idle time and the number of page faults in clusters [26].

However, the load-balancing models presented in [21, 26] are similar in the sense that the weights of system resources and buffer size are statically configured with a dynamical workload. In contrast, the new feedback control mechanism proposed in this study judiciously configures these parameters in accordance with the workload of the cluster. Trace-driven simulations show that, compared with the CM and IO policies, the proposed scheme with a feedback control mechanism significantly enhances the overall performance of a cluster system under both memory-intensive and I/O-intensive workload.

Some work has been done to make use of feedback control mechanisms in operating systems and distributed environments [12, 20]. For example, Steere et al. proposed a scheduling scheme that dynamically adjusts CPU allocation and period of threads using the feedback of an application's rates of progress with respect to its inputs and/or outputs [20]. Li and Nahrstedt studied a feedback control algorithm to support end-to-end QoS in a distributed environment [12]. However, the feedback controls of resource weights and buffer sizes have not been addressed in these works. In contrast, this paper has presented the experimental results that verify the benefits of the proposed feedback control mechanism for both resource weights and buffer sizes in a highly dynamic environment.

3. System Model. We consider the issue of feedback control method to improve the performance of load balancing schemes in a cluster connected by a high-speed network, where each node not only maintains its individual job queue that holds jobs until they finish execution, but also perceives 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 three main resources, namely, CPU, main memory, and disk I/O. It is assumed that a round-robin scheduling (time-sharing) is employed as the CPU scheduling policy [9, 27], and the disk of

each node is modeled as a single M/G/1 queue [11]. Since jobs may be delayed because of waiting in queues (to share resources with other jobs) or being migrated to remote nodes, the slowdown imposed on a job u is defined as below,

$$slowdown(u) = \frac{t_f(u) - t_a(u)}{t_{CPU}(u) + t_{IO}(u)} \tag{3.1}$$

where $t_f(u)$ and $t_a(u)$ are the finish and arrival times of the job, and $t_{CPU}(u)$ and $time_{IO}(u)$ are the times spent by job u on CPU and I/O, respectively, without any resource sharing.

In expression 3.1, the numerator corresponds to the total time the job spends running, accessing I/O, waiting, or migrating, and the denominator corresponds to the execution time for job u in a dedicated setting. The definition of slowdown is an extension of the one used in [9, 26, 27], where I/O access time is not considered.

For simplicity, we assume that all nodes are homogeneous, having identical computing power, memory capacity, and disk I/O performance characteristics. This simplifying assumption should not restrict the generality of the proposed model, because if a cluster is heterogeneous, the relative load of a given job imposed on a node with high processing capability is less than that imposed on a node with low performance. The proposed scheme may be extended to handle heterogeneous system by incorporating a simple conversion mechanism for relative load [16].

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 [9, 26, 27]. Additionally, we assume that the input data of each job has been stored on the local disk of the node to which the job is submitted. This assumption is conservative in nature, since we conducted an experiment to show that, under I/O-intensive workload, the performance of the proposed schemes with such assumption is approximately 10% less effective than that of the schemes without it.

For a newly arrived job u at a node i, load balancing schemes attempt to ship it to a remote node with the lightest load if node i is heavily loaded, otherwise job u is admitted into node i and executed locally. To avoid useless migration that may potentially degrade the system performance, the load balancing schemes consider transferring a job only if the load discrepancy between the source node and the destination node is greater than the load of the newly arrived job plus the migration cost, therefore guaranteeing that each migration improves the expected slowdown of the job. If an appropriate candidate remote node is not available or the migration is evaluated to be useless, the load balancing schemes will not initiate the job migration.

4. Adaptive Load Balancing Scheme.

4.1. Weighted Average Load-balancing Scheme. In this section, we present WAL, a weighted average 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. For a newly arrived job u at a node i, the WAL-FC scheme balances the system load in the following five steps.

1. First, the load of node i is updated by adding job u's load, assigning the newborn job to the local node.
2. Second, a migration is to be initiated if node i's load is overloaded. Node i is overloaded, if: (1) its load is the highest; and (2) the ratio between its load and the average 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 [25], is obtained from an experiment where the threshold is varied from 1.0 to 2.0.
3. Third, a candidate node j with the lowest load is chosen. In the case where there are more than two nodes with the lowest load, we randomly select one node to break the tie. If a candidate node is not available, WAL-FC will be terminated and no migration will be carried out.
4. Fourth, WAL-FC determines if job u is eligible for migration. A job is eligible for migration if its migration is able to potentially reduce the job's slowdown.
5. Finally, job u is migrated to the remote node j, and the load of nodes i and j is updated in accordance with job u's load.

WAL-FC calculates the weighted average load index in the first step. The load index of each node i is defined as the weighted average of CPU and I/O load, thus:

$$load(i) = W_{CPU} \times load_{CPU}(i) + W_{IO} \times load_{IO}(i), \tag{4.1}$$

where $load_{CPU}(i)$ is CPU load defined as the number of running jobs and $load_{IO}(i)$ is the I/O load defined as the summation of the individual implicit and explicit I/O load contributed by jobs assigned to

node i. W_{CPU} and W_{IO} are resource weights used to indicate the significance of the corresponding resource.

It is noted that the memory load is expressed by the implicit I/O load imposed by page faults. Let $l_{page}(i,u)$ and $l_{IO}(i,u)$ denote the implicit and explicit I/O load of job u assigned to node i, respectively. $load_{IO}(i)$ can be defined by equation 4.2, where M_i is a set of jobs running on node i:

$$load_{IO}(i) = \sum_{u \in M_i} l_{page}(i,u) + \sum_{u \in M_i} l_{IO}(i,u). \tag{4.2}$$

Let $r_{MEM}(u)$ denote the memory space requested by job u, and $n_{MEM}(i)$ represent the memory space in bytes that is available to all jobs running on node i. It is to be noted that the memory space, $n_{MEM}(i)$, can be configured in accordance with the buffer size that is adaptively tuned by the feedback control mechanism proposed in Section 4.2. 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, $l_{page}(i,u)$ can be defined as follows, where μ_i denotes the page fault rate of the node, and $load_{MEM}(i)$ is the memory load denoted as the sum of the memory requirements of the jobs running on node i.

$$l_{page}(i,u) = \begin{cases} 0 & \text{if } load_{MEM}(i) \le n_{MEM}(i), \\ \frac{\mu_i \times \sum_{v \in M_i} r_{MEM}(v)}{n_{MEM}(i)} & \text{otherwise.} \end{cases} \tag{4.3}$$

$l_{IO}(i,u)$ in Equation 4.2 is a function of I/O access rate, denoted λ_u), and I/O buffer hit rate $h(i,u)$ that will be discussed in Section 4.1. Thus, $l_{IO}(i,u)$ is approximated by the following expression:

$$l_{IO}(i,u) = \lambda_u \times (1 - h(i,u)). \tag{4.4}$$

In what follows, we quantitatively determine whether a job is eligible for migration. When a job u is assigned to node i, its expected response time $r(i,u)$ can be computed in Equation 4.5.

$$r(i,u) = t_u \times E(L_i) + t_u \times \lambda_u \times E(s^i_{disk} + \frac{\Lambda^i_{disk} \times E((s^i_{disk})^2)}{2(1-\rho^i_{disk})}), \tag{4.5}$$

where t_u and λ_u are the computation time and I/O access rate of job u, respectively. $E(s^i_{disk})$ and $E((s^i_{disk})^2)$ are the mean and mean-square I/O service time in node i, and ρ^i_{disk} is the utilization of the disk in node i. $E(L_i)$ represents the mean CPU queue length L_i, and Λ^i_{disk} denotes the aggregate I/O access rate in node i. Since the expected response time of an eligible migrant on the source node has to be greater than the sum of its expected response time on the destination node and the migration cost, job u is eligible for migration if:

$$r(i,u) > r(j,u) + c_u, \tag{4.6}$$

where j represents a destination node, and c_u is the migration cost (time) modeled as follows,

$$c_u = e + d_u \times (\frac{1}{b^{ij}_{net}} + \frac{1}{b^i_{disk}} + \frac{1}{b^j_{disk}}), \tag{4.7}$$

where e is the fixed cost of migrating the job and loading it into the memory on another node, b^{ij}_{net} denotes the available bandwidth of the network link between node i and j, b^i_{disk} is the available disk bandwidth in node i. In practice, b^{ij}_{net} and b^j_{disk} can be measured by a performance monitor [3]. Accordingly, the simulator discussed in Section 5 estimates b^{ij}_{net} and b^j_{disk} by storing the most recent values of the disk and network bandwidth. d_u represents the amount of data initially stored on disk to be processed by job u. Thus, the second term on the right hand side of Equation 4.7 represents the migration time spent on transmitting data over the network and on accessing source and destination disks.

4.2. Problem Description and Examples. The feedback control mechanism that aims at minimizing the mean slowdown focuses on adjusting the resource weights and the buffer sizes. To help describe the problem of fixed resource weights and I/O buffer sizes, we first present the following examples that motivate the proposed solution to improve the system performance.

Assume a cluster with six identical nodes [9, 17, 26, 27], to which the IO load-balancing policy is applied. The average page-fault rate and I/O access rate are chosen to be 2.0 No./ms (Number/Millisecond) and 2.8 No./ms, respectively. The total memory size for each node is 640 Mbyte, and other parameters of the cluster are given in Section 5.1. We modified the traces used in [9, 27], adding a randomly generated I/O access rate to each job. The traces used in [9] have been collected from one workstation on six different time intervals. 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 4 Mbytes [27].

To evaluate the impact of resource weights (see Equation 4.1) on the system performance, we conducted a simulation experiment where the resource weights were statically set. Figure 4.1 plots the relationship between the resource weight of I/O and the mean slowdown experienced by all the jobs in the trace. The result indicates that the mean slowdown consistently decreases as the I/O resource weight increases from 0 to 1 with increments of 0.2. We attributed this observation to the fact that, under I/O-intensive workload conditions, the I/O resource weight with a high value is able to accurately reflect the significance of the disk I/O resources in the system.

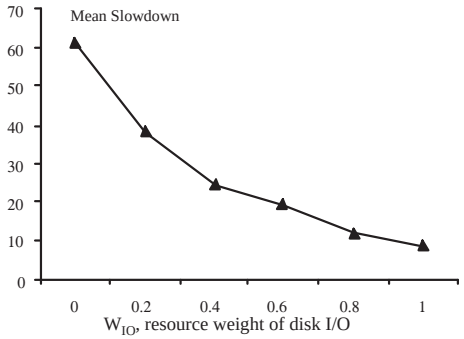

FIG. 4.1. *Mean slowdowns as a function of the I/O resource weight. Average page-fault rate* = 2.0*No./ms, average I/O access rate* = 2.8 *No./ms.*

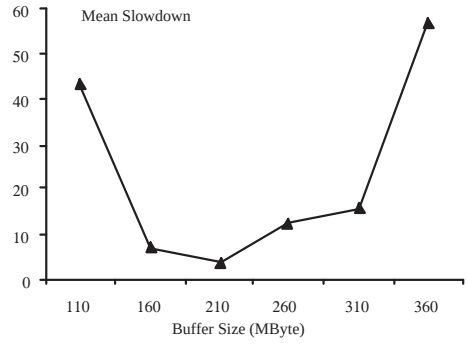

FIG. 4.2. *Mean slowdowns as a function of the buffer size. Average page-fault rate is* 5.0 *No./ms, average I/O access rate is* 2.3*No./ms*

The memory of each node is divided into two portions, with one serving as I/O buffer and the other being used to store working sets of running jobs. Without loss of generality, we assume that the buffer sizes of six nodes are identical. We conducted a second experiment, in which the buffer sizes were statically configured. Figure 4.2 shows the buffer size chosen in the experiment and the corresponding mean slowdowns obtained from the simulator.

The curve in Figure 4.2 reveals that the buffer size has a large effect on the mean slowdowns of the IO-aware policy. When buffer size is smaller than 210 MByte, the slowdown decreases with the increasing value of the buffer size. In contrast, the slowdown increases as the buffer size increases if the buffer size is greater than 210 MByte. Optimally, the mean slowdown of this given workload reaches the minimum value when buffer size is 210 MByte. A large buffer size results in a high buffer hit rate and reduces I/O processing time, thereby causing a positive effect on the performance. On the other hand, given a fixed value of the total available main memory size, a larger buffer size implies a smaller the amount of memory used to store the working sets of running jobs, which in turn leads to a larger number of page faults. In general, a large buffer size may introduce both positive and negative effect on the mean slowdown at the same time, and the overall performance depends on the resultant effect.

Although the static configuration of resource weights and buffer sizes is an approach to tuning the performance of clusters where workload conditions can be modeled and predicted, this approach performs poorly and inefficiently for highly dynamic environments where workloads are unknown at compile time. Therefore, a feedback control algorithm is developed in this study to adaptively configure resource weights and buffer sizes.

4.3. A Feedback Control Mechanism. The high level view of the architecture for the feedback control mechanism is presented in Figure 4.3, where the architecture comprises a load-balancing scheme, a resources-sharing controller, and a feedback controller. The resource-sharing controller consists of a CPU scheduler, a memory allocator and an I/O controller. The slowdown of a newly completed job and the history slowdowns are fed back to the feedback controller, which then determines the required control action ΔW_{IO} and $\Delta bufsize$. $\Delta W_{IO} > 0$ means the IO-weight needs to be increased, and otherwise the IO-weight should be decreased. Since the sum of W_{CPU} and W_{IO} is 1, the control action ΔW_{CPU} can be obtained accordingly. Similarly, $\Delta bufsize > 0$ means the buffer size needs to be increased, and otherwise the buffer size is to be decreased.

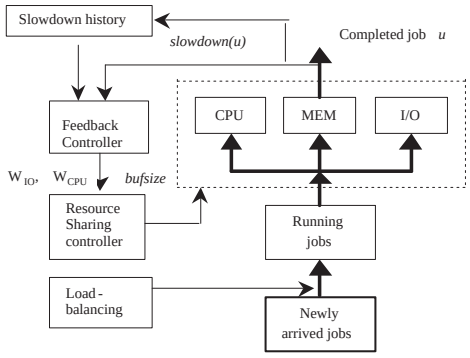

FIG. 4.3. *Architecture of the feedback control mechanism*

The first goal of the feedback controller is to manipulate the resource weights in a way that makes it possible to minimize the mean slowdown of jobs. The system model for an open loop balancer is approximately given by the following equation,

$$slowdown(z) = -wg(L)W_{IO}(z) + wd(L), \tag{4.8}$$

where $wg(L)$ and $wd(L)$ are the gain factor and disturbance factor of the I/O resource weight under workload L, respectively. The values of wg and wd largely depend on workload conditions and the applied load-balancing policy. Thus, wg and wd can be obtained based on simulation models for open-loop load balancers. The control rule for the resource weight is formally modeled below,

$$\Delta W_{IO,u} = G_w(1 - \frac{\overline{S_u}}{\overline{S_{u-1}}})\frac{\Delta W_{IO,u-1}}{|\Delta W_{IO,u-1}|}, \tag{4.9}$$

$$W_{IO,u} = W_{IO,u-1} + \Delta W_{IO}, \tag{4.10}$$

where $\Delta W_{IO,u}$ is the control action, $\overline{S_u}$ denotes the average slowdown, $\frac{\Delta W_{IO,u-1}}{|\Delta W_{IO,u-1}|}$ indicates whether the previous control action has increased or decreased the resource weight, and G_w denotes the controller gain for the I/O resource weight. In the experiments presented shortly in the next section, G_w is tuned to be 0.5 for better performance. Let $W_{IO,u}$ be the resource weight upon the arrival of job at the system, the resource weight will be updated to $W_{IO,u-1} + \Delta W_{IO}$. Without loss of generality, we make use of a linear model to capture the characteristics of varying workload conditions. The model is given by the following equation,

$$slowdown(z) = -wg_0(L)W_{IO}(z) + wd_0 + \Delta wd, \tag{4.11}$$

The feedback controller attempts to manipulate the resource weights in the following three steps. First, when a job u is accomplished, the controller calculates the slowdown s_u of this newly completed job, Second, s_u is stored in the slowdown history table, and the average slowdown $\overline{S_u}$ is computed accordingly. Note that $\overline{S_u}$ reflects a specific pattern of the recent slowdowns in the dynamic workload. The table size is a tunable parameter, and the oldest slowdown will be replaced by the latest one if the history table overflows. In our simulation model presented in Section 5.1, the history table size is fixed to 50. Finally, the controller generates

control actions $\Delta W_{IO,u}$ and $\Delta W_{CPU,u}$, which are based on the previous control action along with the comparison between $\overline{S_u}$ and $\overline{S_{u-1}}$. More precisely, the performance is regarded to be improved by the previous control action if $\overline{S_{u-1}} > \overline{S_u}$, therefore the controller continues increasing W_{IO} if it has been increased by the previous control action, otherwise W_{IO} is decreased. Similarly, $\overline{S_{u-1}} < \overline{S_u}$ means that the performance has been worsened since the latest control action, suggesting that W_{IO} has to be increased if the previous control action has reduced W_{IO}, and vice versa.

Besides configuring the weights, the second goal of the feedback control mechanism is to dynamically set the buffer size of each node based on the unpredictable workload. The mechanism is aiming at improving buffer utilizations and reducing the number of page faults by maintaining an effective usage of memory space for running jobs and their data.

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

$$slowdown(z) = -bg(L)bufsize(z) + bd(L), \tag{4.12}$$

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

$$\Delta bufsize_u = G_b(\overline{S_{u-1}} - \overline{S_u})\frac{\Delta bufsize_{u-1}}{|\Delta bufsize_{u-1}|}, \tag{4.13}$$

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

As can be seen from 4.3, the feedback control generates control action $\Delta bufsize$ in addition to ΔW_{CPU} and ΔW_{IO}. The adaptive buffer size makes noticeable impacts on both the memory allocator and I/O controller, which in turn affect the overall performance (See Figure 4.2). The feedback controller generates a control action $\Delta bufsize$ based on the previous control action along with the comparison between $\overline{S_u}$ and $\overline{S_{u-1}}$. Specifically, $\overline{S_{u-1}} > \overline{S_u}$, means the performance is improved by the previous control action, thereby increasing the buffer size if it has been increased by the previous control action, otherwise the buffer size is reduced. Likewise, $\overline{S_{u-1}} < \overline{S_u}$, indicates that the latest buffer control action leads to a worse performance, implying that the buffer size has to be increased if the previous control action has reduced the buffer size, otherwise the controller decreases the buffer size.

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

5. Experiments and Results. To evaluate the performance of the proposed load-balancing scheme with a feedback control mechanism, we have conducted a trace-driven simulation, in which the performance metric used is slowdown that is defined earlier in section 3. We have evaluated the performance of the following load-balancing policies:

1. CM: the CPU-memory-based load-balancing policy [27] without using buffer feedback controller. If the memory is imbalanced, CM assigns the newly arrived job to the node that has the least accumulated memory load. When CPU load is imbalance and memory load is well balanced, CM attempts to balance CPU load.
2. IO: the IO-based policy [11] without using the feedback control mechanism. The IO policy uses a load index that represents only the I/O load. For a job arriving in node i, the IO scheme greedily assigns the job to the node that has the least accumulated I/O load.
3. WAL: the Weighted Average Load-balancing scheme without the feedback controller [21].
4. WAL-FC: the Weighted Average Load-balancing scheme with the feedback control mechanism.
5. NLB: The non-load-balancing policy without using the feedback controller.

5.1. Simulation Model. Before presenting the empirical results, the trace-driven simulation model and the workload are presented.

To study dynamic load balancing, Harchol-Balter and Downey [9] implemented a trace-driven simulator for a distributed system with 6 nodes in which round-robin scheduling is employed. The load balancing policy studied in that simulator is CPU-based. Zhang et. al [27] extended the simulator, incorporating memory recourses into the simulation system. Based on the simulator presented in [27], our simulator incorporates the following new features: (1) The above polices are implemented in the simulator. (2) The interconnect is assumed to be a fully connected network. (3) A simple disk model is added into the simulator. (4) An I/O buffer model, which will be presented shortly in this section, is implemented on top of the disk model. The traces used in the simulation are modified from [9][27], and it is assumed that the I/O access rate is randomly chosen in accordance with a uniform distribution. We assume that the I/O access rate of each job is independent of the job's memory space requirement and CPU service time. Although this simplification deflates any correlations between I/O requirement and other job characteristics, we can examine the impact of I/O requirement on system performance by configuring the mean I/O access rate as a workload parameter.

The simulated system is configured with parameters listed in Table 5.1. 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 5.1
Data Characteristics

Parameters	**Value**	**Parameters**	**Value**
CPU Speed	800 MIPS	Page Fault Service Time	8.1 ms
RAM Size	640 MByte	Seek and Rotation time	8.0 ms
Initial Buffer Size	160 MByte	Disk Transfer Rate	40MB/Sec.
Context switch time	0.1 ms	Network Bandwidth	1Gbps

Disk accesses of each job are modeled as a Poisson process. Data sizes d_u^{RW} of the I/O requests in job u are randomly generated based on a Gamma distribution with the mean size of 250 KByte and the standard deviation of 50 Kbyte. The sizes chosen in this way reflect typical data characteristics for MPEG-1 data [2], which is retrieved by many multimedia applications.

Since buffer can be used to reduce the disk I/O access frequency (See Equation 4.4), we approximately model the buffer hit probability of I/O access for job u running on node i by the following formula:

$$h(i,u) = \begin{cases} \frac{r_u}{r_u+1} & \text{if } d_{buf}(i,u) \geq d_{data}(u), \\ \frac{r_u}{r_u+1} \times \frac{d_{buf}(i,u)}{d_{data}(u)} & \text{otherwise,} \end{cases} \tag{5.1}$$

where r_u is the data re-access rate, $d_{buf}(i,u)$ is the buffer size allocated to job u, and $d_{data}(u)$ is the amount of data job u 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. $d_{data}(u)$ linearly depends on access rate, computation time and average data size of I/O accesses d_u^{RW}, and $d_{data}(u)$ is inversely proportional to I/O re-access rate. $d_{buf}(i,u)$ and $d_{data}(u)$ are estimated using the following two equations:

$$d_{buf}(i,u) = \frac{\lambda_u d_u^{RW} d_{buf}(i)}{\sum_{k \in M_i} \lambda_k d_u^{RW}}, \tag{5.2}$$

$$d_{data}(u) = \frac{\lambda_u t_u d_u^{RW}}{r_u + 1}. \tag{5.3}$$

From Equations 5.1, 5.2 and 5.3, hit rate $h(i,u)$ becomes:

$$h(i,u) = \begin{cases} \frac{r_u}{r_u+1} & \text{if } d_{buf}(i,u) \geq d_{data}(u), \\ \frac{r_u d_{buf}(i)}{t_u \sum_{j \in M_i} \lambda_j d_j^{RW}} & \text{otherwise.} \end{cases} \tag{5.4}$$

Figure 5.1 shows the effects of buffer size on the buffer hit probabilities of the NLB, CM and IO policies. When buffer size is smaller than 150 Mbyte, the buffer hit probability increases almost linearly with the buffer

size. The increasing rate of the buffer hit probability drops when the buffer size is greater than 150 Mbyte, suggesting that further increasing the buffer size can not significantly improve the buffer hit probability when the buffer size approaches to a level at which a large portion of the I/O data can be accommodated in the buffer.

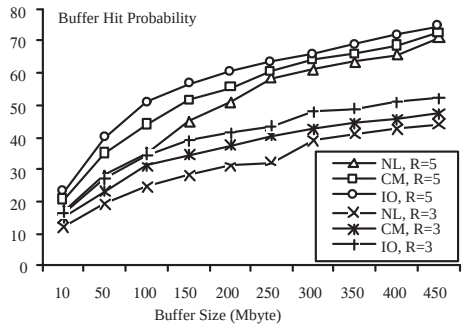

FIG. 5.1. *Buffer Hit Probability as a function of the Buffer Size, page-fault rate is 4.0 No./ms, I/O access rate is 2.2No./ms.*

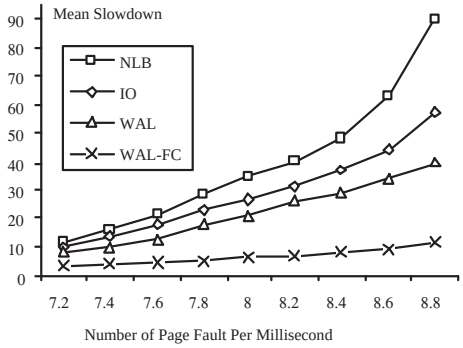

FIG. 5.2. *Mean slowdowns as a function of the page-fault rate, I/O access rate of 0.1 No./ms.*

5.2. Memory Intensive Workload. To simulate a memory intensive workload, the I/O access rate is fixed to a comparatively low level of 0.1 No./ms. The page-fault rate is set from 7.2 No./ms to 8.8 No./ms with increments of 0.2 No./ms. The performance of CM is omitted, since it is very close to that of WAL.

Figure 5.2 reveals that the mean slowdowns of all the policies increase with the page-fault rate. This is because as I/O demands are fixed, high page-fault rate leads to a high utilization of disks, causing longer waiting time on I/O processing.

When the page-fault rate is high, WAL outperforms IO and NLB, and the WAL-FC has better performance than both WAL and IO. For example, the WAL policy reduces slowdowns over the IO policy by up to 37.2% (with an average of 31.5%), and the WAL-FC policy improves the performance in terms of mean slowdown over IO by up to a factor of 4 (400%). The reason is that the IO policy only attempts to balance explicit I/O load, ignoring the implicit I/O load that resulted from page faults. When the explicit I/O load is low, balancing explicit I/O load does not make a significant contribution to balancing the overall system load. In addition, NLB is consistently the worst among the six policies, since NLB leaves three shared resources extremely imbalanced and does not improve the buffer utilization by the adaptive configuration of buffer sizes.

More interestingly, the policies that use the feedback control mechanism algorithm considerably improve the performance over those without employing the feedback controller. For example, WAL-FC improves the system performance over WAL by up to 274% (with an average of 220%). Consequently, the slowdowns of NLB, WAL, and IO are more sensitive to the page-fault rate than WAL-FC.

5.3. I/O-Intensive Workload. To stress the I/O-intensive workload in this experiment, the I/O access rate is fixed at a high value of 2.8 No./ms, and the page-fault rate is chosen from 1.6 No./ms to 2.1 No./ms with increments of 0.1No./ms. The low page-fault rates imply 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. Figure 5.3 plots slowdown as a function of the page-fault rate. The results of IO are omitted from Figure 5.3, since they are nearly identical to those of WAL.

First, the results show that the WAL scheme significantly outperforms the NLB and CM policies, suggesting that NLB and CM are not suitable for I/O intensive workload. For example, as shown in Figure 5.3, WAL improves the performance of CM in terms of the mean slowdown by up to a factor of 9 (with an average of 476%). This is because the CM policies only balance CPU and memory load, ignoring the imbalanced I/O load of clusters under the I/O intensive workload.

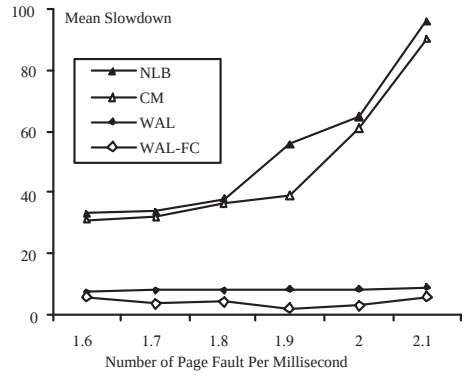

FIG. 5.3. *Mean slowdown as a function of the page-fault rate, I/O access rate is 2.8 No./ms.*

Second, Figure 5.3 shows that WAL-FC significantly outperforms WAL. For example, WAL-FC delivers a performance improvement over WAL by up to 282% (with an average of 125%). Again, this is because the WAL-FC scheme applies the feedback controller to meet the high I/O demands by changing the weights and the I/O buffer sizes to achieve a high buffer hit probability. This result suggests that improving the I/O buffer utilization by using the feedback control mechanism can potentially alleviate the performance degradation resulted from the imbalanced I/O load.

Third, the results further show the slowdowns of NLB and CM are very sensitive to the page-fault rate. In other words, the mean slowdowns of NLB and CM all increase noticeably with the increasing value of I/O load. One reason is, as I/O load are fixed, a high page-fault rate leads to high disk utilization, causing longer waiting time on I/O processing. A second reason is, when the I/O load is imbalanced, the explicit I/O load imposed on some node will be very high, leading to a longer paging fault processing time. Conversely, the page-fault rate makes insignificant impact on the performance of WAL, and WAL-FC. Since the high I/O load imposed on the disks is diminished either by balancing the I/O load or by improving the buffer utilization. This observation suggests that the feedback control mechanism is capable of boosting the performance of clusters under I/O-intensive workload even in the absence of any dynamic load-balancing schemes.

5.4. Memory and I/O intensive Workload. The two previous sections presented the best cases for the proposed scheme since the workload was either highly memory-intensive or I/O-intensive but not both. In these extreme scenarios, the feedback control mechanism provides more benefits to clusters than load-balancing policies do. This section attempts to show another interesting case in which the cluster has a workload with both high memory and I/O intensive jobs. The I/O access rate is set to 1.5 No./ms. The page fault rate is from 7.2 No./ms to 8.4 No./ms with increments of 0.2 No./ms.

Figure 5.4 shows that the performances of CM, IO, and WAL are close to one another. This is because the trace, used in this experiment, comprises a good mixture of memory-intensive and I/O-intensive jobs. Hence, while CM takes advantage of balancing CPU-memory load, IO can enjoy benefits of balancing I/O load. Interestingly, under this specific memory and I/O intensive workload, the resultant effect of balancing CPU-memory load is almost identical to that of balancing I/O load.

A second observation is that, under the memory and I/O intensive workload, load-balancing schemes achieve higher level of improvements over NLB. The reason is that when both memory and I/O demands are high, the buffer sizes in a cluster are unlikely to be changed, as there is a memory contention among memory-intensive and I/O-intensive jobs. Thus, instead of fluctuating widely to optimize the performance, the buffer sizes finally converge to a value that minimizes the mean slowdown.

Third, incorporating the feedback control mechanism in the existing load-balancing schemes is able to further boost the performance. For example, compared with WAL, WAL-FC further decreases the slowdown by up to 54.5% (with an average of 30.3%). This result suggests that, to sustain a high performance in clusters, compounding a feedback controller with an appropriate load-balancing policy is desirable and strongly recommend.

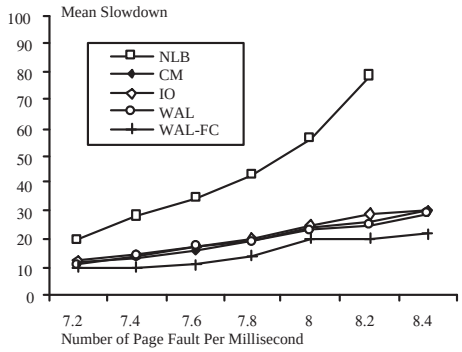

FIG. 5.4. *Mean slowdowns as a function of the page-fault rate, I/O access rate of 1.5 No./ms.*

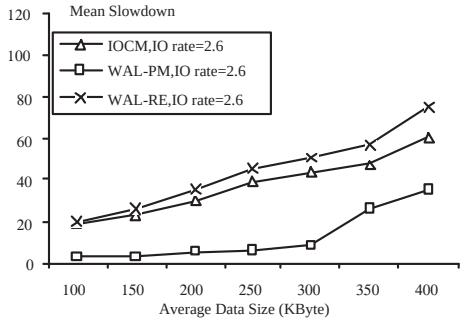

FIG. 5.5. *Mean slowdown as a function of the size of average data size. Page fault rate is 0.5 No./ms, and I/O rate is 2.6 No./ms.*

5.5. Average Data Size. In the previous experiments, the data sizes are chosen based on typical multimedia applications. It is noted that I/O load depends on I/O access rate and the average data size of I/O requests, which in turn rely on the I/O access patterns of applications. The purpose of this experiment is to study the performance improvements achieved by the feedback control mechanism for other types of applications if they exhibit different characteristics. Specifically, Figure 5.5 shows the impact of average data size on the performance of the feedback control mechanism. The page fault rate and the I/O access rate are set to 0.5 No./ms and 2.6 No./ms., respectively. The average data size is chosen from 100 KByte to 400 KByte with increments of 50 KByte.

Figure 5.5 indicates that, for three examined load-balancing policies, the mean slowdown increases as the average data size increases. This is because, under circumstance that both page fault rate and I/O access rate are fixed, a large average data size yields a high utilization of disks, causing longer waiting times on I/O processing. More importantly, Figure 5.5 shows that the performance improvement gained by the feedback control mechanism becomes more noticeable when the average data size is large. This result suggests that the proposed approach is not only beneficial for multimedia applications, but also turns out to be useful for a variety of applications that are data intensive in nature.

6. Conclusions. In this paper, we have proposed a feedback control mechanism to dynamically adjust the weights of recourses and the buffer sizes in a cluster with a general and practical workload that includes memory and I/O intensive workload conditions. The primary objective of the proposed approach is to minimize the number of page faults for memory-intensive jobs while improving the buffer utilization of I/O-intensive jobs. The feedback controller judiciously configures the weights to achieve an optimal performance. Meanwhile, under a workload where the memory demand is high, the buffer sizes are decreased to allocate more memory for memory-intensive jobs, thereby leading to a low page-fault rate.

To evaluate the performance of the mechanism, we compared the proposed WAL-FC scheme with WAL, CM, and IO. For comparison purposes, the NLB policy that does not consider load balancing is also simulated. A trace-driven simulation provides extensive empirical results demonstrating that WAL-FC is effective in enhancing performance of existing dynamic load-balancing policies under memory-intensive or I/O-intensive workload. In particular, when the workload is memory-intensive, WAL-FC reduces the mean slowdown over the CM and IO policies by up to a factor of 9. Further, we have made the following observations:

1. When the page-fault rate is higher and the I/O rate is very low, WAL and CM outperform IO and NLB, and WAL-FC has better performance than WAL;
2. When I/O demands are high, WAL and IO are significantly superior to CM and NLB. And WAL-FC has noticeably better performance than that of IO;
3. Under an I/O intensive workload, the mean slowdowns of NLB and CM all increase noticeably with I/O load. Conversely, the page-fault rate makes insignificant impact on the performance of IO, WAL, and WAL-FC.
4. Under the workload with a good mixture of memory and I/O intensive jobs, WAL-FC achieves high level of improvements over NLB.

5. The performance improvement gained by the feedback control mechanism becomes pronounced when the average data size is relatively large. Future studies in this area may be performed in several directions. First, the feedback control mechanism will be implemented in a cluster system. Second, we will study the stability of the proposed feedback controller. Finally, it will be interesting to study how quickly the feedback controller converges to the optimal value in clusters.

7. Acknowledgements. This work was partially supported by an NSF grant (EPS-0091900), a start-up research fund (103295) from the research and economic development office of the New Mexico Institute of Mining and Technology, a Nebraska University Foundation grant (26-0511-0019), a UNL Academic Program Priorities Grant, and a Chinese NSF 973 project grant (2004cb318201). 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] 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, 2002.
[3] J. Basney and M. Livny, *Managing Network Resources in Condor*, Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing, Pittsburgh, Pennsylvania, August 2000, pp 298-299.
[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] D. Eager, E. Lazowaska, and J. Zahorjan, *A comparison of receiver-initiated and sender-initiated adaptive load sharing*, Proceedings of the 1985 ACM SIGMETRICS conference on Measurement and modeling of computer systems, Austin, Texas, 1985.
[6] D. J. Evans and Wunbutt, *Load balancing with network partitioning using host groups*, Parallel computing, 20:325-345, March 1994.
[7] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the theory of NP-Completeness*, W.H. Freeman, 1979.
[8] R. L. Graham, *Bounds on Multiprocessing Timing Anomalies*, SIAM J. Applied Math., Vol.17, No.2, pp.416-429, 1969.
[9] M. Harchol-Balter and A. Downey, *Exploiting Process Lifetime Distributions for Load Balacing*, ACM transaction on Computer Systems, vol. 3, no. 31, 1997.
[10] C. Hui and S. Chanson, *Improved Strategies for Dynamic Load Sharing*, IEEE Concurrency, vol.7, no.3, 1999.
[11] 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.
[12] B. Li and K. Nahrstedt, *A Control Theoretical Model for Quality of Service Adaptations*, in IEEE International Workshop on Quality of Service, May 1998.
[13] F.C.H. Lin and R.M. Keller, *The Gradient Model Load Balancing Method*, IEEE Trans. Software Engineering, vol. 13, no. 1, pp. 32-38, Jan. 1987.
[14] F. Muniz and E.J. Zaluska, *Parallel Load Balancing: An Extension to the Gradient Model*, Parallel Computing, vol. 21, pp. 287-301, 1995.
[15] R. Pollak, *A Hierarchical Load Balancing Environment for Parallel and Distributed Supercomputer*, Proc. of the International Symposium on Parallel and Distributed Supercomputing, Fukuoka, Japan, September 1995.
[16] X. Qin, H. Jiang, and D. R. Swanson, *An Efficient Fault-tolerant Scheduling Algorithm for Real-time Tasks with Precedence Constraints in Heterogeneous Systems*, Proc. the 31st Int'l Conf. on Parallel Processing (ICPP 2002), Vancouver, Canada, Aug 2002, pp. 360-368.
[17] X. Qin, H. Jiang, Y. Zhu, and D. Swanson, *Dynamic Load Balancing for I/O-Intensive Tasks on Heterogeneous Clusters*, Proc. of the 10th International Conference on High Performance Computing (HiPC 2003), Dec.17-20, 2003, Hyderabad, India.
[18] P. Scheuermann, G. Weikum, P. Zabback, *Data Partitioning and Load Balancing in Parallel Disk Systems*, The VLDB Journal, pp. 48-66, July, 1998.
[19] H. Shen, S. Lor, and P. Maheshwari, *An architecture-independent graphical tool for automatic contention-free process-to-processor mapping*, Journal of Supercomputing, Vol. 18, No. 2, 2001, p. 115-139.
[20] D. C. Steere, A. Goel, J. Gruenberg, et. al., *A Feedback-driven Proportion Allocator for Real-Rate Scheduling*, Operating Systems Designand Implementation, New Orleans, Louisiana, Feb 1999.
[21] 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.
[22] 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.
[23] G. Voelker, *Managing Server Load in Global Memory Systems*, Proc. ACM SIGMETRICS Conf. Measuring and Modeling of Computer Systems, May 1997.
[24] M. Willebeck-LeMair and A. Reeves, *Strategies for Dynamic Load Balancing on Highly Parallel Computers*, IEEE Trans. Parallel and Distributed Systems, vol. 4, no. 9, pp. 979-993, Sept. 1993.
[25] X. Wu, V. Taylor, and R. Stevens, *Design and Implementation of Prophesy Automatic Instrumentation and Data Entry System*, Proc. of the 13th International Conference on Parallel and Distributed Computing and Systems, Anaheim, CA, August 2001.

[26] 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.
[27] 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.

Edited by: Hong Shen
Received: February 27, 2004
Accepted: June 6, 2004