JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 6, No. 4, December 2012, pp. 287-293

A Comparative Performance Study for Compute Node Sharing

Jeho Park*
Computing and Information Services, Harvey Mudd College, Claremont, CA, USA
Jeho_Park@hmc.edu

Shui F. Lam
Computer Engineering and Computer Science, California State University Long Beach, Long Beach, CA, USA
lam@csulb.edu

Abstract

We introduce a methodology for the study of the application-level performance of time-sharing parallel jobs on a set of compute nodes in high performance clusters and report our findings. We assume that parallel jobs arriving at a cluster need to share a set of nodes with the jobs of other users, in that they must compete for processor time in a time-sharing manner and other limited resources such as memory and I/O in a space-sharing manner. Under the assumption, we developed a methodology to simulate job arrivals to a set of compute nodes, and gather and process performance data to calculate the percentage slowdown of parallel jobs. Our goal through this study is to identify a better combination of jobs that minimize performance degradations due to resource sharing and contention. Through our experiments, we found a couple of interesting behaviors for overlapped parallel jobs, which may be used to suggest alternative job allocation schemes aiming to reduce slowdowns that will inevitably result due to resource sharing on a high performance computing cluster. We suggest three job allocation strategies based on our empirical results and propose further studies of the results using a supercomputing facility at the San Diego Supercomputing Center.

Category: Smart and intelligent computing

Keywords: Resource sharing; Resource allocation; Time-sharing cluster; Job scheduling; High performance computing; Percentage slowdown

I. INTRODUCTION

In this paper, we introduce a methodology for the study of the application-level performance of overlapped parallel processing jobs sharing compute nodes in high performance clusters. Throughout this study, we assume that parallel jobs arriving at a cluster need to share the compute nodes with other jobs, and compete with them for processor time and other limited resources such as memory and I/O. For such shared environments, scheduling studies [1-4] have focused more on system-level performance such as throughput and utilization than on application or user perspective performance like response time and slowdown of user jobs completion. Users submitting parallel jobs to a high performance cluster, however, are more interested in how fast their applications finish than in how many applications can run efficiently on the entire system. Therefore, our study focuses on a user perspective performance metric, slowdown; i.e., we perform a comparative performance study on the application slow-

Open Access http://dx.doi.org/10.5626/JCSE.2012.6.4.287 http://jcse.kiise.org

This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

Received September 10 2012, **Revised** 26 September 2012, **Accepted** 19 October 2012
*Corresponding Author

Copyright © 2012. The Korean Institute of Information Scientists and Engineers pISSN: 1976-4677 eISSN: 2093-8020

down and the relative slowdowns of jobs in different job mix environments.

The percentage slowdown S is calculated simply by

$$S = \frac{T_s - T_d}{T_d} \times 100 \quad (1)$$

where T_s is response time in the shared mode and T_d is response time in the dedicated mode. We analyze parallel job performance by examining the percentage slowdown due to another parallel job in the same node of the system.

It is worth noting that Weinberg and Snavely [5] took a similar approach to study slowdown effects by running two different benchmark jobs at the same time on SDSC's DataStar cluster. Our study, however, differs from theirs in that ours focuses on the time-sharing effects on a set of single-processor compute nodes while theirs investigates the space-sharing effects for shared resources such as memory and I/O on a multiprocessor, multicore node.

In the following sections, we explain the background of our comparative performance study, and provide the details of the methodology used for the study. We then describe the tools developed for the experimentation and the parallel jobs representing different characteristics of message passing interface (MPI) jobs. In Section V, we show the results of a series of concurrent job executions on a time-sharing, non-dedicated Linux cluster. Then we explain the setup of an SDSC supercomputing resource for the future study of the parallel job slowdown. A potential future extension of this research would be to apply the conclusion to job management systems and test its applicability in reducing job slowdown rates for superimposed parallel jobs on a high performance computing (HPC) cluster.

II. BACKGROUND

The Beowulf cluster system we used for our results was composed of 16 commodity computers interconnected via a gigabit Ethernet switch. Each compute node had a single 2.4 GHz Pentium 4 processor and 1 GB RAM [6]. As each node had one processor, it was a useful test bed for our study in finding the time-sharing effects of concurrent parallel processes when the execution of these processes are scheduled by the individual local operating systems and not an external scheduler. We will refer to this scheduling scheme as uncoordinated local scheduling in distributed systems.

There are studies on the pros and cons of various scheduling schemes especially in the context of coscheduling abound. For example, Anglano [1] conducted simulations and reported a large amount of performance degradation due to the uncoordinated scheduling of processes, especially when the parallel application is communication-bound. On the other hand, Wong and Goscinski [2] claimed that uncoordinated process scheduling does not introduce extreme performance loss; the overall performance degradation of local scheduling was similar to gang-scheduling's performance degradation, and in some cases, local scheduling worked better than gang-scheduling. Currently, the advantage of coordinated scheduling remains unsettled. Consequently more rigorous evaluation of uncoordinated scheduling, especially for application-level performance, is needed.

To better understand the application-level performance degradation, we investigated the performance impact due to the communication and computation characteristics of parallel jobs. We chose four jobs from the NAS Parallel Benchmark (NPB): IS, EP, CG, and MG [7]. All four jobs have distinctive characteristics in communication and computation. We explain the details of their characteristics in Section IV.

Our experiment procedure is simple and straightforward. First, we execute a new job (injected job) while another job (base job) is running on a set of compute nodes. While those two jobs are running at the same time, we sample their runtime statistics, average system load, and the number of packets they send and receive. After both of the parallel jobs finish their execution, we examine the data to make sure that the *injected* job left the system prior to the base job; this is to ensure that the statistics for injected job's slowdown is not biased due to dedicated runs of the injected job in case of the base job's early termination. Finally, we analyze the data to calculate slowdown of the injected job for different base jobs.

This controlled experiment is designed to collect necessary empirical data for the performance analysis of a single injected job of given characteristics when it is running in an environment with a single base job of certain characteristics. It may be possible to run multiple base jobs with multiple injected jobs on a set of nodes to simulate a shared computing environment (e.g., [8]). Running multiple base jobs and multiple injected jobs, however, will complicate the experiment, making it difficult to identify the factors responsible for performance degradation. Therefore, throughout our comparative performance study, we examined two single jobs (an injected job and a base job) sharing a set of nodes and found the performance degradation patterns due to the interference which arose from their demand on the shared resources. In the next section, we explain our methodology in detail.

III. METHODOLOGY FOR PERFORMANCE STUDY

It is obvious that a job injected into a system while a base job is running will have to time share the processor and compete for other limited resources. Both jobs are therefore expected to experience performance degradation. The only question is how much. To find an answer, using our tools, we ran a series of tests and took measur-

able data. In these experiments we measure and report only the performance degradation of the injected jobs since they are the jobs we can ensure to have run in a shared resource environment from start to stop.

The tools include a parallel /proc file scanner and a simulation driver. The parallel /proc file scanner scans and parses /proc/net/dev and /proc/stat virtual system files at every sampling time. The /proc/net/dev virtual system file contains network volume counts such as the number of bytes and packets it received and transmitted. The /proc/stat file keeps track of the amount of time that the CPU has spent in user mode, nice mode, system mode, idle task and interrupt services. The /proc virtual system file approach, however, is only meaningful for coarse grain time resolution of 100 ms or larger. This coarse grain sampling time resolution works fine for the purpose of our data sampling because we are interested in application level statistics, not network link-level statistics. Frequent access to /proc file is not recommended because the scanner may affect the processor time of the jobs being monitored.

The second tool, simulation driver, was originally developed to spawn a series of different parallel jobs arriving at the system according to a user-defined job arrival time distribution. This creates a multi-user environment under the assumption of processor timesharing on compute nodes without any particular job scheduling scheme to manage job allocation. With the tool, we were able to generate reproducible and controllable system loads in both computation and communication to study and model network traffic and resource contention incurred by multi-user driven parallel jobs [8].

For this comparative performance study, the simulation driver was modified to execute a predefined set of parallel jobs in regular time intervals. This allowed us to consistently inject a new job at the right time for multiple tests so that we could calculate the average slowdown for the same combination of jobs with consistency.

The parallel jobs we tested were taken from NPB kernels that contain jobs with different computation and communication characteristics. The NPB suite (ver. 2.4) consists of eight benchmark problems: five numerical algorithm kernels (CG, EP, IS, MG, and FT) and three simulated computational fluid dynamics applications (SP, BT, and LU). Each benchmark problem has five levels of problem sizes: S, W, A, B, and C from small to large problem size, respectively. We collected their computation and communication data to categorize them in terms of processor and network usage patterns. We then picked four of them to represent four types of parallel jobs: EP, IS, CG, and MG. In the next section, we illustrate the characteristics of the four NPB jobs.

Fig. 1. Four representative parallel jobs and their characteristics according to computation and communication. IS: integer sort, CG: conjugate gradient, MG: multi-grid, EP: embarrassingly parallel.

IV. CHARACTERISTICS OF EP, IS, CG, AND MG

The four benchmark kernels, EP, IS, CG, and MG, have distinguishable characteristics in processor usage and message passing communication usage: computation-bound job (EP), communication-bound job (IS), communication-bound mixture job (CG), and computation-bound mixture job (MG). Fig. 1 illustrates these characteristics in a diagram.

The EP benchmark uses all of the available processor power with a couple of negligible communications. On the other hand, the IS benchmark extensively uses collective communication for sorting. The MG and CG benchmarks use both computation and communication in their executions. MG takes more processor power during its computation than CG does; MG uses 70–80% of the processor for its computation for class B, while CG uses only 40–50% for class B. We attempted to compare them in both class A and class B problem sizes because the two classes were proper size problems for our Beowulf cluster test bed.

A. Computation-Bound Job: EP

The embarrassingly parallel (EP) benchmark kernel is highly parallelized for computation and has negligible communication throughout its execution. This benchmark tests floating point operation performance by generating a large number of pseudorandom numbers concurrently on multiple nodes. The EP benchmark does communications in its initial stage using the MPI_Barrier and in its final stage using four back-to-back MPLAllreduce calls. The

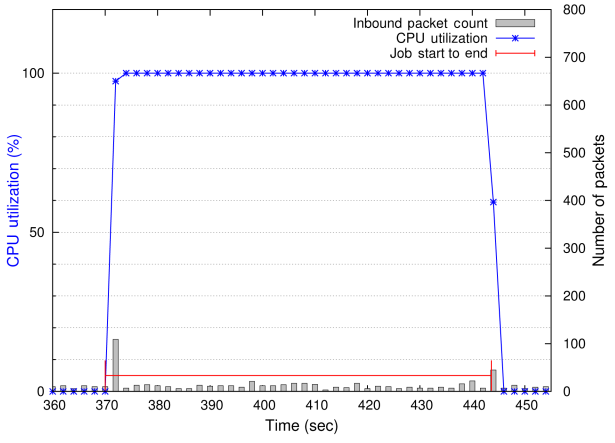

Fig. 2. A dedicated embarrassingly parallel (EP, class B) run.

collective communication overhead is negligible compared to its computation part. In terms of CPU utilization, the EP kernel utilizes almost 100% of the CPU resource during its entire execution due to the computation-bound characteristic of this kernel.

A comprehensive description of the EP kernel's behavior is depicted in Fig. 2. The plot shows synchronized CPU usage and communication activities in one figure. In Fig. 2, the left vertical axis indicates CPU utilization levels (marked by the 'x' marking symbols in the plot) at the time of sampling. The right vertical axis represents the number of packets indicated by the impulse bars in the plot. The red horizontal line with h and end points shows the starting and ending of the job. Notice that the CPU usage of the EP kernel is 100% and the average number of packets per 2-second period is approximately 10. Fig. 2 visually confirms that EP is a highly computation-bound job.

B. Communication-Bound Job: IS

The integer sort (IS) kernel benchmark uses collective communication calls (MPLAlltoall, MPI_Alltoallv, MPLAllreduce, and MPLReduce) throughout its execution, with an exception of a few point-to-point communications (MPLSend, MPI_wait, and/or MPLIrecv) at the finalizing stage. Moreover, IS sends and receives a huge number of small size messages (≈ 1 kB) for the parallel sorting of integers, which are generated separately on other nodes at the initial computation period. The IS benchmark represents a communication-bound job since it uses only about 8% of the CPU time throughout its execution on average.

Fig. 3 shows a visualization of IS kernel's runtime data in the dedicated mode. Notice that the number of packets received is almost 60,000 in 2 seconds during communication peak times.

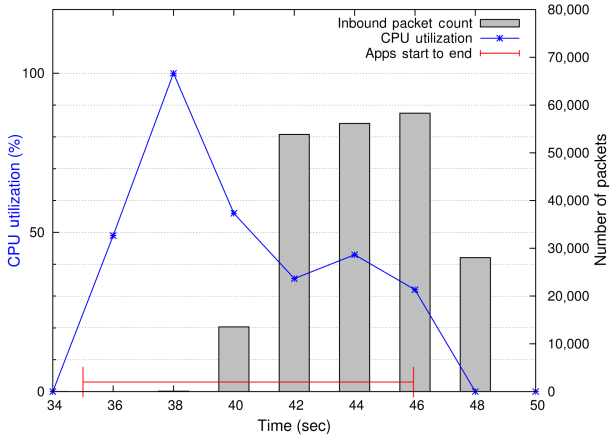

Fig. 3. A dedicated integer sort (IS, class B) run.

C. Mixture Jobs: CG and MG

So far we have seen two extreme cases, computation-bound EP kernel and communication-bound IS kernel. In this section, we investigate the computation and communication characteristics of two mixture jobs, CG and MG.

The conjugate gradient (CG) kernel can be categorized as a communication-bound mixture job due to its lower processor usage level compared to the multi-grid (MG) kernel, and a smaller number of packet transmissions than IS for the class A and B problems. The CG benchmark mainly performs point-to-point communications throughout its parallel execution at (almost) uniform intervals; an exception is one MPLReduce call at its finalizing stage.

Fig. 4 shows that the CPU utilization is about 50% throughout the CG benchmark's runtime. Our measurements show that the number of packets a node receives in each 2-second interval is around 11,000. In contrast, the packet transmission rate of the IS kernel is about 55,000 packets per 2-second interval during communication peak

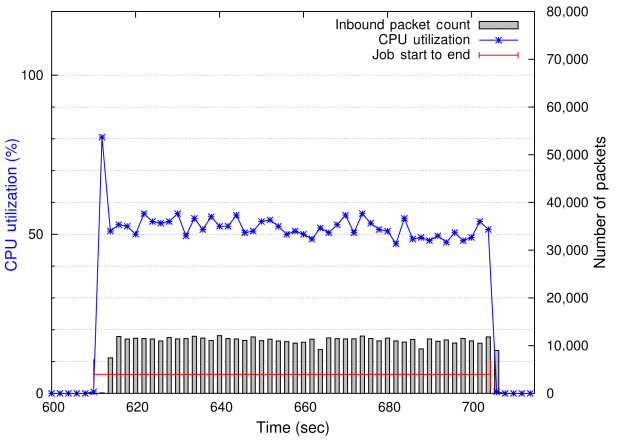

Fig. 4. A dedicated conjugate gradient (CG, class B) run.

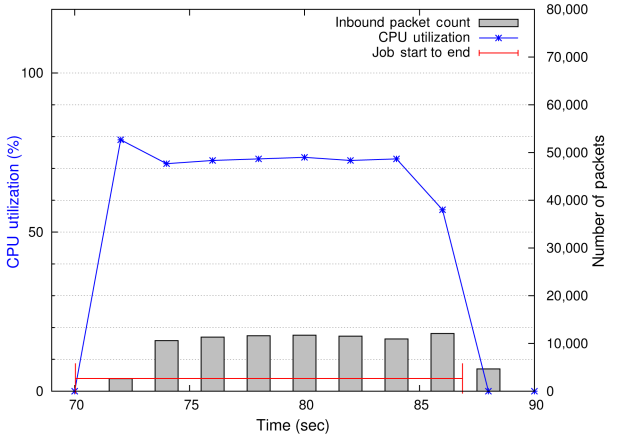

Fig. 5. A dedicated multi-grid (MG, class B) run.

times. The CG benchmark regularly sends and receives messages in three sizes: small-size messages (8 bytes and 16 bytes) and two types of big-size messages (56,000 bytes for class A and 300,000 bytes for class B).

On the other hand, the MG benchmark can be categorized as a computation-bound mixture job because its CPU utilization level is relatively high (around 70%) for class A and class B problems, and it also does a fair amount of communication as shown in Fig. 5. The MG benchmark performs a number of point-to-point communications along with several collective communications throughout its execution. Unlike the CG benchmark, the communication pattern (or occurrences) of the MG benchmark is not uniform and the size of messages varies widely. Therefore, the amount of calculation time between data exchange calls also varies for different data exchange sizes.

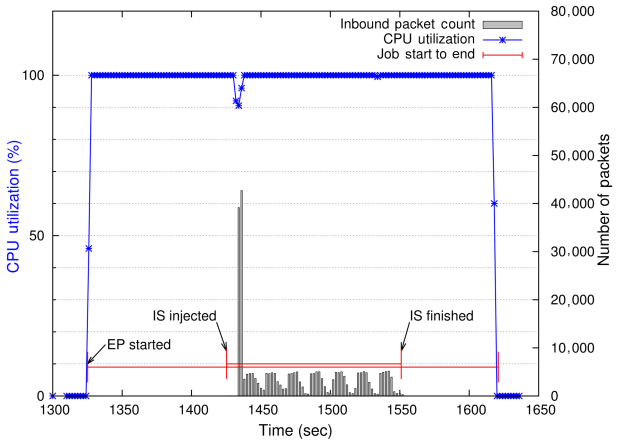

Fig. 6. A shared integer sort (IS, class B) run with a highly computation-bound job, embarrassingly parallel (EP, class C). The two horizontal red lines in the plot indicate that the two jobs, EP and IS, were concurrently running at the time of sampling.

V. RESULTS FROM THE COMPARATIVE PERFORMANCE STUDY

Using the methodology and tools described in the previous sections, we collected performance data from the Beowulf cluster and analyzed them to gain insights for better resource allocation strategies of parallel jobs sharing the processor and other resources on a compute node (or a set of compute nodes). Table 1 presents the slowdown of the injected jobs (in three different problem sizes) for different base jobs. The numbers are the average percentage slowdown calculated by Equation 1 with data collected from multiple simulation runs.

The EP benchmark experienced the least amount of slowdown when injected to the system where IS, CG or MG were running. The EP took twice the time it would take in the dedicated mode when it was injected to the system where another EP was running. This was a natural consequence because the Linux local scheduler gave the same priority to both the base job and the injected job. Such computation-bound jobs exploit 100% of the CPU time whenever the CPU is assigned to them. Hence, the runtime should be doubled from the dedicated runtime when there are two such jobs in the system competing.

Noticeably, the slowdown was almost negligible when EP was injected to the node where a communication-bound job (e.g., IS) was already running. With mixture jobs (i.e., CG and MG), the EPs performance experienced relatively small degradation (<35%). Therefore, we may conclude that highly computation-bound jobs like the EP kernel would not be prone to slowdown due to sharing the processor and other resources with another type of job on time-sharing Linux clusters.

The communication-bound IS benchmark worked satisfactorily with mixture jobs; its completion was delayed by less than 48% of dedicated runs for reasonable size problems (class A and B). Hence, if a communication-bound job needs to run with other jobs on a cluster system, the job management system would want to allocate it to a set of nodes (or a part of a set of nodes) where a mixture job is running to ensure the least performance degradation of the communication-bound job due to resource sharing.

The IS class B, however, experienced an unacceptable amount of slowdown with the computation-bound base job; it finished its computation and communication almost ten times slower than its dedicated mode runs on average. Fig. 6 depicts an instance of such extreme delays. It shows that the base job (EP class C) started at around 1,325 seconds and the injected job, IS class B, launched 100 seconds later at around 1,425 seconds. The pattern of the communication packets transmitted per sampling time is clearly different from its dedicated run (Fig. 3) throughout the life of the shared IS execution. We studied the extreme case more carefully and found that the performance of sending and receiving a large number of small messages using point-to-point communication depends largely on the MPI message protocol especially under heavy CPU loads. The *lamd* request progression interface (RPI) module in LAM/MPI helped reduce the delay substantially although the *lamd* module was considered to be a slow protocol scheme due to its asynchronous user datagram protocol (UDP)-based message passing [9].

Among all kinds of base jobs, the communication-bound mixture job (CG) experienced the largest performance degradation with the communication-bound job (IS). The reason for CG's largest slowdown rate with IS may be attributed to the timing mismatch created by the high-rate packet transmission of the IS benchmark interfering with the structured (synchronized) communication pattern of the CG benchmark.

The CG experienced less slowdown with MG than with communication-bound or computation-bound jobs.

Table 1. Percentage slowdown (%) of injected MPI jobs for different base jobs

	EP injected			IS injected			CG injected			MG injected		
Base job	W	A	B	W	A	B	W	A	B	W	A	B
EP	93	100	99	43	102	956	42	88	85	184	89	654
IS	4	2	4	45	81	96	130	141	143	155	72	108
CG	1	5	10	55	14	33	42	51	86	55	18	38
MG	2	32	34	60	14	48	56	32	36	36	78	75

MPI: message passing interface, EP: embarrassingly parallel, IS: integer sort, CG: conjugate gradient, MG: multi-grid.

This property led us to conclude that if a communication-bound mixture job needs to be assigned to a set of nodes to share resources on it, running with another mixture job would be the best choice. Similarly, the computation-bound mixture job (MG) works better with CG than any other kind of job, yielding less than 38% of slowdown for reasonably large problems (classes A and B). Therefore, the job placement strategy for CG also applies to MG.

Additionally, from Table 1, notice that the MG class B benchmark experienced substantial performance degradation when it was injected to a node where a computation-bound job was running; the slowdown rate of MG with EP was 654%. In [6], we also showed that the slowdown may be less severe when different communication protocol schemes were used as in the extreme case of IS (class B) over EP.

VI. FUTURE WORK

The results from our comparative performance study clearly showed that the slowdown rates largely depend on the characteristics of the base jobs running concurrently on a set of compute nodes when processor time-sharing was allowed. With our empirical results and tools in hand, the next phase of our study is to expand our methodology to HPC clusters with multiple multicore nodes (multi-CPU), multicore nodes for both time-sharing and space-sharing performance study. To this end, we have begun gathering performance data of NPB jobs from San Diego Supercomputer Center's Trestles.

Trestles is a dedicated XSEDE (a single, virtual cyber-infrastructure that is composed of high performance and high throughput computing resources shared by scientists; http://www.xsede.org) cluster having 324 nodes and over 20 TB memory with quad data rate (QDR) InfiniBand interconnection. Each node has 32 processor cores (four 8-core AMD Magny-Cours Opteron CPUs). Currently jobs are managed by the Catalina scheduler [10] and resources are handled by the TORQUE resource manager. We plan to adopt our simulation model to a cluster of similar scale and characteristics, and compare performance regarding the use of the Catalina scheduler (coordinated scheduling) vs. uncoordinated local scheduling.

VII. CONCLUSION

In this paper, we presented a methodology and the set of tools for a comparative performance study of time-sharing parallel jobs on a set of compute nodes. We used four NPB jobs that represent different types of parallel jobs on the system. We conducted our experiments by injecting a parallel job into the system while a base job is running, and collected performance data during the run. A combination of different types of injected and base jobs were used in our experiment. With the collected performance data, slowdowns of the injected jobs were calculated and analyzed, and their relative performance degradations were identified.

Our empirical results from this methodology led to a couple of interesting observations relevant to job scheduling. First, highly computation-bound jobs exploit processor time-sharing so that it suffers relatively little slowdown when running with other types of jobs in the same nodes. Second, a highly communication-bound job works reasonably well over mixture jobs with only moderate performance degradation (no more than 50% slowdown for relatively large problem sizes) though the performance of the mixture job appears to be significantly impacted by the communication-bound job when they run simultaneously. Last, mixture jobs run faster with mixture jobs than with highly communication-bound or highly computation-bound jobs. In any case, when a new job is added to the compute nodes currently running a base job, both jobs will suffer performance degradation. The impact on the base job due to the injected job during the time of resource sharing can also be found in Table 1 with the roles of the two jobs reversed.

These observations suggest three simple job allocation strategies: 1) a highly computation-bound job would be better scheduled to run on a separate set of nodes as they substantially impact other types of parallel jobs in the same node; 2) though a highly communication-bound job suffers only moderately when running with a mixture job, the slowdown of the mixture job is rather severe, therefore such a combination is desirable when a highly communication-bound job has higher priority over mixture jobs in the system; and 3) a mixture job may be scheduled with another mixture job without substantial performance

impact due to resource sharing. In conclusion, the amount of performance degradation of a parallel (MPI) job due to other parallel (MPI) jobs in the same node can be expressed as "Mixture job > Highly communication-bound job > Highly computation-bound job."

We plan to expand our study to include comparative performance data from multi-core, space-sharing clusters and to apply our results to job schedulers (e.g., SDSC's Catalina scheduler) in order to study its effectiveness and applicability in reducing slowdown of parallel jobs when limited resources are shared by multiple parallel jobs.

REFERENCES

1. C. Anglano, "A comparative evaluation of implicit coscheduling strategies for networks of workstations," *Proceedings of the 9th International Symposium on High-Performance Distributed Computing*, Pittsburgh, PA, 2000, pp. 221-228.
2. A. K. L. Wong and A. M. Goscinski, "Concurrent execution of multiple NAS parallel programs on a cluster," *Proceedings of the 5th International Conference on Computational Science*, Atlanta, GA, 2005, pp. 435-442.
3. B. B. Zhou, X. Qu, and R. P. Brent, "Effective scheduling in a mixed parallel and sequential computing environment," *Proceedings of the 6th Euromicro Workshop on Parallel and Distributed Processing*, Madrid, Spain, 1998, pp. 32-37.
4. G. S. Choi, S. Agarwal, J. H. Kim, C. R. Das, and A. B. Yoo, "Performance comparison of coscheduling algorithms for non-dedicated clusters through a generic framework," *International Journal of High Performance Computing Applications*, vol. 21, no. 1, pp. 91-105, 2007.
5. J. Weinberg and A. Snavely, "Symbiotic space-sharing on SDSC's datastar system," *Proceedings of the 12th International Conference on Job Scheduling Strategies for Parallel Processing*, Saint-Malo, France, 2006, 192-209.
6. J. Park, "An empirical approach to communication and performance modeling for message passing parallel applications on cluster systems," Ph.D. dissertation, Claremont Graduate University, Claremont, CA, 2009.
7. D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow, "The NAS Parallel Benchmarks 2.0," NASA Ames Research Center, Moffett Field, CA, Report NAS-95-020, 1995.
8. J. Park, S. Lam, and J. Angus, "Self-similarity in message passing parallel processing communication," *Proceedings of the International Conference on Communications in Computing*, Las Vegas, NV, 2008, pp. 94-100.
9. The LAM/MPI Team, Open Systems Lab, "LAM/MPI User's Guide version 7.1.2," http://brooks.chem.lsa.umich.edu/Cluster/status/docs/7.1.2-lam-user.pdf.
10. Technology Transfer and Intellectual Property Services, "Catalina scheduler: future home of Catalina software distribution page," http://www.sdsc.edu/catalina/.

Jeho Park

Jeho Park received a Ph.D. in Engineering and Industrial Applied Mathematics from Claremont Graduate University in 2009. Currently he is the Scientific Computing Specialist at Harvey Mudd College with responsibility for providing consultation and training on mathematical and scientific computing, high performance computing, and parallel processing for faculty, students and research groups. His research interests include performance modeling, job scheduling, high performance computing and education.

Shui F. Lam

Shui F. Lam received a Ph.D. in Computer Science from the Pennsylvania State University in 1975. Currently she is Professor of Computer Science at the California State University, Long Beach. Her research interests include multiprocessor scheduling, optimization & simulation modeling of scheduling and transportation problems, and high performance computing. She has co-authored a book on computer capacity planning and published articles in the Journal of the ACM, Communications of the ACM, SIAM Journal on Computing, and others.