

Remote Memory Management and Prefetching Techniques for Jobs in Grid

S.Radha
National Institute of Technology, Tiruchirappalli, India
radha.nitt@gmail.com

S.Mary Saira Bhanu
National Institute of Technology, Tiruchirappalli, India
msb@nitt.edu

N.P.Gopalan
National Institute of Technology, Tiruchirappalli, India
gopalan@nitt.edu

Abstract

Predominant resources for execution of any application are computational power and memory. On one side, computational power has grown many folds faster than memory capacity. On the other side, application's memory requirements have kept on increasing from time to time. Application's minimum memory requirement influences job scheduling decision in grid. But once the application starts executing it faces memory pressure i.e. increase in memory requirement. This could be handled by remote memory paging - moving pages from memory loaded machine to remote machine with unused memory. Highly unpredictable network latency in grid has direct impact on the remote memory access latency. The idea of prediction and prefetching can be adapted to reduce this latency. Profile based and Markov based prediction models are explored in this paper. The experiments on memory intensive applications show that the Markov based model has better accuracy and profile based prediction provide good coverage.

1. Introduction

The drastic growth and reach of the internet and existence of powerful computers, high speed networks have enabled the use of geographically scattered resource as single unified resource.

While the cost of high end servers, storages are still high, there are also so many idle resources available. Idle resources include unused CPU cycles, idle main memory and unused storage [4]. Grid computing solves this problem.

Grid-computing focuses on resource sharing in distributed systems in an effective, efficient, flexible, secure and coordinated fashion [6, 15]. Resource management is the fundamental challenge in Grid computing due to the dynamic nature of grid [9].

Any application for its execution requires CPU cycles and memory. In the recent years the cost of CPU cycles has descended whereas cost of memory is not yet cheap. But memory requirements of applications have grown with its complexity. This initiates a need to utilize all the idle memory [4].

Initial job scheduling decision for a job in grid computing is done based on CPU cycles needed and minimum memory requirements [9]. After scheduling the job on a particular node there may be a need of more memory than what is available in that node. This is called as *memory pressure* [2].

In a grid, memory pressure can arise when local job starts executing utilizing the free memory thereby reducing the available memory for grid application. There is a need for rescheduling in the grid in the presence of memory pressure.

The approach proposed in this paper eliminates the overhead attached to rescheduling. Rather than migrating job to different machine the idle memory in the remote machine can be utilized.

But the network performance largely governs the performance of remote memory paging [14]. Network latency in grid is highly unpredictable as there are various types of networks [9]. Most of the grid systems today try to make use of network infrastructure provided by internet, network latency is still more unpredictable in the Internet.

Remote memory is a new layer between disk and primary memory in the memory hierarchy of the node [1]. But there is a speed gap between these layers due to the network latency. This speed gap can be brought down by prefetching - the process of bringing memory pages before application demands for it. This scheme minimizes the remote memory access latency and aids in intelligent resource sharing for semantic grids.

1.1. Related Work

The theme of consuming idle memory has been thought of for quite sometime as memory is an

expensive and critical resource. In the arena of cluster computing there is a bunch of work done.

The work done by Anurag Acharya [4] discusses the existence of idle memory and the benefits acquired by utilizing it. Also shows that the machine with large amounts of memory has large amount of memory not in active use.

Eric A. Anderson [5] explores the idea of NetworkRAM. NetworkRAM allows addressing of all the memory available in all nodes in the network as a single global address space. His proposal discusses the various implementations by which utilization of idle memory in remote machines can be done.

Li Xiao's attempt [3] supports the fact that insufficient memory shall increase the number of page faults and hence jobs experience long latencies. Any scheme that reduces number of page faults shall increase the performance significantly. Later an interesting study done by Xiaodong Zhang [1] explores two schemes namely the Network RAM and memory reservation for dynamic load sharing. The memory requirement of an application initially mentioned is used to determine and schedule the job on a particular node. But this study makes it clear that the memory requirement of application increases once it starts executing. Another similar study by the same author [2], explain where the existing models fail. All the existing works discussed, are done using NetworkRAM scheme for cluster environment.

Many research works are done in the area of prediction and prefetching also. Jasmine Y.Q. Wang [10] suggests that prediction of future references based on an application's reference history is simple and efficient in the presence of good prediction algorithm in place. This effort strongly support Markov based prediction algorithms to improve running time of IO intensive applications and shows some promising results in global memory system (GMS).

Another study by Doug Joseph [12] shows how prefetching using Markov predictors implemented in hardware has improved the performance of processor caches.

Our proposal is different from the other attempts as it introduces the prefetching technique to speedup the remote memory paging. The prediction and prefetching scheme would conceal the impact of network latency on execution times of jobs in grid while exploiting idle remote memory. Also this scheme is completely transparent to applications and does not demand the presence of GMS or NetworkRAM.

The paper is organized as follows: The approach used for the remote memory paging and prefetching are presented in Section 2, Implementation details are elaborated in Section 3, Performance analysis details and results are discussed in Section 4.

2. Approach

When an application needs more memory for continuation of its execution there are two options.

One, the whole process can be stopped, all its context details can be saved, a remote machine can be identified and this job can be moved to the remote machine and restarted. This is called *job migration*. This scheme is very attractive in homogenous environment but in grid environment where members are heterogeneous by nature, this scheme may not be that attractive. Identification of a remote machine with same or compatible architecture, operating system, required software and more memory would be difficult. Grid computing is devised to provide low cost solution to solve large problems. But a requirement of a machine with large memory may lead to a need for high end servers which are still costly. Also not all application may be simple enough to freeze and restart in a different machine. Application's context may be large and hence migrating to a different machine may exhibit severe performance degradation. The job migration can become a chain action as the memory pressure may arise repeatedly.

Other scheme is to perform Remote Memory Paging. Application continues to execute in the same machine, but the memory pressure is handled by using memory available in the remote machine (Figure 1). Few old machines which are idle and are not powerful enough to execute any application could lend their memory for execution of the application. This scheme does not add any context save/restore overhead as the application is not moved anywhere else for execution.

2.1. Remote Memory Paging

The Remote memory paging could be a potential option in the presence of memory pressure due to the following facts:

- Internet has made almost all machines part of network
- Existence of idle memory in the machines in the network

2.1.1. Design issues. The remote memory paging can be implemented as user library or as kernel instrumentation.

- User library - Applications need to make explicit calls to use the remote memory. This is easier to implement as all the burden of identification, selecting remote node with idle memory and utilizing the idle memory is for the application programmers.
- Kernel Instrumentation - Remote Memory Paging is transparent to the application. This implementation does not require any modification to the application.

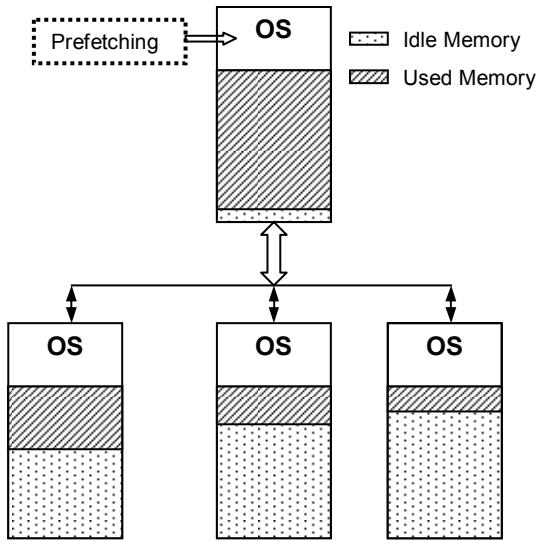


Figure 1: System Design

A *threshold time* need to be determined. A machine with free memory is the potential node to lend memory only if the free memory is available for a certain amount of time (threshold time).

Technique to move the memory page to remote node should be efficient and add very little *overhead* to overall execution.

When an application requests for a memory page, the memory page could be in the main memory of the same machine or in the secondary memory of same machine or in the main memory of the remote machine. The *memory management unit* of the operating system should accommodate the required *changes* in order to facilitate this.

In non dedicated computing farms, the user shouldn't face *performance degradation* while reclaiming resources or executing their jobs in that machine. So, in the remote nodes that lend the idle memory, the local jobs in those machines have a higher priority of using the resources than the guest data accommodated in the idle memory.

The remote node shouldn't *modify the data* in the memory page or shouldn't move the memory pages in the memory it has lend to another node. Required measures need to be taken to handle these situations.

The Grid environment has inherent *dynamism* in the availability of resources. So, the remote memory paging system should be robust in nature.

2.2.Prediction and Prefetching

Remote Memory Paging is enhanced further by prediction strategies for Prefetching.

2.2.1. Profile based Prediction. Page fault occurs when an application requests for a memory page to the OS. The history of page faults (profile) forms the basis for the prefetching.

Though correct prefetches enhance the overall application performance, too many incorrect prefetches will degrade the performance significantly [13]. Incorrect prefetch include a memory page prefetched to main memory much earlier than application needs it, as this leads to removal of some other application's memory pages from the main memory affecting its performance. So, there exists a need to give importance to the temporal component.

2.2.2. Markov based Prediction. Markov based prediction is one implementation of prediction by partial matching [10].

Markov based prediction model proposed for prefetching has following parameters:

P_a : Probability for an application to request for a memory page a . Determined based page-fault trace history.

n : Number of future predictions to be made

o : Number of memory page fetches of the past to be considered towards predicting next memory page.

t_p : The difference in time between the first of the o memory page request and the current page request

P_{min} : Minimum probability a page should have to be considered for prefetching

Δ : Minimum time an application gets to run in the processor

Prefetch of a particular memory page a is performed if the $(P_a \geq P_{min})$ and $(t_p \leq \Delta)$. P_a of a is calculated by the frequency of occurrence of a after o memory page requests [10].

3. Implementation

Remote memory paging is implemented using combination of Network Block Device (NBD) and RamDisk (RD).

NBD virtualizes the remote disk as local disk and RD makes main memory behave like a disk.

Available idle memory in remote node is identified, formatted as RD swap space, mapped to the node under memory pressure as a NBD. The node with idle memory will be nbd-server serving the RD swap space and node under memory pressure would be the nbd-client using the NBD as its swap space.

The advantage of achieving remote memory paging using NBD and RD is that the application is transparent of remote memory paging. This makes it simple for all application to use remote memory paging as application requires no modification.

Prefetching scheme is implemented as kernel instrumentation. First, the application's memory access trace is recorded without disturbing the execution of application. This operation must impart very little overhead to the application executing. The memory access trace also contains the timestamp of each memory access. Using this data the time after which a particular memory page is needed by that application is determined.

When the OS scheduler selects an application for execution, it gets a chance to run at least for a determined time quantum Δ before getting preempted. This value determines the temporal component of the prefetching.

Consider that an application accesses memory pages in the following manner, page a_1 at time t_1 , a_2 at time t_2 , and so on.

$$\begin{matrix} t_1 & t_2 & t_3 & t_4 & \dots & t_n \dots \\ a_1 & a_2 & a_3 & a_4 & \dots & a_n \dots \end{matrix}$$

Prefetching of memory pages till t_n when application demands a_1 is useful only if the difference between t_n and t_1 is less than or equal to time quantum Δ .

$$t_n - t_1 \leq \Delta$$

In the profile based prefetching model when an application demands (page fault) for a particular memory page, that page is fetched along with memory pages it would need within the current time quantum. This strategy shall eventually avoid early prefetches.

Markov-based prefetching model works similar to the previous scheme but for few changes. The application's page fault trace is used to design a Markov model. Each memory page request is assigned priority P_a based on frequency of occurrence. The t_p value is calculated from the page fault trace. When an application requests a particular memory page its immediate past o memory pages are fed to Markov model. The Markov model calculates P_a and t_p of the memory page request and predicts the next n pages the application needs and the resultant pages are prefetched.

4. Performance Analysis

4.1 Experiment Setup

Experiment setup has a cluster of Pentium IV 1.7 GHz processor with 256MB of primary memory, 1GB swap space in local disk. Operating system is Suse Linux 10.

The Linux kernel is modified to trace the application memory reference behavior and to perform prefetching. Application's memory reference pattern is collected from the kernel by *proc* file system driver.

Time stamp is collected by reading the time stamp count register.

To compare performance of Remote Memory paging with the remote memory paging with prefetching, the available physical memory for the node is brought down and swap devices are altered.

For Markov based prediction model the n value is taken as 10% of the total number of page faults. Third order Markov model is considered ($o=3$) as first and second orders are too conservative and has weak predictive power [11]. Higher the order of Markov model, accuracy is higher but the coverage is lesser [11, 13]. The value of Δ is calculated based on the standardized frequency of timer interrupt of the Linux kernel [8].

To observe the effect of both prefetching schemes two memory intensive applications *pifft* and *pcompress2* of *freebench* are considered.

Pifft program calculates the value of Π based on Fast Fourier Transform (FFT). The working set of *pifft* increases along with the number of digits of Π to be calculated. *Pcompress2* uses *memcpy*, *quicksort*, *memcmp* which are basic operations for many memory intensive applications [14].

4.2. Performance Results

Table 1 shows the working set size and the number of page faults of the applications.

As shown in table 2, collecting the application's memory reference trace adds very minimal overhead. The overhead in the execution time is 0.1 seconds for *pifft* and 0.4 seconds for *pcompress2*.

Figure 2 and 3 depicts the effect of the temporal component while prefetching. For these two programs

Table 1: Memory usage statistics of *pifft*, *pcompress2*

Program	Working Set Size (MB)	Number of Page faults
<i>pifft</i>	52	14666
<i>pcompress2</i>	32	59594

Table 2: Effect of collecting memory reference trace on execution time

Program	Without Trace Overhead	With Trace Overhead
<i>pifft</i>	100.537s	100.603s
<i>pcompress2</i>	61.502s	61.957s

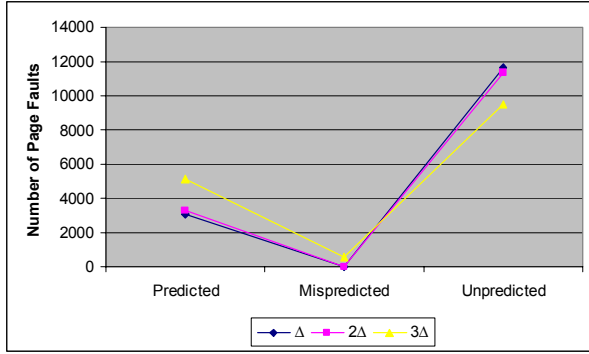


Figure 2: Effect of prefetching addresses of pifft for different time intervals

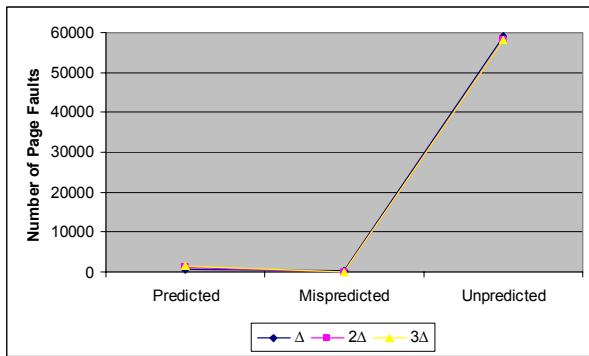


Figure 3: Effect of prefetching addresses of pcompress2 for different time intervals

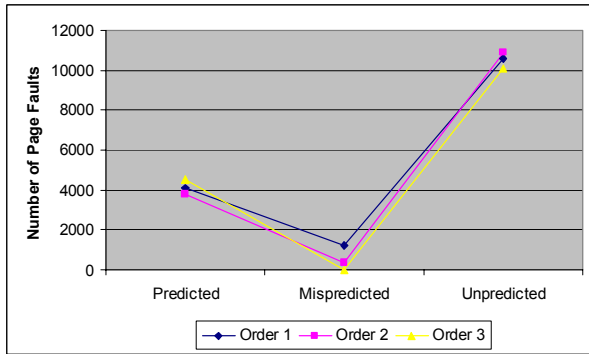


Figure 4: Performance of Markov model of order $o=1, 2$ and 3 for pifft

there is no noticeable difference in performance while prefetching pages within Δ , 2Δ or 3Δ . So, we set temporal component value as Δ for running the prefetching experiments.

Figure 4 is the comparison of Markov models with order 1, 2 and 3. Markov model with $o=3$ has lesser number of missed predictions.

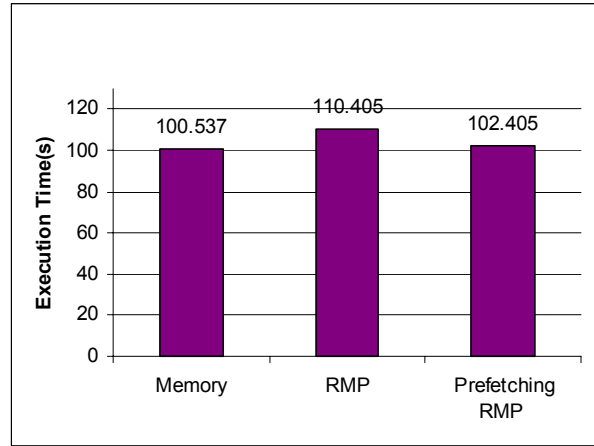


Figure 5: Execution times of pifft while executing in local memory, local memory along with Remote Memory Paging (RMP) and local memory, RMP with profile based prefetching

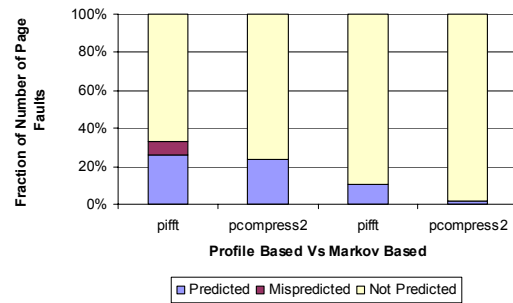


Figure 6: Performance of prefetching in terms of page faults in profile based approach and Markov Based approach

Though the accuracy of model improves with the order of Markov model the time taken to make prediction decision increases with order.

Figure 5 exposes the results of execution of pifft in terms of execution time with and without prefetching over remote memory paging. To run this experiment a memory pressure is introduced and the system is forced to use remote memory. The presence of prefetching brings down the speed gap between local memory and remote memory. Local memory is just 1.02 times faster than remote memory paging with prefetching.

Figure 6 is the performance comparison of profile based approach and Markov model based approach in terms of number of page faults. Profile based approach has more coverage but missed predictions do exist. In Markov based model though coverage is less the missed predictions are insignificant.

5. Conclusion and Future Work

The central theme of building a Grid computing infrastructure is the better resource utilization. Though there are number of ways of doing it, the idea devised in this paper is novel in the arena of grid computing. The efficient resource utilization is made possible by the scheme to handle the high memory needs of the applications. Though lots of high end servers are manufactured these days, they are quite expensive.

But the Remote Memory Paging idea would provide memory without any additional cost. So, the cost to performance benefits of this scheme is substantial.

In grid computing the applications are distributed across multiple machines and so the network latency governs the application performance. The prefetching targets at hiding the impact of network latency, hence the overall execution time reduces significantly. Results show that the remote memory paging performance is brought close to performance of main memory by prefetching. The simplicity and adaptive nature of the thought makes it more attractive.

In our future work, we plan to explore the options available to assure reliability for remote memory paging.

References

- [1] Li Xiao, Songqing Chen and Xiaodong Zhang “Adaptive Memory Allocations in Clusters to Handle Unexpectedly Large Data-Intensive Jobs”, *IEEE Transactions on Parallel And Distributed Systems*, Vol. 15, No. 7, 2004 , pp. 577-592.
- [2] Li Xiao, Songqing Chen and Xiaodong Zhang “Dynamic Cluster Resource Allocations for Jobs with Known and Unknown Memory Demands”, *IEEE Transactions on Parallel and Distributed Systems*, Vol 13, No. 3, 2002, pp. 223-240.
- [3] Li Xiao, Xiaodong Zhang and Stefan A. Kubricht “Incorporating Job Migration and Network RAM to Share Cluster Memory Resources”, *Proc. of the Ninth IEEE International Symposium on High-Performance Distributed Computing (HPDC9)*, 2000, pp. 71-78.
- [4] A. Acharya and S. Setia “Availability and Utility of Idle Memory in Workstation Clusters”, *Proc. ACM SIGMETRICS Conf. Measuring and Modeling of Computer Systems*, 1999, pp. 35-46.
- [5] Eric A. Anderson Jeanna M. Neefe “An Exploration of Network RAM”, UC Berkeley Technical Report, UCB/CSD-98-1000, 1994.
- [6] Ian Foster, Carl Kesselman (Ed.), Morgan Kaufmann, San Francisco, *The Grid: Blueprint for a New Computing Infrastructure*, Second edition, 2004.
- [7] Luis Ferreira et. al. *Introduction to Grid Computing with Globus*, IBM Redbook series, 2002.
- [8] Daniel P. Bovet, Marco Cesati, *Understanding the Linux Kernel*, Third Edition, O'Reilly Media, November 2005.
- [9] Jarek Nabrzyski, Jennifer M. Schopf, Jan Weglarz, *Grid Resource Management: State of the Art and Future Trends*, first ed. Springer, September 2003.
- [10] Jasmine Y. Q. Wang, Joon Suan Ong, Yvonne Coady, Michael J. Feeley, “Using idle workstations to implement predictive prefetching”, *Proc. of the Ninth IEEE International Symposium on High Performance Distributed Computing*, 2000.
- [11] Greta Batiels, Anna Karlin, Henry Levy, Geoffrey Voelker, Darrell Anderson, Jeffrey Chase, “Potentials and Limitations of Fault-Based Markov Prefetching for Virtual Memory Pages”, *Proc. of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 1999, pp. 206 – 207.
- [12] Doug Joseph, Dirk Grunwald, “Prefetching using Markov predictors”, *Proc. of the 24th IEEE annual international symposium on Computer architecture*, 1997.
- [13] Thomas R. Puzak, A. Hartstein, P. G. Emma, V. Srinivasan “When prefetching improves/degrades performance”, *Proc. of the 2nd conference on Computing frontiers*, 2005, pp: 342 – 352.
- [14] S. Liang, R. Noronha and D. K. Panda, “Swapping to Remote Memory over InfiniBand: An Approach using a High Performance Network Block Device”, *IEEE International Conference on Cluster Computing (Cluster 2005)*, Sept. 2005.
- [15] H. Zhuge, “Resource Space Grid: Model, Method and Platform”, *Concurrency and Computation: Practice and Experience*, 2004.