# Parallel Network RAM: Effectively Utilizing Global Cluster Memory for Large Data-Intensive Parallel Programs

John Oleszkiewicz, Li Xiao and Yunhao Liu
Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824 USA
{oleszkie, lxiao, liuyunha}@cse.msu.edu

## 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 and better utilizes available RAM resources. This approach will allow larger problems to be solved while reducing the computational, communication and synchronization overhead typically involved in parallel applications.*

## 1. Introduction

Many scientific computing applications demand a large amount of processing power, memory space and I/O accesses. One standard approach to alleviate processor and memory load is to parallelize these applications into multiple parallel processes so that the may be run on multiple nodes in a computing cluster. An advantage of this approach is CPU and memory resources are evenly distributed and used. Another advantage is the nature of load balancing in parallel computing. However, these two advantages may not serve the best performance interests of these parallel processes because the balanced workload distribution among processes may result in unbalanced resource utilization in a cluster. Specifically, there are trade-offs between the number of parallel jobs and memory usage.

For a given problem size, in order to ensure each node has enough memory space, the number of parallel processes in a parallel job may be set very high. This results in less work and more required synchronization per parallel process The CPU on each node may be underutilized. Parallel speedup is very hard to improve as the number of parallel processes increases to a certain value, due to increasing communication and synchronization overhead. Good performance cannot be guaranteed by using a large number of nodes.

On the other hand, if the number of parallel processes is limited, the CPU will be better utilized but nodes may run out of available memory and may be forced to use the local disk as a swapping site. 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 [3, 4, 15].

Network RAM [1, 11] has been proposed to schedule 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. The remote RAM is treated as a new layer in the memory hierarchy between RAM and disk. Resulting page accesses are slower than RAM, but much faster than disk [11, 23].

Network RAM techniques cannot be directly applied to parallel jobs to achieve satisfactory performance for several reasons. One serious issue is that parallel processes from the same parallel job synchronize regularly. If each node seeks network RAM independently, it is likely that an uneven amount of network RAM will be granted to the nodes. With this uneven allocation, parallel processes will run at different speeds. However, parallel jobs as a whole 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 parallel processes individually seek out network RAM with no coordination among themselves, a potentially large amount of network traffic will result. This may induce congestion on the cluster. Parallel applications require high performance networks to run efficiently. Congestion could seriously impact the performance of jobs on the system.

We propose a new peer-to-peer solution, called Parallel Network RAM (PNR), so parallel jobs can utilize idle remote memory. In this scheme, each node requests memory resources as well as provides memory for others. Requests are indirect in that each node contacts a local 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 load is 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 time and higher system throughput.

We make several contributions in this paper.

- We show that existing techniques cannot maximize the performance gain of parallel jobs in terms of both parallel speedup and execution time given a cluster with unbalanced resource utilization.
- We propose a novel and effective solution to this problem called Parallel Network RAM (PNR). PNR coordinates the allocation of network RAM to overcome limitations of past solutions. This allows CPU cycles to be provided by a small subset of nodes, while the global memory space is open to the memory demand of any parallel job.
- We build a simulator that models a cluster and our proposed PNR algorithms. Conducting trace-driven simulations, we compare the performance of the PNR system to a disk paging-only solution (DP) and a network RAM solution similar to previous work (NR).

The rest of the paper is organized as follows. Section 2 describes background information related to our work. Section 3 presents our proposed PNR algorithms. Section 4 describes our trace-driven simulator and justifies our simulation parameters and metrics. Section 5 presents our performance evaluation. Section 6 discusses the implications of our simulation results and section 7 concludes the paper.

## 2. Background and Related Work

The majority of work in this area has focused on parallel job scheduling. In this section, we describe various parallel job schedulers and previous solutions to the problem of overloaded memory on cluster systems.

### 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 simple scheduling system is a space sharing system. Space sharing allows more than one job to be scheduled on the multicomputer 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 jobs monopolizing the system.

Gang scheduling combines both space sharing and time sharing. Each job is alloted 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. The maximum number of time slots allowed is known as the maximum multiprogramming level (MPL). Setting the MPL to a low level is a convenient way to reduce PCS overhead and reduce overall memory load on nodes. A system with an MPL value of one is a simple space sharing system with no time sharing.

There are a variety of processor allocation strategies and variations of gang scheduling. These include first fit, best fit, left-right by size, left-right by slots, load-based allocation strategies, ”buddy” systems such as distributed hierarchical control, and migration-based algorithms [8, 24].

### 2.2. Previous Solutions to the Memory Problem

Previous studies agree that unmodified disk paging results in severely reduced performance [4, 18]. 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 scheduler have been suggested. If no memory information about incoming jobs is known, then the simplest solution is to keep the MPL to a minimum [14]. Another solution attempts to guess memory usage information based on information about the job provided by the user and information contained in the program executable. The solution then uses this information in scheduling decisions [3]. If other information on jobs, such as speedup information, is known ahead of time, then another solution can use that information [15].

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 [20].

Much work has been done for sequential job scheduling with memory considerations. Regarding network RAM implementations, the Global Memory System (GMS) [7] and the Remote Memory Pager [13] attempt to reduce the page fault overhead by using remote paging techniques. DoDo [1] is designed to improve system throughput by harvesting idle memory space in a distributed system. Here, the owner processes have the highest priority for using the 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 [2]. This solution is a job-migration-based load sharing approach.

Recently, we have developed several load sharing alternatives by considering both CPU and memory resources with known and unknown memory demands [21, 22]. The objective of our 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. Parallel Network RAM

We propose a novel and effective technique called Parallel Network RAM (PNR) to better utilize the resources of both CPU and memory and minimize communication and synchronization overhead. Our objective is to fundamentally improve the efficiency of large scale scientific computing. Instead of evenly allocating parallel tasks among the nodes, we consider CPU and memory resource allocations separately. Under this non-uniform scheme, the number of CPUs to be assigned to a parallel job will be optimized in order to better utilize increasingly powerful CPU cycles and to minimize the communication and synchronization overhead among cluster nodes. Each job can utilize both the local memory space from the assigned CPUs and the remote memory space in other nodes. With PNR, speedup can be scaled very well as the available remote memory increases, and performance can also be scaled very well as the problem size increases.

It is assumed that the PNR algorithms are implemented as a subsystem or module of each node's kernel. PNR runs as just another part of the virtual memory system. This minimizes disruption to established cluster systems. Parallel applications will not have to be re-compiled with special PNR libraries and the execution of remote paging will be transparent to them. The assumed centralized scheduler of the system will not have to be changed since PNR is not a part of it and will not coordinate with it in any way.

In brief, all nodes host PNR servents. All servents act as PNR clients and servers. Some servents may act as managers. When a new process is started, nodes participating in that job will contact their local PNR client if in need of additional memory resources. Clients will contact their local managers and managers will contact PNR servers on the behalf of multiple clients. Similarly, when a job is stopped, nodes will notify their clients if they no longer need the allocated network RAM. Clients will contact their managers and the managers will contact servers on the clients' behalf.

Previous work, such as [1, 7], has introduced centralized proxies to network RAM allocation and management. However, to our knowledge, the use of distributed proxies to coordinate network RAM allocation for threads within a parallel job is unique.

### 3.1. Architecture Detail

A PNR client attempts to allocate and deallocate network RAM on the behalf of its hosting node. The node treats allocated network RAM as additional virtual memory - just as it would with disk space. When a process starts execution on a node, it allocates the amount of memory it will use during its execution. If the node's PNR client determines that this allocation will lead to disk usage, the client contacts a manager and requests network RAM. Once network RAM is allocated, the client is informed by the manager what machines are serving network RAM. The client may then start sending pages to the server(s) for storage and later retrieval. A similar process is followed for network RAM deallocation.

PNR managers do the majority of allocation and deallocation work. Managers take requests from multiple clients and send requests to PNR servers on the clients' behalf. Managers exist to balance memory requests from multiple parallel processes that belong to one job.

When a new job starts, one of the job's servents volunteers to be a PNR manager. The volunteering is randomized but all clients affected by the new job must agree on which servent is the manager. Managers decide which PNR servers to contact based on local memory load tables. These tables are maintained via extra information sent in by PNR communications. Specifically, every message sent to a manager will also include the sending node's current memory load. Managers record this information and broadcast their updated load tables to all servents after each job start and stop. In this way, all managers will have a general idea of memory allocation on each node. Managers use a randomized worst-fit algorithm for memory allocation and will pick the servers they believe have the largest amount of unallocated memory.

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. Currently the unallocated memory threshold is set to a low value - a tenth of a megabyte. This parameter can be changed if mem-

ory appears to be consistently overallocated on PNR servers. After the memory is allocated, servers receive requests to read and write the allocated network RAM directly from clients. Servers grant all valid deallocation attempts.

## 4. Simulation Methodology

We have developed a trace-driven simulator to experiment with clusters and their interaction with the proposed PNR algorithms. This section describes the simulator and the experimental environment.

### 4.1. Workload Model

Many workload traces and synthetic workload generators exist for use by simulators. However, no standard memory usage benchmarks currently exist [5]. We decided to use a large trace with memory allocation information that has been assembled and discussed by Feitelson [9]. The trace has been gathered from the CM-5 parallel platform at the Los Alamos National Lab. It contains 201,387 lines, each of which represents information of one parallel job (e.g. number of processors used, CPU seconds, amount of memory allocated, etc.) This information was recorded through the majority of 1996.

We will use a subset of jobs from this well-studied trace. This subset is arbitrarily chosen as the first 5000 jobs of the workload. Since the workload profile at a given site tends to be fairly stable over time [12], using a subset of jobs should be indicative of the load on the system.

### 4.2. System Model

We model a system architecture similar to the CM-5 but change some system parameters where possible to model a more modern cluster architecture. Just as the trace assumes, each node runs at 33 Mhz and has 32 MB of local memory. An addition to the modeled system compared to the CM-5 are hard drives local to each node. It is assumed each disk has an infinite capacity and runs at 7200 RPM with a seek time of 9ms and has a transfer rate of 50 MB/s. It should be noted that these performance figures are representative of current commodity hard drives (example: [17]) and are not performance figures from the time period in which the CM-5 was used. The interconnecting network is assumed to be a simple Ethernet 100 Mbps star topology. Each link has a latency of 50ns and the central switch has a processing delay of 80 microseconds. For the sake of simulator speed, we scale down the trace from its original 1024 nodes to 64 nodes. Job processor usage is scaled down accordingly. Note that other job characteristics, such as memory usage and communication frequency, are held constant.

### 4.3. Scheduler Models

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 schedulers, we use FCFS as our queuing discipline and a simple best-fit node packing scheme. It is assumed that the scheduler has no knowledge of the memory requirements of jobs.

Each time slice in the gang scheduler runs for a 60 second quantum as suggested by [16]. The time required to perform a PCS is fixed at 4ms [4]. The maximum number of time slices is set to two. This number is conservative and limits paging activity [14]. Alternative scheduling and slot unification are provided.

### 4.4. Job Behavior Model

Each parallel job is composed of multiple parallel processes. It is assumed that each process allocates a static amount of memory at start time. Each process accesses memory at its node independently. Previous studies have shown that parallel scientific applications generate memory references every three to five CPU cycles. They also suggest that the cache hit ratio for these applications ranges from approximately 50% to 65% [6, 19]. We assume that our parallel applications access memory every four CPU cycles and have a cache hit ratio of 50%.

All processes in a job synchronize with each other at regular intervals. The synchronization pattern is a simple master/worker pattern, where multiple workers send and receive messages from one master. In our experiments, we set the time interval between synchronizations to be one CPU second (once every 32 million cycles). This is not a heavy synchronization load. Future experiments should use more complex and more frequent loads.

It is assumed no paging activity will result when all jobs fit into available memory on each node. When more memory is allocated than is available, there is a chance a non-cache memory access will result in a page fault. We use an exponential function that is dependent on both the average memory access rate and the amount of process memory currently paged out to determine the time until the next page fault.

### 4.5. Metrics

There are no universally valid and accepted metrics. In fact, different metrics can give contradictory results [10]. In this section we list and justify the metrics we used.

First is average response time, or total wallclock time from submit to finish. This metric is used very often but can

overemphasize large jobs. In parallel workloads, small jobs account for the majority of jobs. [10]

Second, to directly compare DP to PNR, we create another metric: "optimization rate". This metric represents the improvement of PNR over DP.

$OptimizationRate = \frac{R_{disk} - R_{pnr}}{R_{disk}} \times 100\%$

Third, we calculate the average and standard deviation of node memory allocation by sampling the memory allocation information of each node every 50,000 simulated time seconds.

## 5. Performance Evaluation

We have defined a set of experiments to evaluate PNR. Specifically, we defined a base experiment and explored variations on this base experiment to gain insight into PNR performance characteristics. We will first describe the results from the base experiment and follow with results from the variations. We compare a system with PNR to a system without PNR that we call "Disk Paging" (DP) since this system's only way of dealing with overallocation is using the disk. We also compare PNR to a system which uses uncoordinated network RAM allocation (NR). In NR, each node will independently seek network RAM. This system is conceptually identical to the system proposed in [13]. By comparing PNR to DP, we demonstrate that PNR has tangible performance benefits over using disk paging alone. By comparing PNR to NR, we demonstrate that the extra coordination used in PNR is essential to improving the performance of network RAM on cluster systems.

### 5.1. Base Experiments

In our base experiment, all parameters are set to the values described in the methodology section. Figure 1 demonstrates that PNR performed comparatively well in terms of response time. DP had an average response time of 60974.8 seconds, NR had an average response time of 65912.1 seconds, and PNR had an average response time of 42226.2 seconds. Optimization rate for PNR is a modest 31% while it is -8.1% for NR!

PNR also made better use of available RAM resources. DP used an average of 17.87 MB (55.8%) per node with a standard deviation of 5.17 MB, NR an average of 16.37 MB (51%) with a standard deviation of 5.75 MB, and PNR an average of 14.71 MB (45.9%) with a standard deviation of 4.98 MB. Average disk usage was also down and more uniform. DP used, on average, 3.51 MB of disk space per node with a standard deviation of 1.51 MB, NR an average of 2.97 MB and standard deviation of 1.92 MB, and PNR an average of 1.91 MB with a standard deviation of 1.13 MB. These results indicate that the overall system load is lower

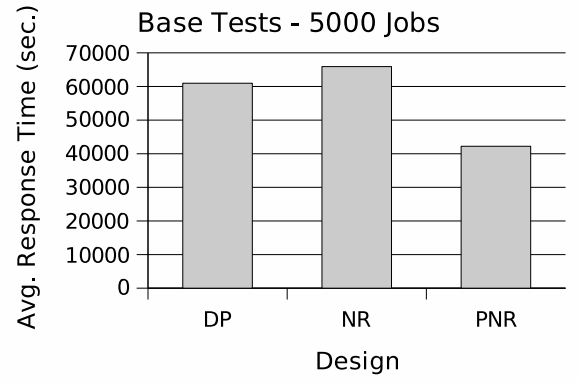


**Figure 1. Response time of base test with 5000 job workload.**

with PNR due to a higher job turnover rate, and that allocation of memory is becoming more uniform - one of the goals of PNR.

PNR experienced only 65% the number of page faults that DP experienced. This can be attributed to the fact that, since PNR jobs will finish faster, they will experience less PCSs. This in turn leads to less page loading due to memory contention by multiple processes.

### 5.2. Network Performance Variations

When we reduce the bandwidth of the links in the standard star topology to 10 Mbps, we observe a dramatic impact on PNR (not shown in figures). For all workloads PNR performs significantly worse than DP. Optimization rate may be anywhere between -10% to -1000%. Response times for PNR reach as high as three million seconds, while the highest DP response times are easily within 100,000 seconds. Significantly, this is the only set of experiments in which PNR is substantially outperformed by DP.

Figure 2 shows that the optimization rate for PNR is consistently superior to DP and NR at high network bandwidths. Interestingly, however, PNR and NR performance decreases at some points as bandwidth increases. The cause of the increased response times is apparently increased disk usage for both NR and PNR. As network performance passes a certain threshold, disk usage goes up. This phenomena makes sense if we consider that if processes are becoming backlogged in this large trace, and if increased network performance results in higher turnover, then longer-running jobs will tend to start running together in greater frequency than under the lower-performing networks. Longer running jobs tend to demand more memory than shorter jobs and when larger jobs start running together more frequently, memory load and response times increase.

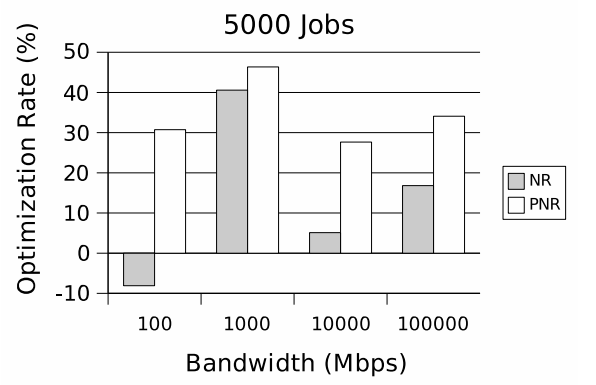


**Figure 2. Optimization rate at varying network speeds with 5000 jobs**

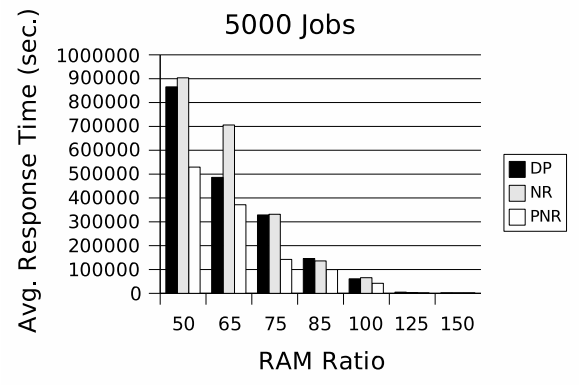


**Figure 4. Response time with varying amount of RAM available with 5000 jobs.**

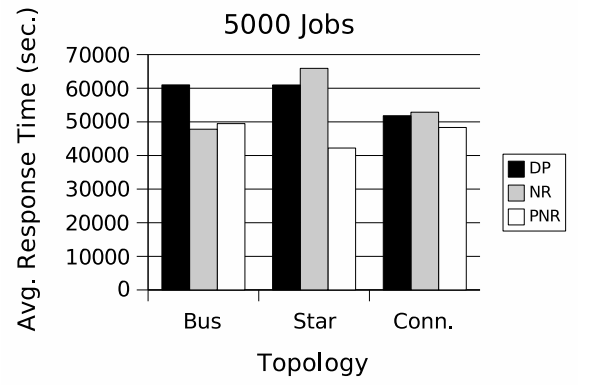


**Figure 3. Response time of methods under different topologies.**

### 5.3. Network Topology Variations

Figure 3 displays the results over various network topologies. Bus and star behave as expected for both DP and PNR. Interestingly, NR has a slight advantage over PNR in the bus environment. However, with the connected topology, PNR response time rises compared with star while DP and NR experience performance benefits. We believe that the same phenomena which caused the performance penalty in high bandwidth networks is at work here.

### 5.4. RAM Variations

We varied the amount of total RAM available at each node to simulate different memory loads and observe how DP and PNR react. The RAM ratio of the x-axis in figures 4-6 is a multiplier we apply to the standard amount of RAM available in the base tests. For instance, RAM ratio 1.5 results in nodes with 48 MB of RAM (up from the original 32 MB).

Two observations are obvious: as memory load decreases, PNR, NR, and DP converge to the same performance metrics and as memory load increases, PNR and NR/DP metrics diverge. When no paging activity occurs, all of the methods are equivalent in performance. For instance, we observe in figure 4 that as the RAM ratio is increased from 0.5 to 1.5, response times for each converge to 1960.6 seconds. This is an important result, since it demonstrates that the extra communication overhead of PNR adds only a negligible amount to response time in situations of light load.

In figure 5, we see the optimization rates converging when the RAM ratio is at 1.35 and higher. PNR does particularly well as RAM is initially decreased (0.75) and increased (1.15). As the RAM ratio is lowered, however, some of the initial advantage is lost We believe that PNR and DP will eventually converge again as RAM becomes so scarce that it can never be shared via PNR. NR has consistently lower optimization rates compared to PNR

### 5.5. Space Sharing Scheduler Experiments

We repeat the base set of experiments and the RAM variation experiments with a simple space sharing scheduler. For all inputs and for all variations of memory load tested, a pattern emerges. NR has a slight advantage under light memory loads (0.75 RAM ratio) but is consistently and significantly outperformed by PNR at heavier memory loads (see figure 6).

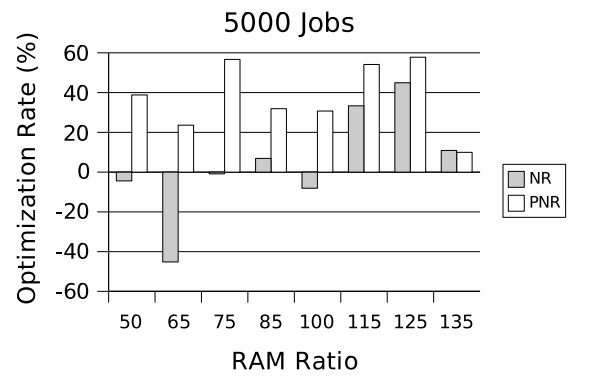


**Figure 5. Optimization rate with varying amounts of RAM with 5000 jobs.**

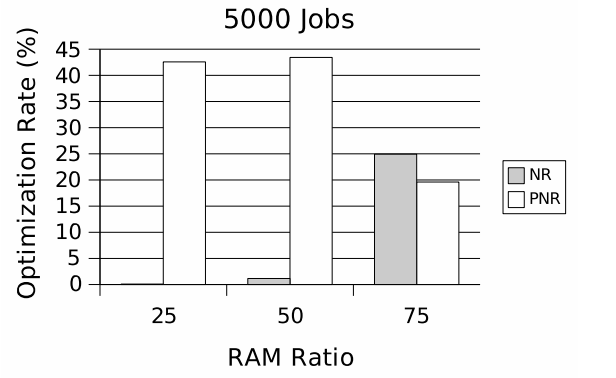


**Figure 6. Optimization rate under simple space sharing scheduler with varying amounts of RAM available.**

## 6. Discussion

Given our observations, it is clear that all designs follow an exponential curve as memory load is changed. As memory load increases, they tend towards infinite response times. As memory load decreases, they will converge on a constant number. PNR's curve is lower than NR/DP's by a constant factor.

This model implies that adding PNR to lightly loaded systems (with high performance networks) will not harm performance and under moderate memory loads PNR can lead to large improvements over disk paging. However, as memory becomes too scarce to share, PNR performance will tend to converge to DP performance.

PNR is very sensitive to network performance. The general shape of results is another exponential curve, where PNR response time tends toward infinity as network service time is increased and converges to some constant number as service time decreases.

Since low network performance results in low PNR performance, PNR should not be considered on systems with low bandwidth or high message RTT times. A standard 100 Mbps network or higher should be regarded as the minimum for satisfactory performance.

Network bandwidth and latency appear to be much more important to the performance of PNR than does the topology of the underlying network. Congestion did not appear to play a large role in our experiments. This situation may be different in a larger network. Topology changes should be studied again on a larger scale.

PNR is clearly the superior method under heavy memory loads when using the space sharing scheduler. But, interestingly, NR consistently beats PNR under light memory loads. Apparently, the added communication and coordination overhead of PNR is useful when RAM is very scarce, but the lightweight NR may be sufficient under light load conditions. This may be due to the fact that PCSs are eliminated on this system. This eliminates a large portion of network activity (i.e. the large amount of page faults after a PCS) and lessens the impact of a bad network RAM allocation decision. These results are interesting and should be studied further.

## 7. Conclusion and Future Work

In this paper we identified a novel way of reducing page fault service time and better utilizing memory resources. 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 implement this idea as a set of scalable peer-to-peer algorithms which can be implemented on a variety of multicomputer systems. We discovered that adding PNR to systems with high performance networks will not hurt overall system performance, and will help speed up memory accesses and more evenly spread load.

While average response time figures are favorable for PNR, this paper did not explore the degree to which PNR penalizes processes running on PNR servers. Performance may be compromised on these nodes in favor overall memory balance. Future work should examine the severity of this problem and propose solutions if necessary.

While we believe our PNR algorithm is scalable, this paper does not explore this aspect of PNR. Future work should concentrate on examining the scalability of our PNR algorithm, both in terms of additional CPUs and additional memory. We must study how additional CPUs increase communication overhead of our PNR strategy and at what scale the fundamental limit of virtual memory address space becomes a concern.

Ultimately, PNR should be compared to competing paging avoidance strategies. These include scheduling strategies that attempt to avoid paging, hardware solutions, and others. It is important to know if one strategy is superior, or if several strategies are best used in conjunction.

**Acknowledgments**

This research was made possible by a grant from the Michigan Space Grant Consortium and National Science Foundation grant ACI-0129883.

## References

[1] A. Acharya and S. Setia. The utility of exploiting idle memory for data-intensive computations. Technical Report TRCS98-02, 1998.

[2] A. Barak and A. Braverman. Memory ushering in a scalable computing cluster. *Journal of Microprocessors and Microsystems*, 22(3-4):175–182, August 1998.

[3] A. Batat and D. G. Feitelson. Gang scheduling with memory considerations. In *14th Intl. Parallel Distributed Processing Symp.*, pages 109–114, 2000.

[4] D. C. Burger, R. S. Hyder, B. P. Miller, and D. A. Wood. Paging tradeoffs in distributed-shared-memory multiprocessors. *The Journal of Supercomputing*, 10(1):87–104, 1996.

[5] 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. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 67–90. Springer-Verlag, 1999. Lect. Notes Comput. Sci. vol. 1659.

[6] G. F. P. F. Darema-Rogers and K. So. Memory access patterns of parallel scientific programs. *Performance Evaluation Review*, 15(1):46–57, 1987.

[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. In *Symposium on Operating Systems Principles*, pages 201–212, 1995.

[8] D. G. Feitelson. Packing schemes for gang scheduling. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 89–110. Springer-Verlag, 1996. Lect. Notes Comput. Sci. vol. 1162.

[9] D. G. Feitelson. Memory usage in the LANL CM-5 workload. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 78–94. Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.

[10] D. G. Feitelson. Metrics for parallel job scheduling and their convergence. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 188–205. Springer Verlag, 2001. Lect. Notes Comput. Sci. vol. 2221.

[11] M. D. Flouris and E. P. Markatos. *High Performance Cluster Computing*, chapter 16, pages 383–408. Prentice Hall, 1999.

[12] V. Lo, J. Mache, and K. Windisch. A comparative study of real workload traces and synthetic workload models for parallel job scheduling. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 25–46. Springer Verlag, 1998. Lect. Notes Comput. Sci. vol. 1459.

[13] E. P. Markatos and G. Dramitinos. Implementation of a reliable remote memory pager. In *USENIX Annual Technical Conference*, pages 177–190, 1996.

[14] Y. Z. A. S. H. F. J. E. Moreira. Improving parallel job scheduling by combining gang scheduling and backfilling techniques. *IPDPS*, pages 133–142, 2000.

[15] E. W. Parsons and K. C. Sevcik. Benefits of speedup knowledge in memory-constrained multiprocessor scheduling. *Performance Evaluation*, 27/28(4):253–272, 1996.

[16] U. Schwiegelshohn and R. Yahyapour. Improving first-come-first-serve job scheduling by gang scheduling. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 180–198. Springer Verlag, 1998. Lect. Notes Comput. Sci. vol. 1459.

[17] Seagate. Barracuda ata v and barracuda ata v plus technical specifications. `http://www.seagate.com/docs/pdf/datasheet/disc/ds_barracudaata5.pdf`.

[18] S. K. Setia. The interaction between memory allocation and adaptive partitioning in message-passing multicomputers. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 146–164. Springer-Verlag, 1995. Lect. Notes Comput. Sci. vol. 949.

[19] K. C. Sevcik and S. Zhou. Performance Benefits and Limitations of Large NUMA Multiprocessors. In *Proceedings of Performance '93*, pages 183–204. Elsevier Science Publ, Sept 93.

[20] F. Wang, M. Papaefthymiou, and M. Squillante. Performance evaluation of gang scheduling for parallel and distributed multiprogramming. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 277–298. Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.

[21] L. Xiao, S. Chen, and X. Zhang. Dynamic cluster resource allocations for jobs with known and unknown memory demands. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):223–240, March 2002.

[22] L. Xiao, S. Chen, and X. Zhang. Adaptive memory allocations in clusters to handle unexpectedly large data-intensive jobs. *IEEE Transactions on Parallel and Distributed Systems*, 15(6), June 2004.

[23] L. Xiao, X. Zhang, and S. A. Kubricht. Incorporating job migration and network RAM to share cluster memory resources. In *HPDC*, pages 71–78, 2000.

[24] B. B. Zhou, D. Welsh, and R. P. Brent. Resource allocation schemes for gang scheduling. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 74–86. Springer Verlag, 2000. Lect. Notes Comput. Sci. vol. 1911.