Memory Servers: A Scope of SOA for High-End Computing

Surendra Byna Xian-He Sun Ryan Nakhoul
Computer Science Department, Illinois Institute of Technology, Chicago
{bynasur, sun, rnakhoul}@iit.edu

Abstract

The expanding gap between microprocessor and disk performance has initiated new techniques of providing memory as a service in high-end computing (HEC). Although the processor and disk densities have improved over the last decades, the improvement of disk performance is inferior to that of processors, which is causing a bottleneck for HEC. With the rapid growth of network technology for cluster computers, the idea of accessing memory of an idle peer node has proven to be faster than accessing a disk. With this motivation, many researchers developed systems that offer idle memory as a service. In this paper, we present a brief survey of various systems that offer memory service in improving the disk access performance and discuss the scope of applying service oriented architecture (SOA) for HEC.

1. Introduction

High-end computing is a major strategic tool for science, engineering, and industry. HEC simulations in various areas of science enable to understand the world around us [1]. HEC systems have emerged with Teraflops of computing power and Petaflop computing is in the near horizon. However, disparity among processors, storage, memory, network, and applications causes a gap between peak performance and sustained system performance, and this gap is growing rapidly [5] (figure 1). While processor and network speeds are growing rapidly, disk performance has been improving very slowly. The processor performance is improving thousands of times faster compared to that of I/O.

To solve this problem, many research groups have proposed providing memory as a service over network in order to reduce disk accesses and to let clients directly access memory remotely. The assumption with this research is that data can be accessed faster on network than from disks [2, 4, 6, 8, 10]. In this paper, we discuss and compare these strategies, and present a possibility of extending this idea with SOA model.

2. Background and Evaluation Criteria

Research has shown that in a cluster of workstations, and in local area networks consisting of multiple computers intercommunicating, a considerable amount of unused memory is available in each node [2]. With very low sustained system performance, much of memory on individual workstations is idle. This fact has lead to investigating solutions that allow applications to take advantage of this collectively large amount of available memory.

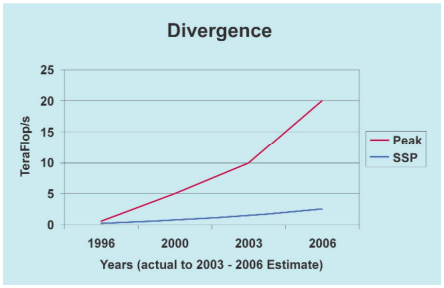

Figure 1. "Divergence Problem" - Increasing gap between peak performance and sustained system performance (SSP)
Source: HECRTF [1]

Data-intensive applications in the areas of graphics rendering, weather prediction systems, simulation software etc. require massive amounts of memory. When no more physical memory is available, these applications begin paging to secondary storage-local disk. The disk access overhead is in the range of 10 ms, where as memory access latency ranges from 44ns to 114.3 ns [12]. Thus, the efficiency of HEC machines in running the applications mentioned above is dramatically low. In addition to that, transferring data rate from or to a hard disk is 320MB/s, while data transfer rate on a network can reach more than 1000MB/s. This is where network memory servers come into play by allowing processes to access remote idle memory not being used by the remote host. Although accessing remote memory resources introduces network communication overhead, it is widely believed that this method is still faster than paging to local disk.

To design a distributed system in which physical memory resources of each individual node can be utilized by other machines requires considering many issues. These issues for any distributed system are: transparency, fault tolerance, replication and consistency, portability, scalability, and network performance. Another criterion under consideration is how these memory servers are implemented. There are six types of implementations [3]: explicit program management, user level implementation,

user-level page management, device driver, modified kernel, and network interface. We base all these issues in comparing existing network memory servers.

3. Comparison of Experimental Systems

Two of the original research projects with the idea of exploiting memory space over network in various computing environments are Network RAM [3], Global Memory Service (GMS) [6]. Based on these models many extensions were proposed. Dodo [2] suggests harvesting idle memory space by using resource monitors, a central memory manager and scheduler. Iftode et al. [7] suggests using part of the nodes of a multi-computer as memory servers. These memory servers form a layer between the RAM and disks. Recently, Xiao et al. [10] have proposed Parallel Network RAM (PNR) to utilize global cluster memory when memory requirement is large. Anemone [7] is the latest implementation of network memory server that uses RAM disk interface. Due to space limitation, we briefly discuss these projects and compare them. A detailed comparison of these memory servers is available at www.cs.iit.edu/~suren/scc06full.pdf.

The Network RAM [3] project is a user-level implementation allowing applications to utilize remote memory resources by making explicit calls in their code. Backup mechanisms, to cope with node failures, were as well investigated, but not implemented. These mechanisms are network backup, and local backup. In the application code, nevertheless, the type of mechanism should be specified.

The Global Memory Management (GMS) [6] project has been developed to offer a way of globally managing memory resources, in a cluster of workstations, in order to avoid disk accesses as much as possible. The solution is a distributed memory management algorithm implemented at the operating system level.

The Dodo [2] project was designed to allow applications that are hungry for memory to use remote memory on the network as an intermediate cache between local memory and disk. Dodo, which is implemented at the user-level, requires applications to explicitly use remote memory by the use of a special library. And to make memory region management easy on programmers, another special library is provided.

The Anemone [7] project aims at exploiting large amount of memory resources available in a high-speed LAN. The main objective is to provide applications that utilize memory resources intensively with memory resources dispersed in the network, in a transparent, fault-tolerant fashion. The most prominent difference between Anemone and other projects mentioned above is that there is no need to modify the operating system or the application code in order to utilize remote memory resources. Instead, Anemone makes use of the memory-resident disk interface (RAM disk) and the Network File System protocol.

The Parallel Network RAM [10] is yet another project dealing with harvesting remote memory. It has been proposed as a solution to the problem of having overloaded nodes in parallel computing platforms. This overloading is caused by the fact that memory is allocated unevenly as the job scheduler in such platform does not know the real memory requirements of the jobs. Thus, Parallel Network RAM is a module that runs as another part of the virtual memory system.

The Memory Servers for Multicomputers [7] project has been developed with the intent of offering memory resources as a fast backing storage to other nodes in the network. This storage logically lies between local memory and disk.

The objective of Remote Memory Pager [9] project is to use remote main memory for paging. The system has been implemented as block device driver linked to the DCE/OSF1 operating system. Therefore no kernel code or application code require modification.

Table 1 shows a comparison of the main aspects of the experimental systems mentioned above. It is worthy to note that the systems that are implemented at the user-level are the most portable. On the other hand, as expected, systems implemented at the kernel level, in general, offer better performance, but suffer from portability issues.

A common architecture of all the network memory servers discussed above contains 1) clients that request for extra memory, 2) servers that provide memory space, and 3) a memory management engine (MME) to help the client locate the memory server and to move the data from the server to the clients (see figure 2). The server is capable of providing remote memory space for multiple clients. For the ease of description, we show one client in Figure 2.

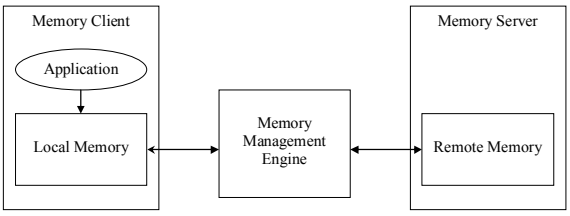

Figure 2. Architecture of Memory Servers

One of the main shortcomings of all the systems we discussed above is that they passively provide memory space for clients. While they perform better than disk access performance the performance gap between CPU and data access still exists and rapidly growing. We proposed strategies to predict future data references adaptively and to push data closer to the CPU on time [11]. To present the full details of this architecture is out of scope for this paper. We refer the readers to [11] for

Table 1. Comparison of Memory Server systems

System	Type of Implementation	Scalability	Portability	Fault Tolerant	Transparency	Modification Required	Speedup
Anemone	The Memory Engine, as of today, is implemented at the user-level, and in the future will be implemented at kernel level	High	High, but mostly on UNIX- based operating system.	Not yet, but in the near future as replication mechanisms and consistency issues are being considered.	High	None	3-3.5 times faster than disk paging.
Dodo	User-level	High	High, but mostly on UNIX- based operating system.	Handles crashes, but nothing related to data loss.	High	Application code has to be modified to make use of the Dodo special libraries	1.1-3.2 times faster than disk paging.
Network RAM	User-level	Low. Global resource manager is a performance bottleneck	High, but mostly on UNIX- based operating system.	No, but replication mechanisms are being considered to handle servers crash. Moreover, Global resource manager is a single point of failure.	High	Application code has to be modified	2-5 times faster than disk paging.
Parallel Network RAM	Kernel-level	Medium. As the number of CPUs increases in the system, performance will degrade.	Medium, since the Parallel Network RAM is a subsystem of the kernel.	No, every memory server is a point of failure. No backup mechanism is implemented.	High	Kernel code modification	1.3-1.45 times faster than disk paging.
Memory Servers for Multicomputers	Kernel-level (for a machine hosting a memory server)	High	Low, specific to NX/2 operating system.	Highly fault tolerant. Replication mechanism is used.	High	Kernel code modification for clients only.	1.1-1.22 times faster than disk paging.
Global Memory Management	Kernel-level	High	Low. Implemented as part of the OSF/1 operating system.	Although no globally managed data is lost (pages are written in local disk whenever they are sent out to remote nodes), initiator node and master node are each a single point of failure.	High	Kernel code modification.	1.5-3.5 times faster than disk paging.
Remote Memory Pager	Device driver	Medium. Clients are hardwired to servers.	Low, works only with DEC/OSF1 operating system	Highly fault tolerant when it comes to memory pages. Redundancy mechanism is used. However, node hosting the servers list is a single point of failure.	High	None. Integrated into the kernel code without modifying the latter.	1.1-2.12 times faster than disk paging.

more details of our memory servers for HEC. In the next section, we discuss scope of developing our adaptive memory server strategies for HEC, using service oriented architecture.

4. SOA for Memory Service

The memory servers explained in the previous section are tightly coupled and have many limitations in fully utilizing the advances of network and processor technologies. The development of memory servers was aimed to improve disk access performance by utilizing the idle memory of peer nodes on cluster computers. However, due to this tight coupling the memory servers suffer from poor fault tolerance and scalability. Moreover, the performance gap between microprocessor and memory is rapidly increasing. The passive provision of memory space on memory servers is not sufficient to improve the overall performance of scientific and engineering applications.

The common memory server architecture (figure 2) and general process of SOA (figure 3) have similarities. In both architectures, there are clients that require memory service similar to a service consumer of SOA, and multiple servers that provide extra memory space for

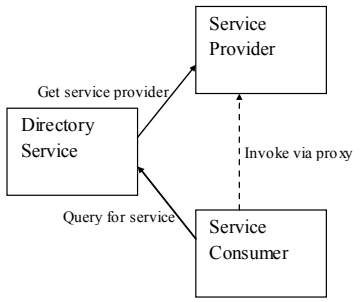

Figure 3. Directory service of SOA

the consumer entities. Memory Management Engine (MME) can act as a directory service between memory clients and memory servers such that MME collects the information of available servers and advertises that information for memory clients to choose a server. Moreover, the MME can also behave as a service provider entity to predict the future data references of memory clients and push that data from its location at memory servers to the clients [11]. This strategy overlaps the microprocessor stall time during data access more effectively and in turn reduces the execution time of application running on the memory clients.

SOA provides security, scalability, fault tolerance and interoperability. These features are beneficial for memory server architecture. Among the memory server systems described in the previous section, some of them suffer from scalability and fault tolerance problems. Providing dedicated servers and offering memory services, such as extra memory space and proactive data movement, improves the scalability and fault tolerance.

While SOA features are helpful for providing memory servers, the success depends on resolving several challenges. It is widely considered that the performance of services over the web (web services) is not beneficial to high performance computing. However, SOA is not limited to web services. By providing a service-oriented infrastructure (SOI) that is modified to fit the goals of HEC, benefits of SOA model can be applied for memory servers.

While MME provides discovery service, performance improvement depends on how often this service is needed. The interaction between memory client and MME to predict future references may affect the performance. To improve the performance effectively, it is possible to have data access profiles of clients before hand. Memory servers can be implemented on any multiprocessor environment such as clusters, shared memory processing machines, multicore processors, and distributed computing environments [11].

5. Conclusion

Memory server technology has been available for more than a decade to provide extra memory space for nodes that require more memory than they have. In this paper, we have briefly discussed many of these systems. With the advances in network, and microprocessor technologies we revisit this idea of using memory servers with the primary goal of improving data movement performance for high-end computing.

We presented the common architecture of existing memory servers (figure 2). The memory management engine (MME) of this architecture has been passively providing memory space for multiple clients in the existing memory servers. Following the service oriented computing concept, we propose that offering memory as a service by utilizing SOA features will benefit high-end computing to reduce processor stall time during data access. In future, we plan to explore the design of memory servers using SOA features and verify performance gain.

7. References

[1] Spencer Abraham, “Facilities for the Future of Science: A Twenty-Year Outlook”, *DOE Office of Science report*, 2003

[2] Anurag Acharya, Samir Koussih and Sanjeev Setia, “Dodo: A User-level System for Exploiting Idle Memory in Workstation Clusters”, *in HPDC ’99,* August 1999

[3] Eric A. Anderson and Jeanna M. Neefe, “An exploration of network RAM”, *Technical report*, Computer Science Division, UC Berkeley, December 1994

[4] D. Comer and J. Griffioen, “A New Design for Distributed Systems: The Remote Memory Model”, *In proceedings of the 1990 Summer USENIX Conference*, pages 127-136, June 1990

[5] DARPA, “High Productivity Computing Systems (HPCS), Vision: Focus on the Lost Dimension of HPC-User & System Efficiency and Productivity”

[6] M. Freeley, W. Morgan, F. Pighin, A.Karlin, and H. Levy, “Implementing Global Memory Management in a Workstation Cluster”, *in proceedings of the 15th ACM Symposium on Operating Systems Principles*, December 1995

[7] Michael Hines, Kartik Gopalan and Mark Lewandowsky, “Anemone: Adaptive Network Memory Engine”, *poster for NSDI, 2nd Symposium on Network Systems Design and Implementation*, Summer 2005

[8] L. Iftode, K. Li, and K. Petersen, “Memory servers for multicomputers”, *in Proceedings of COMPCON 93*, 1993, pp. 538—547

[9] Evangelos P. Markatos and George Dramitinos, “Implementation and Evaluation of a Remote Memory Pager”, *Technical Report FORTH/ICS 129*, Institute of Computer Science (ICS), Heraklio, Crete, GR-711-10.

[10] John Oleszkiewicz ,Li Xiao, and Yunhao Liu, "Parallel Network RAM: Effectively Utilizing Global Cluster Memory for Large Data-Intensive Parallel Programs", *33rd International Conference on Parallel Processing (ICPP 2004)*, Montreal, August, 2004.

[11] Xian-He Sun, and Surendra Byna, “Data-access Memory Servers for Multi-processor Environments”, *Technical Report (TR-IIT/CS-05-01)*, Illinois Institute of Technology, Chicago

[12] “Hard disk”, (http://www.answers.com/topic/hard-disk).