Journal of Network and Computer Applications, vol. 31, no. 1, pp. 32-46, January 2008.

Performance Comparisons of Load Balancing Algorithms for I/O-Intensive Workloads on Clusters

Xiao Qin

Department of Computer Science and Software Engineering
Auburn University, Auburn, AL 36849
xqin@auburn.edu
http://www.eng.auburn.edu/~xqin

Abstract

Load balancing techniques play a critically important role in developing high-performance cluster computing platforms. Existing load balancing approaches are concerned with the effective usage of CPU and memory resources. Due to imbalance in disk I/O resources under I/O-intensive workloads, the previous CPU- or memory-aware load balancing schemes suffer significant performance drop. To remedy this deficiency, in this paper we propose a novel load-balancing algorithm (hereinafter referred to as IOLB) for clusters, which aims at maintaining high resource utilization under a wide range of workload conditions. Specifically, IOLB is conducive to reducing the average slowdown of all parallel jobs submitted to a cluster by balancing load in disk resources. This can, in turn, not only achieve the effective usage of global disk resources but also reduce response times of I/O-intensive parallel jobs. To theoretically study the optimization of the IOLB algorithm, we qualitatively comparing IOLB with two conventional CPU- and memory-aware load-balancing schemes. We prove that when the workloads become CPU-intensive or memory-intensive in nature, IOLB gracefully degrades towards the existing load-balancing schemes. Experimental results based on trace-driven simulations demonstratively show that the IOLB algorithm significantly improves the resource utilization of a cluster under I/O-intensive workloads. Furthermore, our results confirm that IOLB is able to maintain the same level of performance as the two existing approaches, because IOLB improves CPU and memory utilization under CPU- and memory-intensive workloads.

1. Introduction

Load balancing techniques play an important role in the design and development of high-performance clusters. A variety of load balancing schemes [1][3][12][17] can be used to improve performance of parallel and distributed systems clusters by assigning work, at run time, to computational nodes with under-utilized resources. Dynamic load-balancing schemes have been extensively investigated, primarily focusing on CPU [12][14], memory [1][28], network [3][8][24], or a combination of CPU and memory [32] resources. Although the existing load-balancing schemes are effective in maintaining high utilization of resources, the previous approaches suffer significant performance drop under I/O-intensive workloads due to imbalance in disk I/O resources. It is worth noting that disk I/O resources become a performance bottleneck under I/O-intensive workloads, since the performance gap between CPU and disk I/O is widening. It is believed that a way of solving the disk I/O bottleneck problem is to leverage load-balancing techniques to achieve effective usage of global disk resources in clusters.

In this paper we propose a novel load-balancing algorithm (hereinafter referred to as IOLB) for parallel jobs running on clusters. The IOLB algorithm aims at improving utilization of disk I/O, CPU, and memory resources in a cluster under a wide spectrum of workload. Specifically, IOLB is conducive to balancing load in a variety of resources, thereby reducing slowdowns of all parallel jobs submitted to a cluster. After qualitatively comparing IOLB with two conventional CPU- and memory-aware load-balancing schemes, we prove that IOLB gracefully degrades towards the existing load-balancing schemes if workloads become CPU- and memory-intensive, respectively. We conducted trace-driven simulations to show that the IOLB algorithm significantly improves the resource utilization of clusters under I/O-intensive workloads.

Moreover, our results confirm that IOLB improves CPU and memory utilization under CPU- and memory-intensive workloads and, therefore, IOLB can maintain the same level of performance as the two existing approaches.

The rest of the paper is organized as follows. Related work in the literature is briefly reviewed in the following section. Section 3 describes a generic model and the IOLB load-balancing algorithm for parallel jobs. Section 4 presents a qualitative comparison between IOLB and the two existing load-balancing schemes. To confirm the analytical comparison, in Section 5 we made use of trace-driven simulations to quantitatively evaluate performance of the IOLB algorithm and the alternative solutions. Finally, Section 6 summarizes the main contributions of this paper and comments on future research directions.

2. Related Work

In the past decade, load balancing techniques in the context of CPU and memory resources has been extensively studied in recent years. For example, Harchol-Balter and Downey studied a preemptive migration policy that is more effective than non-preemptive migration policies under CPU-intensive workloads [12]. Zhang *et al.* [32] proposed new load sharing policies that are concerned with effective usage of both CPU and memory resources. The above load-balancing schemes are able to achieve high system performance under CPU- and memory-intensive workload conditions, respectively.

A number of approaches to balancing load in disk I/O resources can be found in the literature [16][33]. Lee *et al.* studied two file assignments algorithms to balance load across all disks, thereby making it possible to improve overall system performance by fully utilizing available hard drives [16]. Zhang *et al.* proposed three I/O-aware scheduling schemes that are aware of the

job's spatial preferences [33].

In recent years, the issue of leveraging I/O cache and buffer to boost performance of storage systems has been reported in the literature. Choi *et al.* proposed a buffer replacement scheme for effective caching of disk blocks[7]. Forney et al. investigated storage-aware caching algorithms in heterogeneous clusters [11]. Ma *et al.* developed an active buffering mechanism to alleviate the disk I/O bottleneck problem using local idle memory and overlapping I/O with computation [18]. We proposed a feedback control mechanism to improve the performance of a cluster by adaptively manipulating the I/O buffer size [23]. Our IOLB load-balancing algorithm is complementary to the aforementioned caching and buffering techniques, meaning that IOLB can provides additional performance improvement when the existing caching and buffering mechanisms.

3. An I/O-aware Load-Balancing Algorithm

3.1 A generic model

A cluster computing platform considered in this study consists of a set $N = \{N_1, N_2, \ldots, N_n\}$ of n homogeneous nodes connected by a high-speed interconnection network like Myrinet. Note that the terms node and machine are used interchangeably throughout this paper. Throughout this paper, N_i represents a set of tasks running on the ith node. Each node in a cluster is composed of a combination of various resources, including processors, memory, network connectivity, and disks. A load manager residing in each node is responsible for load balancing and monitoring available resources of the node. Each job is associated to a home machine, through which the job is submitted to the cluster. A node becomes the home machine of a job either because the job is initially created on the node or because data to be accessed by the job is stored in the node. A

similar home model was proposed by Lavi and Barak in context of load balancing [15]. In the case that a job is submitted through its home node, the corresponding load manager for the home node is invoked to allocate the job to a node or a group of nodes with the least load. At any time, a node is either having its load manager performed or executing a task. When the load manager is carried out, the underlying computation may be concurrently performed or suspended. It is feasible to make the load manager and other tasks executed in parallel, since the load manager can be running in the background by an inexpensive coprocessor [31]. In addition, it is reasonable to assume that all load managers in a cluster is capable of keeping track of global load information by monitoring local resources and sharing load information through a direct communication network [22].

In this study we are concerned with a class of embarrassing parallel applications, each of which is represented in form of a set $T = \{\tau_1, \tau_2, ..., \tau_m\}$ of tasks (also referred to as processes) that are independent of one another. Some real-world examples of embarrassing parallel applications can be found in [29]. It is worth noting that the proposed load-balancing algorithm can be readily integrated with a communication load balancer to deal with parallel applications with dependent tasks. A task τ_i in T is modeled as a tuple $(c_i, s_i, \lambda_i, d_i)$, where c_i is the computational time, s_i is the requested memory space measured by MBytes, λ_i is the arrival rate of disk requests measured by number of disk accesses per ms (No./ms), and d_i is the disk request's data size in KBytes. Note that the parameters c_i and s_i are used to describe task τ_i's requirements for CPU and memory, whereas the I/O requirement of τ_i is characterized by the other two parameters - λ_i and d_i.

3.2 Problem Formulation

Given a task τ_i, we denote t_i^{ded} as the time to execute the task on a dedicated computing cluster, and t_i as the time to execute the task on the same cluster in a time-sharing setting. In our model, we consider three resources: CPU, memory, and disk I/O. Let $t_i^{ded,CPU}, t_i^{ded,page}$, and $t_i^{ded,IO}$ be the times of a task spent on CPU, page fault handling, and disk I/O processing in a dedicated mode. The values of $t_i^{ded,CPU}, t_i^{ded,page}$, and $t_i^{ded,IO}$ can be respectively derived from τ_i's requirement parameters, including c_i, s_i, λ_i, and d_i. Let t_i^{CPU}, t_i^{page}, and t_i^{IO} denote the times spent on the three resources in a time-shared mode. Since tasks running on computational nodes may be delayed by contention for resources, the slowdown imposed on task τ_i is expressed as the ratio between the task's execution time in the time-shared mode and its execution time on the same cluster in the dedicated mode. Thus, the slowdown of τ_i is written as

$$sd_i = \frac{t_i}{t_i^{ded}} = \frac{t_i^{CPU} + t_i^{page} + t_i^{IO}}{t_i^{ded,CPU} + t_i^{ded,page} + t_i^{ded,IO}}. \quad (1)$$

Given a parallel application with task set $T_i = \{\tau_1, \tau_2, ..., \tau_{mi}\}$, the slowdown of the application is calculated as the average slowdown of all the independent tasks in T. Thus, the slowdown of the parallel application is written as

$$sd(T_i) = \frac{1}{m_i}\sum_{j=1}^{m_i} sd_j = \frac{1}{m_i}\sum_{j=1}^{m_i} \frac{t_j^{CPU} + t_j^{page} + t_j^{IO}}{t_j^{ded,CPU} + t_j^{ded,page} + t_j^{ded,IO}}. \quad (2)$$

For a special case where all the independent tasks in the set T_i are identical, (i.e., $\forall 1 \le j \le m_i : t_j^{ded,CPU} = t^{ded,CPU}$, $t_j^{ded,page} = t^{ded,page}, t_j^{ded,IO} = t^{ded,IO}$), Eq. (2) can be rewritten as follows

$$sd(T_i) = \frac{1}{m_i \cdot (t^{ded,CPU} + t^{ded,page} + t^{ded,IO})}\sum_{j=1}^{m_i} t_j^{CPU} + t_j^{page} + t_j^{IO}. \quad (3)$$

The goal of the proposed load-balancing algorithm is to reduce the average slowdown of all parallel applications submitted to a cluster. This can, in turn, minimize the average response time of the running applications. Specifically, our algorithm aims at optimizing the following average slowdown of a sequence of parallel applications (e.g., T_1, T_2, …, T_q) executed on a cluster

$$\text{minimize } sd = \frac{1}{q}\sum_{i=1}^{q} sd(T_i) = \frac{1}{q}\sum_{i=1}^{q}\left(\frac{1}{m_i}\sum_{j=1}^{m_i}\frac{t_j^{CPU} + t_j^{page} + t_j^{IO}}{t_j^{ded,CPU} + t_j^{ded,page} + t_j^{ded,IO}}\right). \quad (4)$$

3.3 A load-balancing algorithm for I/O-intensive workloads

Now we present a load-balancing algorithm (hereinafter referred to as IOLB) for a wide variety of workload conditions including I/O-intensive, CPU-intensive, and memory-intensive, workloads. The objective of the proposed IOLB algorithm is to balance the load of three types of resources across all nodes in a cluster such that the average slowdown of submitted parallel jobs is minimized. Since the goal of this study is to analytically evaluate the performance of the IOLB algorithm, we are focused on a remote execution mechanism in which a task can be running on a remote node where it started execution. Thus, preemptive migrations of tasks are not supported in the IOLB algorithm. Nevertheless, the IOLB load-balancing algorithm can be readily integrated with a preemptive migration mechanism, thereby providing further performance improvement. Recently, we studied a load-balancing algorithm with preemptive migration, and details of this algorithm can be found in [22].

To facilitate the description of the IOLB algorithm, we first introduce the following three load indices with respect to CPU, memory, and I/O resources. The CPU load index L_i^{CPU} of node i is defined as the sum of remaining CPU lifetimes of tasks running on the node. Thus, L_i^{CPU} is expressed as $L_i^{CPU} = \sum_{\tau_j \in N_i} r_j$, where r_j is the expected remaining CPU lifetime of task τ_j. The

memory load index L_i^{page} of node i is defined as the sum of page fault processing times t_j^{page} of tasks on node i. Hence, we have $L_i^{page} = \sum_{\tau_j \in N_i} t_j^{page}$. Similarly, the I/O load index L_i^{IO} of node i is the sum of I/O processing times of tasks on the node. Therefore, the I/O load index can be written as

$$L_i^{IO} = \sum_{\tau_j \in N_i} \left(r_j \cdot \lambda_j \cdot t_{j,IO} \right), \tag{5}$$

where $t_{j,IO}$ is task τ_j's the I/O processing time of each disk request. The value of $t_{j,IO}$ in Eq. (5) is computed by

$$t_{i,IO} = t_{seek} + t_{rot} + \frac{d_i}{B_{disk}}, \tag{6}$$

where t_{seek} and t_{rot} are the seek time and rotational latency, and $\frac{d_i}{B_{disk}}$ is the data transfer time depending on data size d_i and disk bandwidth B_{disk}.

In light of the three load indices described above, we propose a new concept of load imbalance factor to quantify the amount of imbalance in a cluster. The load imbalance factor of a resource is a product of the fraction of time spent on using the resource in a cluster and the discrepancy between the maximum and the minimum loads of the resource among all nodes in the cluster. More specifically, the load imbalance factors for CPU, memory, and I/O resources can be written as Eq. (7)-(9).

$$LIF_{CPU} = \frac{\sum_{i=1}^{n} L_i^{CPU}}{\sum_{i=1}^{n} L_i^{CPU} + \sum_{i=1}^{n} L_i^{page} + \sum_{i=1}^{n} L_i^{IO}} \cdot \sum_{i=1}^{n} \left| L_i^{CPU} - \overline{L}_i^{CPU} \right|, \tag{7}$$

$$LIF_{page} = \frac{\sum_{i=1}^{n} L_i^{page}}{\sum_{i=1}^{n} L_i^{CPU} + \sum_{i=1}^{n} L_i^{page} + \sum_{i=1}^{n} L_i^{IO}} \cdot \sum_{i=1}^{n} \left| L_i^{page} - \overline{L}_i^{page} \right|, \quad (8)$$

$$LIF_{IO} = \frac{\sum_{i=1}^{n} L_i^{IO}}{\sum_{i=1}^{n} L_i^{CPU} + \sum_{i=1}^{n} L_i^{page} + \sum_{i=1}^{n} L_i^{IO}} \cdot \sum_{i=1}^{n} \left| L_i^{IO} - \overline{L}_i^{IO} \right|, \quad (9)$$

where $\overline{L}_i^{CPU} = \frac{1}{n}\sum_{i=1}^{n} L_i^{CPU}$, $\overline{L}_i^{page} = \frac{1}{n}\sum_{i=1}^{n} L_i^{page}$, and $\overline{L}_i^{IO} = \frac{1}{n}\sum_{i=1}^{n} L_i^{IO}$. On the right-hand sides of Eqs. (7)-(9), the first terms, which reflect the importance of the three types of resources, are the fractions of times spent on computing, page fault handling, and disk I/O processing, respectively. The second terms on the right-hand side of Eqs. (7)-(9) are used to measure the amount of imbalance in the three resources.

The load imbalance factor *LIF* of a cluster can be derived from Eqs. (7)-(9) as the sum of the load imbalance factors of the three resource types. Thus, we have

$$LIF = LIF_{CPU} + LIF_{page} + LIF_{IO}. \quad (10)$$

Now we are positioned to delineate the IOLB load-balancing algorithm, of which the pseudocode is shown in Fig. 1. Given an embarrassing parallel application with a set *T* of independent tasks submitted to a local node N_i of a cluster, the IOLB algorithm make an effort to balance workload of the cluster's resources by allocating each task in *T* to a computational node such that the task's expected response time (also known as turn around time) is minimized. In other words, IOLB aims to redistribute load among all the node in a cluster, thereby allowing the submitted parallel application to efficiently run on the cluster.

For each task τ_j in set *T*, the IOLB algorithm repeatedly performs Steps 2-12 described as

follows. First, the response time R_j^i of τ_j on the local node N_i is approximated by Step 2. The estimation of R_j^i is important because it will be used to justify whether a remote execution of τ_j is worthwhile (see Steps 6, 11, and 16).

Algorithm: IO-Aware Load Balancing (IOLB)
Input: A local node N_i, a job with task set T submitted to N_i.

1. **for** (each task $\tau_j \in T$) **do**
2. calculate the response time R_j^i of τ_j on node N_i;
3. **if** $c_j \cdot \lambda_j \cdot t_{j,IO} > 0$ **and** $LIF_{IO} = \max(LIF_{CPU}, LIF_{page}, LIF_{IO})$ **and** $L_i^{IO} = \max_{a=1}^{n}(L_a^{IO})$ **then**
4. choose node N_k such that $L_k^{IO} = \min_{a=1}^{n}(L_a^{IO})$;
5. calculate the response time R_j^k of τ_j on node N_k;
6. **if** $c_j \cdot \lambda_j \cdot t_{j,IO} < (c_j \cdot \lambda_j \cdot t_{j,IO} + L_i^{IO} - L_k^{IO})/2$ **and** $R_j^i > R_j^k + e$ **then**
7. dispatch task τ_j to node N_k and remotely execute τ_j on N_k;
8. **else** locally execute τ_j on N_i;
9. **else if** $t_j^{page} > 0$ **and** $LIF_{page} > LIF_{CPU}$ **and** $L_i^{page} = \max_{a=1}^{n}(L_a^{page})$ **then**
10. choose node N_k such that $L_k^{page} = \min_{a=1}^{n}(L_a^{page})$;
11. calculate the response time R_j^k of τ_j on node N_k;
12. **if** $t_j^{page} < (t_j^{page} + L_i^{page} - L_k^{page})/2$ **and** $R_j^i > R_j^k + e$ **then**
13. dispatch task τ_j to node N_k and remotely execute τ_j on N_k;
14. **else** locally execute τ_j on N_i;
15. **else if** $L_i^{CPU} = \max_{a=1}^{n}(L_a^{CPU})$ **then**
16. choose node N_k such that $L_k^{CPU} = \min_{a=1}^{n}(L_a^{CPU})$;
17. calculate the response time R_j^k of τ_j on node N_k;
18. **if** $r_j < (r_j + L_i^{CPU} - L_k^{CPU})/2$ **and** $R_j^i > R_j^k + e$ **then**
19. dispatch task τ_j to node N_k and remotely execute τ_j on N_k;
20. **else** locally execute τ_j on N_i;
21. Update the load status;
22. **end for**

Fig. 1. Pseudocode of the IOLB load-balancing algorithm.

Second, Step 3 is responsible for initiating the process of balancing workload of disk I/O resources. Specifically, Steps 4-7 are invoked to balance the load of I/O resources in the case that all the following three conditions hold in Step 3. Condition 1 (i.e., $c_j \cdot \lambda_j \cdot t_{j,IO} > 0$) states that the I/O load of τ_j must be greater than zero (see Theorems 2 and 3). Condition 2 (i.e., $LIF_{IO} = \max(LIF_{CPU}, LIF_{page}, LIF_{IO})$) says that the load imbalance factor of disk I/O must be higher than those CPU and memory resources. Condition 3 (i.e., $L_i^{IO} = \max_{a=1}^{n}(L_a^{IO})$) means that the I/O load of the local node is the highest among those of all the nodes.

Third, if it becomes a necessity to balance I/O load, then Step 4 chooses the most approximate remote node N_k with the lightest load with respect to disk I/O, followed by estimating the response time R_j^k of τ_j on the candidate node N_k. Step 6 is of critical importance to ensure performance improvement achieved by having τ_j executed remotely. More specifically, before Step 7 dispatches τ_j and has it remotely executed on N_k, Step 6 must make sure that the following two conditions are satisfied. Condition 1 (i.e., $c_j \cdot \lambda_j \cdot t_{j,IO} < (c_j \cdot \lambda_j \cdot t_{j,IO} + L_i^{IO} - L_k^{IO})/2$) guarantees that the load discrepancy between L_i^{IO} and L_k^{IO} is reduced. Condition 2 (i.e., $R_j^i > R_j^k + e$, where e is the remote execution overhead) ensures that the expected response time of τ_j on the selected remote node N_k is less than the response time of τ_j on the local node N_i. In the case that the remote execution of τ_j is not beneficial, Step 8 has τ_j locally executed on N_i.

Fourth, Step 9 decides if the following three conditions are satisfied before a meaningful

remote execution is performed. Condition 1 (i.e., $t_j^{page} > 0$) says that τ_j must exhibit page fault behavior (see Theorems 5 and 6). Page fault behaviors occur when the memory space required by running tasks exceeds the amount of available memory space. Condition 2 (i.e., $LIF_{page} > LIF_{CPU}$) indicates that the load imbalance factor of memory resources must be higher than that CPU resources. Condition 3 (i.e., $L_i^{page} = \max_{a=1}^{n}\left(L_a^{page}\right)$) states that the load of page-fault processing in the local node is the highest among those of all the nodes. If the above conditions hold, Steps 10-14 aim to balance the load of memory resources by transferring task τ_j from the overloaded node to a remote node that are lightly loaded with respect to memory. Step 12 is carried out to guarantee that the remote execution of τ_j leads to performance improvement.

Fifth, if there is no way of balancing the disk I/O and memory resources in the cluster, Steps 15-20 attempt to evenly distribute the CPU load. When the local node is overloaded with respect to CPU resource (See Step 15), task τ_j is dispatched to and executed by a remote node with the lightest CPU load. Step 19 makes the remote execution possible if such a remote execution is beneficial (see Step 18).

Last, Step 21 maintains updated load information that is broadcasted to the local node and other nodes in the cluster.

The following theorem proves the time complexity of the IOLB load-balancing algorithm.

Theorem 1. Given a cluster and a parallel application submitted to the cluster, the time complexity of the IOLB algorithm is *O*(*nm*), where *n* is the number of nodes in the cluster, *m* is the number of tasks in the application, and the values of *n* and *m* are much larger than 2.

Proof. It takes *O*(*1*) time to compute the response time of a task on a node. The time complexity

of determining that a local node is overly loaded is $O(n)$, since there are n nodes in the cluster (see Step 3). Step 4 takes $O(n)$ time to choose the most appropriate node with the minimal load. Steps 6 and 7 take $O(1)$ time. Hence, the time complexity of balancing disk I/O resources is $O(2+2n)$ (see Steps 7-12). Similarly, the time complexities of balancing memory and CPU resources are both $O(2+2n)$. Since there are m tasks in the parallel application, the time complexity of the IOLB algorithm is $O(2+2n)O(m) = O(2(1+n)m)$. The values of n and m in most cases are much larger than 2 and, therefore, the time complexity becomes $O(nm)$.

4. An Analytical Comparison

In this section, we first prove important properties of the IOLB algorithms (see Lemmas 1-2, Theorems 2-6). Next, we qualitatively compare IOLB with two existing load-balancing algorithms (see Theorems 7 and 9).

4.1 Properties

Theorem 2. Let c_j, λ_j, and $t_{j,IO}$ be the execution time, I/O arrival rate of task τ_j, and the I/O processing time of each disk request. If the value of $c_j \cdot \lambda_j \cdot t_{j,IO}$ is zero, then the allocation of τ_j has no impact on balancing disk I/O resources.

Proof. Before the allocation of τ_j, the amount of imbalance with respect to disk I/O resources can be measured by $\sum_{i=1}^{n} \left| L_i^{IO} - \overline{L}_i^{IO} \right|$, where $\overline{L}_i^{IO} = \frac{1}{n} \sum_{i=1}^{n} L_i^{IO}$. Without loss of generality, we assume τ_j is allocated to N_k, and prior to the arrival of τ_j the I/O load of N_k is $L_i^{IO} = \sum_{\tau_j \in N_i} \left(r_j \cdot \lambda_j \cdot t_{j,IO} \right)$ (see Eq. 5). The disk I/O load index after dispatching task τ_j to node N_k becomes

$L_i'^{IO} = c_j \cdot \lambda_j \cdot t_{j,IO} + \sum_{\tau_j \in N_i} \left(r_j \cdot \lambda_j \cdot t_{j,IO} \right)$. Since the value of $c_j \cdot \lambda_j \cdot t_{j,IO}$ is zero, we have $L_i'^{IO} = c_j \cdot \lambda_j \cdot t_{j,IO} + \sum_{\tau_j \in N_i} \left(r_j \cdot \lambda_j \cdot t_{j,IO} \right) = \sum_{\tau_j \in N_i} \left(r_j \cdot \lambda_j \cdot t_{j,IO} \right) = L_i^{IO}$. Thus, the values of L_i^{IO} and $L_i'^{IO}$ are identical, meaning that the allocation of τ_j has no impact on balancing disk I/O resources.

Corollary 1. If the I/O arrival rate λ_j of task τ_j is zero, then the allocation of τ_j has no impact on balancing disk I/O resources.

Proof. The value of $c_j \cdot \lambda_j \cdot t_{j,IO}$ becomes zero if the I/O arrival rate λ_j of task τ_j is zero. Then, the proof is immediate from Theorem 1.

Theorem 3. Suppose there is a task τ_j (initially submitted to node N_i) to be allocated in a cluster; the disk I/O processing time of τ_j is $c_j \cdot \lambda_j \cdot t_{j,IO}$. Then $c_j \cdot \lambda_j \cdot t_{j,IO} > 0$ is a necessary condition for allocating task τ_j in a way to balance load with respect to disk I/O.

Proof. To prove the correctness of Theorem 3, we have to show that allocating τ_j in a way to balance load in disk I/O that $c_j \cdot \lambda_j \cdot t_{j,IO}$ is larger than 0. This can be proved by contradiction and, hence, let us assume that t_j^{page} equals to 0. Since $c_j \cdot \lambda_j \cdot t_{j,IO} = 0$, Theorem 2 shows that the allocation of τ_j has no impact on balancing disk I/O resources, meaning that there is no way of allocating τ_j such that the imbalance load in disk I/O is alleviated. Hence, we obtain the contradiction that completes the proof.

Theorem 4. Let t_j^{page} and $L_{j,mem}$ be the page fault processing time and memory requirement of task τ_j. Let L_i^{mem} denote the accumulation of the memory space allocated to tasks running on the

node N_i. Thus, we have $L_i^{mem} = \sum_{j \in N_i} L_{j,mem}$. The page fault processing time t_j^{page} is computed as,

$$t_j^{page} = \begin{cases} 0, & \text{if } L_i^{mem} \le M_i, \\ p_i \cdot \dfrac{\sum_{j \in N_i} L_{j,mem}}{M_i} \cdot r_j \cdot \left(t_{seek} + t_{rot} + \dfrac{d_{page}}{B_{disk}} \right), & \text{otherwise,} \end{cases} \tag{11}$$

where p_i is the page fault rate, M_i is the total memory space available on node N_i, r_j is the expected remaining CPU lifetime, t_{seek} and t_{rot} are the seek time and rotational latency, d_{page} is the page size, and $\frac{d_{page}}{B_{disk}}$ is the data transfer time.

Proof. First, we have to prove that t_j^{page} is zero if $L_i^{mem} \le M_i$. When the total available memory space M_i can meet the memory demands of tasks running on the *i*th node (i.e., $L_i^{mem} < M_i$), no page fault occurs in the node. In this case τ_j exhibits no page fault behavior, and the page fault processing time t_j^{page} is zero.

Second, let us prove that $t_j^{page} = p_i \cdot \dfrac{\sum_{j \in N_i} L_{j,mem}}{M_i} \cdot r_j \cdot \left(t_{seek} + t_{rot} + \dfrac{d_{page}}{B_{disk}} \right)$ if $L_i^{mem} > M_i$. If L_i^{mem} is larger than M_i, then the node encounters page faults. The number of page faults π_j is proportional to (1) the page fault rate p_i, (2) the accumulated memory space L_i^{mem} allocated to all the running tasks on N_i, and (3) the expected remaining CPU lifetime r_j. Furthermore, π_j is inversely proportional to the total available memory space M_i. Therefore, the number of page faults π_j of task τ_j can be written as $\pi_j = p_j \cdot \dfrac{L_i^{mem}}{M_i} \cdot r_j$. The I/O processing time μ of each page fault is the summation of the seek time t_{seek}, the rotational latency t_{rot}, and the data transfer time

$\frac{d_{page}}{B_{disk}}$. Thus, the I/O processing time μ is expressed by $\mu = t_{seek} + t_{rot} + \frac{d_{page}}{B_{disk}}$. Hence, the page fault processing time t_j^{page} is written as follows if $L_i^{mem} > M_i$

$$t_j^{page} = \pi_j \cdot \mu = p_j \cdot \frac{L_i^{mem}}{M_i} \cdot r_j \cdot \mu$$
$$= p_i \cdot \frac{\sum_{j \in N_i} L_{j,mem}}{M_i} \cdot r_j \cdot \left(t_{seek} + t_{rot} + \frac{d_{page}}{B_{disk}} \right),$$

which completes the proof of the theorem 3.

Theorem 5. Let τ_j denote a task to be allocated in a cluster. For each node N_i in a cluster, if the total available memory space M_i is able to meet the memory demands of τ_j and tasks running on N_i (i.e., $L_{j,mem} + L_i^{mem} \le M_i$), then the allocation of task τ_j has no impact on balancing memory resources.

Proof. It is proved that $L_i^{mem} \le M_i$, since we have $L_{j,mem} + L_i^{mem} \le M_i$ for each node N_i in the cluster. In light of Theorem 3, we can prove that before the allocation of τ_j, it is true that $\forall \tau_k \in N_i : t_k^{page} = 0$. Hence, we have $\forall 1 \le i \le n : L_i^{page} = \sum_{\tau_k \in N_i} t_k^{page} = 0$. The amount of imbalance with respect to memory resources can be measured by $\sum_{i=1}^{n} \left| L_i^{page} - \overline{L}_i^{page} \right|$, where $\overline{L}_i^{page} = \frac{1}{n} \sum_{i=1}^{n} L_i^{page}$. Prior to the allocation of τ_j, we have $\sum_{i=1}^{n} \left| L_i^{page} - \overline{L}_i^{page} \right| = 0$. Similarly, after the allocation of τ_j, we have $\forall 1 \le i \le n : L_i^{page} = \sum_{\tau_k \in N_i} t_k^{page} = 0$, because $L_{j,mem} + L_i^{mem} \le M_i$. Consequently, the value of $\sum_{i=1}^{n} \left| L_i^{page} - \overline{L}_i^{page} \right|$ is still zero after the allocation of τ_j, meaning that

the allocation of τ_j has no impact on balancing memory resources.

Before proceed to the proof of a necessary condition for allocating tasks such that the global usage of memory resources in a cluster is improved, we first prove the following two lemmas.

Lemma 1. Suppose there is a task τ_j (initially submitted to node N_i) to be allocated in a cluster; the page fault processing time of τ_j on N_i is t_j^{page}. If the page fault processing time t_j^{page} of τ_j on N_i equals to zero, then the page fault processing times of all tasks running on N_i equal to zero. More formally, we have $t_j^{page} = 0 \rightarrow \forall \tau_k \in N_i : t_k^{page} = 0$.

Proof. Let L_i^{mem} be the accumulative memory space allocated to tasks running on N_i prior to the allocation of τ_j. Since t_j^{page} equals to zero, we show that the sum of $L_{j,mem}$ and L_i^{mem} is smaller than the total available memory space in N_i (i.e., $L_{j,mem} + L_i^{mem} \leq M_i$). Hence, we have $L_i^{mem} \leq M_i$, meaning that the page fault processing time of each task running on N_i is zero (i.e., $\forall \tau_k \in N_i : t_k^{page} = 0$). Hence, the proof.

Lemma 2. Suppose there is a task τ_j (initially submitted to node N_i) to be allocated in a cluster; the page fault processing time of τ_j on N_i is t_j^{page}. If the page fault processing time t_j^{page} of τ_j on N_i equals to zero, then the memory load index L_i^{page} of node *i* equals to zero (i.e., $L_i^{page} = 0$).

Proof. The memory load index L_i^{page} is measured by $L_i^{page} = \sum_{\tau_k \in N_i} t_k^{page}$. As per Lemma 1, it is proved that the page fault processing time of all tasks running on N_i are zero (i.e., $\forall \tau_k \in N_i : t_k^{page} = 0$). Therefore, we have $L_i^{page} = \sum_{\tau_k \in N_i} t_k^{page} = 0$.

Theorem 6. Suppose there is a task τ_j (initially submitted to node N_i) to be allocated in a cluster; the page fault processing time of τ_j on N_i is t_j^{page}. Then $t_j^{page} > 0$ is a necessary condition for allocating task τ_j in a way to balance load in terms of memory resources.

Proof. The proof of Theorem 5 is immediate from Lemma 2. To prove the correctness of Theorem 5, we have to show that allocating τ_j in a way to balance load in memory resource implies that t_j^{page} is larger than 0. This can be proved by contradiction and, hence, let us assume that t_j^{page} equals to 0. Since $t_j^{page} = 0$, Lemma 2 shows that the load index with respect to memory resource is 0, implying that there is no way of allocating τ_j such that load in memory resource is balanced. Hence, we obtain the contradiction that completes the proof.

4.2 A qualitative comparison

Now we qualitatively compare the IOLB algorithm with two existing scheduling approaches: the CPU-based load-balancing algorithm (hereinafter referred to as CLB) [12] and the memory-based load-balancing algorithm (hereinafter referred to as MLB) [32]. The CLB load-balancing policy strives to improve global usages of CPU resources by balancing CPU load across all nodes in a cluster. The MLB load-balancing policy is conducive to balancing workload with respect to memory resources in a cluster when the cluster experiences a large number of page faults due to insufficient memory space. Both CLB and MLB are respectively concerned with effective usages of global CPU and memory resources in clusters without addressing the issue of balancing disk I/O resources. Consequently, the existing load-balancing approaches become inadequate for mixed workloads with CPU-, memory-, and I/O-intensive applications.

Theorem 7. Suppose there is a task τ_j (initially submitted to node *i*) to be allocated in a cluster;

node *k* has the lightest load in disk I/O (i.e., $L_k^{IO} = \min_{a=1}^{n}\left(L_a^{IO}\right)$). If the following five conditions are satisfied, then the IOLB algorithm outperforms the CLB and MLB algorithms.

(1) $c_j \cdot \lambda_j \cdot t_{j,IO} > 0$, (2) $LIF_{IO} = \max\left(LIF_{CPU}, LIF_{page}, LIF_{IO}\right)$, (3) $L_i^{IO} = \max_{a=1}^{n}\left(L_a^{IO}\right)$, (4) $c_j \cdot \lambda_j \cdot t_{j,IO} < \left(c_j \cdot \lambda_j \cdot t_{j,IO} + L_i^{IO} - L_k^{IO}\right)/2$, and (5) $R_j^i > R_j^k + e$.

Proof. First, theorem 3 shows that $c_j \cdot \lambda_j \cdot t_{j,IO} > 0$ is a necessary condition for balancing disk I/O load. Second, $LIF_{IO} = \max\left(LIF_{CPU}, LIF_{page}, LIF_{IO}\right)$ indicates that balancing disk I/O load can achieve more performance improvement than balancing memory or CPU resources. Third, $L_i^{IO} = \max_{a=1}^{n}\left(L_a^{IO}\right)$ means that the local node is overly loaded in terms of disk I/O. Since both the CLB and MLB algorithms do not take disk I/O load into account, leaving disk I/O resources severely imbalanced under conditions (1), (2), and (3). the suffer significant performance drop under I/O-intensive workload due to the imbalance of I/O load. Furthermore, condition (4) guarantees that allocating task τ_j to node k with the lightest disk I/O load can efficiently alleviate the imbalance problem, whereas condition (5) ensures that the expected response time of τ_j on the candidate remote node *k* is less than the response time of τ_j on the local node *i*. Consequently, if the above conditions hold, then the IOLB algorithm outperforms the CLB and MLB algorithms.

Theorem 8. If each task τ_j running on a cluster imposes no load on disk resources (i.e., $c_j \cdot \lambda_j \cdot t_{j,IO} = 0$), then the IOLB algorithm degrades to the MLB algorithm.

Proof. If the value of $c_j \cdot \lambda_j \cdot t_{j,IO}$ for each task τ_j is 0, the first condition in Step 3 in the IOLB

algorithm does not hold, then Steps 4-8 used to balance disk I/O load across disks in the cluster are not performed by IOLB. In this case the IOLB algorithm strives to balance load with respect to memory resources if the memory load exceeds the amount of available memory space. This means that the behavior of IOLB is the same as that of MLB if the value of $c_j \cdot \lambda_j \cdot t_{j,IO}$ for each task τ_j is 0. Hence, in case that each task τ_j running on a cluster imposes no load on disk resources (i.e., $c_j \cdot \lambda_j \cdot t_{j,IO} = 0$), the performance of IOLB and MLB are identical. This completes the proof of Theorem 5.

Theorem 9. If each task τ_j running on a cluster imposes no load on disk and memory resources (i.e., $c_j \cdot \lambda_j \cdot t_{j,IO} = 0$ and $t_j^{page} = 0$), then the IOLB algorithm degrades to the CLB algorithm.

Proof. First, if the value of $c_j \cdot \lambda_j \cdot t_{j,IO}$ for each task τ_j is 0, the first condition in Step 3 in the IOLB algorithm does not hold, then Steps 4-8 used to balance disk I/O load across disks in the cluster are not performed by IOLB. Second, if the page fault processing time t_j^{page} of τ_j is zero, Steps 9-14 are not invoked to improve the global usage of memory resources in the cluster. In this case the IOLB algorithm makes an effort to evenly distribute the CPU load. Therefore, the behavior of IOLB is the same as that of CLB. This completes the proof of Theorem 8.

5. Experimental Results

In this section we quantitatively compare the IOLB algorithm with the two existing load-balancing approaches (see Section 4.2): CLB and MLB. We conducted trace-driven simulations using a simulated cluster with 32 nodes providing a time-sharing environment. A similar simulation environment was delineated in [23]. The traces used in our experiments were

extrapolated from those traces reported in [12][32]. The number of tasks of each parallel job in traces is selected randomly with a uniform distribution between 2 to 32. CPU times and memory demands of jobs are specified in the traces. We assume that the disk request arrival rate of each job is generated randomly with a uniform distribution. This assumption is reasonable because the mean disk request arrival rate can be manipulated and examined as a system parameter. In our empirical studies, we varied the mean disk request arrival rate from 0.8 to 1.25 No./ms. The data size of disk requests in each job is selected randomly based on a Gamma distribution with the mean size of 256Kbyte. The performance metric by which we evaluate system performance is mean slowdown of all the jobs in a trace.

Table 1. Mean slowdowns of parallel jobs under the CLB, MLB, and IOLB schemes.

λ	0.80	0.85	0.90	0.95	1.0	1.05	1.10	1.15	1.20	1.25
CLB	62	74	86	99	113	132	149	168	187	210
MLB	62	75	87	100	114	132	149	171	192	218
IOLB	47	61	73	80	94	101	117	130	151	172

The number of nodes in the simulated cluster is 32; number of tasks in each job is selected randomly with a uniform distribution between 2 to 32; disk request arrival rate is varied from 0.8 to 1.25 No./ms; disk request size is chosen randomly with a Gamma distribution with mean size of 256Kbyte; the page fault rate is set to 0.5 No./ms; the page size is 4KByte.

Table 1 shows impacts of the disk request arrival rate on the mean slowdowns of submitted parallel jobs under the three evaluated load-balancing schemes. It is intuitive that regardless of the load-balancing approaches, the slowdowns of the parallel jobs increase with the I/O load goes up. This is because high disk request arrival rates leads to high disk I/O loads, which in turn cause long I/O processing time and long waiting time on disk I/O resources. More importantly, the experimental results reveal that the IOLB algorithm is superior to the CLB and MLB load-balancing schemes. These results indicate that the existing load-balancing policies are inadequate for I/O-intensive workloads. The performance improvements can be attributed to the fact that

CLB and MLB do not address the issue of balancing disk I/O load under I/O-intensive workload conditions.

Table 2. Mean slowdowns of parallel jobs under the CLB, MLB, and IOLB schemes.

page fault rate	2.6	2.7	2.8	2.9	3	3.1	3.2	3.3	3.4	3.5
CLB	22	23	23	24	24	24	24	27	27	28
MLB	8.5	8.6	8.8	10.8	10.9	11.1	13	13.5	14.2	14.3
IOLB	8.5	8.5	8.7	10.8	10.9	11	13.1	13.5	14.2	14

Disk request arrival rate is fixed to 0.01 No./ms; the page fault rate varies from 2.6 to 3.5 No./ms; the page size is 4KByte.

Recall that Theorem 8 proves that if all parallel jobs running on a cluster impose no load on disk I/O resources, then the performance of IOLB is identical to that of MLB. Now we experimentally validate the correctness of Theorem 8 using memory-intensive workloads. To achieve this goal, we varied the page fault rate from 2.6 to 3.5 No./ms with an increment of 0.1 No./ms. The disk request arrival rate is fixed to a low value - 0.01 No./ms.

Table 2 shows performance impacts of the page fault rate on the mean slowdowns of parallel jobs running on the simulated cluster. For all the three examined load-balancing schemes, results in Table 2 indicate that MLB and IOLB outperform CLB under memory-intensive workloads. These performance improvements are possible because MLB and IOLB are concerned with the global memory usage in the cluster by balancing memory resources. The improved memory usage in turn significantly reduces time spent in page fault processing. This trend becomes more pronounced when the page fault rate is increased.

6. Conclusions

Most existing load balancing approaches are inadequate for I/O-intensive workloads due to imbalance of I/O loads and low usage of global disk resources. To address this issue, in this paper we proposed a new load-balancing algorithm (referred to as IOLB) for clusters. The

proposed load-balancing algorithm aims to achieve the effective usage of global disk resources in a cluster. This can, in turn, minimize the average slowdown of all parallel jobs running on a cluster and reduce the average response time of the jobs. In addition to balancing loads in disk resources under I/O-intensive workloads, the IOLB algorithm improves the CPU and memory utilization under CPU- and memory-intensive workload conditions. Consequently, IOLB is able to maintain the same level of performance as two existing CPU- and memory-aware load-balancing schemes. We conducted trace-driven simulations where traces are composed of parallel applications with a wide variety of I/O demands. Empirical results demonstratively show that compared with the two existing load-balancing approaches, the IOLB algorithm significantly improves the resource utilization of a cluster under I/O-intensive workloads. When the workloads become CPU-intensive or memory-intensive in nature, IOLB gracefully degrades towards the existing load-balancing schemes.

Future studies can be performed in the following directions. First, we will evaluate the performance of IOLB on a large-scale cluster with more than 1000 nodes. Second, in this study we assume that network communication cost is negligible. Therefore, we intend to further extend our load-balancing algorithm in a way to balance load in network resources. Third, a heterogeneity-aware load-balancing algorithm will be investigated to deal with parallel jobs running on heterogeneous clusters, in which nodes have various processing capabilities.

Acknowledgements

The work reported in this paper was supported by the US National Science Foundation under Grants No. CCF-0742187 and No. CNS-0713895, Auburn University under a startup grant, and the Intel Corporation under Grant No. 2005-04-070.

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] A. Acharya et al., "Tuning the Performance of I/O-intensive Parallel Applications," *Proc. 4th IOPADS*, pp. 15–27, Philadelphia, PA, May 1996.

[3] J.M. Bahi, S. Contassot-Vivier, R. Couturier, "Dynamic load balancing and efficient load estimators for asynchronous iterative algorithms," *IEEE Trans. Parallel and Distributed Systems*, vol. 16, no. 4, pp. 289-299, April 2005.

[4] J. Basney and M. Livny, "Managing Network Resources in Condor", *Proc. IEEE Symp. High Performance Distributed Computing*, pp 298-299, Aug. 2000.

[5] B. Bode, D. M. Halstead, R. Kendall, and Z. Lei, "The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters," *Proc. 4th Annual Linux Showcase & Conference*, 2000.

[6] C. Chang, B. Moon, A. Acharya, C. Shock, A. Sussman, and J. Saltz, "Titan: a High-Performance Remote-Sensing Database," *Proc. 13th Int'l Conf. Data Eng.*, Apr. 1997.

[7] J. Choi, S.H. Noh, S.L. Min, and Y. Cho, "Design, Implementation, and Performance Evaluation of a Detection-Based Adaptive Block Replacement Scheme," *IEEE Trans. Computers*, vol. 51, no. 7, July 2002.

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

[9] A.C. Dusseau, R.H.Arpaci, and D.E. Culler, "Effective Distributed Scheduling of Parallel Workload," *Proc. ACM SigMetrics Conf. Measuring and Modeling of Computer Systems*, pp.25-36, May 1996.

[10] S.M. Figueira and F. Berman, "A Slowdown Model for Applications Executing on Time-Shared Clusters of Workstations," *IEEE Trans. Parallel and Distributed Systems*, vol. 12, no. 6, pp. 653-670, June 2001.

[11] B. Forney, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Storage-Aware Caching: Revisiting Caching for Heterogeneous Storage Systems," *Proc. 1st File and Storage Technology*, 2001.

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

[13] B. Hendrickson and D. Womble, "The torus-wrap mapping for dense matrix calculations on massively parallel computers," *SIAM J. Sci. Comput.*, vol. 15, no. 5, Sept. 1994.

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

[15] R. Lavi and A. Barak, "The Home Model and Competitive Algorithm for Load Balancing in a Computing Cluster," *Proc. Int'l Conf. Distributed Computing Systems*, Apr. 2001.

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

[17] J.-M. Liu X.-L. Jin, and Y.-S. Wang, "Agent-based load balancing on homogeneous minigrids: macroscopic modeling and characterization," *IEEE Trans. Parallel and Distributed Systems*, vol. 16, no. 7, pp. 586-598, July 2005.

[18] X. Ma, M. Winslett, J. Lee, and S. Yu, "Faster Collective Output through Active Buffering," *Proc. Int'l Symp. Parallel and Distributed Processing*, 2002.

[19] A. Mueller, "Fast sequential and parallel algorithms for association rule mining: A comparison," *Technical Report CS-TR-3515*, University of Maryland, College Park, Aug. 1995.

[20] B.K. Pasquale and G.C. Polyzos. "Dynamic I/O Characterization of I/O Intensive Scientific applications," *Proc. Int'l Conf. Supercomputing*, pp. 660-669, 1994.

[21] X. Qin, H. Jiang, Y. Zhu, and D. Swanson, "Dynamic Load balancing for I/O- and Memory-Intensive workload in Clusters using a Feedback Control Mechanism," *Proc. 9th Int'l Euro-Par Conf. Parallel Processing,* Klagenfurt, Austria, Aug. 2003.

[22] X. Qin, H. Jiang, Y. Zhu, and D.R. Swanson, "Improving the Performance of I/O-Intensive Applications on Clusters of Workstations," *Cluster Computing: The Journal of Networks, Software Tools and Applications*, vol. 8, no. 4, Oct. 2005.

[23] X. Qin, H. Jiang, Y. Zhu, and D. R. Swanson, "Towards Load Balancing Support for I/O-Intensive Parallel Jobs in a Cluster of Workstations," *Proc. 5th IEEE Int'l Conf. Cluster Computing*, pp.100-107, Dec. 2003.

[24] X. Qin and H. Jiang, "Improving Effective Bandwidth of Networks on Clusters using Load Balancing for Communication-Intensive Applications," *Proc. 24th IEEE Int'l Performance, Computing, and Communications Conf.*, pp.27-34, April 2005.

[25] J.O. Roads, et al., "A Preliminary Description of the Western U.S. Climatology", *Proc. 9th Annual Pacific Climate Workshop*, Sept. 1992.

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

[27] M. Uysal, A. Acharya, and J. Saltz. "Requirements of I/O Systems for Parallel Machines: An Application-driven Study," *Technical Report, CS-TR-3802*, University of Maryland, College Park, May 1997.

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

[29] B. Wilkinson, and M. Allen, "Parallel Programming, Techniques and Applications Using Networked Workstations and Parallel Computers", *Prentice-Hall, Inc.*, 1999.

[30] S. Wu and U. Manber, "agrep - A fast approximate pattern-matching tool," *USENIX Conference Proc.*, pp. 153–162, San Francisco, CA, Winter 1992.

[31] C.-Z Xu, B. Monien, and R. Luling, "An Analytical Comparison of Nearest Neighbor Algorithms for Load Balancing in Parallel Computers," *Proc. Int'l Symp. Parallel Processing*, 1995.

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

[33] Y. Zhang, A. Yang, A. Sivasubramaniam, and J. Moreira, "Gang Scheduling Extensions for I/O Intensive Workloads," *Proc. 9th Workshop Job Scheduling Strategies for Parallel Processing*, 2003.