Effectively Utilizing Global Cluster Memory for Large Data-Intensive Parallel Programs

John Oleszkiewicz, *Member*, *IEEE*, Li Xiao, *Member*, *IEEE*, and Yunhao Liu, *Member*, *IEEE*

Abstract—Large scientific parallel applications demand large amounts of memory space. Current parallel computing platforms schedule jobs without fully knowing their memory requirements. This leads to uneven memory allocation in which some nodes are overloaded. This, in turn, leads to disk paging, which is extremely expensive in the context of scientific parallel computing. To solve this problem, we propose a new peer-to-peer solution called Parallel Network RAM. This approach avoids the use of disk, better utilizes available RAM resources, and will allow larger problems to be solved while reducing the computational, communication, and synchronization overhead typically involved in parallel applications. We proposed several different Parallel Network RAM designs and evaluated the performance of each under different conditions. We discovered that different designs are appropriate in different situations.

Index Terms—Parallel programs, peer-to-peer, cluster, network RAM, scheduling, simulation.

---◆---

1 INTRODUCTION

CLUSTER systems with networked server nodes are becoming more popular for high-performance scientific computing applications for both economic and technical reasons [1]. One standard approach to reducing the runtime of such applications is to parallelize them into multiple parallel processes so many cluster nodes can run parts of the application simultaneously. An advantage of this approach is the CPU and memory resources of the job are evenly distributed and used. However, this advantage may not serve the best performance interests of the cluster because a balanced workload distribution among many parallel processes may result in unbalanced resource utilization of the system. Uniform use of resources at the parallel process level does not necessarily mean the system itself is evenly utilized.

We can attempt to adjust system memory load by adjusting the number of processes in a parallel process, but there are trade-offs between the number of processes and memory usage. For a given problem size, in order to ensure each node has enough memory space to accommodate a process, we could partition the parallel process into a large number of processes. This results in less work and more required synchronization for each process. As a consequence, the CPU on each node may be underutilized. Parallel speedup is very hard to improve as the number of processes increases, due to the increasing communication and synchronization overhead.

To ensure good CPU utilization, we must limit the number of processes in a parallel process. However, as the problem size increases, nodes may run out of available memory and be forced to use the local disk as a swapping site [2], [3]. Performance will suffer from frequent page faults since hard disks are orders of magnitude slower than RAM. Research has shown that disk paging results in unsatisfactory performance on parallel platforms and should be avoided [2], [3], [4].

We can increase the number of nodes in the system, but doing so may be impossible or expensive given the cluster setup. In addition, it is unlikely that the additional nodes will offer any real benefit over the long term since it is likely the users of the cluster will simply increase their usage of the system to match the new resources available.

The key problem is memory usage and this problem has two parts: memory fragmentation and paging overhead. The aggregate memory capacity of the system may be enough to satisfy memory demands, but cluster memory is distributed into discrete chunks. Usage of these chunks may be uneven and inefficient. On the most heavily loaded nodes, disk paging is invoked which incurs a high cost.

Network RAM [5], [6] has been proposed for use by sequential jobs in clusters to even memory load and reduce paging overhead. This technique allows applications to allocate more memory than is available on the local machine while avoiding paging to disk by allocating idle memory of other machines over a fast interconnecting network. This remote RAM is treated as a new layer in the memory hierarchy between RAM and disk. Resulting page accesses are slower than RAM, but faster than disk [5], [7], [6], [8], [9].

Existing network RAM techniques should not be directly applied to parallel jobs for several reasons. One critical issue is that processes from the same parallel job synchronize regularly. If cluster nodes seek network RAM independently, an uneven amount of network RAM may be granted to nodes hosting processes. With this uneven allocation, the

---

- *J. Oleszkiewicz is with Smiths Aerospace, 3290 Patterson Ave., SE Grand Rapids, MI 49512-1991. E-mail: john@oleszkiewicz.net.*
- *L. Xiao is with the Department of Computer Science and Engineering, 3115 Engineering Building, Michigan State University, East Lansing, MI 48824. E-mail: lxiao@cse.msu.edu.*
- *Y. Liu is with the Department of Computer Science, Hong Kong University of Science and Technology, Kowloon, Hong Kong. E-mail: liu@cs.ust.hk.*

processes will run at different speeds. However, the parallel jobs they belong to will only run at the speed of the slowest process, due to synchronization. The nodes with extra network RAM waste it since their hosted processes will spend most of their time waiting for other processes. Therefore, coordination is required to grant overloaded nodes equal portions of memory to allow hosted processes to run at equal speeds.

Another issue is network congestion. If cluster nodes individually seek out network RAM with no coordination among themselves, a potentially large amount of unnecessary network traffic will result. This may induce congestion on the cluster. Parallel applications require high performance networks to run efficiently and congestion could seriously impact performance.

We propose a new peer-to-peer solution called Parallel Network RAM (PNR) that allows overloaded cluster nodes to utilize idle remote memory. With this scheme, each node may request memory resources from remote nodes and provide memory resources for others. Requests are indirect in most proposed PNR methods: Each node contacts a manager (super-peer) node and requests that it allocate network RAM on its behalf. Managers coordinate the allocation of network RAM of several nodes and ensure that memory resources are distributed evenly to the nodes hosting parallel processes belonging to the same parallel job. PNR will allow more jobs to execute concurrently without resorting to disk paging. This will lead to decreased average response times and higher system throughput.

This paper makes the following contributions:

- We first identify the unbalanced resource utilization problem in a cluster with a mixed workload of jobs with different resource requirements. Existing techniques cannot maximize the performance gain of parallel jobs in such an environment in terms of both parallel speedup and execution time.
- We propose a novel and effective solution to this problem called Parallel Network RAM (PNR). PNR makes it possible for parallel jobs in a cluster to utilize memory resources from available remote nodes. The CPU cycles will be provided by a small subset of nodes while the global memory space of the cluster is open to the memory demands of any parallel job. Since the speed gap between accessing local memory and remote memory is shrinking and the speed gap between accessing local disk and remote memory continues to enlarge, the proposed scheme is expected to be beneficial for large scale scientific computing applications now and in the future.
- We build a simulator that models a cluster and proposed PNR algorithms. Conducting trace-driven simulations, we compare the performance of four different PNR designs to each other and to a solution that only uses disk paging. We identify which designs perform best under different circumstances.

The rest of this paper is organized as follows: Section 2 reviews related work. Section 3 describes Parallel Network RAM, the algorithms used to implement PNR, and the strengths and weaknesses of different proposed PNR designs. Section 4 describes the methodology, metrics, and experiments used to test our PNR algorithms. Section 5 describes the results of the experiments. Section 6 discusses the results and draws conclusions about the various PNR designs. Section 7 concludes the paper and outlines future work.

2 BACKGROUND AND RELATED WORK

In this section, we describe various parallel job schedulers, previous solutions to the problem of overloaded memory on cluster systems, and network RAM as one specific solution.

2.1 Parallel Scheduling Algorithms

The primary duty of the scheduler on a cluster system is to ensure high system throughput and low overall response times of submitted jobs. One scheduling approach is the space sharing model. Space sharing allows multiple jobs to be scheduled on the cluster at one time. Each node is devoted to one parallel process and each job runs until completion without preemption. The space sharing model is vulnerable to large, long-running jobs monopolizing the system. A bad scheduling decision is difficult to correct [1].

Another approach, gang scheduling, combines both space sharing and time sharing to avoid the problems associated with large jobs [10], [11], [12], [13]. Each job is allotted a time slot and the job may execute in its time slot. Nodes within each time slot are space shared. When a time quantum has expired, all jobs in the current slot are preempted and replaced with jobs in the next slot. The preemption process is called a "Parallel Context Switch" (PCS) and involves a certain amount of overhead. There is at most one process running on each node at any given time, although some schemes relax this constraint [14].

The time slot mechanism ensures no large job may monopolize the system for a long period of time. The maximum number of time slots allowed is known as the "Multiprogramming Level" (MPL). A system with an MPL value of one is a space sharing system.

2.2 Previous Solutions to the Memory Problem

Previous studies agree that unmodified disk paging results in severely reduced performance on parallel systems [3], [15]. Generally speaking, previous solutions to this problem either attempt to avoid disk paging entirely or attempt to reduce its effect.

Various ways to avoid paging by altering the system scheduler have been suggested. If no processor or memory information about incoming jobs is known, then the simplest solution is to keep MPL to a minimum [16]. If processor and memory estimates are provided by the user, they can be used in scheduling decisions. The use of user estimates of runtimes is the basis of a scheduling technique called backfilling. However, it is well known that such information is unreliable. In fact, many backfilling schemes take advantage of systemic inaccurate runtime estimations [17]. Another solution to the memory problem guesses memory usage information based on information about the job provided by the user and information contained in the program executable [2]. Another uses speedup information known about jobs ahead of time to make scheduling decisions [4], [18]. Given this information,
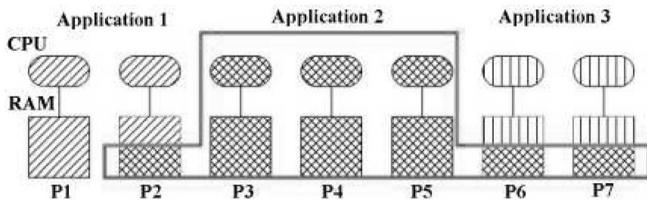
Fig. 1. Diagram of Parallel Network-RAM. Application 2 is assigned to nodes of P3, P4, and P5, but utilizes the available memory spaces in other nodes, such as P2, P6, and P7.

the scheduler can choose to give more processors to efficient jobs to increase utilization or to memory-intensive jobs to increase throughput.

One method that is aimed at reducing the disk paging penalty is called block paging. In this scheme, the system groups sets of pages together and acts upon these groups as units. Groups are defined by the system based on memory reference behavior of jobs [19].

Another method used to reduce paging penalties is Network RAM. Regarding network RAM implementations, the Global Memory System (GMS) [7] and the Remote Memory Pager [8] attempt to reduce page fault overhead by using remote paging techniques. DoDo [5] is designed to improve system throughput by harvesting idle memory space in a distributed system. In DoDo, processes running on the local system have the highest priority for using CPUs and memory on their workstations. This divides the global memory system into different local regions.

A memory ushering algorithm is used in MOSIX for memory load sharing [20]. This solution is a job-migration-based load sharing approach. Recently, several load sharing alternatives have been developed. These techniques consider both CPU and memory resources with known and unknown memory demands [21], [22]. The objective of the designs is to reduce the number of page faults caused by unbalanced memory allocations of distributed jobs so that overall performance can be significantly improved.

3 DESIGN RATIONALE

We propose a novel and effective technique called Parallel Network RAM (PNR) to better utilize both CPU and memory and minimize communication and synchronization overhead for parallel pragrams. We demonstrate the basic idea of PNR in Fig. 1. In this figure, Application 2 runs on nodes P3, P4, and P5, but utilizes available memory space in nodes P2, P6, and P7, where nonmemory intensive Applications 1 and 3 reside. With PNR, instead of three nodes being overloaded while four are underloaded, all seven nodes are fully utilized.

Our objective is to fundamentally improve the efficiency of large-scale scientific computing. Under our proposed solution, CPU and memory resource allocations are considered separately. CPU utilization requirements can be taken into account during the scheduling phase, and memory requirements can be handled as needed during execution. In this way, CPU usage can be optimized to maximize utilization and minimize communication and synchronization overhead while memory usage can include

both the local memory space from the assigned CPUs and the remote memory space in other nodes (as needed). With PNR, speedup can be scaled as the available remote memory increases, and performance can also be scaled as the problem size increases.

Because CPU usage and memory usage are considered separately, PNR does not coordinate with or receive information from the assumed centralized scheduler of the system. This allows PNR to be implemented on a variety of platforms, including traditional supercomputers and computing clusters, without changes to their existing scheduling policies.

3.1 Generic Description

In brief, all nodes in the system host PNR servents. All servents act as PNR clients and servers. Some servents may act as managers. Managers act as proxies for clients to communicate with servers. The purpose of managers is to coordinate client requests.

A PNR client attempts to allocate and deallocate network RAM on the behalf of its hosting node. The node uses allocated network RAM as additional virtual memory just as it would with disk space. Once network RAM is allocated, the client is informed what machines are serving the network RAM and how much was allocated. The client will send pages to the server(s) for storage and later retrieval.

Most PNR methods we propose use "managers," which are proxies between clients and servers. Depending on the PNR strategy, the manager may immediately act on client allocation and deallocation requests or wait for a certain number of requests before acting. Most strategies require that all clients associated with threads within a parallel job must contact the same manager before the manager is allowed to act on any one of the requests. When all requests are received from relevant clients, the manager aggregates the requests and attempts to select a PNR server from its availability list. The server is contacted and may grant, partially grant, or deny the manager's request. If too little network RAM is allocated by the server, the PNR manager will attempt to contact another server. This process is repeated until enough network RAM is granted by multiple servers or until no servers are left to query. The manager will divide received network RAM evenly among the requesting clients.

Servers receive requests from managers for network RAM. If the server has more unallocated RAM than a certain threshold, it will grant the network RAM request and allocate the memory to the manager. After network RAM is allocated, servers receive requests to read and write the data directly from clients. Servers grant all valid deallocation attempts.

During each message-passing interaction, servents piggyback current memory load information onto other messages. Managers use this up-to-date memory load information to update their network RAM availability lists. Managers may share this information with other managers via broadcasts.
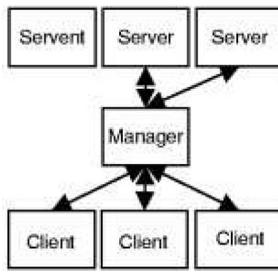
Fig. 2. Centralized PNR Design (CEN).

3.2 Designs

In this section, we propose four different PNR designs. Each design has a different architecture and has different amounts of communication overhead associated with it.

In the centralized (CEN) strategy, only one manager exists, and it coordinates all client requests (see Fig. 2). All servents know the identity of this manager. Since only one active manager exists, the centralized manager does not broadcast any memory load information it receives. However, as the system size grows, the network connection leading to the node hosting the manager will become a bottleneck. Note that the computational and memory overhead of hosting this central manager are not taken into account by our experiments.

In the client (CLI) strategy, each client acts as a manager and sends allocation requests directly (see Fig. 3). The client does not coordinate with other clients and attempts to allocate as much network RAM as possible for its hosting node. It does not share memory load information with other servents. This strategy allows clients to allocate network RAM quickly and eliminates manager synchronization overhead. It is scalable and simple to implement, but does not coordinate network RAM allocation. Some clients may receive large amounts of network RAM and others may not. This may worsen overall performance, since much of the network RAM allocated is wasted.

In the local managers strategy (MAN), whenever a job starts or stops, one of the servents running on a node associated with that job will volunteer to act as the manager (see Fig. 4). Each servent involved must agree on which servent will act as the manager. The manager will take their requests, allocate network RAM, and divide up the received network RAM evenly among the requesting clients. At the end of each allocation and deallocation, the manager will broadcast memory load information to each servent in the system since each can potentially act as a manager. This is
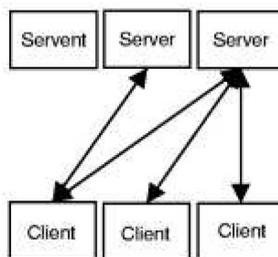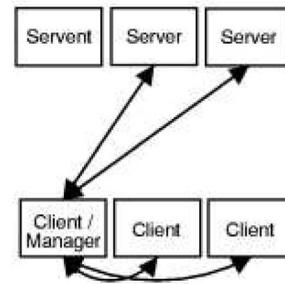

Fig. 4. Local manager PNR design (MAN).

the major drawback of this approach. Sending a message to each servent can introduce congestion into the system, especially if no real broadcasting facility exists and broadcasts must be implemented via multiple point-to-point messages.

In the backbone strategy (BB), only a subset of servents act as managers (see Fig. 5). This subset of servents is well known and all clients contact these servents for their network RAM requests. Clients associated with a new parallel job will contact a manager for service. As in the MAN design, all clients associated with a job must agree on which manager to contact. Managers will share memory load information by broadcasting to other managers. If the backbone of managers consists of only one member, then this strategy is equivalent to the centralized strategy. If the backbone of managers consists of all servents, then this strategy is equivalent to the local managers strategy.

This scheme can potentially be a "best of both worlds" solution, compared to the centralized and local managers solutions. It is more scalable than the centralized solution since load is shared among many servents, and it uses fewer messages for synchronization than the local managers solution since broadcast messages only need to be sent to a subset of nodes. It also has the advantage of being customizable to the cluster setup. If the network is small, then a small backbone (perhaps a backbone of one) may be all that is required. As the network becomes larger, then the number of servents in the backbone can be increased appropriately to manage scalability.

4 METHODOLOGY

We have created a simulator to test our proposed designs [23]. This section describes the models the simulator uses and the experiments we have designed.
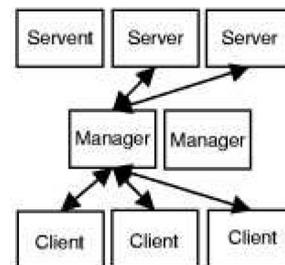

Fig. 3. Client-only PNR design (CLI).

Fig. 5. Backbone PNR Design (BB).

4.1 Models

Many workload traces and synthetic workload generators exist for use by parallel platform simulators. However, no standard memory usage benchmarks currently exist [24]. We use a large trace collected from the CM-5 parallel platform at the Los Alamos National Lab. This trace has been assembled and discussed by Feitelson [25], contains 201,387 jobs recorded through the majority of 1996, and is one of the few traces that records memory usage information.

We will use a subset of jobs from this well-studied trace. Since the workload profile at a given site tends to be fairly stable over time, using a subset of jobs should be indicative of the average load on the system [26]. However, using a workload trace may be biased toward that collected site's policies and not representative of all workloads [24], [27], [26].

To complement the CM-5 workload trace, we model a system architecture similar to the CM-5. Each node in the simulated system runs at 33 Mhz and has 32 MB of local memory. It is assumed each node has a disk of infinite capacity that runs at 7200 RPM with a seek time of 9 ms and has a transfer rate of 50 MB/s. The interconnecting network is assumed to be a simple Ethernet 100 Mbps star topology. Each link has a latency of 50 nanoseconds and the central switch has a processing delay of 80 microseconds. No collisions occur on the simulated links. Messages are queued if the link is currently in use. Broadcast messages are simulated using N point-to-point messages from the broadcasting node to each other node. Since the CM-5 had 32 processors dedicated solely to system tasks, it is assumed that the operating system of the CM-5 imposes no process or memory load on the nodes.

It is assumed there is one centralized scheduler for the system. In our simulations we experiment with two schedulers: a simple space-sharing scheduler and a gang scheduler. For both, we use first-come first-serve (FCFS) as our queuing discipline. FCFS has been shown to be a simple, efficient ordering which guarantees fairness [16]. Most simple node packing schemes lead to identical performance, so we use best-fit packing [10]. It is assumed that the schedulers have no knowledge of the memory requirements of jobs and, so, these requirements are not taken into account when scheduling decisions are made.

Each time slice in the gang scheduler runs for a 60 second quantum as suggested by [28]. The time required to perform a PCS is fixed at 4 ms [3]. The maximum number of time slices is set to two. This number is conservative and limits paging activity [16]. Alternative scheduling and slot unification are provided.

Each parallel job is composed of multiple parallel processes. It is assumed that each process allocates a static amount of memory at start time. Previous studies have shown that parallel scientific applications generate memory references every three to five CPU cycles and have a cache hit ratio that ranges from approximately 50 percent to 65 percent [29], [30]. We assume that our parallel applications access memory every four CPU cycles and have a cache hit ratio of 50 percent.

TABLE 1
Table of Experiments

RAM	150%, 135%, 125%, 115%, 100%, 85%, 75%, 65%, 50%
Network	10, 100, 1,000, 10,000, 100,000
Topology	Bus, Star, Connected
Space Sharing	50%, 75%, 100%

All processes in a job synchronize with each other at regular intervals. We use a simplistic master/worker synchronization pattern. One process is chosen as the "master" process and all other threads in the job are "workers." After a one CPU second, each worker sends a message to the master and waits for a response. After the master receives all messages, the workers are allowed to proceed.

4.2 Metrics

There are no universally valid and accepted metrics. In fact, different metrics can give contradictory results [31], [27], [32]. Our primary metric is average parallel job response time, or total wallclock time from submit to finish. This metric is used often but can overemphasize large jobs. In parallel workloads, small jobs account for the majority of jobs [10], [31], [27], [24].

To directly compare DP ("disk paging," a system without PNR) to the various PNR designs, we create another metric based on average response time (R): optimization ratio, which is defined as

$$Optimization\ Ratio(PNR\ Design) = \frac{R_{DP} - R_{PNR}}{R_{DP}} \times 100\%.$$

We also calculate the average and standard deviation of node memory allocation and disk allocation by sampling the memory allocation information of each node every 50,000 simulated time seconds. This metric will help determine if PNR is reducing disk usage and making memory usage more uniform across the cluster.

4.3 Experimental Setup

To determine the effectiveness of PNR in comparison to DP, we define several experiments. The basic set of experiments is defined in Table 1. These experiments reflect our main points of interest: performance under varying memory loads, under varying network speeds, under different network topologies, and under different scheduling strategies. The basic experiment table becomes multidimensional as we apply it to several different paging methods, workloads of 4,000 and 5,000 jobs, and to 64 and 128 node systems. This section describes and explains each experiment category and dimension.

To test each scheme under varying memory loads we vary the amount of RAM available at each node while holding memory demands of jobs constant. The parameter given signifies the relative amount of RAM present at each node in the experiment. For instance, in the 150 percent experiment, we adjust the default 32 MB of RAM to 48 MB (or 150 percent of the original value).

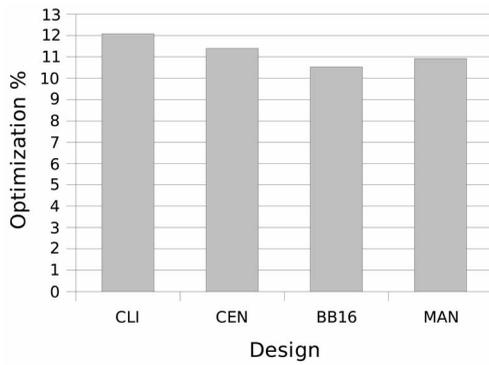To test each scheme under varying network performance, we alter link bandwidth and switch processing

Fig. 6. Base experiment—64 nodes and 4,000 jobs.

Fig. 7. Base experiment—128 nodes and 4,000 jobs.

delay. The base value of the network performance is 100 Mbps with an 80 microsecond switch processing delay. The parameter value 1,000 signifies performance 10 times the base value—with link bandwidth at 1,000 Mbps and switch processing delay at 8 microseconds. The values of 10,000 and 100,000 follow the same pattern. Parameter value 10 has link bandwidth of 10 Mbps but does not alter the switch processing delay.

We examine each scheme under three different topologies: a bus, a star, and a fully connected network. The star is the base topology and the connected network has two links per pair of nodes.

We examine each scheme under two different scheduling strategies: a gang scheduler and a space sharing scheduler. The gang scheduler is used as the base scheduler. The space sharing scheduler is simply the gang scheduler with a MPL of one. When we use the space sharing scheduler, we run the scheduler under varying RAM workloads. Following the same pattern as the gang scheduling experiments, we vary the RAM ratio from 50 percent to 100 percent. We do not simulate anything beyond 100 percent RAM because of the CM-5 scheduling policy which prevented any processes larger than the RAM available from running.

Each experiment defined in Table 1 is run on 4,000 and 5,000 job workloads. We discovered in early experiments that jobs start to become backlogged under the gang scheduler with the 5,000 job workload. By comparing the 4,000 and 5,000 job workloads we can understand how each solution performs under different types of system loads. To test scalability, each experiment is run on a 64 node cluster and a 128 node cluster.

We run five different paging methods against each experiment. The base method is disk paging (DP). We test four PNR methods: the centralized method (CEN), the client-only method (CLI), the local managers method (MAN), and a backbone method (BB). The backbone method defines the number of PNR managers to be $\frac{1}{4}$th the total number of nodes in the system. For instance, on the 64 node system, the number of managers is defined as 16 (BB16).

5 RESULTS

This section describes the results gathered from experiments described in Section 4. Because of the large volume of result information, this section demonstrates key concepts. It does not produce an exhaustive recounting of all
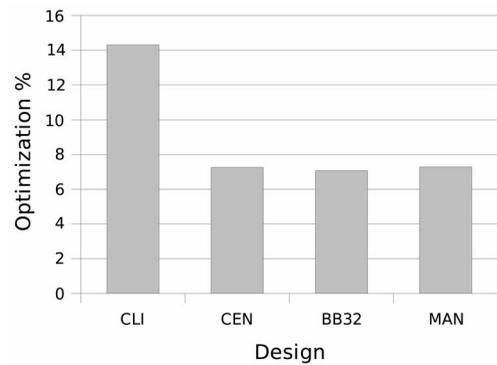
experiments. Section 5.1 describes the basic set of experiment results. Sections 5.2 and 5.3 describe results from experiments that varied the memory load of the basic system. Section 5.4 describes results from experiments that varied the network configuration of the basic system.

5.1 Base Experiments

We define the "base" set of experiments as experiments where all of the default parameters described in Section 4 are set. Note that we only describe the relative performance of PNR designs to DP. The absolute response times are very different depending on the workload.

Figs. 6 and 7 show that, for the light workload, each PNR design offers a performance improvement over DP. Interestingly, CLI offers the best optimization ratio in both cases. These results go against the expectation that CLI will always lead to inferior performance because of a lack of peer coordination.

Further, CLI did not lead to significantly uneven memory allocation as was expected. For the smaller cluster, all PNR designs had a comparable memory usage. For the larger cluster, CLI had the lowest average (17.25 percent) and standard deviation (13.59 percent) of memory allocation of all methods. For comparison, MAN had an average of 19.63 percent and a standard deviation of 14.53 percent. DP had an average of 23.56 percent and standard deviation of 16.34 percent.

This pattern changes when a heavier workload is introduced. Fig. 8 shows that, for the smaller cluster, BB16 is the best design at a 33.38 percent speedup over DP. This
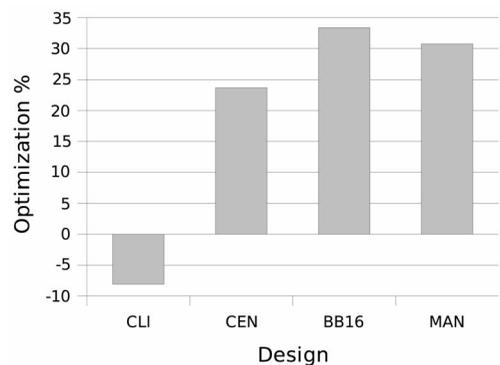

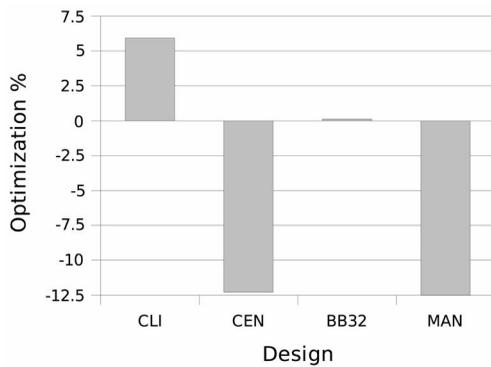Fig. 8. Base experiment—64 nodes and 5,000 jobs.

Fig. 9. Base experiment—128 nodes and 5,000 jobs.

Fig. 10. RAM experiments—64 nodes and 4,000 jobs.

result is more in line with our initial hypothesis where a hybrid of centralized and decentralized approaches would work best.

However, for the larger cluster (Fig. 9), CLI is the only method that experiences significant performance enhancement. The others experience no benefit or performance degradation. The underlying problem is the star network of the base experiment. This network has a bottleneck which limits network performance. As we will show in later experiments, PNR is very sensitive to network performance. Poor network performance will invariably lead to poor PNR performance. Because the CLI method uses the network the least, it manages to perform the best in this situation.

One important result to note is that the PNR methods will tend to significantly reduce the number of page faults experienced by the cluster. Depending on the experiment, the PNR clusters experienced approximately 50 percent to 80 percent the number of page faults DP experienced. This is because PNR jobs tend to finish faster. The faster the PNR jobs finish, the less PCSs they endure. This leads to less page loading due to memory contention of multiple processes.

We observe some interesting statistics for each PNR design with the heavier workload. For the smaller cluster, the average memory allocation for the PNR designs (45.15-51.15 percent) was lower than DP's (55.84 percent). However, all PNR designs expect MAN had a higher standard deviation (16.18-18 percent) compared to DP (16.15 percent). For the larger cluster, each PNR design had a lower memory usage standard deviation than DP. However, compared to DP (57.62 percent), some PNR designs had higher average memory allocations (MAN at 60.84 percent) while others did not (BB32 at 54 percent).

These memory usage statistics come as a surprise. It was expected that PNR would always smooth out memory usage. We expected lower memory usage averages and lower standard deviations in all or most cases.

To summarize:

- CLI can offer performance benefits comparable or superior to other PNR methods.
- The use of PNR leads to significantly less page faults.
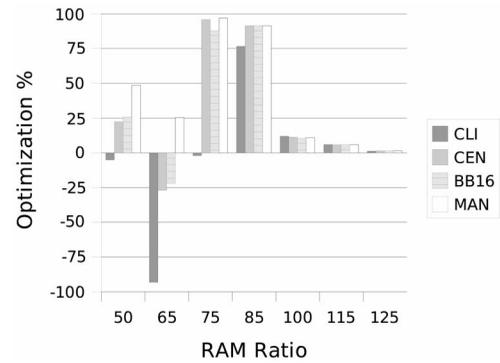- The use of PNR does not necessarily smooth out memory usage.

5.2 Load Variations

For this set of experiments, we vary the load on the simulated cluster by varying the amount of RAM at each node. Some observations can be made across all load experiments. First, we observe optimization ratios for all PNR designs converge to 0 percent as RAM is increased. In our figures, areas where the methods have converged to 0 percent are omitted. Second, as available RAM is decreased, all paging designs increase average response time exponentially. PNR provides a performance benefit under heavy loads, but is eventually eclipsed by DP when RAM is too scarce to share.

For the 128 node results, note that there is data missing at 65 percent and 50 percent RAM ratios for both the 4,000 and 5,000 job workloads due to simulation time constraints.

For the light workload (Figs. 10 and 11), a very large boost in performance is observed for RAM ratio 85 percent and 75 percent. These optimization ratios (approaching 100 percent) are among the highest observed for all experiments. At 65 percent RAM ratio and below, PNR performance is more variable—sometimes underperforming DP. Clearly, the PNR methods are beginning to converge to the DP response time since RAM is becoming too scarce to share.

For the heavier workload (Figs. 12 and 13), the PNR methods get a performance boost as RAM is initially increased and then eventually converge to 0 percent optimization ratio. As RAM is decreased, the pattern observed for the smaller workload is repeated. The only difference is that the performance benefits are less
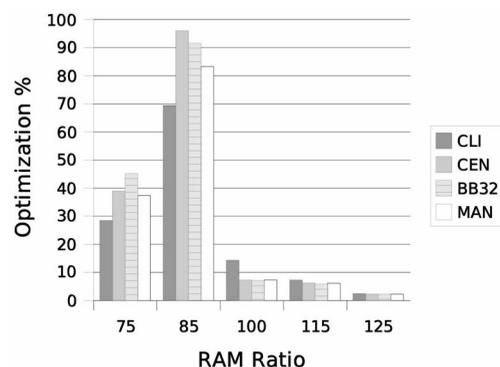

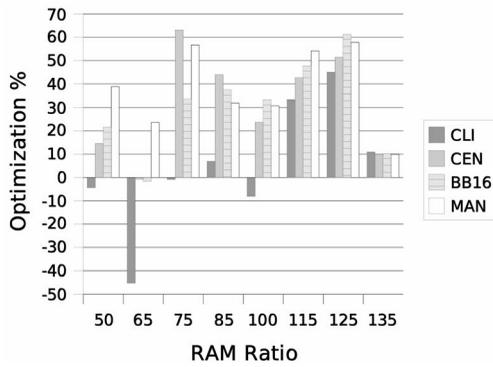Fig. 11. RAM experiments—128 nodes and 4,000 jobs.

Fig. 12. RAM experiments—64 nodes and 5,000 jobs.

Fig. 14. Space sharing experiments—64 nodes and 4,000 jobs.

pronounced. The maximum optimization ratio observed is approximately 60 percent.

Interestingly, for the larger cluster (Fig. 13) a large performance penalty is felt by all of the PNR designs. In terms of absolute response times, all methods improved between 115 percent and 125 percent RAM ratios, but DP improved more than all of the PNR methods. Since the absolute response times of all methods are smaller in this case, it is easier to get an exaggerated optimization ratio. More work will be needed to determine if this observed penalty is a serious problem.

Taking Figs. 10, 11, 12, and 13 into account, we can make multiple general conclusions:

- PNR offers its best relative benefit under moderate memory loads.
- PNR offers less relative benefit for very heavy and very light loads.
- CLI offers the least (or no) benefit under most scenarios.
- MAN is the only PNR method to consistently give positive performance results in all heavy load scenarios.
- The other PNR methods are more difficult to judge since their performance varies.

5.3 Load Variations with a Space Sharing Scheduler

Using a space-sharing scheduler changes load characteristics. For all workloads, recall that no individual job is larger than the CM-5's available RAM. With a space sharing
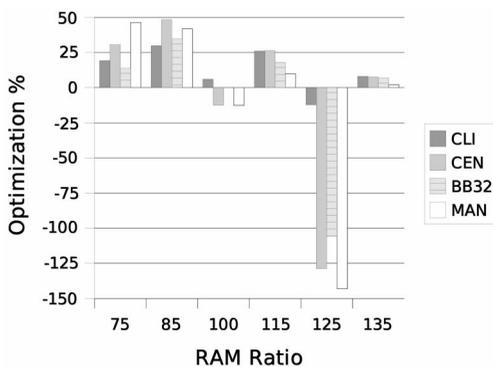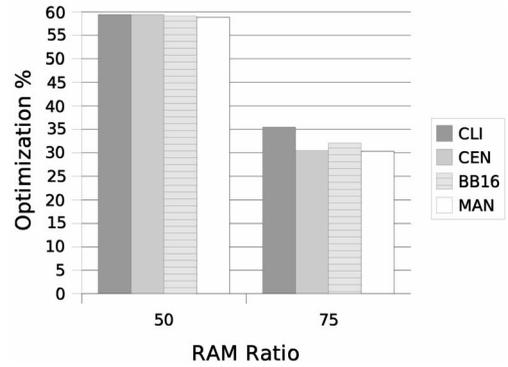
scheduler, no two jobs will be loaded on one node. Therefore, at RAM ratio 100 percent, no node is overloaded.

The same observations are true for all experiments shown in Figs. 14, 15, 16, and 17. Each PNR method offers a performance benefit over DP. CLI offers the best performance when load is lower (RAM ratio 75 percent) and the other PNR methods offer superior performance when load is higher (RAM ratio 50 percent). The size of the cluster or the workload only effects the magnitude of the performance benefit.

We can conclude the following:

- Coordinating network RAM allocation is essential as RAM becomes more scarce.
- CLI can provide acceptable or superior performance benefits under lighter loads.

5.4 Network Variations

For all cluster setups and workloads, we observe that PNR is extremely sensitive to poor network performance. The performance of PNR at 10 Mbps of bandwidth was so poor (an order of magnitude worse than DP), it was not included in the figures. The result of using a bus network on a large cluster (Figs. 23 and 25) was similar. Using a bus network for a smaller cluster introduced a smaller performance penalty, but its use for PNR is still not recommended.

Figs. 18 and 19 show the effect of increasing network performance under a lighter workload. Clearly, each PNR method becomes equivalent as network performance is increased. The coordination overhead of the various PNR methods must be the limiting factor in these

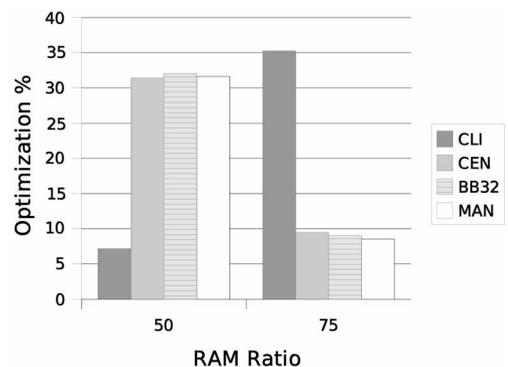Fig. 13. RAM experiments—128 nodes and 5,000 jobs.

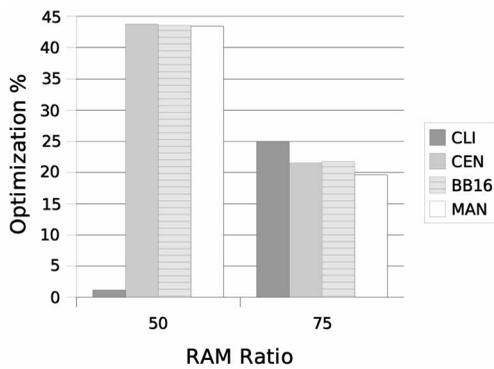Fig. 15. Space sharing experiments—128 nodes and 4,000 jobs.

Fig. 16. Space sharing experiments—64 nodes and 5,000 jobs.

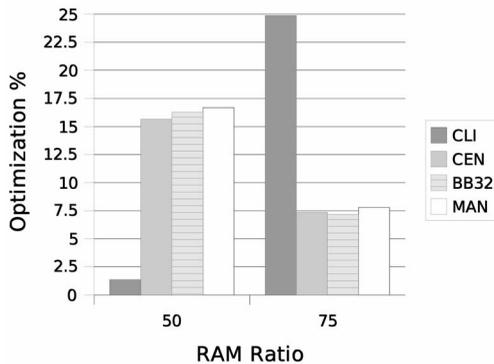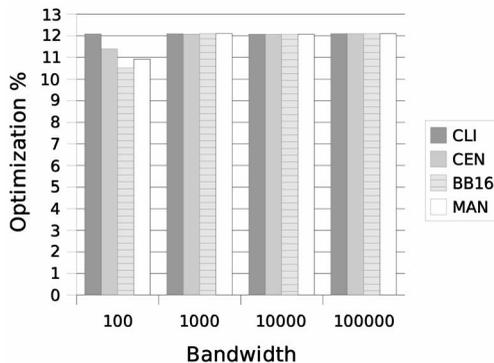Fig. 17. Space sharing experiments—128 nodes and 5,000 jobs.

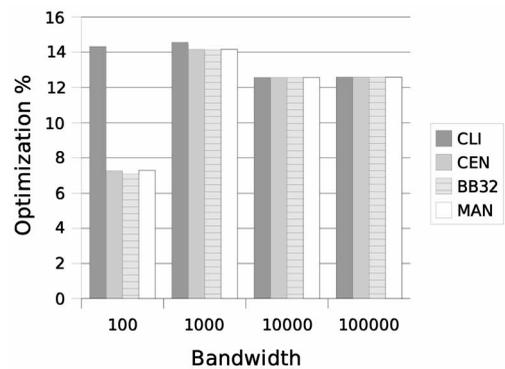Fig. 18. Network experiments—64 nodes and 4,000 jobs.

Fig. 19. Network experiments—128 nodes and 4,000 jobs.
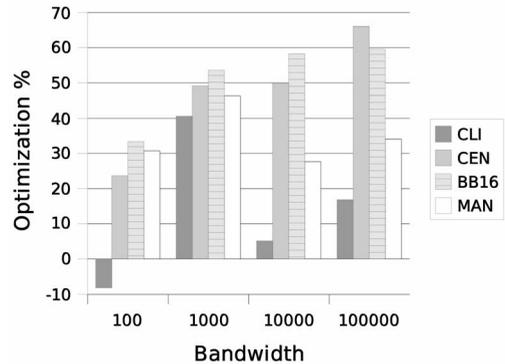

Fig. 20. Network experiments—64 nodes and 5,000 jobs.

Fig. 21. Network experiments—128 nodes and 5,000 jobs.

experiments. As communication improves, this factor decreases in importance.

Figs. 20 and 21 show the effect of increasing network performance for the heavier workload. The general trend is increased PNR performance given increased network performance. Generally speaking, CLI underperforms each other PNR method in these tests. Under this heavier workload, the main limiting factor is no longer coordination overhead. Instead, it is the scarcity of RAM. The methods that coordinate the allocation of memory exhibit superior performance compared to the method that does not.

Figs. 22 and 23 show the results of changing the network topology of the cluster under a lighter workload. For the connected network, we observe results similar to those at the high bandwidth network. Each PNR design has roughly equivalent performance on both the 64 and 128 node clusters. Since the connected network does not increase bandwidth and since the optimization ratio is similar to that

found on the high-performance networks, we can conclude that the significant factor in this experiment is congestion.

Experiments with different topologies under a heavier workload are shown in Figs. 24 and 25. The results under the connected network are very similar to the equivalent bandwidth experiments. CEN and BB16 come out as the best on the small cluster, and all methods are roughly equivalent on the large cluster.

A summary of network experiments is:

- PNR should not be used on low-performance networks.
- Under light load, the PNR methods' metrics converge as network performance is increased.
- Under heavy loads, CLI is inferior to the coordinating methods.
- Under heavy loads and on small clusters, CEN and BB16 seem to perform better as network performance is increased.
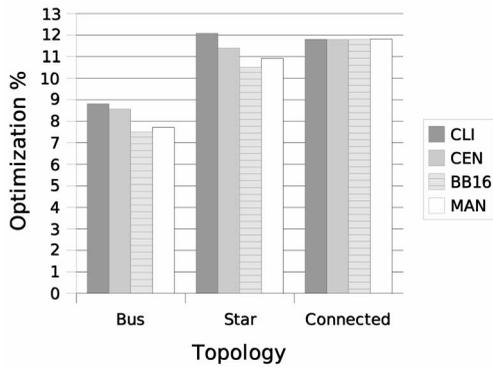
Fig. 22. Topology experiments—64 nodes and 4,000 jobs.

Fig. 24. Topology experiments—64 nodes and 5,000 jobs.

- Under heavy loads and on large clusters, all coordinating PNR methods have comparable performance as network performance is increased.

6 ANALYSIS

We have described our results in Section 5. We now interpret these results to discover general models. We also identify cluster configurations on which PNR would be useful.

Given our observations, it is clear that both PNR and DP follow an exponential curve as memory load is changed. As memory load increases, PNR and DP both tend toward infinite response times. As memory load decreases, PNR and DP response times will converge on a constant number. PNR does not offer a fundamentally different performance curve. It offers a curve lower than DP's within certain load bounds. Adding PNR to systems that are loaded within these bounds (and have adequate communication networks) should lead to a performance benefit. As observed, this improvement can approach 100 percent optimization ratio under moderate loads. However, as memory becomes too scarce to share, PNR performance will eventually be surpassed by DP performance.

PNR is very sensitive to network performance. The general shape of results is another exponential curve. PNR response time tends toward infinity as network service time is increased and converges to some constant number as service time decreases. DP does not follow this model. Since low network performance results in low PNR performance,
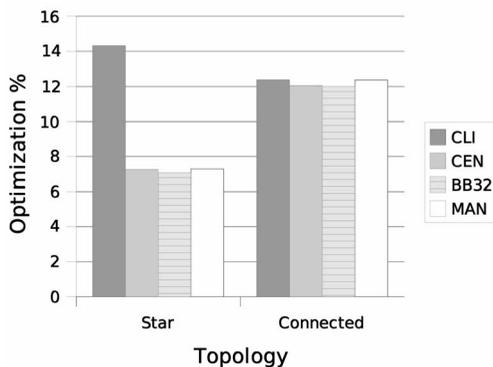
PNR should not be considered on networks with low bandwidth or communication bottlenecks. A standard 100 Mbps network with a scalable topology should be considered a minimum for satisfactory performance.

A space sharing system reduced overall memory load. Memory load is reduced because only one process is allowed on a node at a time. As RAM becomes more scarce, each PNR method except CLI appears to perform equivalently. The main limiting factor on the space sharing system is network RAM allocation coordination. Under heavy load, all of the schemes that use coordination appear to work equally well. Under light load, CLI is the best choice for a space sharing system. Depending on the workload and cluster setup, the best performance gains can be anywhere between 15-60 percent.

On the gang scheduling systems, network performance is crucial to PNR performance. When a high-performance network is available, PNR can produce pronounced performance gains. Under the lighter workload, the maximum optimization ratio observed was typically in the 10-12.5 percent range, although when RAM became scarce, optimization ratio could approach 100 percent. For the heavier workload, the maximum optimization ratio observed was 75 percent.

For heavily loaded systems, PNR can significantly reduce the response time of jobs as compared to DP. However, it may make sense to attempt to avoid the bulk of paging activity by using a space sharing system. The avoidance of paging can lead to better performance overall that PNR can enhance.
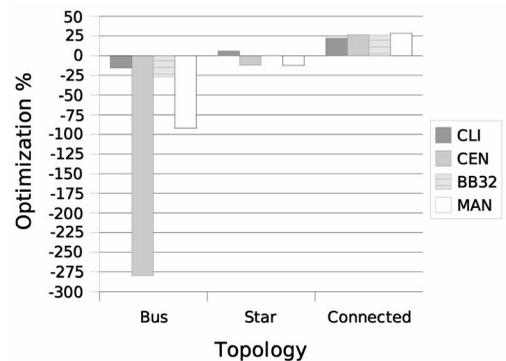

Fig. 23. Topology experiments—128 nodes and 4,000 jobs.

Fig. 25. Topology experiments—128 nodes and 5,000 jobs.

TABLE 2
PNR Design Usage

	Low Load	High Load
Fast Network	ANY	CEN, BBX, MAN
Slow Network	CLI	BBX

One surprising result of our simulations is that the proposed PNR designs do not always smooth out memory usage as measured by the standard deviation of system memory usage. For some experiments, PNR memory usage was even more nonuniform than DP's memory usage. This indicates that more work must be done to ensure PNR itself does not create more overloaded nodes.

The two main differentiating factors among the PNR designs are coordination of allocation of memory resources and the communication overhead required to coordinate. When network performance is high and RAM is relatively plentiful, we observe that each PNR design has equivalent performance. This is because coordination of memory resources is not a crucial issue when RAM is plentiful, and coordination overhead is less important with a fast network. Thus, the two differentiating factors are eliminated.

CLI does surprisingly well in certain situations. If RAM is plentiful, CLI benefits since it requires very little overhead to allocate network RAM. When the network is slow, CLI performs better than its counterparts because it contributes relatively little to network congestion. If it is known that a system has a slow network and will be lightly loaded, CLI may be the best method to implement due to its low overhead costs.

However, CLI performs poorly in heavy-load situations where RAM is scarce. In these situations, it becomes necessary for network RAM allocation to be coordinated to avoid wasting it. That leaves us with CEN, BBX, and MAN. When the network is fast, all three can potentially be used effectively. When the network is slow, only BBX and MAN should be considered since CEN will introduce significant congestion on the network resources that service the central server.

It is difficult to choose between the BBX and MAN methods. We will tentatively claim that BBX has an advantage over MAN in slow networks since fewer messages must be broadcast for synchronization purposes.

A summary of our recommendations is in Table 2.

7 CONCLUSION AND FUTURE WORK

In this paper, we identified a novel way of reducing page fault service time and average response time in a cluster system running parallel processes. This method, which we call Parallel Network RAM, uses remote idle RAM as another tier in the memory hierarchy for parallel jobs in clusters. We proposed several different PNR designs and evaluated the performance of each under different conditions. We discovered that different designs are appropriate in different situations and that simply applying PNR to an existing system is not a guarantee of increased performance.

We can condense our findings to a core set of observations:

- Using a coordinating PNR method under heavier loads is essential for good performance.
- Coordinating PNR methods offer the best performance enhancement when under moderate load. Performance gains can be as high as 100 percent.
- CLI can provide acceptable or superior results under light load only.
- All PNR methods offer little benefit under very heavy or very light loads.
- Good network performance is crucial for good PNR performance.

This paper only used a specific subset of a single trace from a particular parallel system. To our knowledge, traces such as the one we used that include memory information are very rare. It would be invaluable to collect a new trace on a modern parallel system running cutting-edge parallel programs. Not only would this allow us to compare how PNR acts on a modern system, it would also allow us to calibrate our simulator and ensure that the models used (e.g., page fault models) are realistic.

The simulator can be improved in many ways. It currently only simulates jobs that use a static amount of memory determined at runtime. It would be useful to know how PNR performs given a more realistic dynamic memory workload. It also simulates a very simplistic job synchronization model. A more intense and complex model would give us more realistic results. It only simulates very simple schedulers. Adding more sophisticated features, such as job migration or memory usage estimates would be fruitful.

REFERENCES

[1] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, K.C. Sevcik, and P. Wong, "Theory and Practice In Parallel Job Scheduling," *Job Scheduling Strategies for Parallel Processing,* pp. 1-34, 1997.
[2] A. Batat and D.G. Feitelson, "Gang Scheduling with Memory Considerations," *Proc. IEEE 2000 Int'l Parallel and Distributed Processing Symp.,* pp. 109-114, 2000.
[3] D.C. Burger, R.S. Hyder, B.P. Miller, and D.A. Wood, "Paging Tradeoffs in Distributed-Shared-Memory Multiprocessors," *The J. Supercomputing,* vol. 10, no. 1, pp. 87-104, 1996.
[4] E.W. Parsons and K.C. Sevcik, "Benefits of Speedup Knowledge in Memory-Constrained Multiprocessor Scheduling," *Performance Evaluation,* vols. 27/28, no. 4, pp. 253-272, 1996.
[5] A. Acharya and S. Setia, "Availability and Utility of Idle Memory in Clusters of Workstations," *Proc. ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems,* 1999.
[6] M.D. Flouris and E.P. Markatos, *High Performance Cluster Computing.* Prentice Hall, pp. 383-408, 1999.
[7] M.J. Feeley, W.E. Morgan, F.H. Pighin, A.R. Karlin, H.M. Levy, and C.A. Thekkath, "Implementing Global Memory Management In A Workstation Cluster," *Proc. Symp. Operating Systems Principles,* pp. 201-212, 1995.

[8] E.P. Markatos and G. Dramitinos, "Implementation of a Reliable Remote Memory Pager," *Proc. USENIX Ann. Technical Conf.,* pp. 177-190, 1996.

[9] L. Xiao, X. Zhang, and S.A. Kubricht, "Incorporating Job Migration and Network Ram to Share Cluster Memory Resources," *Proc. Ninth IEEE Int'l Symp. High Performance Distributed Computing,* pp. 71-78, 2000.

[10] D.G. Feitelson, "Packing Schemes for Gang Scheduling," *Job Scheduling Strategies for Parallel Processing,* D.G. Feitelson and L. Rudolph, eds., pp. 89-110, 1996.

[11] B.B. Zhou, D. Welsh, and R.P. Brent, "Resource Allocation Schemes for Gang Scheduling," *Job Scheduling Strategies for Parallel Processing,* D.G. Feitelson and L. Rudolph, eds., pp. 74-86, 2000.

[12] A.B. Downey, "Lachesis: A Job Scheduler for the Cray T3E," *Job Scheduling Strategies for Parallel Processing,* D.G. Feitelson and L. Rudolph, eds., pp. 47-61, 1998.

[13] A. Hori, H. Tezuka, and Y. Ishikawa, "Overhead Analysis of Preemptive Gang Scheduling," *Job Scheduling Strategies for Parallel Processing,* D.G. Feitelson and L. Rudolph, eds., pp. 217-230, 1998.

[14] Y. Wiseman and D. Feitelson, "Paired Gang Scheduling," Technical Report 2001-10, Hebrew Univ., 2001.

[15] S.K. Setia, "The Interaction between Memory Allocation and Adaptive Partitioning in Message-Passing Multicomputers," *Job Scheduling Strategies for Parallel Processing,* D.G. Feitelson and L. Rudolph, eds., pp. 146-164, 1995.

[16] Y. Zhang, A. Sivasubramaniam, H. Franke, and J.E. Moreira, "Improving Parallel Job Scheduling by Combining Gang Scheduling and Backfilling Techniques. " *Proc. IEEE 2000 Int'l Parallel and Distributed Processing Symp.,* pp. 133-142, 2000.

[17] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, "Characterization of Backfilling Strategies for Parallel Job Scheduling," *Proc. 2002 Int'l Workshops Parallel Processing,* 2002.

[18] E.W. Parsons and K.C. Sevcik, "Coordinated Allocation of Memory and Processors in Multiprocessors," *Proc. 1996 ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems,* pp. 57-67, 1996.

[19] F. Wang, M. Papaefthymiou, and M. Squillante, "Performance Evaluation of Gang Scheduling for Parallel and Distributed Multiprogramming," *Job Scheduling Strategies for Parallel Processing,* D.G. Feitelson and L. Rudolph, eds., pp. 277-298, 1997.

[20] A. Barak and A. Braverman, "Memory Ushering in a Scalable Computing Cluster," *J. Microprocessors and Microsystems,* vol. 22, nos. 3-4, pp. 175-182, Aug. 1998.

[21] L. Xiao, S. Chen, and X. Zhang, "Dynamic Cluster Resource Allocations for Jobs With Known and Unknown Memory Demands," *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 3, pp. 223-240, Mar. 2002.

[22] L. Xiao, S. Chen, and X. Zhang, "Adaptive Memory Allocations in Clusters to Handle Unexpectedly Large Data-Intensive Jobs," *IEEE Trans. Parallel and Distributed Systems,* vol. 15, no. 6, June 2004.

[23] J. Oleszkiewicz and L. Xiao, "Pnrsim: A Parallel Network Ram Simulator," Technical Report MSU-CSE-04-13, Computer Science and Eng., Michigan State Univ., East Lansing, Apr. 2004.

[24] S.J. Chapin, W. Cirne, D.G. Feitelson, J.P. Jones, S.T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby, "Benchmarks and Standards for the Evaluation of Parallel Job Schedulers," *Job Scheduling Strategies for Parallel Processing,* D.G. Feitelson and L. Rudolph, eds., pp. 67-90, 1999.

[25] D.G. Feitelson, "Memory Usage in the LANL CM-5 Workload," *Job Scheduling Strategies for Parallel Processing,* D.G. Feitelson and L. Rudolph, eds., pp. 78-94, 1997.

[26] V. Lo, J. Mache, and K. Windisch, "A Comparative Study of Real Workload Traces and Synthetic Workload Models for Parallel Job Scheduling," *Job Scheduling Strategies for Parallel Processing,* D.G. Feitelson and L. Rudolph, eds., pp. 25-46, 1998.

[27] D.G. Feitelson and L. Rudolph, "Metrics and Benchmarking for Parallel Job Scheduling," *Job Scheduling Strategies for Parallel Processing,* pp. 1-24, 1998.

[28] U. Schwiegelshohn and R. Yahyapour, "Improving First-Come-First-Serve Job Scheduling by Gang Scheduling," *Job Scheduling Strategies for Parallel Processing,* D.G. Feitelson and L. Rudolph, eds., pp. 180-198, 1998.

[29] F. Darema-Rogers, G.F. Pfister, and K. So, "Memory Access Patterns of Parallel Scientific Programs," *Performance Evaluation Rev.,* vol. 15, no. 1, pp. 46-57, 1987.

[30] K.C. Sevcik and S. Zhou, "Performance Benefits and Limitations of Large NUMA Multiprocessors," *Performance Evaluation,* vol. 20, pp. 185-205, 1994.

[31] D.G. Feitelson, "Metrics for Parallel Job Scheduling and Their Convergence," *Job Scheduling Strategies for Parallel Processing,* D.G. Feitelson and L. Rudolph, eds., pp. 188-205, 2001.

[32] J.P. Jones and B. Nitzberg, "Scheduling for Parallel Supercomputing: A Historical Perspective Of Achievable Utilization," *Job Scheduling Strategies for Parallel Processing,* D.G. Feitelson and L. Rudolph, eds., pp. 1-16, 1999.

[33] J. Oleszkiewicz, L. Xiao, and Y. Liu, "Parallel Network Ram: Effectively Utilizing Global Cluster Memory for Large Data-Intensive Parallel Programs," *Proc. 2004 Int'l Conf. Parallel Processing,* pp. 353-360, 2004.

John Oleszkiewicz received the BS degree in computer science from Grand Valley State University in Allendale, Michigan in 2002. He received the MS degree in computer science from Michigan State University in 2004. He is currently employed as a software engineer at Smiths Aerospace in Grand Rapids, Michigan. His research interests include computing clusters, high-performance computing, distributed systems, and peer-to-peer computing. He is a member of the IEEE.

Li Xiao received the BS and MS degrees in computer science from Northwestern Polytechnic University, China, and the PhD degree in computer science from the College of William and Mary in 2002. She is an assistant professor of computer science and engineering at Michigan State University. Her research interests are in the areas of distributed and Internet systems, overlay systems and applications, system resource management, and design and implementation of experimental algorithms. She is a member of the ACM, the IEEE, the IEEE Computer Society, and IEEE Women in Engineering.

Yunhao Liu received the BS degree in automation from Tsinghua University, China, in 1995, the MA degree from Beijing Foreign Studies University, China, in 1997, and the PhD degree in computer science from Michigan State University in 2004. He was a regional manager in China Telecom from 1997 to 1998, and a Deputy Director of E-Post office in the State Postal Bureau of China from 1998 to 2001. He is now an assistant professor in the Department of Computer Science at Hong Kong University of Science and Technology. His research interests are in the areas of peer-to-peer computing, pervasive computing, distributed systems, network security, grid computing, and high-speed networking. He is a member of the IEEE and the IEEE Computer Society.

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