Communication-Aware Load Balancing for Parallel Applications on Clusters

Xiao Qin, *Senior Member*, *IEEE*, Hong Jiang, *Member*, *IEEE*, Adam Manzanares, *Student Member*, *IEEE*, Xiaojun Ruan, *Student Member*, *IEEE*, and Shu Yin, *Student Member*, *IEEE*

Abstract—Cluster computing has emerged as a primary and cost-effective platform for running parallel applications, including communication-intensive applications that transfer a large amount of data among the nodes of a cluster via the interconnection network. Conventional load balancers have proven effective in increasing the utilization of CPU, memory, and disk I/O resources in a cluster. However, most of the existing load-balancing schemes ignore network resources, leaving an opportunity to improve the effective bandwidth of networks on clusters running parallel applications. For this reason, we propose a communication-aware load-balancing technique that is capable of improving the performance of communication-intensive applications by increasing the effective utilization of networks in cluster environments. To facilitate the proposed load-balancing scheme, we introduce a behavior model for parallel applications with large requirements of network, CPU, memory, and disk I/O resources. Our load-balancing scheme can make full use of this model to quickly and accurately determine the load induced by a variety of parallel applications. Simulation results generated from a diverse set of both synthetic bulk synchronous and real parallel applications on a cluster show that our scheme significantly improves the performance, in terms of slowdown and turn-around time, over existing schemes by up to 206 percent (with an average of 74 percent) and 235 percent (with an average of 82 percent), respectively.

Index Terms—Cluster, communication-aware computing, parallel computing, load balancing.

1 Introduction

Significant cost advantages, combined with rapid advances in middleware and interconnect technologies, have made clusters a primary and fast growing platform for high-performance scientific computing. Scheduling [6] and load balancing [18], [19] are two key techniques used to improve the performance of clusters for scientific applications by fully utilizing machines with idle or underutilized resources. A number of distributed load-balancing schemes for clusters have been developed, primarily considering a variety of resources, including the CPU [13], memory [1], disk I/O [18], or a combination of CPU and memory resources [31]. These approaches have proven effective in increasing the utilization of resources in clusters, assuming that network interconnects are not potential bottlenecks in clusters. However, a recent study demonstrated that the need to move data from one component to another in clusters is likely to result in a major performance bottleneck [10], [23], indicating that data movement through the interconnection of a cluster can become a primary bottleneck. Thus, if the opportunity for improving effective bandwidth of networks is fully exploited, the performance of parallel jobs on clusters could be enhanced.

A large number of scientific applications have been implemented for execution on clusters and more are underway. Many scientific applications are inherently computationally and communicationally intensive [8]. Examples of such applications include 3D perspective rendering, molecular dynamics simulation, quantum chemical reaction dynamics simulations, and 2D fluid flow using the vortex method, to name just a few [8]. The above bottleneck becomes more severe if the resources of a cluster are time/space-shared among multiple scientific applications and the communication load is not evenly distributed among the cluster nodes. Furthermore, the performance gap between the effective speed of CPU and network resources continues to grow at a faster pace, which raises a need for increasing the utilization of networks on clusters using various techniques.

The main motivation for our study is to improve the efficiency and usability of networks in cluster computing environments with high communication demands. The effort to improve the utilization of networks can be classified into hardware and software approaches. In this paper, we focus on an approach designed at the software level. In particular, we develop a communication-aware load-balancing scheme (referred to as COM-aware) to achieve high effective bandwidth communication without requiring any additional hardware. Our approach can substantially improve the performance of parallel applications running on a large-scale, time/space-shared cluster, where a number of parallel jobs share various resources. COM-aware load balancing enables a cluster to utilize most idle, or underutilized, network resources while keeping the usage of other types of resources reasonably high. We propose a behavioral

- *X. Qin, A. Manzanares, X. Ruan, and S. Yin are with the Department of Computer Science and Software Engineering, Shelby Center for Engineering Technology, Samuel Ginn College of Engineering, Auburn University, AL 36849-5347.*
  E-mail: {xqin, acm0008, xzr0001, szy0001}@auburn.edu.
- *H. Jiang is with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, 103 Schorr Center, 1101 T Street, Lincoln, NE 68588-0150. E-mail: xietao@nmt.edu.*

Manuscript received 31 July 2008; revised 20 Mar. 2008; accepted 21 May 2009; published online 23 July 2009.
Recommended for acceptance by F. Lombardi.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2008-07-0387.
Digital Object Identifier no. 10.1109/TC.2009.108.

0018-9340/10/$26.00 © 2010 IEEE Published by the IEEE Computer Society

model for parallel applications to capture the typical characteristics of various requirements of CPU, memory, network, and disk I/O resources.

Typical load-balancing approaches that only focus on maximizing the utilization of one type of resource are not sufficient for a wide range of applications. Increasing evidence shows that high-performance computing requires clusters to be capable of executing multiple types of applications submitted by users [20], [26], [31] simultaneously. For this reason, the COM-aware scheme is designed in a way that delivers high performance under a large variety of workload scenarios, in order to provide one possible solution to the problem.

The rest of the paper is organized as follows: Section 2 presents a summary of related work. Section 3 introduces an application behavioral model and a system model that aim at capturing the resource requirements for applications and the simulated cluster environment. Section 4 presents our communication-aware load-balancing scheme. Section 5 introduces the simulation model and the methods used for gathering the data for performance evaluation. Section 6 summarizes the performance evaluation of the proposed scheme by simulating a cluster running both synthetic bulk synchronous and real-world communication-intensive parallel applications. Finally, Section 7 summarizes the main contributions of this paper and comments on future research.

2 Related Work

In general, load-balancing techniques fall into to categories: centralized load balancing and distributed load balancing. Centralized schemes typically require a head node that is responsible for handling the load distribution. As the cluster size scales up, the head node quickly becomes a bottleneck, causing a significant performance degradation. To solve this scalability problem, the workload of the load balancer can be delegated to multiple nodes in a cluster, hence the notion of distributed dynamic load balancing. In addition, a centralized scheme has the potential problem of poor reliability because permanent failures of the central load balancer can result in a complete failure of the load-balancing mechanism. Therefore, the focus of this paper is on designing a decentralized communication-aware load balancer for time-/space-shared (i.e., nondedicated) clusters.

There is a large body of established research focusing on the issue of distributed load balancing for CPU and memory resources. For example, Harchol-Balter and Downey [13] developed a CPU-based preemptive migration policy that was shown to be more effective than nonpreemptive migration policies. Zhang et al. [31] studied load-sharing policies that consider both CPU and memory services among the nodes. Although these schemes can effectively utilize memory and/or CPU resources at each node in the system, none of them has considered the effective usage of I/O or network resources.

In our previous paper, we proposed an I/O-aware load-balancing scheme to meet the needs of a cluster system with a variety of workload conditions [18]. This approach is able to balance implicit disk I/O load as well as that of explicit disk I/O. This is due to the fact that when the memory space is unable to meet the memory requirements of the jobs running on a node, a large number of page faults will exist, which in turn, will lead to a heavy implicit I/O load. Additionally, we have extended the above I/O-aware load-balancing strategy from three different perspectives. First, we have incorporated preemptive job migration to further improve the system performance over nonpreemptive schemes [19]. Second, we have developed an approach for hiding the heterogeneity of resources, especially that of I/O resources [21]. Third, we have devised two simple yet effective load-balancing schemes for parallel I/O-intensive jobs running on clusters [20]. All the above approaches are effective under workloads without high communication intensity and balancing the communication load is not considered.

A communication-sensitive load balancer has been proposed by Cruz and Park [7]. The balancer uses runtime communication patterns between processes to balance the load. Orduña et al. have proposed a criterion to measure the quality of network resource allocation to parallel applications [17]. Based exclusively on this criterion, they have developed a scheduling algorithm to fully utilize the available network bandwidth. Our proposed scheme is different from their work in that our scheme attempts to simultaneously balance two different kinds of I/O load, namely, communication and disk I/O.

Many researchers have shown that message-passing architectures and programming interfaces are useful tools to develop parallel applications. Brightwell et al. have implemented the Portals message-passing interface on a commodity PC Linux cluster [3]. For higher level message-passing layers, Portals can reduce the overhead for receiving messages due to the availability of programmable NICs with significant processing power. Buntinas et al. have designed and implemented an application-bypass broadcast operation, which is independent of the application running at a process to make progress [4]. The approach presented in this paper can be considered complementary to the existing message-passing techniques in the sense that an additional performance improvement can be achieved by combining our communication-aware load-balancing technique with the Portals MPI and the application-bypass broadcast.

3 System Models and Assumptions

Since the communication and I/O demands of applications may not be known in advance, in this section, we introduce an application model, which aims at capturing the typical characteristics of the communication, disk I/O, and CPU activity within a parallel application. The parameters of the model for a parallel application can be obtained by repeatedly executing the application offline for multiple inputs. Alternatively, a light weight and online profiler can be used to monitor the application behavior, thereby updating the model. Note that the model is used in our trace-driven simulations.

Because the *Message Passing Interface* (MPI) specification simplifies many complex issues for scientific application development, application designers and developers have adopted the MPI programming model [30], [28] as the de facto standard for parallel programming on clusters. Consequently, the development of numerous scientific applications for clusters has been largely based on MPI [12].

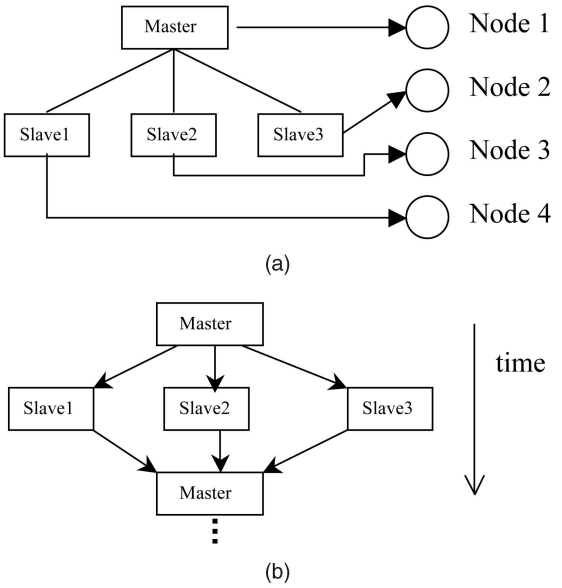


Fig. 1. SPMD application model. A master-Slave communication topology. (a) Four processes are allocated to four nodes. (b) The master periodically communicates to and from the slaves.

In the MPI-based application model, large data sets are decomposed across a group of nodes in a cluster. Processes (the terms process and task are used interchangeably throughout this paper) in a parallel application communicate with one another through the message-passing interface (Fig. 1). In this study, we focus on a bulk-synchronous style of communication, which has been used in the popular Bulk-Synchronous Parallel model [9], [27], [22] and conforms with the MPI style of interprocess communication patterns. This model is a good fit for a variety of bulk-synchronous parallel applications, especially MPI-based parallel programs, in which a master creates a group of slave processes across a set of nodes in a cluster (Fig. 1a). The slave processes concurrently compute during a computation phase, and then, processes will be synchronized at a barrier so that messages can be exchanged among these processes during the communication phase (Fig. 1b).

To design a load-balancing scheme that improves the effective usage of network resources in a cluster, we have to find a way to quantitatively measure the communication requirements of parallel applications. Based on the communication requirements of all the applications running on the cluster, we can estimate the communication load of each node.

In what follows, we introduce an application behavioral model, which captures the load of CPU, network, and disk I/O resources generated by a parallel application. This model is important because the load-balancing scheme proposed in Section 4 makes full use of the model to quickly calculate the load induced by parallel applications running on the system. Our model is reasonably general in the sense that it is applicable for both communication and I/O-intensive parallel applications. Let N be the number of phases for a process (slave task) of a parallel job running on a node, and T_i be the execution time of the ith phase. T_i can be obtained by the following equation, where T^i_{CPU}, T^i_{COM},

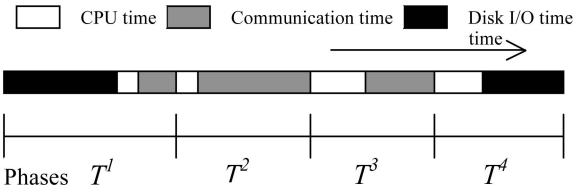


Fig. 2. Example for the application behavior model. Number of phases is $\mathrm{N} = 4$.

and T^i_{Disk} are the times spent using CPU, communication, and disk I/O resources:

$$T^i = T^i_{CPU} + T^i_{COM} + T^i_{Disk}. \tag{1}$$

Therefore, the total execution time of the process is estimated as $T = \sum_{j=1}^{N} T^i$. Let R_{CPU}, R_{COM}, and R_{Disk} be the requirements of CPU, communication, and disk I/O, and these requirements can be quantitatively measured as:

$$\begin{aligned} &R_{CPU} = \sum_{i=1}^{N} T^i_{CPU}, \quad R_{COM} = \sum_{i=1}^{N} T^i_{COM}, \quad and \\ &R_{Disk} = \sum_{i=1}^{N} T^i_{Disk}. \end{aligned} \tag{2}$$

An example of the above notation is illustrated in Fig. 2, which is for demonstrative purposes only. In this example, the process has four phases. In the first phase, the application spends a large fraction of time reading data from the disk. The second phase spends a small amount of time on computation and the rest on communication. Thus, phase two is communication-intensive. The third phase is composed of balanced CPU and communication requirements. The last phase is I/O-intensive in the sense that it spends a large fraction of time writing data to the disk.

We are now in a position to derive the execution time model for a parallel job (the terms job, application, and program are used interchangeably) running on a dedicated cluster environment. Given a parallel job with p identical processes, the execution time of the job can be calculated as:

$$T_p = s_p + \sum_{i=1}^{N} \left\{ MAX^p_{j=1}\left(T^i_{j,CPU} + T^i_{j,COM} + T^i_{j,Disk}\right)\right\}, \tag{3}$$

where $T^i_{j,CPU}$, $T^i_{j,COM}$, and $T^i_{j,Disk}$ denote the execution time of process j in the ith phase on the three prospective resources. The first term on the right-hand side of the equation represents the execution time of sequential components including synchronization delay and computation time in the master process; the second term corresponds to the execution time of parallel components. Disk IO times depend on the IO access rate, average IO data size, disk seek and rotation time, and the Disk transfer rate. Therefore, we do not assume by any means that the disk I/O times are fixed.

Let $s_{p,CPU}$, $s_{p,COM}$, and $s_{p,Disk}$ denote the sequential components execution time on CPU, communication, and disk I/O accesses, thus:

$$s_p = s_{p,CPU} + s_{p,COM} + s_{p,Disk}. \tag{4}$$

Our goal is to model a nondedicated cluster, where each job has a home node that it prefers for execution [15]. This is because either the input data of a job has been stored in the home node, or the job was created on its home node. When a parallel job is submitted to its home node, the load balancer allocates the job to a group of nodes with the least load. If the load balancer finds the local node to be heavily loaded, an eligible process will be migrated to a node with the lightest load.

A cluster in the most general form is composed of a set of nodes each containing a combination of multiple resource types, such as CPU, memory, network, and disk. Each node has a load balancer that is responsible for balancing the load of available resources. The load balancer periodically receives reasonably up-to-date global load information from the resource monitor [14], [25]. When the number of communicating nodes in the system becomes large or when the system load is heavy, the communication overhead tends to be excessive. To reduce this undesirable overhead, Shin and Chang proposed the notion of buddy sets and preferred lists [24].

The network in our model provides full connectivity in the sense that any two nodes are connected through either a physical link or a virtual link. This assumption is arguably reasonable for modern interconnection networks (e.g., Myrinet [2] and InfiniBand [29]) that are widely used in high-performance clusters. Both Myrinet and Infiniband networks provide pseudofull connectivity, allowing simultaneous transfers between any pair of nodes.

We assume that the CPU and memory burden placed by data communication is relatively low, and consequently, the CPU and memory load (due to data communication) is not considered. This assumption is reasonable since the communication can be directly handled by the network interface controllers without the local CPUs intervention or the buffer in the main memory [11].

4 Communication-Aware Load Balancing

In this section, we propose a dynamic, communication-aware load-balancing scheme for nondedicated clusters. Each node in the cluster serves multiple processes in a time-sharing fashion so that these processes can dynamically share the cluster resources. To facilitate our load-balancing technique, we need to measure the communication load imposed by these processes.

Let a parallel job formed by p processes be represented by $t_0, t_1, , t_{p-1}$, and where ni is the node to which t_i is assigned. Without loss of generality, we assume that t_0 is a master process, and $t_j (0 < j < p)$ is a slave process. Let $L_{j,p,COM}$ denote the communication load induced by t_j, which can be computed with the following formula, where $T^i_{j,COM}$ is the same as the one used in (3):

$$L_{j,p,COM} = \begin{cases} \sum_{i=1}^{N} T^i_{j,COM}, & j \neq 0, n_j \neq n_o, \\ 0, & j \neq 0, n_j = n_0, \\ \sum_{1 \leq k < p, n_k \neq n_j} \sum_{i=1}^{N} T^i_{k,COM}, & j = 0. \end{cases} \quad (5)$$

The first term on the right-hand side of the equation corresponds to the communication load of a slave process t_j when the process t_j and its master process are allocated to different nodes. The second term represents a case where the slave and its master process are running on the same node, and therefore, the slave process exhibits no communication load. Similarly, the third term on the right-hand side of the (5) measures the network traffic in and out of the master node, which is the summation of the communication load of the slave processes that are assigned to nodes other than the master node.

Intuitively, the communication load on node $i, L_{i,COM}$, can be calculated as the cumulative load of all the processes currently running on the node. Thus, $L_{i,COM}$ can be estimated as follows:

$$L_{i,COM} = \sum_{\forall\, j: n_j = i} L_{j,p,COM}. \quad (6)$$

Given a parallel job arriving at a node, the load-balancing scheme attempts to balance the communication load by allocating the jobs processes to a group of nodes with a lower utilization of network resources. Before dispatching the processes to the selected remote nodes, the following two criterions must be satisfied to avoid useless migrations:

Criterion 1. Let nodes i and j be the home node and candidate remote node for process t, the communication load discrepancy between nodes i and j is greater than t's communication load.

Criterion 1 guarantees that the load on the home node will be effectively reduced without making other nodes overloaded. We can formally express this criterion as:

$$\left(L_{i,COM} - L_{j,COM}\right) > L_{t,p,COM}. \quad (7)$$

Criterion 2. Let nodes i and j be the home node and candidate remote node for process t, then the estimated response time of t on node j is less than its local execution. Hence, we have the following inequality, where R^i_t and R^j_t are the estimated response time of t on nodes i and j, respectively. $R^{i,j}_{t,mig}$ is the migration cost for process t:

$$R^j_t < R^i_t + R^{i,j}_{t,mig}. \quad (8)$$

The migration cost $R^{i,j}_{t,mig}$ is the time interval between the initiation and the completion of ts migration, which is comprised of a fixed cost for running t at the remote node j and the process transfer time that largely depends on the process size and the available network bandwidth measured by the resource monitor.

It is a known fact that, in practice, clusters are likely to have a mixed workload with memory, I/O, and communication-intensive applications. Therefore, our communication-aware load-balancing approach is designed in such a way to achieve high performance under a wide spectrum of workload conditions by considering multiple application types. Specifically, to sustain high and consistent performance when the communication load becomes relatively low or well balanced, our scheme additionally and simultaneously considers disk I/O and CPU resources. In other words, the ultimate goal of our scheme is to balance three different resources simultaneously under a wide spectrum of workload conditions. In practice, one resource

```
if (L_i,COM = MAX(L_i,COM, L_i,CPU, L_i,Disk)) begin
if (the submitted job is communication-intensive) begin
node i makes an effort to balance the communication load;
else if (the submitted job is I/O-intensive) begin
        node i keeps disk I/O resources well balanced;
    else if (the submitted job is CPU-intensive)
balance CPU resources;
    end
end
else if (the workload is memory-intensive) begin
    Balance memory resources;
 else if (the submitted job is I/O-intensive) begin
        node i keeps disk I/O resources well balanced;
     else if (the submitted job is CPU-intensive)
balance CPU resources;
    end
end
end
```

Fig. 3. Pseudocode of the communication-aware load-balancing scheme.

is likely to become a bottleneck if the fraction of the execution time spent on this resource is substantially higher than that on the other two resources. For this reason, our scheme envisions a type of resource that exhibits higher load than that of any other resources as the first-class resource, and the scheme attempts to first balance the first-class resource. Let $L_{i,CPU}$ and $L_{i,Disk}$ be the load of node i in terms of CPU and disk I/O activities. The pseudocode of our communication-aware scheme is presented in Fig. 3.

5 Performance Evaluation

To evaluate the performance of the proposed load-balancing scheme, we have conducted extensive trace-driven simulations. This section describes our simulation model, workload conditions, and performance metrics.

A 32-node cluster was simulated under a variety of workload conditions. Before presenting the workload and performance metrics in detail, we briefly describe the simulation model first. To study dynamic load balancing, Harchol-Balter and Downey [13] have implemented a simulator of a distributed system with six nodes, where a CPU-based load-balancing policy is studied. Zhang et al. [31] extended the simulator by incorporating memory recourses. We have added new features to this simulator. First, the new communication-aware load-balancing scheme is implemented. Second, a fully connected network is simulated and the size of the simulated cluster is now variable. The simulated cluster is configured using the various parameters listed in Table 1. The parameters for CPU, memory, disk, and network interconnect are chosen in such a way that they resemble a typical cluster node.

It is well understood that the performance of a cluster system is affected by the workload submitted to the system. Therefore, designing a realistic workload plays an important role in our performance evaluation. To evaluate the performance impacts of the communication-aware load-balancing scheme, we extrapolate traces from those reported in [12], [30]. It is noted that the original traces are obtained by recording information regarding job submissions to one workstation at different time intervals [13]. In particular, the traces in our experiments consist of the arrival time, arrival node, requested memory size, actual running time, I/O access rate, average I/O data size, message arrival rate, the average message size, and the number of tasks in each parallel job. To simulate a multiuser time-sharing environment, the traces contain a collection of parallel jobs. Without loss of generality, we first consider a bulk-synchronous style of communication (See Section 3), where the time interval between two consecutive synchronization phases is determined by the message arrival rate. The experimental results reported in Section 6.5 validate our argument that the communication-aware load-balancing scheme can also be applied to communication-intensive applications with a variety of communication patterns.

For jobs that contain communication requirements, messages issued by each task are modeled as a Poisson Process with a mean message arrival rate. The message size is randomly generated according to a Gamma distribution with a mean size of 512 Kbyte, which reflects typical data characteristics for many applications, such as 3D perspective rendering [16]. The durations and memory requirements of the jobs are defined in the trace files, and the communication requirement of each job is randomly generated. Although this simplification deflates any correlations between communication requirements and other job characteristics, we still are able to manipulate the communication requirements as an input parameter and examine the impact that changing communicational intensity has on the system performance (See Sections 6.1 and 6.3). To validate the results based on synthetic parallel applications, we simulate five real scientific applications [8] which have various computation, memory, disk I/O, and communication requirements (See Section 6.5).

The goal of the proposed load-balancing scheme is to improve the performance of submitted jobs as such; we need to choose good metrics to intuitively capture the performance of jobs running on a cluster. The first performance metric considered in our experiments is the job *turn-around time* [5]. The turn-around time of a job is the time elapsed between the jobs submission and its completion. The turn-around time is a natural metric for the job performance due to its ability to reflect a users view of how long a job takes to complete.

A second important performance metric to be introduced in our study is *slowdown*. The slowdown of a parallel job running on a nondedicated cluster is a commonly used

TABLE 1
System Characteristics

Parameter	Values Assumed	Paremeter	Values Assumed
CPU Speed	1000 MIPS	Time slice of CPU time sharing	10 ms
RAM Size	1 Gbytes (Gegabytes)	Context switch time	0.1 ms
Network Bandwidth	1Gbps	Disk seek time and rotation time	8.0 ms
Page fault service time	8.1 ms (millisecond)	Disk transfer rate	40 Mbytes/Sec.

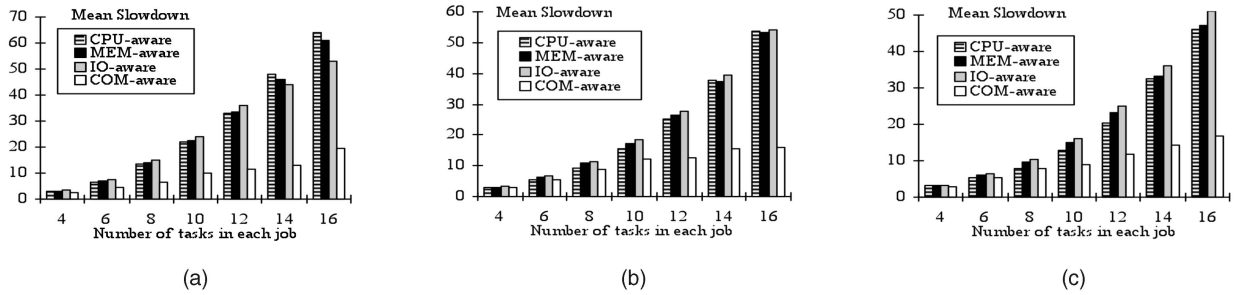


Fig. 4. Mean slowdown versus number of tasks. (a) Message arrival $\text{rate} = 0.1$ No./ms. (b) Message arrival $\text{rate} = 0.3$ No./ms. (c) Message arrival $\text{rate} = 0.5$ No./ms.

metric for measuring the performance of the job [13]. The slowdown of a job is defined by: $S_p = T'_p/T_p$, where T'_p is the job's turn-around time in a resource-shared setting and T_p is the turn-around time of running in the same system but without any resource sharing. Note that Tp can be estimated by either using (3) (See Section 3) or offline execution using multiple inputs, and T'_p can be measured at runtime.

6 Experimental Results

To empirically evaluate the performance of our communication-aware load-balancing scheme (referred to as COM-aware), we compared COM-aware against three existing load balancers that can achieve high CPU, memory, and disk I/O utilizations, respectively. We refer to the existing load-balancing policies as the CPU-aware [13], MEM-aware [31], and I/O-aware [20] load balancers.

This section presents experimental results obtained from feeding a simulated cluster of 32 nodes with various traces that reflect a wide range of workload conditions. First, the performance under communication-intensive workload situations is studied. Second, we investigate the impact of varying the network bandwidth and message size parameters have on the performance of the simulated cluster. Third, we measure the performance of the cluster under mixed workload conditions with I/O, memory, and communication-intensive applications. Finally, we simulate several real-world communication-intensive parallel applications to validate the results from the synthetic application cases.

6.1 Communication-Intensive Workloads

In the first experiment, we intend to stress the communication-intensive workload by setting the message arrival rate at values of 0.1, 0.3, and 0.5 No./ms, respectively. Both page fault rate and I/O access rate are set at a low value of 0.01 No./Sec. This workload reflects a scenario where jobs have high communication-to-computation ratios.

Figs. 4 and 5 plot the mean slowdown and turn-around time (measured in Seconds) as a function of the number of tasks in each parallel job. Each graph in Figs. 4 and 5 reports the results for communication-intensive jobs running on a cluster of 32 nodes using four different load-balancing schemes. These figures indicate that all of the load-balancing schemes experience an increase in the mean slowdown and turn-around time when the number of tasks in a job increases. This is because when the CPU, memory, and disk I/O demands are fixed and at the same time the number of tasks in each parallel job is increased, it will lead to high communication demands. This causes longer waiting times for sending and receiving data.

Importantly, we observe from Figs. 4 and 5 that the COM-aware load balancer is significantly better than the CPU-aware, MEM-aware, and IO-aware load balancers under the communication-intensive workload situation. For example, when each parallel job consists of 16 tasks and the message arrival rate is 0.5 No./ms, the COM-aware approach improves the performance, in terms of mean slowdown, over the CPU-aware, MEM-aware, and IO-aware schemes by more than 174, 182, and 206 percent, respectively. We attribute this result to the fact that the CPU-aware, MEM-aware, and IO-aware schemes only balance the CPU, memory, and disk I/O resources, respectively. Thus, these schemes totally ignore the imbalanced communication load resulting from communication-intensive parallel applications.

A third observation drawn from Figs. 4 and 5 is that when most parallel applications running on the cluster are of a small-scale, the COM-aware scheme only marginally

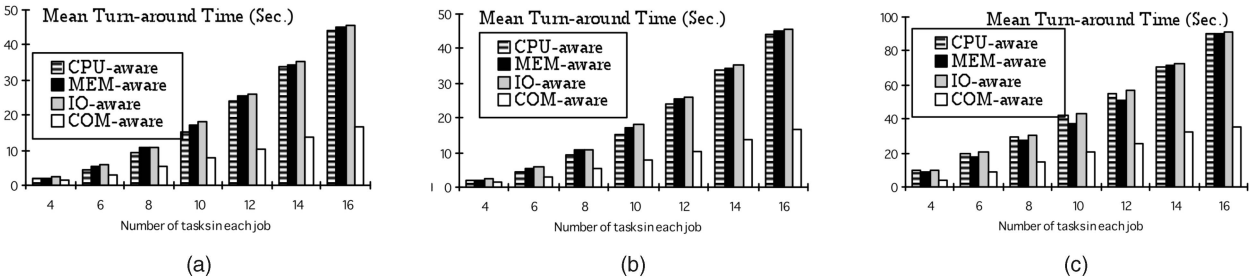


Fig. 5. Mean turn-around time versus number of tasks. (a) Message arrival $\text{rate} = 0.1$ No./ms. (b) Message arrival $\text{rate} = 0.3$ No./ms. (c) Message arrival $\text{rate} = 0.5$ No./ms.

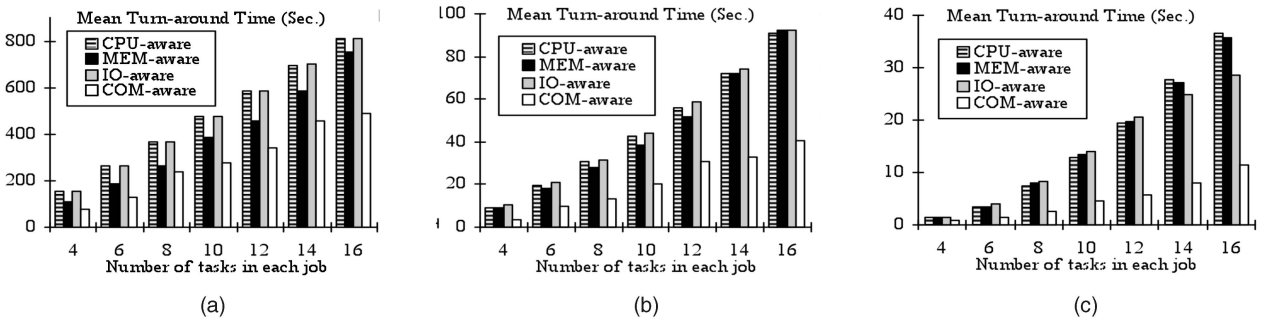


Fig. 6. Mean turn-around time versus network bandwidth. (a) Message $\mathrm{Bandwidth} = 10$ Mbps. (b) Message $\mathrm{Bandwidth} = 100$ Mbps. (c) Message $\mathrm{Bandwidth} = 1$ Gbps.

outperforms the CPU-aware and MEM-aware load balancers. For example, if the number of tasks in each parallel job is 4 and the message arrival rate is set to 0.1 No./ms, the performance gains in terms of mean slowdown and turn-around time achieved by the COM-aware scheme are as low as 3 and 4 percent, respectively. When parallel applications in the system are scaled up, the performance gap between the COM-aware and the three other schemes quickly increases. The implication of this performance gap is that large-scale parallel applications with high communication demands can greatly benefit from the proposed load-balancing technique.

6.2 Varying Network Bandwidth

While the first experiment assumes that all messages are transmitted over a network at an effective rate of 1 Gbps, the second set of results is obtained by varying the network bandwidth of the cluster. As a result, we will investigate the impact of network bandwidth on the performance of the four load-balancing schemes. In order to explore this issue, we set the network bandwidth to 10 Mbps, 100 Mbps, and 1 Gbps. Since the mean slowdown metric has similar patterns as compared to the mean turn-around time, Fig. 6 only shows the performance with respect to the turn-around time for the CPU-aware, MEM-aware, IO-aware, and COM-aware policies.

As shown in Fig. 6, the mean turn-around time of the four policies shares a common trait that the turn-around time is inversely proportional to network bandwidth. This result can be explained by the fact that when the communication demands are fixed, increasing the network bandwidth effectively reduces the communication-intensive applications time spent transmitting data. In addition, a high network bandwidth results in less synchronization time among tasks of a parallel job and low migration costs in these four load-balancing policies. Fig. 6 further reveals that the performance improvement achieved by COM-aware becomes more pronounced when the network bandwidth is high. For example, if the network bandwidth of the cluster is set to 10 Mbps, 100 Mbps, and 1 Gbps, the average performance improvements gained by the COM-aware load balancer over the three existing policies are 61.4, 116.1, and 182.9 percent, respectively.

6.3 Varying Average Message Size

Communication load depends on the message arrival rate and the average message size, which, in turn, depends on the communication patterns. The purpose of this experiment is to show the impact that varying the average message size has on the performance of the four load-balancing schemes. Fig. 7 shows that the mean turn-around time increases as the average message size increases. This is caused by the fact that as the message arrival rate is unchanged, a large average message size yields high network utilization in the system, causing longer waiting times on message transmissions. A second observation in Fig. 7 is that the benefits of the COM-aware scheme become increasingly pronounced when communication-intensive applications running on the cluster send and receive larger data messages. This is because the larger the average message size, the higher the network utilization, which, in turn, results in longer waiting times in network queues.

6.4 Workload with a Mix of Job Types

In the results provided in Sections 6.1, 6.2, and 6.3, memory and I/O-intensive jobs are omitted because the focus of the

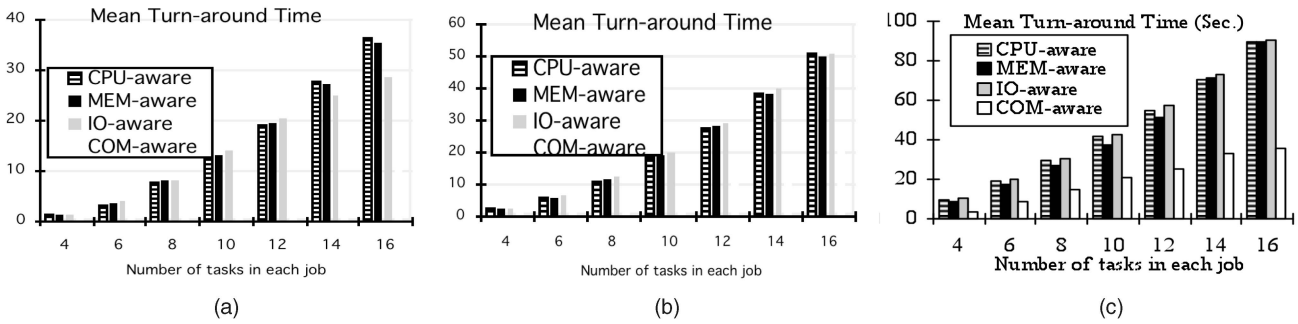


Fig. 7. Mean turn-around time versus message size. Message size is (a) 512 KB, (b) 2 MB, and (c) 5 MB.

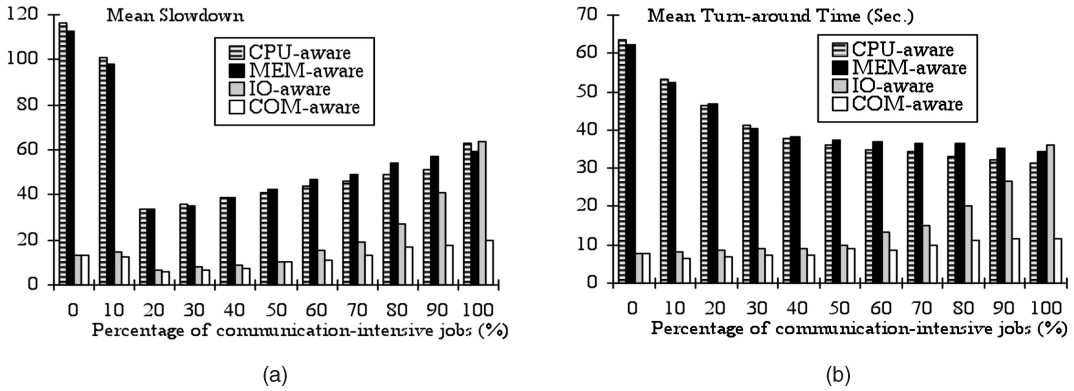


Fig. 8. Performance versus percentage of communication-intensive jobs. (a) Slowdown. (b) Turn-around time.

previous experiments is on communication-intensive workloads. However, the results obtained for workloads with only communication-intensive jobs are not always valid for clusters where multiple types of jobs are submitted. Clusters may have to execute a mix of job types, and different job types in the system affect the performance of each other. Therefore, in this section, we measure the impact of the proposed scheme on a cluster of 32 nodes where memory and I/O-intensive jobs are submitted along with communication-intensive jobs. Again, we compare the mean slowdown and turn-around time of the COM-aware scheme against the three existing load-balancing policies. The message arrival rate and network bandwidth in this experiment are set to 0.1 No./ms and 1 Gbps.

Fig. 8 shows the mean slowdown and turn-around time as functions of the percentage of communication-intensive jobs. It is noted that the 11 traces in Fig. 8 contain a collection of I/O and communication-intensive jobs. In general, the results show that our COM-aware policy performs the best in all cases. In particular, both the COM-aware and IO-aware policies noticeably outperform the CPU-aware and MEM-aware policies when the workload is I/O-intensive. This indicates that the COM-aware load balancer can maintain the same level of performance as the IO-aware scheme for I/O-intensive applications.

Interestingly, Fig. 8 shows that the performance improvement of IO-aware, over CPU-aware and MEM-aware strategies, consistently decreases with the increase in the percentage of communication-intensive jobs. The reason for this is that the IO-aware load balancer does not view network devices as important components of clusters even when the vast majority of parallel applications have high communication demands. In contrast, the mean slowdown and turn-around time of the COM-aware scheme are not sensitive to the percentage of communication-intensive jobs.

We now turn to study another mixed workload consisting of memory and communication-intensive jobs. Fig. 9 shows the mean slowdown and turn-around time as a function of the percentage of communication-intensive jobs. Again, the first observation is that the COM-aware policy performs the best in all cases. When the workload is memory-intensive, the performance of COM-aware scheme is very close to that of the MEM-aware scheme, and the MEM-aware strategy is better than the CPU-aware and IO-aware strategies if the percentage of communication-intensive jobs is less than 40 percent. This implies that the CPU-aware and IO-aware schemes are not suitable for applications with high memory demands. This is because both the MEM-aware and COM-aware schemes make an effort to achieve high usage of global memory.

The results also show that the performance of the MEM-aware load balancer is worse than that of the CPU and COM-aware load balancers when the percentage of communication-intensive jobs is larger than 40 percent. The results are expected because the MEM-aware scheme does not attempt to share network resources among the different nodes. We have obtained similar results under other workloads containing a mix of CPU, memory, I/O, and

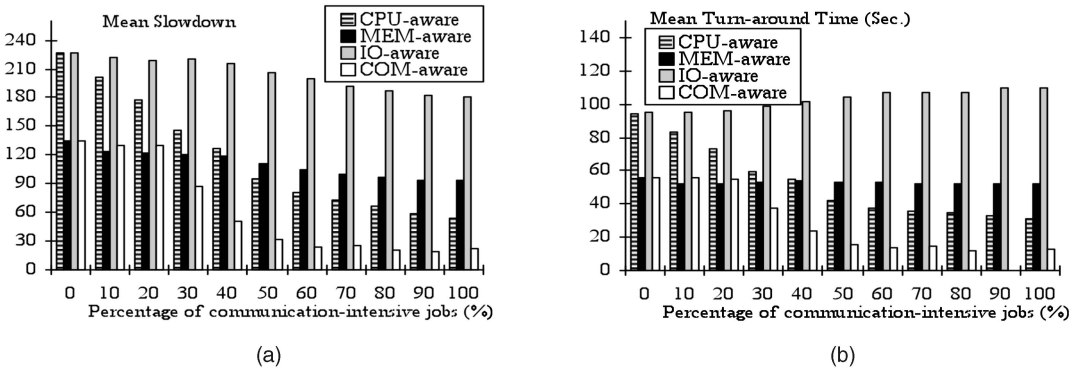


Fig. 9. Performance versus percentage of communication-intensive jobs. (a) Slowdown. (b) Turn-around time.

TABLE 2
Descriptions of Real Communication-Intensive Applications

Application	Description
Render	This application integrates a 6000× 6000 pixel 24-bit color image with 16-bits of elevation data per pixel to obtain 480×640 pixel 24-bit output images.
Molecular	This application is used to compute properties for liquids and polymers.
Semi	This application is a 3-D semiconductor device simulator.
React	This application is used to predict the behavior of chemical reactions.
Vortex	This application models the evolution of vortices in a 2-dimensional fluid.
ALL	The workload of a cluster where all the five applications are submitted.

communication-intensive jobs. Due to the space limitations, we present a subset of our results in this section.

6.5 Experiments on Real Parallel Applications

To validate the experimental results based on the synthetic parallel applications, we simulate five real communication-intensive parallel applications [8]. In this section, we evaluate the impact the COM-aware load balancer has on five real communication-intensive applications, which have different computation, disk I/O, and communication patterns. To simulate the previously mentioned communication-intensive applications, we generated six job traces where the arrival patterns of the jobs are extrapolated from traces collected from the University of California at Berkeley [13]. Each of the five traces consist of one type of real application described in Table 2, whereas the sixth trace is a collection of the five different types of applications. A 32-node cluster is simulated to run the applications.

Fig. 10a shows the mean slowdown of the simulated applications running on the cluster using the four load-balancing policies. It is observed from Fig. 10 that the COM-aware load-balancing approach works quite well for all communication-intensive applications. In the case of executing a single application, the COM-aware scheme exhibits a performance improvement of up to 75.9 percent (with an average of 42.2 percent, See Fig. 10b) in mean slowdown over the three existing policies. Note that the percentage improvement over the CPU-aware scheme can be calculated using the following equation, $1-(MS_{COM}/MS_{CPU})$, where MS_{COM} and MS_{CPU} are the mean slowdowns of the communication-aware and CPU-aware load balancers. We can use a similar way to derive the improvements over the MEM-aware and I/O-aware schemes. The performance gain of the COM-aware load balancer is mainly attributed to the effective usage of network resources in addition to the CPU, memory, and I/O resources in the cluster.

7 Conclusions

This paper has introduced a behavioral model for parallel applications with large requirements of network, CPU, and disk I/O resources. The model is particularly beneficial to clusters where resource demands of applications may not be known in advance. Thus, load balancers can make full use of this model to quickly and accurately determine the load induced by a variety of parallel applications. Furthermore, we have addressed the issue of improving the effective bandwidth of networks on clusters at the software level without requiring any additional hardware. Specifically, we have proposed a dynamic communication-aware load-balancing scheme, referred to as COM-aware, for nondedicated clusters where the resources of the cluster are shared among multiple parallel applications being executed concurrently. A simple yet efficient means of measuring communication load imposed by processes has been presented. We have reported the preliminary simulation results obtained on both synthetic bulk synchronous and real parallel applications. The experimental results show that the COM-aware approach can improve the performance over the three existing schemes by up to 206 and 235 percent, in terms of slowdown and turn-around time, respectively, under workloads with high communication demands. If the workload is memory-intensive or I/O-intensive in nature, COM-aware strategy dynamically and adaptively considers the memory or disks as first-class resources in clusters,

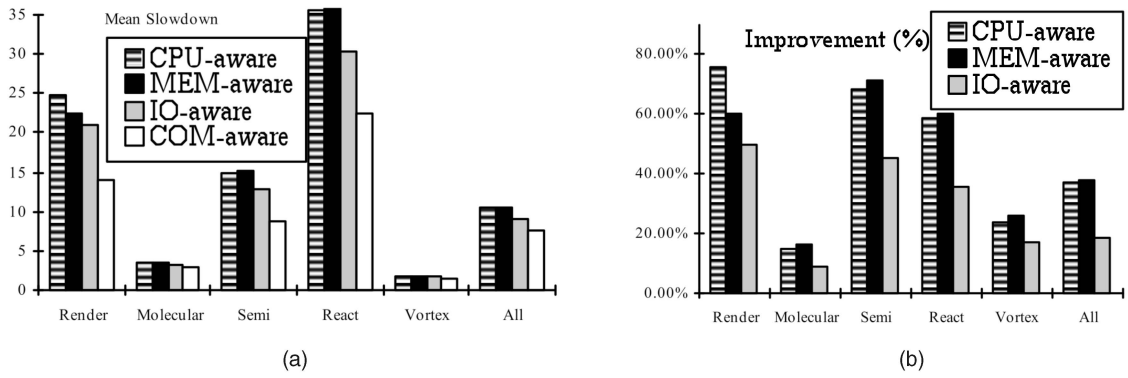


Fig. 10. (a) Slowdowns of real applications. (b) Performance improvements.

thereby sustaining the same level of performance as the existing memory-aware and I/O-aware schemes.

The experimental results presented in this paper largely depend on the simulator. As a future research direction, we will develop a prototype of the communication-aware load balancer. We will compare results obtained from the prototype with those reported from the simulator.

ACKNOWLEDGMENTS

The work reported in this paper was supported by the US National Science Foundation under Grants CCF-0845257 (CAREER), CNS-0757778 (CSR), CCF-0742187 (CPA), CCF-0621526 (HECURA), CNS-0831502 (CyberTrust), OCI-0753305 (CI-TEAM), DUE-0837341 (CCLI), and DUE-0830831 (SFS), as well as Auburn University under a startup grant and a gift (Number 2005-04-070) from the Intel Corporation. The source code of the ioBalanceSim simulator is available at www.eng.auburn.edu/~xqin/software/ioBalanceSim.

REFERENCES

[1] A. Acharya and S. Setia, "Availability and Utility of Idle Memory in Workstation Clusters," *Proc. ACM SIGMETRICS,* pp. 35-46, 1999.

[2] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and W.-K. Su, "Myrinet: A Gigabit-Per-Second Local Area Network," *IEEE Micro,* vol. 15, no. 1, pp. 29-36, Feb. 1995.

[3] R. Brightwell, B. Lawry, A.B. MacCabe, and R. Riesen, "Portals 3.0: Protocol Building Blocks for Low Overhead Communication," *Proc. 16th Int'l Parallel and Distributed Processing Symp. (IPDPS '02),* pp. 164-173, 2002.

[4] D. Buntinas, D.K. Panda, and R. Brightwell, "Application-Bypas Broadcast in MPICH over GM," *Proc. Third Int'l Symp. Cluster Computing and the Grid (CCGRID '03),* pp. 2-9, 2003.

[5] W. Cirne and F. Berman, "When the Herd Is Smart: Aggregate Behavior in the Selection of Job Request," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, no. 2, pp. 181-192, Feb. 2003.

[6] J. Cohen, E. Jeannot, N. Padoy, and F. Wagner, "Messages Scheduling for Parallel Data Redistribution between Clusters," *IEEE Trans. Parallel and Distributed Systems,* vol. 17, no. 10, pp. 1163-1175, Oct. 2006.

[7] J. Cruz and K. Park, "Towards Communication-Sensitive Load Balancing," *Proc. 21st Int'l Conf. Distributed Computing Systems,* pp. 731-734, Apr. 2001.

[8] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina, "Architectural Requirements of Parallel Scientific Applications with Explicit Communication," *Proc. 20th Ann. Int'l Symp. Computer Architecture (ISCA '93),* pp. 2-13, 1993.

[9] A.C. Dusseau, R.H. Arpaci, and D.E. Culler, "Effective Distributed Scheduling of Parallel Workloads," *Proc. ACM SIGMETRICS,* pp. 25-36, 1996.

[10] W.-C. Feng, J. (Gus) Hurwitz, H. Newman, S. Ravot, R.L. Cottrell, O. Martin, F. Coccetti, C. Jin, X. (David) Wei, and S. Low, "Optimizing 10-Gigabit Ethernet for Networks of Workstations, Clusters, and Grids: A Case Study," *Proc. 2003 ACM/IEEE Conf. Supercomputing (SC '03),* p. 50, 2003.

[11] P. Geoffray, "OPIOM: Off-Processor I/O with Myrinet," *Future Generation Computer Systems,* vol. 18, no. 4, pp. 491-499, 2002.

[12] W. Grop and E. Lusk, "The Message Passing Interface(MPI) Standard," *Argonne Nat'l Lab,* 2001.

[13] M. Harchol-Balter and A.B. Downey, "Exploiting Process Lifetime Distributions for Dynamic Load Balancing," *ACM Trans. Computer Systems,* vol. 15, no. 3, pp. 253-285, 1997.

[14] C.H. Hsu and J.W. Liu, "Dynamic Load Balancing Algorithms in Homogeneous Distributed Systems," *Proc. Sixth Int'l Conf. Distributed Computing Systems,* pp. 216-223, May 1986.

[15] R. Lavi and A. Barak, "The Home Model and Competitive Algorithms for Load Balancing in a Computing Cluster," *Proc. 21st Int'l Conf. Distributed Computing Systems (ICDCS '01),* pp. 127-134, 2001.

[16] P. Li and D. Curkendall, "Parallel 3-D Perspective Rendering," *Proc. First Int'l Delta Applications Workshop,* pp. 52-58, 1992.

[17] J.M. Orduna, V. Arnau, A. Ruiz, R. Valero, and J. Duato, "On the Design of Communication-Aware Task Scheduling Strategies for Heterogeneous Systems," *Proc. 2000 Int'l Conf. Parallel Processing (ICPP '00),* pp. 391-398, 2000.

[18] X. Qin, "Design and Analysis of a Load Balancing Strategy in Data Grids," *Future Generation Computer Systems,* vol. 23, no. 1, pp. 132-137, 2007.

[19] X. Qin, "Performance Comparisons of Load Balancing Algorithms for I/O-Intensive Workloads on Clusters," *J. Network Computer Application,* vol. 31, no. 1, pp. 32-46, 2008.

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

[21] X. Qin, Y. Jiang, H. Zhu, and D. Swanson, "Dynamic Load Balancing for I/O-Intensive Tasks on Heterogeneous Clusters," *Proc. 10th Int'l Conf. High Performance Computing (HiPC '03),* Dec. 2003.

[22] K.D. Ryu and J.K. Hollingsworth, "Exploiting Fine-Grained Idle Periods in Networks of Workstations," *IEEE Trans. Parallel and Distributed Systems,* vol. 11, no. 7, pp. 683-698, July 2000.

[23] L. Schaelicke and A.L. Davis, "Design Trade Offs for User-Level I/O Architectures," *IEEE Trans. Computers,* vol. 55, no. 8, pp. 962-973, Aug. 2006.

[24] K.G. Shin and Y.-C. Chang, "Load Sharing in Distributed Real-Time Systems with State-Change Broadcasts," *IEEE Trans. Computers,* vol. 38, no. 8, pp. 1124-1142, Aug. 1989.

[25] J.A. Stankovic, "Simulations of Three Adaptive, Decentralized Controlled, Job Scheduling Algorithms," *Computer Networks and ISDN Systems,* vol. 8, no. 3, pp. 199-217, 1984.

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

[27] L.G. Valiant, "A Bridging Model for Parallel Computation," *Comm. ACM,* vol. 33, no. 8, pp. 103-111, 1990.

[28] J.S. Vetter and F. Mueller, "Communication Characteristics of Large-Scale Scientific Applications for Contemporary Cluster Architectures," *J. Parallel and Distributed Computing,* vol. 63, no. 9, pp. 853-865, 2003.

[29] J.S. Vetter and F. Mueller, "High Performance Implementation of MPI Datatype Communication over Infiniband," *Proc. Int'l Parallel and Distributed Processing Symp.,* Apr. 2004.

[30] J.S. Vetter and A. Yoo, "An Empirical Performance Evaluation of Scalable Scientific Applications," *Proc. 2002 ACM/IEEE Conf. Supercomputing (SC '02),* pp 1-18, 2002.

[31] X.-D. Zhang, L. Xiao, and Y.-X. Qu, "Improving Distributed Workload Performance by Sharing Both CPU and Memory Resources," *Proc. 20th Int'l Conf. Distributed Computing Systems (ICDCS '00),* pp. 233-241, 2000.



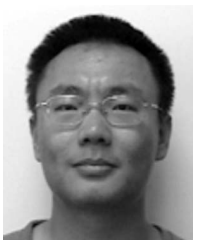
Xiao Qin received the BS and MS degrees in computer science from Huazhong University of Science and Technology in 1992 and 1999, respectively, and the PhD degree in computer science from the University of Nebraska-Lincoln in 2004. Currently, he is an assistant professor in the Department of Computer Science and Software Engineering at Auburn University. Prior to joining Auburn University in 2007, he had been an assistant professor with New Mexico Institute of Mining and Technology (New Mexico Tech) for three years. He received the US National Science Foundation (NSF) CAREER Award in 2009. In 2007, he received an NSF CPA Award and an NSF CSR Award. His research interests include parallel and distributed systems, storage systems, fault tolerance, real-time systems, and performance evaluation. His research is supported by the (NSF), Auburn University, and Intel Corporation. He had served as a subject area editor of the *IEEE Distributed System Online* (2000-2001). He has been on the program committees of various international conferences, including IEEE Cluster, IEEE IPCCC, and ICPP. He is a senior member of the IEEE.



Hong Jiang received the BS degree in computer engineering from Huazhong University of Science and Technology, Wuhan, China, in 1982, the MAS degree in computer engineering from the University of Toronto, Canada, in 1987, and the PhD degree in computer science from the Texas A&M University, College Station, in 1991. Since August 1991, he has been at the University of Nebraska-Lincoln, where he served as the vice chair of the Department of Computer Science and Engineering (CSE) from 2001 to 2007 and is a professor of CSE. His present research interests include computer architecture, computer storage systems and parallel I/O, parallel/distributed computing, cluster and Grid computing, performance evaluation, real-time systems, middleware, and distributed systems for distance education. He serves as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems*. He has more than 150 publications in major journals and international Conferences in these areas, including the IEEE-TPDS, IEEE-TC, JPDC, ISCA, FAST, ICDCS, OOPLAS, ECOOP, ICS, HPDC, ICPP, etc., and his research has been supported by the US National Science Foundation (NSF), DOD, and the State of Nebraska. He is a member of the ACM, the IEEE, the IEEE Computer Society, and the ACM SIGARCH.



Adam Manzanares received the BS degree in computer science from the New Mexico Institute of Mining and Technology, United States, in 2006. He is currently working toward the PhD degree at the Department of Computer Science and Software Engineering, Auburn University. During the Summers of 2002-2007, he has worked as a student intern at the Los Alamos National Laboratory. His research interests include energy-efficient computing, modeling and simulation, and high-performance computing. He is a student member of the IEEE.



Xiaojun Ruan received the BS degree in computer science from Shandong University in 2005. He is currently working toward the PhD degree at the Department of Computer Science and Software Engineering, Auburn University. His research interests are in parallel and distributed systems, storage systems, real-time computing, performance evaluation, and fault tolerance. His research interests focus on high-performance parallel cluster computing, storage system, and distributed system. He is a student member of the IEEE.



Shu Yin received the BS degree in communication engineering and the MS degree in signal and information processing from Wuhan University of Technology (WUT) in 2006 and 2008, respectively. He is currently working toward the PhD degree at the Department of Computer Science and Software Engineering, Auburn University. His research interests include storage systems, reliability modeling, fault tolerance, energy-efficient computing, and wireless communications. He is a student member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.