This paper appeared in the *Proceedings of the 24th IEEE International Performance, Computing, and Communications Conference (IPCCC 2005),* pp.27-34, Phoenix, Arizona, April 7-9, 2005

Improving Effective Bandwidth of Networks on Clusters using Load Balancing for Communication-Intensive Applications

Xiao Qin
Department of Computer Science,
New Mexico Institute of Mining and Technology,
801 Leroy Place, Socorro, New Mexico 87801
xqin@cs.nmt.edu
http://www.cs.nmt.edu/~xqin

Hong Jiang
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, NE 68588-0115,
jiang@cse.unl.edu

Abstract

Clusters have emerged as a primary and cost-effective infrastructure for parallel applications, including communication-intensive applications that transfer a large amount of data among nodes of a cluster via interconnection networks. Conventional load balancers have been proven effective in increasing utilization of CPU, memory, and disk I/O resources in a cluster. However, most of the existing load-balancing schemes ignore network resources, leaving open an opportunity for improving 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 performance of communication-intensive applications by increasing effective utilization of networks in cluster environments. Our load-balancing scheme can make use of an application model to quickly and accurately determine the load induced by a variety of parallel applications. Simulation results on executing a wide range of parallel applications on a cluster show that the proposed scheme can significantly improve the performance in slowdown and turn-around time over three existing schemes by up to 206% (with an average of 74%) and 235% (with an average of 82%), respectively.

1. Introduction

Scheduling [19] and load balancing [17] are two key techniques used to improve the performance of clusters for scientific applications by fully utilizing machines with idle or under-utilized resources. A number of distributed load-balancing schemes for clusters have been developed, primarily considering a variety of resources, including CPU [10], memory [1], disk I/O [16], or a combination of CPU and memory [23]. These approaches have been proven effective in increasing utilization of resources in clusters, assuming that networks connecting nodes 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 [8], 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 executions on clusters and more are underway. Many scientific applications are inherently computationally and communicationally intensive [7]. Examples of such applications include 3-D perspective rendering, molecular dynamics simulation, quantum chemical reaction dynamics simulations, and 2-D fluid flow using the vortex method, to name just a few [7]. The above bottleneck becomes more severe if the resources of a cluster are time-/space-shared among multiple scientific applications and communication load is not evenly distributed among the cluster nodes. Furthermore, the performance gap between CPU and network continues to widen at a faster pace, which raises a need of 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 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 performances of parallel applications running on a large-scale, time-/space-shared cluster, where a number of parallel jobs share various resources. COM-aware enables a cluster to utilize most idle or under-utilized network resources while keeping the

usage of other type of resources reasonably high. We employed an application behavioral 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 or another 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 [17][21][23] simultaneously. For this reason, the COM-aware scheme is designed in a more sophisticated way that delivers high performance under a large variety of workload scenarios to provide one possible solution to the problem.

The rest of the paper is organized as follows. In the section that follows, related work in the literature is briefly reviewed. Section 3 introduces the system model, and section 4 proposes a communication-aware load-balancing scheme. Section 5 presents performance measurements of the proposed scheme by simulating a cluster of 32 nodes with bulk synchronous parallel applications. Finally, Section 6 concludes the paper with comments on future directions for this work.

2. Related work

In general, load-balancing techniques fall into two camps: centralized and distributed load balancing. Centralized schemes require a head node that is responsible for handling load distribution. As cluster size scales up, the head node quickly becomes a bottleneck. To solve this scalability problem, distributed dynamic load balancing can be used to delegate workload of load balancing to multiple modes in a cluster. In addition, a centralized scheme has a potential problem of poor reliability because permanent failures of the central load balancer can render in the entire load balancing mechanism dysfunctional. Therefore, the focus of this paper is on designing a decentralized communication-aware load balancer for time-/space-shared clusters.

There is a large body of established research focusing on the issue of distributed load balancing for CPU and memory resources. Harchol-Balter et al. [10] developed a CPU-based preemptive migration policy that was shown to be more effective than non-preemptive migration policies. Zhang et al. [23] 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 prior study, we proposed a family of I/O-aware load-balancing policies to meet the needs of a cluster system with a variety of workload conditions [16][17][18]. These approaches are effective under workload without high communication intensity, but balancing communication load is not considered.

A communication-sensitive load balancer has been proposed by Cruz and Park [6]. The balancer uses run-time communication pattern between processes to balance load. Orduña et al. have proposed a criterion to measure the quality of network resource allocation to parallel applications [14]. 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 showed that message-passing architectures and programming interfaces are useful techniques to develop parallel applications. Brightwell et al. implemented Portals message-passing interface on a commodity PC Linux cluster [3]. For higher-level message-passing layers, the Portals can reduce overhead for receiving messages due to the availability of programmable NICs with significant processing power. Buntinas et al. 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 Model and Assumptions

Since communication and I/O demands of applications may not be known in advance, we can make use of an application model proposed in [15] to capture characteristics of communication, disk I/O, and CPU activities within parallel applications. The parameters of the model for a parallel application can be obtained by repeatedly executing the applications off-line for multiple inputs. Alternatively, a lightweight and on-line profiling technique can be used to monitor applications' behaviors, thereby updating the model. Note that the model is used in our trace-driven simulations.

Our goal is to model a non-dedicated cluster, where each job has a "home" node that it prefers for execution [12]. This is because either input data of a job has been stored in its home node, or the job was created on the 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 [17].

A cluster in the most general form is comprised of a set of nodes each containing a combination of multiple types of resources, such as CPU, memory, network connectivity, and disks. 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 resource monitor [11][20].

It is assumed in our model that networks in clusters provide 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 like Myrinet [2] and InfiniBand [22], which provide pseudo-full connectivity, allowing simultaneous transfers between any pair of nodes.

Since communication can be directly handled by network interface controllers without local CPU's intervention or the buffer in the main memory [9], we assume that CPU and memory load induced by data communication is negligible.

4. Communication-aware load balancing

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

In this study we focus on a bulk-synchronous style of communication, in which a master creates a group of slave processes across a set of nodes in a cluster. Let a parallel job be formed by p processes $t_0, t_1, \ldots, t_{p-1}$, and n_i 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 Equation (1), where $T^i_{j,COM}$ is the time spent on communication.

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 Equation (1) 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 different than the one executing the master.

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

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

$$L_{i,COM} = \sum_{\forall All\ j: n_j = i} L_{j,p,COM} \quad (2)$$

Given a parallel job arriving at a node, the load-balancing scheme attempts to balance the communication load by allocating the job's processes to a group of nodes with 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 commutation 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:

$$\left(L_{i,COM} - L_{j,COM}\right) > L_{t,p,COM} \quad (3)$$

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_t^i and R_t^j 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.

$$R_t^j < R_t^i + R^{i,j}_{t,mig} \quad (4)$$

The migration cost, $R^{i,j}_{t,mig}$, is the time interval between the initiation and the completion of t's 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 a 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, a resource 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 Figure 1.

```
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 if
  end if
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 if
    end if
end if
```

Figure 1. Pseudo code of the communication-aware load balancing scheme.

5. Experimental results

5.1. Experiment platform

We simulate a cluster with 32 nodes under a variety of workload conditions. The simulated cluster is configured in such a way to resemble a typical modern day cluster.

To evaluate the performance impacts of the communication-aware load-balancing scheme, we extrapolate traces from those reported in [10][23]. 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 number of tasks in each parallel job.

For jobs with 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 512Kbyte, which reflects typical data characteristics for many applications, such as 3-D perspective rendering [13].

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 job's submission and completion. The turn-around time is a natural metric for the job performance due to its ability to reflect a user's 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 non-dedicated cluster is a commonly used metric for measuring the performance of the job [10]. 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.

5.2. Communication-intensive workloads

In the first experiment we intend to stress the communication-intensive workload by setting the message arrival rate at the value of 0.5No./ms. 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.

Figures 2 and 3 plot the mean slowdown and turn-around time (measured in Seconds) as functions of the number of tasks in each parallel job on a cluster of 32 nodes. Figures 2 and 3 indicates that all the load-balancing schemes experience an increase in the mean slowdown and turn-around time when the number of tasks in jobs increases. This is because when CPU, memory, and disk I/O demands are fixed, the increasing number of tasks in each parallel job leads to high communication demands, causing longer waiting time on sending and receiving data.

Importantly, we observe from Figures 2 and 3 that

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

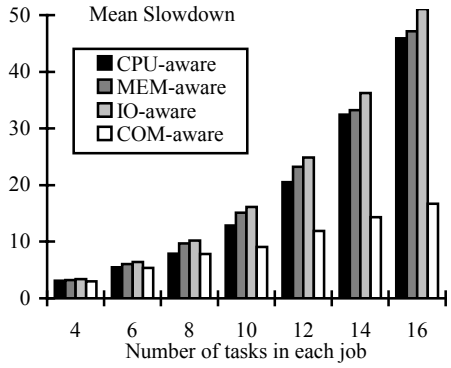


Fig. 2. Mean slowdown vs. number of tasks. Network bandwidth is 1Gbps, and average message size is 512 Kbytes.

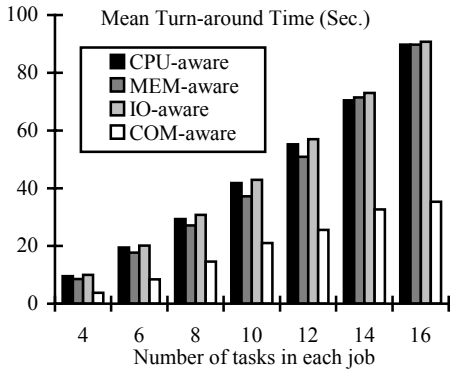


Fig. 3. Mean turn-around time vs. number of tasks. Network bandwidth = 1Gbps, and average message size = 512 Kbytes.

5.3. Varying network bandwidth

While the first experiment assumes that all messages are transmitted over a network at an effective rate of 1Gbps, 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 10Mbps and 1Gbps, respectively. Since the mean slowdown has similar patterns as those of the mean turn-around time, Figure 4 only shows the performance with respect to the turn-around time for the CPU-aware, MEM-aware, IO-aware, and COM-aware policies.

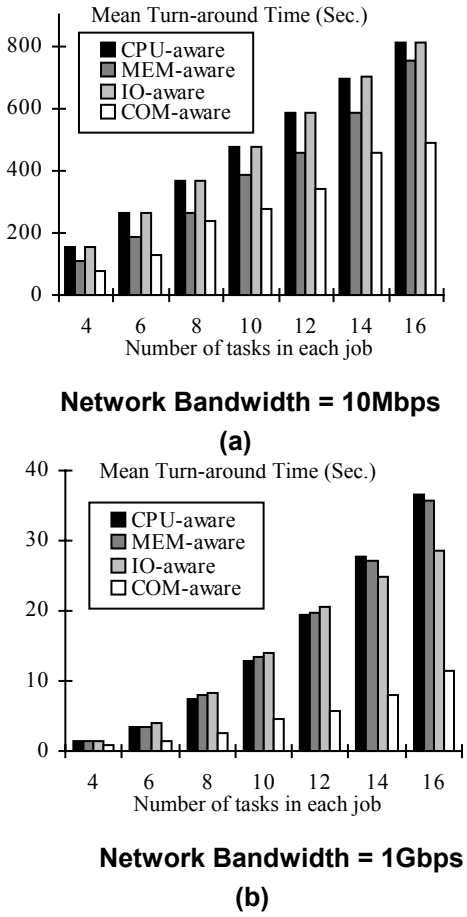


Figure 4. Mean turn-around time vs. network bandwidth. Message arrival rate = 0.1No./ms, and average message size = 512 Kbytes.

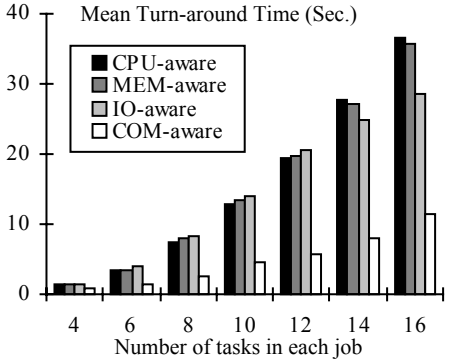
As shown in Figure 4, the mean turn-around time the four policies share a common feature in the sense that turn-around time is inversely proportional to network bandwidths. This result can be explained by the fact that when the communication demands are fixed, increasing the network bandwidth effectively reduces 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 cost in these four load-balancing policies.

Figure 4 further reveals that the performance

improvement achieved by COM-aware becomes more profound when the network bandwidth is very high. For example, if the network bandwidth of the cluster is set to 10Mbps and 1Gbps, the average performance improvements gained by COM-aware over the three existing policies are 61.4% and 182.9%, respectively.

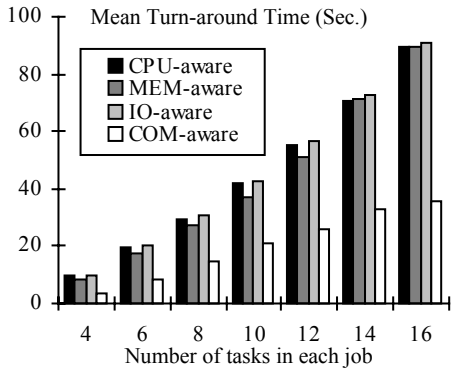
5.4. Varying average message size

Communication load depends on message arrival rate and the average message size, which in turn depends on communication patterns. The purpose of this experiment, therefore, is to show the impact of average message size on the performance of the four load balancing schemes.



Message Size = 512 KB

(a)



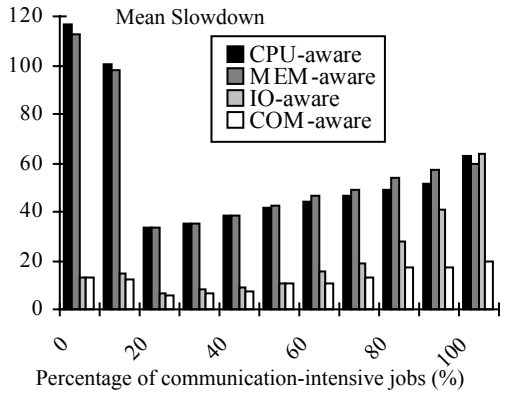
Message Size = 5 MB

(b)

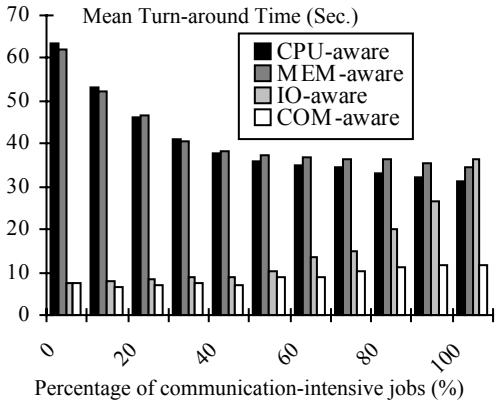
Figure 5. Mean turn-around time vs. message size. Message arrival rate = 0.1No./ms, and network bandwidth = 1Gbps.

Figure 5 shows that the mean turn-around time increases as the average message size increases. The reason is that as message arrival rate is unchanged, a large average message size yields high network utilizations in the system, causing longer waiting times on message transmissions.

A second observation from Figure 5 is that the benefits of the COM-aware scheme become increasingly significant when communication-intensive applications running on the cluster tend to send and receive larger data volumes on the network. This is because the larger the average message size, the higher the network utilization, which in turn results in longer waiting time in network queues.



(a)



(b)

Figure 6. Performance vs. percentage of communication intensive jobs. (a) mean slowdown, (b) mean turn-around time of I/O and communication intensive jobs.

5.5. Workload with a mix of job types

In the results provided in Sections 5.2-5.4, memory- and I/O-intensive jobs are omitted because the focus of the pervious experiments is more on communication-intensive workload situations. However, the results obtained for

workloads with only communication-intensive jobs are not always valid for clusters where multiple types of jobs are submitted. The reason is that realistic clusters may have to execute a mix of job types, and different job types in the systems affect one another. 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 also submitted along with communication-intensive jobs.

Again, we compare the mean slowdown and turn-around time of COM-aware against those of the three existing load-balancing policies. The message arrival rate and network bandwidth in this experiment are set to 0.1 No./ms and 1Gbps.

Figure 6 shows the mean slowdown and turn-around time as functions of the percentage of communication-intensive jobs. It is noted that the eleven traces in Figure 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 COM- and IO-aware noticeably outperform CPU-aware and MEM-aware when the workload is I/O-intensive, indicating that COM-aware can maintain the same level of performance as the IO-aware scheme for I/O-intensive applications.

Interestingly, Figure 6 shows that the performance improvement of IO-aware over CPU- and MEM-aware consistently decreases with the increase in the percentage of communication-intensive jobs. The reason is that IO-aware does not view network devices as important components of clusters even when vast majority of running applications have high communication demands. In contrast, the mean slowdown and turn-around time of COM-aware are not sensitive to the percentage of communication-intensive jobs.

We now turn to study another mixed workload with memory- and communication-intensive jobs. Similarly, Figure 7 shows the mean slowdown and turn-around time as functions 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 is very close to that of the MEM-aware scheme, and MEM-aware is better than CPU-aware and IO-aware if the percentage of communication-intensive jobs is less than 40%, implying that the CPU-aware and IO-aware schemes are not suitable for applications with high memory demands. This is because both MEM-aware and COM-aware take efforts to achieve high usage of global memory.

The results also show that the performance of MEM-aware is worse than that of CPU- and COM-aware when the percentage of communication-intensive jobs is larger than 40%. The results are expected because MEM-aware does not attempt to share network services among the different nodes. We have obtained similar results under other workload with a mix of CPU-, memory-, I/O-, and communication-intensive jobs. Due to the space limits, we present the partial results in this section.

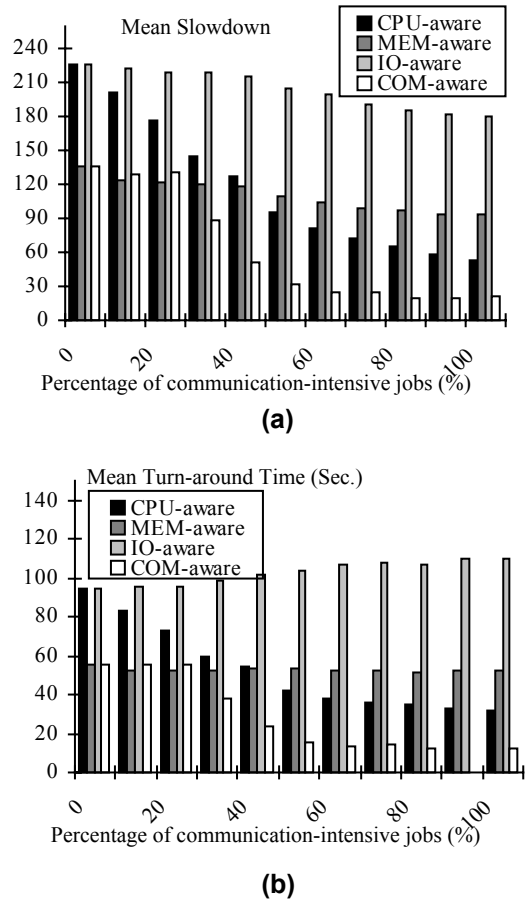


Figure 7. Performance vs. percentage of communication intensive jobs. (a) mean slowdown and (b) mean turn-around time of memory and communication intensive jobs. Message arrival rate = 0.1No./ms, network bandwidth = 1Gbps.

6. Conclusion

We have addressed the issue of improving effective bandwidth of networks on clusters at software level without requiring any additional hardware. Specifically, we have proposed a dynamic communication-aware load-balancing scheme, referred to as *COM-aware*, for non-dedicated clusters where the resources of the cluster are shared among multiple parallel applications being executed concurrently.

Since CPU, communication, and I/O demands of applications may not be known in advance, we took an

advantage of the application behavioral model proposed in [15] to quickly and accurately determine the requirements of various resources imposed by parallel applications.

Furthermore, a simple yet efficient means of measuring communication load imposed by processes have been presented. We have reported the preliminary simulation results obtained on bulk synchronous applications. The experimental results show that under workload with high communication demands, the *COM-aware* approach can improve performance in terms of slowdown and turn-around time over three existing schemes by up to 206% and 235%, respectively. If workload is memory-intensive or I/O-intensive in nature, COM-aware dynamically and adaptively considers memory or disks as the first-class resources in clusters, thereby sustaining the same level of performance as the existing memory-aware and I/O-aware schemes.

As a future research direction, we plan to extend the approach to a more widely distributed environment such as a peer-to-peer system.

Acknowledgements

This work was partially supported by an NSF grant (EPS-0091900), a Nebraska University Foundation grant (26-0511-0019), and a UNL Academic Program Priorities Grant.

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] N.J. Boden, D. Cohen, and W.K. Su., "Myrinet: A Gigabit-per-second Local Area Network," *IEEE Micro*, Vo. 15, No. 1, February 1995.

[3] R. Brightwell, R. Riesen, "Portals 3.0: Protocol Building Blocks for Low Overhead Communication," *Proc. Int'l Parallel and Distributed Processing Symp. Workshops, April, 2002, Florida.*

[4] D. Buntinas, D. K. Panda and R. Brightwell, "Application-Bypass Broadcast in MPICH over GM," *Proc. Int'l Symp. Cluster Computing and the Grid,* May 2003.

[5] W. Cirne, F. Berman, "When the herd is smart Aggregate behavior in the selection of job request," *IEEE Trans. Parallel and Distributed Systems*, No.2, Feb. 2003.

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

[7] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina, "Architectural Requirements of Parallel Scientific Applications With Explicit Communication," *Proc. the 20th Int'l Symp. on Computer Architecture,* 1993.

[8] W. Feng, G. Hurwitz, et al., "Optimizing 10-Gigabit Ethernet for Networks of Workstations, Clusters and Grids: A Case Study," *Proc. SC 2003: High-Performance Networking and Computing Conf.*, 2003.

[9] P. Geoffray, "OPIOM: Off-Processor I/O with Myrinet", *Future Generation Computer Systems*, 2002(19).

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

[11] C.-Y.H. Hsu and J.W.-S. Liu, "Dynamic Load Balancing Algorithms in Homogeneous Distributed Systems," *IEEE Proc. 6th Int'l Conf. Distributed Computing Systems*, pp.216-223, 1986.

[12] R. Lavi and A. Barak, "The Home Model and Competitive Algorithm for Load Balancing in a Computing Cluster," *Proc. the 21st Int'l Conf. Distributed Computing Systems*, Apr. 2001.

[13] P. Li and D. Curkendall, "Parallel 3-D perspective rendering," *Proceedings of the 1st International Delta Applications Workshop*, Technical Report CCSF-14-92, California Institute of Technology, pp.52-58, 1992.

[14] J.M. Orduña, V. Arnau, A. Ruiz, R. Valero, "On the Design of Communication-Aware Task Scheduling Strategies for Heterogeneous Systems," *Proc. Int'l Conf. Parallel Processing*, 2000.

[15] X. Qin, "Improving Network Performance through Task Duplication for Parallel Applications on Clusters," *Proc. the 24th IEEE Int'l Performance, Computing, and Communications Conference (IPCCC 2005),* Phoenix, Arizona, April, 2005.

[16] X. Qin, H. Jiang, Y. Zhu, and D. Swanson, "Dynamic Load Balancing for I/O-Intensive Tasks on Heterogeneous Clusters," *Proc. the 10th Int'l Conf. on High Performance Computing* (HiPC 2003), pp. 300-309, Dec., 2003.

[17] 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. the 5th IEEE Int'l Conf. Cluster Computing* (Cluster 2003), pp.100-107, Dec. 2003.

[18] X. Qin, H. Jiang, Y. Zhu, and D. R. Swanson, "Dynamic Load balancing for I/O- and Memory-Intensive workload in Clusters using a Feedback Control Mechanism," *Proc. the 9th Int'l Euro-Par Conf. Parallel Processing (Euro-Par 2003),* pp. 224-229, Klagenfurt, Austria, Aug. 2003.

[19] N. T. Spring, R. Wolski, "Application Level Scheduling of Gene Sequence Comparison on Metacomputers," *Proc. Int'l Conf. Supercomputing*, pp. 141-148, 1998.

[20] J.A. Stankovic, "Simulation of Three Adaptive, Decentralized Controlled, Job Scheduling Algorithms," *Computer Networks*, Vol.8, pp.199-217, 1984.

[21] 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.

[22] J. Wu, P. Wyckoff, and D. K. Panda, "High Performance Implementation of MPI Datatype Communication over InfiniBand," *Proc. Int'l Parallel and Distributed Processing Symp.,* April, 2004.

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