

Beyond Beowulf:

An innovative architecture to meet the Linux Cluster challenge

John Levesque, Julie Bou
Times N Systems™, Inc., USA

Abstract

This paper presents an innovative architecture used to improve a Linux Cluster, eliminating the traditional weaknesses, including system latency, poor CPU utilization, and sluggish I/O. At the same time, this architecture keeps a Linux Beowulf Cluster's strengths of low contention for system resources, high availability and high scalability, without performance degradation.

This new hardware and software paradigm, referred to as Shared Memory Clustering (SMC), is pioneered by Times N Systems, Inc. of Austin, Texas. With its inherent parallel architecture, Shared Memory Clustering virtualizes storage, increases I/O performance, and enhances the communications speed of Multi-level Parallelism (MLP) and Message Passing Interface (MPI) applications.

While it has many commercial applications, Shared Memory Clustering also performs extremely well as a super-strength Beowulf Linux Cluster running scientific and technical applications.

1 Introduction

Beowulf Clusters, first developed near the end of the last century, link together tens, hundreds, even thousands of processors to create a parallel computing system. The biggest benefit of Beowulf Cluster technology is the use of commercial-off-the-shelf (COTS) servers in the system, each of which contains one or multiple CPUs in a distributed memory environment. The use of industry standard servers provides a significant cost savings over traditional parallel computing systems. The major chink in a Beowulf Cluster's armor is its lack of

a robust parallel interconnect, which compromises the performance of both message passing and I/O activities. Like Beowulf, these Clusters have become heroes of strength. Now, through the sorcery of Shared Memory Clustering (read hard work and applied science), a Beowulf Cluster immerses with a new balance of power and performance.

1.1 Throwing Down the Gauntlet

Today, Beowulf Clusters are still haunted by fiends never banished from traditional parallel computing platforms. While they may rightly boast of low contention for system resources, high availability and good scalability, their undoing is communication latency due to messaging between processors, and imbalanced CPU utilization. Add to this the bane of I/O performance—something Beowulf Clusters haven't begun to address.

John M. May of Lawrence Livermore National Laboratory puts it this way in his book *Parallel I/O for High Performance Computing*:

Parallel programs cannot afford to treat I/O as an afterthought. The speed, memory size, and disk capacity of parallel computers continue to grow rapidly, but the rate at which disk drives can read and write data is improving much more slowly. As a result, the performance of carefully tuned parallel programs can slow dramatically when they read or write files, and the problem is likely to get worse.

1.2 Taking Up the Gauntlet

Shared Memory Clustering accepts this technological challenge, providing an innovative approach to solve I/O problems, more closely coupling servers in a Beowulf Cluster. SMC metamorphoses Beowulf Clusters into superior computing machines, utilizing their strengths and minimizing their weaknesses. When an application is ported to a Linux Cluster, typical concerns are latency and bandwidth of the communications fabric. If the applications require a significant amount of I/O, this must also be considered in light of network file sharing.

Shared Memory Clusters were originally introduced to perform tasks that were menial, but necessary for the last century, like Web hosting, file serving, and storage virtualization. Yet, when coupled with a fast Pentium IV processor and parallel programming techniques, they become excellent platforms for the alchemy of scientific and technical computing.

2 Hardware Swords & Shields

Transforming an existing Linux Cluster into a Shared Memory Cluster requires only simple magic on the part of hardware. Merely add a PCI-based Server Adapter to each node in the cluster, connecting it via fiber optic cables to a directly accessible pool of memory in the Shared Memory Node (SMN). This allows all servers to share resources across the entire system. When an application running on one server seeks certain memory addresses, the request is routed to the PCI bus through the fiber optic cables to the Shared Memory Node.

2.1 The Keep—The Shared Memory Node

The Shared Memory Node is the castle keep, enabling resources to be shared across the system. It facilitates efficient inter-node communications and synchronization protocols. Using a point-to-point protocol between servers and the Shared Memory Node, SMC eliminates the overhead of normal shared media network packets. This interconnect logic provides data transfers of up to 100 megabytes per second with a latency of 2.3 microseconds. Using redundant Shared Memory Nodes increases system reliability and speeds SMN read operations across a loaded system. Each Shared Memory Node houses a number of Shared Memory Node Controllers, an atomic complex and a pool of shared memory.

2.1.1 Shared Memory Node Controllers

One Shared Memory Node Controller provides an access point for up to four servers to communicate with the SMN. This controller card acts as the messenger of data between shared memory and the server. Each card has fiber interface transceivers and several custom chips. The fiber transceivers are connected to a serializer/deserializer (SERDES) chip, which is in turn connected to one of the chips containing custom logic. Data is received or transmitted via signals through the fiber cable. The received signal is deserialized by the SERDES chip and passed on to the custom chip. The data rate for this transmission is 125 megabits per second. To transmit data, the process is reversed. The custom chip presents a data byte to the SERDES chip to be broadcast by the fiber transceiver. The custom chips on the card provide the interface to the SERDES chips and the link to the SMN motherboard.

2.1.2 Hardware Atomic Complex

The atomic complex on the SMN commands all synchronization functions and non-memory transactions that are essential to maintaining memory coherency and message passing among nodes. Synchronization primitives enable atomic operations and maintain memory coherency. The atomic complex provides primitives for semaphores, unique counters, doorbells, SMN performance profiling, and important global information. The semaphore primitives provide locks for the servers, enabling software maintenance of shared memory

coherence. Unique counters enable the selection of a single master for a variety of tasks. Doorbell primitives are simple message passing commands. With a doorbell, one server signals another to initiate a predefined task. While all doorbells are candidates to become interrupts, the hardware efficiently combines approximately nine out of ten of these into a single interrupt. The performance profiling primitives provide diagnostics on the system with minimal impact on performance.

2.1.3 Memory

The Shared Memory Node addresses up to 1 gigabyte of shared, error checking and correcting (ECC) memory. This pool of memory is directly accessible to all servers. Direct Memory Access (DMA) reads and writes transfer data to and from the shared memory, with a burst feature providing even faster memory accesses. Auguring the future foretells that the Shared Memory Node will migrate to a system of memory banks, increasing its size and bandwidth.

2.2 Keys to the Keep—Server Nodes

Each node is a contributing member of a team, providing resources that are shared as needed to enhance system-wide performance. While each is an independent server, possessing power to process data, acquire a lock for controlling data access, or to move data as necessary, each pledges fealty to the team.

Each node holds a Server Adapter card, which sits on the node's motherboard and manifests itself to the CPU as a large chunk of memory. While a server's local memory is not directly accessible by others, the Shared Memory Node facilitates communications among the servers and allows access the SMN's pool of memory via Direct Memory Access operations. Operating system extensions and the hardware atomic complex synchronize data for assured coherence.

2.2.1 Server Adapter

The Server Adapter is a 32-bit 5V PCI card. One of these adapters is installed on every node. This card primarily functions as a memory controller.

The Server Adapter has a fiber interface gigabit transceiver, a SERDES chip and a custom chip. The custom chip facilitates communication between the PCI bus and the SERDES chip, initiating data transfers, directly accessing memory, and answering doorbell requests from the SMN through the PCI bus. It monitors the position of each server's shared memory window.

3 Software Sorcery

More powerful than the SMC's hardware is its software's ability to enchant the operating system into using the Shared Memory Node's resources without kernel modifications to the application. Shared Memory Clustering software architecture is designed to dramatically reduce interference by using a load/store data exchange mechanism.

3.1 Shape-shifting Capacity—Storage Virtualization

Shared Memory Clustering software facilitates the creation of Distributed Hard Drives and Distributed RAM Drives, which are visible to all nodes across the system. A Distributed Hard Drive is made up of raw, unformatted partitions taken from any number of different nodes. These multi-terabyte virtual parallel drives use the multiple I/O channels and multiple disk heads to read and write data across many machines.

A Distributed RAM Drive consists of blocks of memory contributed from the local memory of the servers across the system. These virtual multi-gigabyte RAM disks allow an application file of perhaps 5 or 6 gigabytes to be loaded into RAM for fast I/O access to the data without ever going to disk.

3.1.1 Flying without Modification

Application software and the operating system are completely unaware of the virtual nature of these drives. By merely installing the I/O drivers, sequential applications perform common Linux I/O tasks in parallel. Simply by allocating a file across a set of nodes using Distributed Drive technology, any file I/O becomes parallel without kernel modifications to the application. This is particularly useful for a file that is accessed extensively by a single processor. Since a sequential file has a single owner, file integrity is only assured when one processor accesses the file at a time. Using SMC, a single processor has the power to access a file spanning several nodes. It invisibly implements parallel reads and writes across multiple I/O channels. Under Shared Memory Clustering, file I/O is treated as parallel according to the configuration of the file, and any single access is distributed across the servers.

Another way that Shared Memory Clustering benefits a Beowulf Cluster's I/O is to become a Linux data block server. For example, by connecting a four node SMC to a Beowulf Cluster, each SMC node handles data block transfers, using the underlying parallel I/O for faster performance.

3.2 New Telepathy—A Shared File System

Parallel applications using a Global File System (GFS) implementation use multiple processors to issue file I/O requests to a single file. GFS is particularly well suited to Shared Memory Clustering, since this file system's lock manager resides in shared memory, readily accessible to all servers. GFS accesses parallel files through the SMN, significantly improving I/O performance.

3.2.1 Message Passing Interface

An optimized Message Passing Interface (MPI) provides low latency data exchange, facilitating efficient communications among servers through the Shared Memory Node. Special MPI buffers reside in the SMN's memory pool, utilizing a proprietary system of doorbell primitives to efficiently facilitate message passing among processors. Latencies of less than 5 microseconds are available now, with a bandwidth of 100 megabytes per second. Without a ritual examination of sheep entrails, divination is somewhat unclear, however, it is predicted that future implementation of Shared Memory Clustering's MPI could also supply latencies of less than 3 microseconds, four times the bandwidth, and an intelligent fabric, which uses the SMN's central processing unit to sum data and perform other functions that accelerate collective communication. These visions will be explored in future development.

3.2.2 Multi-level Parallelism

Shared Memory Clustering works its magic to achieve optimal performance by exploiting the advantages of using common memory to perform data exchanges. One approach that works particularly well with SMC architecture is Multi-level Parallelism (MLP), pioneered by Jim Taft of Advanced Management Technology Inc.—NAS.

Each processor reads and writes to the shared memory using one of MLP's four functions: allocate memory, read, write, and barrier. In Shared Memory Clustering technology, these four functions are implemented with the low level shared memory access library. MLP requires less data movement than MPI. It also directly allocates memory, reads and writes data directly into shared memory from the arrays, and synchronizes with low level calls.

In a finite difference application, data is contributed from surrounding blocks by ghost cells (also called stencils or halos), which contain input to the solution of each block's calculation. Using MLP, these ghost cells reside in shared memory. One processor writes its information to the ghost cell where the adjoining block can read it.

Large programs, like a finite difference application, are frequently dominated by simple global communications. For example, in the Parallel Ocean Program (POP) scaling is heavily dependent on a global summation of a single scalar value processed by individual processors. This scalar operation is performed in MPI with the MPI_ALLREDUCE call. In a Shared Memory Cluster, all processors

quickly store their scalar value in shared memory. Then, one processor reads all scalars with a single read, sums the value and stores the result back in shared memory where all processors can read the global sum as necessary. When using a large number of processors, this global sum is significantly faster than the traditional $\text{LOG}(n\text{procs})$ logarithmic communications process.

Another important operation handled more efficiently by Shared Memory Clustering is a barrier. Barriers are implemented in shared memory by having all processors read and write a variable that contains a single bit for each processor. Toggling the bit signals that the processor is at the barrier. When all bits are toggled, all processors can proceed.

3.3 Book of Spells—Programming Library

A library of low level data access functions is available for Fortran or C to optimize user applications for Shared Memory Clustering. These include routines to allocate memory, read and write memory, allocate locks, and access and release locks. Applications often require only simple processor communication where full MPI is overkill. By using the SMC library, a particular function is applied for greater performance without the tax of MPI administration.

4 The Alchemy of Applications on a Shared Memory Cluster

Just as our predecessor alchemists were commissioned to render base metals into gold or find the elixir of life, let us demonstrate (a bit more successfully) how a Beowulf Cluster can be given new life and extended profitability by the use of Shared Memory Clustering, using optimized MPI, MLP and shared memory functions.

Timing comparisons between a Shared Memory Cluster and a Beowulf Cluster show several important things. The first set of benchmarks compares the performance of MPI applications running on both SMC architecture and a traditional Beowulf Cluster. The second set demonstrates MLP's advantage over MPI. The final set shows how processor speed affects application performance, regardless of how the programming code has been optimized.

4.1 Turning a Beowulf Cluster into Gold

When comparing the performance of an MPI application running on both a Shared Memory Cluster and a Beowulf Cluster, the times of the SMC are superior as shown by benchmarks for several elementary MPI functions.

4.1.1 Round One: The Barrier Function Challenge

Notice the faster times for a 16-processor barrier on the Shared Memory Cluster versus the Beowulf Cluster. This is significant because so many applications spend a large amount of time synchronizing distributed processors using the barrier function.

(Benchmarks to be included.)

4.1.2 Round Two: The Global Reduction Function Challenge

An even clearer comparison is the global reduction function. Due to the shared memory between processors, all processors except the master write a word to shared memory. Then the master processor reads all the words with one read, sums the data and broadcasts the results. This is all that is needed to complete the operation. On the other hand, the process on a traditional Network Cluster is more complex, requiring Log N communication steps and a broadcast.

(Benchmarks to be included.)

4.2 Conjuring with MLP & MPI

While any sorcerer's apprentice could compare the timings of an SMC system with a traditional Beowulf Cluster running large applications, we'd like to take it a step further and address several important MLP and MPI functions. To demonstrate Shared Memory Clustering's superior performance, our mages were tasked with a series of challenges comparing a Beowulf Cluster with a Shared Memory Cluster, both using MPI and MLP.

The following tables compare the performance of SMC with a traditional Beowulf Cluster, both using MLP and MPI to execute several operations. Clearly, the increase gained by using SMC over Beowulf Clustering is further extended by employing MLP. These comparisons illustrate that the performance benefits gained by moving existing MPI applications to SMC architecture are further extended by implementing the MLP programming paradigm.

(Benchmarks to be included.)

4.3 Pentium III vs. Pentium IV Tournament

Total application performance on a parallel processing system is dependent on two attributes. The first is the speed of the nodes' processors. The second is how well the code is optimized to run on those processors. Many necromancers in the parallel computing industry practice the black art of retrograding their code so when it is multiplied across a number of processors, the factor of "improvement" will be greater. Do not be fooled by these parlor tricks. The more efficiently the parallel processing code is written, the faster the processes will be, regardless of the multiplier presented by the processors' speed. Given that, if optimized code is run on a faster processor, the results are even better.

For scientific and technical applications on a Shared Memory Clustering system, Pentium IV processors are highly recommended.

Below are several comparisons of applications comparing Pentium III and Pentium IV operations.

(Benchmarks to be included.)

5 Concluding the Transformation

The new Shared Memory Clustering architecture increases performance, handling parallel I/O requests for applications, both by itself and in conjunction with a Beowulf Cluster. It virtualizes storage, allowing applications to have more efficient and faster access to data. SMC's interconnect performs MPI applications with aplomb and excels in MLP applications written for remote shared memory. These systems provide an excellent platform for scientific and technical computing.

While we may laugh at the awe surrounding those revered as sorcerers of old, we pay tribute to the advances these alchemists and engineers made, paving the way for modern science and technology.