Network RAM Based Process Migration for HPC Clusters

[1] Hamid Sharifian*
sharifian@comp.iust.ac.ir
[2] Dr. Mohsen Sharifi
msharifi@iust.ac.ir
[1,2] School of Computer Engineering, Iran University of Science and Technology

Received: 04/Dec/2012 Accepted: 09/Feb/2013

Abstract
Process migration is critical to dynamic balancing of workloads on cluster nodes in any high performance computing cluster to achieve high overall throughput and performance. Most existing process migration mechanisms are however unsuccessful in achieving this goal proper because they either allow once-only migration of processes or have complex implementations of address space transfer that degrade process migration performance. We propose a new process migration mechanism for HPC clusters that allows multiple migrations of each process by using the network RAM feature of clusters to transfer the address spaces of processes upon their multiple migrations. We show experimentally that the superiority of our proposed mechanism in attaining higher performance compared to existing comparable mechanisms is due to effective management of residual data dependencies.

Keywords: High Performance Computing (HPC) Clusters, Process Migration, Network RAM, Load Balancing, Address Space Transfer.

1. Introduction

A standard approach to reducing the runtime of any high performance scientific computing application on a high performance computing (HPC) cluster is to partition the application into several portions that can be run in parallel by multiple cluster nodes simultaneously.

HPC clusters generally consist of three main parts: a collection of off-the-shelf (COTS) computing and storage nodes, a network connecting the nodes, and a cluster manager system software that manages all nodes and presents a single system image to applications while exploiting the parallel processing power of multiple nodes.

The cluster manager system software provides a set of global services that aim at making resource distribution transparent to all applications, managing resource sharing between applications, deploying as much cluster resources as possible for demanding applications, and scheduling parallel processes on all cluster nodes. The services include global resource management, distributed scheduling, load sharing, process migration, and network RAM.

Dynamic load sharing can be achieved by moving processes from heavily-loaded nodes to lightly-loaded nodes at runtime. This can lead to fault resilience, ease of system administration, and data access locality in addition to an enhanced degree of dynamic load distribution [1].

Upon migration of a process, the process must be suspended and its context information in the source node extracted and transferred to the destination node. The process can only then resume executing from the point it was suspended. Two critical challenges of process migration are the transfer of the process address space from the source node to the destination node, and access to the opened files in the destination node after process migration [2].

In this paper we only focus on resolving the first challenge of process migration by introducing a new process migration mechanism by using the network RAM feature of HPC clusters, wherein the aggregate main memory of all cluster nodes in a cluster represents the network RAM of that cluster [3]. In addition to

* Corresponding Author

achieve comparably higher performance than existing process migration mechanisms, our proposed mechanism is intended to allow for multiple-migration of each process that is a missing feature in existing process migration mechanisms that is accounted as a source of inefficiency.

The rest of paper is organized as follows. Sections 2 and 3 introduce process migration and network RAM. Section 4 reviews related works on process migration with respect to transfer of process address space. Sections 5 and 6 present our network RAM based mechanism and its evaluation, and Section 7 concludes the paper.

2. Process Migration

Process migration is the act of transferring an active process between two computers and restoring its execution in a destination node from the point it left off in the source node. The goals of process migration are closely tied to applications that use migration. The primary goals include resource locality, resource sharing, dynamic load balancing, fault resilience, and ease of system administration [1]. Any process migration mechanism can thus be benchmarked and evaluated with respect to the degree it satisfies these goals.

Considering an HPC cluster, process migration has three main phases [4] (note that these phases are applicable to process migration in all types of networks of computers in general including HPC clusters that are the main focus of this paper):

1. *Detaching Phase* that involves the suspension and the extraction of the context of a migrant process in its current node. These activities must be done in a way that none of the other processes running on the current node or in other nodes of the cluster experience any execution inconsistencies. At the start of this phase, the execution of the migrant process is frozen.
2. *Transfer Phase* that involves the transfer of the extracted context of the migrant process to the destination node.
3. *Attaching Phase* that involves the reconstruction of the migrant process on the destination node. The reconstruction in turn involves the allocation of resources on the target node to the migrant process, informing the beneficiaries and/or brokers of the migrant process about the current executing place of the migrant process, and resuming the execution of the migrant process on the destination node from the point it left off on the source node.

The time interval between freezing a migrant process on a source node and resuming its execution on a destination node is called the freeze time representing the status wherein the migrant process is neither executing on the source nor executing on the destination node. The longer the freeze time the lower will be the performance of the process migration.

The context of a process to be migrated includes the process's running state; stack contents; processor registers; address space; heap data; and process's communication state (like open files or message channels). The whole context must be transferred to the destination node before the process can continue its execution on the destination node. The process address space is the largest part of the process context that might have hundreds of megabyte of data [5] taking longest to be transferred to the destination node. This can adversely affect the performance of process migration. Therefore, the performance of any process migration mechanism largely depends on how long it takes to transfer the context of migrant processes to destination nodes.

Various data transfer techniques have been presented in the literature that try to reduce the high cost of address space transfer [6]. A well-known technique is to transfer only parts of process address space to allow resumption of processes on destination nodes without waiting for the transfer of the whole process address space and context. Though very attractive on grounds of improving the performance of process migration, this technique leaves parts (pages) of process address space on different nodes when multiple migrations in the lifetime of process is allowed and occurs. In other words, the process address space is scattered on multiple nodes resulting in residual dependencies. Management of residual dependencies increases the implementation complexity of process migration that in turn results in performance degradation of process migration.

Our proposed mechanism is particularly useful to strong migrations wherein the entire process state (rather than code and some initialization data in weak code migration) must be transferred to destination.

3. Network RAM

Large-memory high-performance applications such as scientific computing, weather prediction simulations, data warehousing and graphic rendering applications need massive fast accessible address spaces [7] that are not provided by even high capacity DRAMs. The runtime performance of these applications degrades quickly when system encounters memory shortage and starts swapping memory to local disks.

In today's clusters with very low-latency networks, the idle memory of other nodes can be used as storage media faster than local disks, called network RAM. The goal of network RAM is to improve the performance of memory intensive workloads by paging to idle memory over the network rather than to disk [8].

Some common uses of network RAM are remote memory paging [9], network memory file systems, and swap block devices [10,11]. Locating unused memory in every node requires that network RAM keeps up-to-date information about all unused memories.

Since network RAM stores data on remote memories, it includes remote memory paging facility to keep information on all remote data. This functionality of network RAM can be used to alleviate the performance overhead of process migration in transferring and managing the address spaces of migrant processes. This describes why we have deployed network RAM technology to propose a novel process migration mechanism for HPC clusters in this paper.

4. Related Works

We can categorize into three categories the works on process migration that have presented solutions to cope with the address space transfer issue in particular into three categories.

4.1 Address Space Transfer Techniques

To avoid the high cost of process address space transfer, several techniques have been introduced. In the *total-copy* technique, which is the simplest and weakest one, the whole process address space is copied to destination node at the migration time [12,13,14,15,16]. The *pre-copy* technique transfers the whole process address space to destination node before starting to migrate the process in order to reduce the freeze time of the process [17]. The *copy-on-reference* technique transfers only the process state to the destination node and pages of process address space are transferred on demand [18,19]. In the *flushing* technique, dirty pages are flushed to disk and the process accesses them on demand from disk instead of memory on the source node [20].

4.2 Prefetching Techniques

Besides techniques for transferring process address space, another technique is proposed to increase the performance of address space transfer while migrant is running on the destination node. This technique includes prefetching of those pages that are likely to be accessed by the migrant to avoid remote page faults. This solution is used in openMosix and it is called Lightweight Prefetching [21].

4.3 DSM-based Techniques

Some HPC clusters have used the distributed shared memory (DSM) mechanism to transfer process address spaces between nodes upon process migrations. Pages stored on DSM need not be transferred at all during process migration. Only pages accessed by the migrant after migration are provided to the migrant process using the DSM mechanisms. *Kerrighed* is a kind of SSI operating system that has used this technique for process migration [2]. *CORAL* [4] and *Mach* [6] use the same technique for process migration and *MigThread* [22] uses a DSM framework for thread migration.

4.4 Comparison

DSM-based and copy-on-reference techniques in support of process address space transfer are more efficient than their counterparts because they do not transfer the whole process address space and avoid storing pages on disk. However, systems such as *Mosix* [15], *Accent* [19] and *RHODOS* [18] that have used the copy-on-reference migration technique allow processes to migrate just once in their lifetime in order to avoid the complex implementation of multiple process migrations or better said the complex management of dependencies of data residual on different nodes if multiple migrations were allowed. On the other hand, systems that have used the DSM-based technique are quite dependent on the implementation of their DSM manager for handling dependencies between residual data on different nodes that are provided to migrant processes on demand. In this paper, we propose a transfer technique in the face of multiple process migration allowance whose performance is higher than existing implementations of the DSM-based technique.

5. Motivation

Process migration has gained popularity for several reasons. Traditional process scheduling mechanisms lack enough flexibility to cope with changing loads of very large HPC clusters and process migration can be beneficial here. Unlike other mechanisms such as check-pointing, process migration needs no server coordination [20] and is more suited to make clusters scalable. By growing the size of data in high performance applications rather than process code size which is quite stable, process migration will be very promising when data are located on several nodes.

In spite of above advantages, process migration has not been widely used in HPC clusters. This is mainly due to the low performance of process migration mechanisms and the complexity of implementing the migration support in commodity operating systems.

In HPC clusters executing applications with huge address spaces, the use of idle memories of remote cluster nodes instead of disk is more attractive. This can be achieved by network RAM. With growing applications with large data, the use of network RAM has become more advantageous. That is why various models have been implemented in recent years [7,9,10,11,23,24]. In such applications whose address spaces are very large and distributed among multiple nodes, process migration is more beneficial because of the smaller sizes of the process states, though data distribution makes data provision to migrant processes more difficult.

By using the network RAM technology, memory is managed without any direct interference of process migration mechanism simplifying the implementation of process migration and improving the overall performance of process migration mechanism.

6. Proposed Mechanism

Network RAM uses memories of remote cluster nodes to store data. When a process migrates to another node, some parts of its address space are remained on the source node. This is similar to the case that the process is on the destination and data are stored in remote memory of the network RAM that becomes accessible to the migrant process on demand. Inspired by this similarity, we propose a new process migration mechanism that uses network RAM for the purpose of transferring process address spaces during process migrations. This approach decreases the implementation complexity of our process migration mechanism, reduces the overhead of residual data dependencies, and improves the performance of migrant processes.

Network RAM has good facilities that can be used in process migration. These facilities include remote memory pager and an efficient module to locate remote pages across an HPC cluster. Thanks to these facilities, we can transfer only the required pages and at the same time, allow processes to migrate multiple times on various nodes.

In our mechanism, there is no need for transfer of the whole process address space and pages required by the migrant process can be accessed through the network RAM remotely. Fig. 1 shows the schema of our proposed mechanism.

When a process is selected for migration, no pages are transferred. Instead, each page in address space of the migrant process is added to the data structures of the network RAM and marked as a *remote page*. Then, page faults of migrant are handled by the network RAM faster than DSM-based solutions.

The network RAM module manages all memory-related issues of process migration mechanism including the transfer of process address space during migration, management of residual dependencies and handling page faults in addition to its own work. This simplifies the implementation of process migration mechanism.

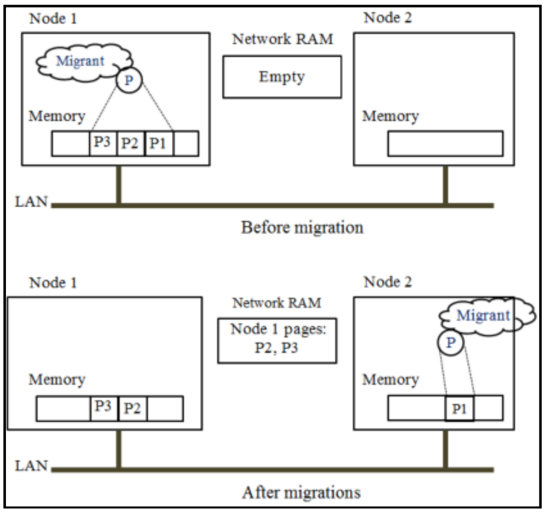

Fig.1. A schematic view of our network RAM-based process migration mechanism

7. Experimental Result

To evaluate our proposed mechanism, we simulated our mechanism by running it on a homogenous cluster with 20 nodes connected to an Ethernet switch via a 10Gbps cluster network connection. Each node had a 3GHz CPU. The cluster has global distributed shared memory, global network RAM and global process and network management features in addition to process migration facility.

We evaluated our mechanism by DGEMM HPC benchmark. Fig. 3 shows the migration times of processes with different sizes of address spaces under DSM-based, network RAM-based and copy-on-reference process migration mechanisms. Among previous solutions only DSM-based migration method support multiple migration of a process in its lifetime on different workstations.

As Fig. 3 shows, the migration times under DSM-based and our proposed mechanisms were almost equal. The migration time has increased linearly with increases in the size of process address space. In both mechanisms, the whole process address space was not transferred to the destination node at migration time. However, the larger the process address space the higher was the migration time implying that bigger address spaces take longer to be managed that is quite logical and sensible. The migration time under copy-on-reference mechanism is not dependent on address space size of process and almost remains unchanged.

So far our experiments showed the same performance for DSM-based vs. network RAM-based process migration mechanisms.

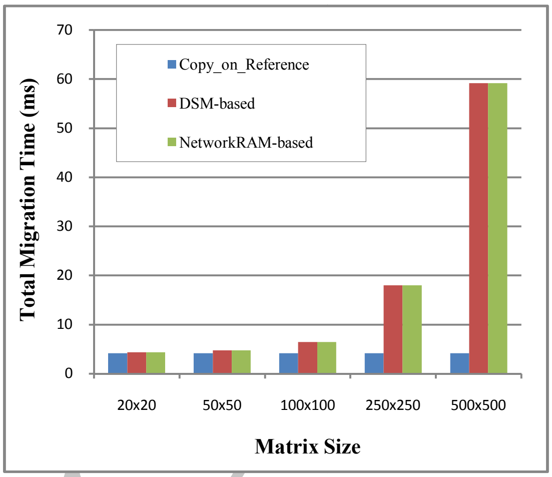

Fig.2. Total migration time for different matrix Sizes

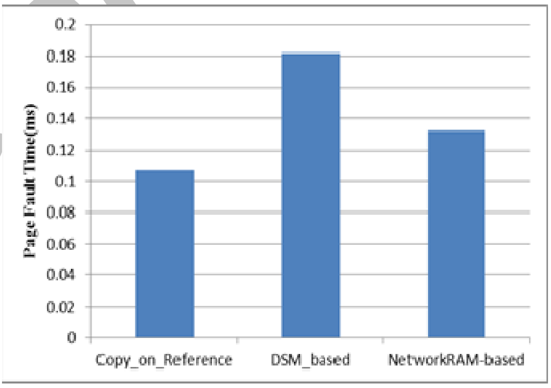

Fig.3. Page fault handle time for process migration mechanisms

The key improvement in our network RAM-based mechanism is handling page faults of the migrant process on destination node. Fig. 3 shows the average page fault handling time for DSM-based, copy-on-reference and network RAM-based mechanisms. As Fig. 3 shows, our proposed mechanism handle page fault of the migrant process faster than DSM-based mechanism. That is because network RAM-based does not consider memory sharing issues and providing pages to the demanding process is performed without locking operations. However, copy-on-reference mechanism has minimum page fault time, because in this way, requested pages are being brought from one specified workstation namely the source workstation.

Due to the fact that page faults may occur thousands of times while executing the migrant process on destination node, improvement of page fault time in our proposed mechanism results in improvement of execution time of process compared to DSM-based mechanism.

Given that our network RAM-based mechanism supports multiple migration of the process, the advantage of our network RAM-based mechanism showed even more when a process was allowed to migrate to more than one cluster node in its lifetime. The more a process is selected to migrate, the more page faults it may experience in its execution.

Fig. 4 shows the execution times of the DGEMM process with 500×500 matrix size after multiple migrations on different cluster nodes in its lifetime. As a result of effective page fault handling by the network RAM, the execution times of the migrant were reduced compared to those of DSM-based mechanism.

8. Conclusion and Future Works

In this paper, we proposed a mechanism that exploited the network RAM facility existing in clusters to transfer process address spaces during process migrations in HPC clusters.

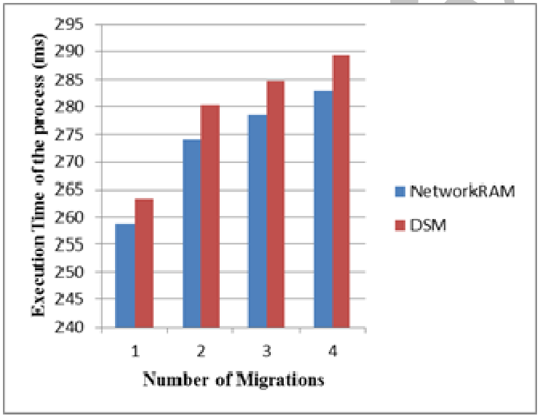

Fig.4. Execution time of DGEMM process after multiple migration in DSM- and Network RAM-based process migrations

Our simulative experiments showed higher overall performance of migrant processes under our proposed mechanism compared to a simulated DSM-based process migration mechanism when processes were allowed to migrate to more than one cluster node in their life-time. This implies that our proposed mechanism is especially attractive to scientific applications with coarse-grain long-lived processes that may require multiple migrations in their life-time; we know as a fact that migration of short-lived processes is not efficient [6]. The network RAM facility we used in our proposed process migration mechanism managed access to remote memory and consequently simplified the implementation of our mechanism.

We can further improve the performance of our proposed process migration mechanism by reducing the numbers of required page transfers by coordinating the network RAM as to where to store remote pages with the task that selects a node as destination upon migration.

Acknowledgments

We hereby acknowledge the help of Mr. Reza Azariun in drafting this paper. We also wish to thank Mr. Ehsan Mousavi and Ms Mirtaheri for their guidance in initiating research on migration in HPC clusters. We also thank ITRC for partially supporting the research whose results are partially reported in this paper.

References

[1] Nalini Vasudevan and Prasanna Venkatesh, "Design and Implementation of a Process Migration System for the Linux Environment," 3rd International Conference on Neural, Parallel and Scientific Computation, August 2006, pp. 1 - 8.

[2] Geoffroy Vall´ee, Christine Morin, Jean-Yves Berthou, Ivan Dutka Malen, and Renaud Lottiaux, "Process Migration based on Gobelins Distributed Shared Memory," Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, May 2002, pp. 325 - 325

[3] Michail D. Flouris and Evangelos P. Markatos, "Network RAM," High Performance Cluster Computing, Architectures and Systems.: Prentice Hall, vol. 1, ch. 16, pp. 383-408, 1999.

[4] Ivan Zoraja, Arndt Bode, and Vaidy Sunderam, "A Framework for Process Migration in Software DSM Environments," Proceedings of 8th Euromicro Workshop on Parallel and Distributed Processing, 2000, pp. 158 - 165.

[5] Ehsan Mousavi Khaneghah, Najmeh Osouli Nezhad, Seyedeh Leili Mirtaheri, Mohsen Sharifi, and Ashakan Shirpour, "An Efficient Live Process Migration Approach for High Performance Cluster Computing Systems," Communications in Computer and Information Science, 2011, vol. 241 part 8, pp. 362 - 373.

[6] Dejan S. Milojicic, Fred Douglis, Yves Paindaveine, Richard Wheeler, and Songnian Zhou, "Process Migration," ACM Computing Surveys (CSUR), vol. 32, no. 3, September 2000, pp. 241 - 299.

[7] Michael R. Hines, Mark Lewandowski, and Katrik Gopalan, "Anemone: Adaptive Network Memory Engine," Proceedings of the twentieth ACM symposium on Operating systems principles, 2005.

[8] Eric A. Anderson and Jeanna M. Neefe, "An Exploration of Network RAM", University of California at Berkeley, 1999.

[9] Hiroko Midorikawa, Motoyoshi Kurokawa, Ryutaro Himeno, and Mitsuhisa Sato, "DLM: A Distributed Large Memory System using Remote Memory Swapping over Cluster Nodes," Proceedings of 2008 IEEE International Conference on Cluster Computing, 2008, pp. 268 - 273.

[10] Hui Jin, Xian-He Sun, Yong Chen, and Tao Ke, "REMEM: Rmote Memory as Checkpointing Storage," 2nd IEEE International Conference on Cloud Computing Technology and Science, 2010.

[11] Changgyoo Park, Shin-gyu Kim, Hyuck Han, Hyeonsang Eom, and Heon Y. Yeom, "Design and Evaluation of Remote Memory Disk Cache," Proceedings of 2010 IEEE International Conference on Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), 20-24 Sept. 2010, pp. 1 - 4.

[12] Michael L. Powell and Barton P. Miller, "Process Migration in Demos/MP," Proceedings of the ninth ACM symposium on Operating systems principles, 1983, New York, NY, USA, pp. 110 - 119.

[13] Yeshayahu Artsy and Raphael Finkel, "Designing a Process Migration Facility: The Charlotte Experience," IEEE Computer, vol. 22, no. 9, September 1989, pp. 47 - 56.

[14] Chris Steketee, Piotr Socko, and Bartosz Kiepuszewski, "Experiences with the Implementation of a Process Migration Mechanism for Amoeba," Proceedings of the 19th ACSC Conference, January-February 1996, Melbourne, Australia, p. 140—148.

[15] Amnon Barak, Oren Laden, and Yuval Yarom, "The NOW MOSIX and its Preemptive Process Migration Scheme," Bulletin of the IEEE Technical Committee on Operating Systems and Application Environments (TCOS), vol. 7, no. 2, Summer 1995, pp. 5 - 11.

[16] Gerald Popek and B. WALKER, The Locus Distributed System Architecture: MIT Press, 1985.

[17] Marvin M. Theimer, Keith A. Lantz, and David R. Cheriton, "Preemptable Remote Execution Facilities for the V-System", Proceedings of the 10th ACM symposium on Operating systems principles, 1985, New York, pp. 2 - 12.

[18] Damien De Paoli and Andrzej Goscinski, "Copy on Reference Process Migration in RHODOS," 1996 IEEE Second International Conference on Algorithms and Architectures for Parallel Processing(ICAPP 96), Jun 1996, pp. 100 - 107.

[19] Edward R. Zayas, "Attacking the Process Migration Bottleneck," Proceedings of the 11th ACM Symposium on Operating systems principles, 1987, New York, USA, pp. 13 - 24.

[20] Fred Douglis and John Ousterhout, "Transparent Process Migration: Design Alternatives and the Sprite Implementation," Software - Practice and Experience, vol. 21, no. 8, August 1991, pp. 757 – 785.

[21] Roy S.C. Ho, Cho-Li Wang, and C.M. Francis Lau, "Lightweight Process Migration and Memory Prefetching in openMosix," IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008) , April 2008, Hong Kong, pp. 1 - 12.

[22] Hai Jiang and Vipin Chaudhary, "MigThread: Thread Migration in DSM Systems," Proceedings of International Conference on Parallel Processing Workshops, 2002, pp. 581 - 588.

[23] Nan Wang et al., "Collaborative Memory Pool in Cluster System," IEEE International Conference on Parallel Processing, 2007, Boston MA, USA, pp. 17 - 24.

[24] Paul Werstein, Xiangfei Jia, and Zhiyi Huang, "A Remote Memory Swapping System for Cluster Computers," Proceedings of Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies, 3-6 Dec. 2007, pp. 75 - 81.