35 Uses of Cluster Interconnects: Distributed Shared Memory Systems, Computing Clusters and Cluster Filesystems / Databases

If you have not read the book (*Performance Assurance for IT Systems*) check the introduction to "More Tasters" on the web site http://www.b.king.dsl.pipex.com/ to understand the scope and objectives of tasters.

This taster follows on from *Cluster Interconnect Technologies,* which discussed the various underlying technologies that are currently available. The objective in this paper is to discuss some of the major uses of Cluster Interconnects: Distributed Shared Memory Systems; Computing Clusters; and Cluster Filesystems / Databases. It should be read in conjunction with the *Back-end Server Clusters* taster where the primary focus is on failover clusters to provide server resilience.

35.1 Starting Terminology

ccNUMA (cache coherent Non-Uniform Memory Access). A system whereby the caches of processors that reside on separate boards, and hence not on the same system bus, are kept coherent (i.e. memory changes are reflected across the entire system). See the *Multiprocessors (Shared Memory)* taster for detailed information.

MPI (Message Passing Interface) is the standard mechanism by which processes communicate across a cluster.

PVM (Parallel Virtual Machine) is an alternative to MPI. It provides a friendlier programming interface that is both flexible and portable. The main disadvantage is that it does not perform as well as MPI.

RDMA (Remote Direct Memory Access) is a mechanism that is used by cluster interconnect products to deliver low latency for access to memory on a remote server node.

R-NUMA (Reflective NUMA) is a hybrid of ccNUMA and S-COMA, transferring memory between nodes at either the cache line or page level. See the *Multiprocessors (Shared Memory)* taster for detailed information.

SDSM (Software-Based Distributed Shared Memory Systems). There are two types of distributed systems where memory is shared across multiple server nodes: hardware-based and software-based.

SMP (Symmetric MultiProcessor) is typically a single image system where the workload can be spread evenly over multiple processors by the operating system. See the *Multiprocessors (Shared Memory)* taster for detailed information.

S-COMA (Simple Cache Only Memory Access). This technique migrates / replicates memory between boards / nodes at the page level. See the *Multiprocessors (Shared Memory)* taster for detailed information.

35.2 Technology Outline

The focus in this taster is on the ability to access areas of memory on remote server nodes to provide a cluster-wide shared memory / cache or to support message passing-based inter-process communication. As shown in Figure 35-1, the facilities can be implemented at various places in the system hierarchy.

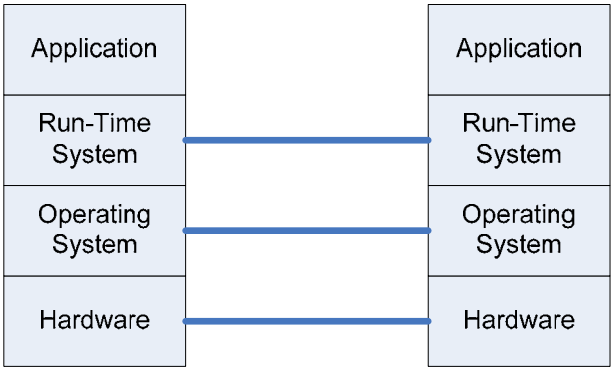

Figure 35-1 Places Where Shared Memory Can Be Implemented

35.2.1 Software-Based Distributed Shared Memory Systems

Distributed Shared Memory Systems typically work at the memory page level. Pages are migrated or replicated between server nodes, as required. Research in this area is split into two main areas: hardware-based and software-based solutions. Let us start with some brief notes on hardware approaches. See the *Multiprocessors (Shared Memory)* taster, which is primarily concerned with single image environments, for a more detailed discussion. In short, ccNUMA is a technique that is used to construct large SMP systems by the use of multiple processor / memory boards. Memory items are moved between boards, as dictated by the workload requirements. The size of an "item" is typically a cache line (32-64 bytes). S-COMA works at the page level rather than the cache line level. R-NUMA is a hybrid that works at both line and page level, using the most appropriate technique.

A considerable number of Software-Based Distributed Shared Memory Systems (SDSM) were developed in the late 1980s and 1990s, many in academia, to provide a cheaper solution which would support parallel computing and cluster filesystems across multiple server nodes, typically although not exclusively the nodes were uniprocessor systems (see Figure 35-2). It is important to note that the majority of implementations do not provide system-wide memory sharing; they tend to limit themselves to sharing a memory region to support parallel application(s). They were generally built as add-ons to a variety of operating systems, typically in the form of run-time systems. They work at the page level, generally relying on the memory management unit (MMU) which controls the virtual memory system. When a memory page fault occurs (because the required page is on another server node) the SDSM is responsible for getting the page from the remote node.

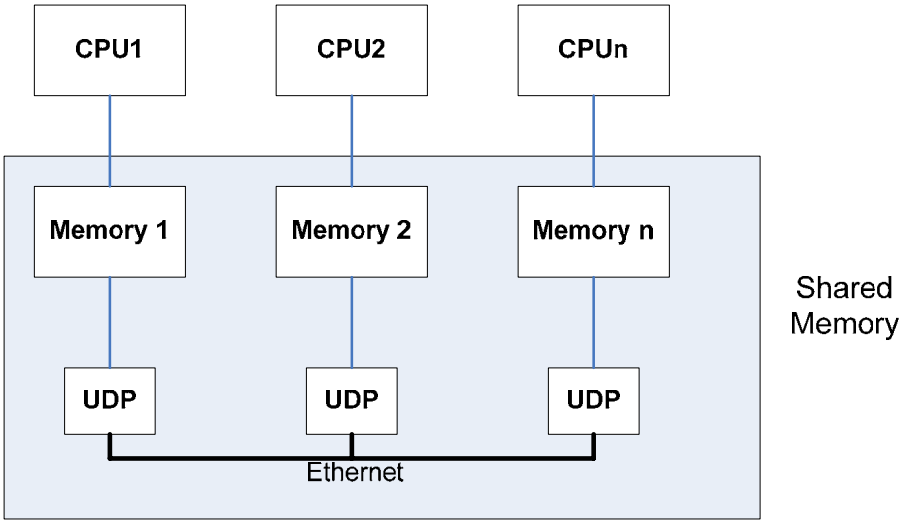

Figure 35-2 Software-Based Distributed Shared Memory (Using UDP over Ethernet)

Apart from the straightforward migration of a page, it can also be replicated in an effort to improve performance across the cluster. However, the presence of multiple instances of a page naturally complicates the issue of memory coherence. The initial consistency model was called sequential consistency. Here, when one node wishes to write a page, messages are sent to all nodes telling them to invalidate their copies; they comply, replying that they have done so; and hence there is now only one instance of the page, which can then be safely written. This process can generate a significant volume of inter-node traffic. The volume of traffic can be exacerbated by "false sharing". Here, two nodes are working with and updating different memory items that happen to be on the same page. This can cause a "ping pong" effect, e.g. node B has to read the revised page again after node A has updated it. When node B does its own write then node A's version must be invalidated, .. and potentially so on.

An improvement on sequential consistency is to use a release consistency model. This involves the use of a lock. Here only one node is allowed access to the data once the lock is set. Only when the lock is released (on completion of the write) does it have to tell another node about the fact that the data has changed. There are two types of release: eager and lazy. The eager variant propagates changes when the process holding the lock releases it, informing every node that is interested. The lazy variant waits until the next process has acquired the lock; this means that only one node needs to be informed.

In addition, there are multiple writer protocols. Here, each node takes a copy of the page and can then modify it, having first taken a copy of the original contents. At a synchronisation point (relating to a lock) it calculates what is called the "diff" (a word by word comparison of the modified and original pages). It then informs the other nodes of the changes. This causes them to invalidate their pages. Each node updates its copy of the page by applying the diff. This approach reduces bandwidth, as a diff is invariably shorter than a whole page. This approach also reduces the effects of false sharing.

All the techniques that have been discussed assume that nodes are not updating the same data item within a page at the same time; it is the responsibility of the application to ensure that this cannot happen.

The original SDSM protocols were based on UDP/IP while Ethernet-based networks provided the cluster interconnect. More recently, variants have been developed to use MPI running over a fast interconnect such as Infiniband.

Distributed Objects is a subject in its own right. It is mentioned here because a number of implementations are based on SDSM techniques, e.g. RThreads and DOSA. DOSA uses a handle table (which contains object ids and their related addresses). All access to objects is via this handle table. The Virtual Memory (VM) system is used for access detection at the page level. Object level protection is obtained via VM mappings; there are three of them (although there is only one physical object): read-only, read-write, and invalid, as shown in Figure 35-3. Object access protection is changed by switching the handle between the mappings. The release consistency model is similar to that which is described previously, albeit it works at the object level.

As a matter of interest, CORBA (arguably the most quoted distributed object system) adopts a totally different approach. It was developed to support a heterogeneous system (different hardware and operating system platforms). It is in essence a client-server system; client processes can invoke operations on objects that are located on other machines. It is similar in this respect to RPC (Remote Procedure Call). ORB (Object Request Broker) is the glue, hiding low level distribution and communications details. IIOP (Internet InterORB Protocol) has been used from V2.0 onwards. It should be noted that there is only one instance of an object (with the potential for scalability issues).

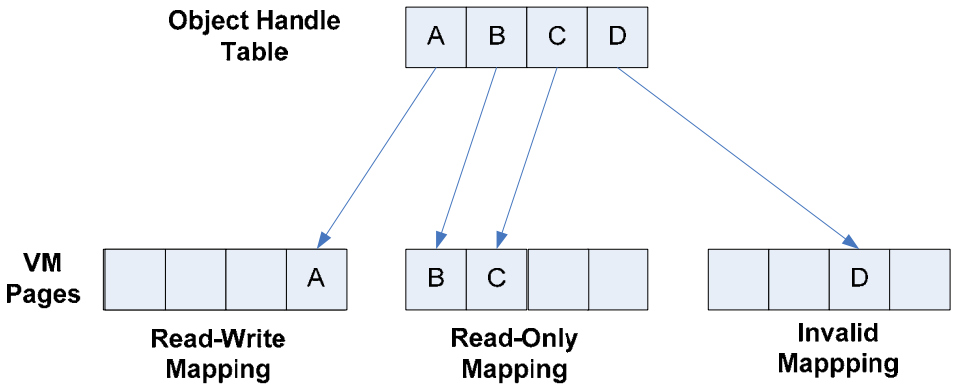

Figure 35-3 Access Protection Of Objects

35.2.2 Computing Clusters

Computing clusters are used mainly, though not exclusively, for scientific applications that have a requirement for parallel programming. Originally, supercomputers were the main, albeit expensive, solution for large scale problems that could not supported on standard single image computer environments. Although alternative solutions appeared, e.g. MPPs (the so-called Massively Parallel Processors) with proprietary inter-processor switches, there was a general clamour for cheaper solutions. In 1994, the Beowulf project within NASA produced the first “cheap” computing cluster, consisting of 16 Intel DX4 processors running Linux, utilising message passing techniques for inter-node communication via 10Mbit Ethernet. See Figure 35-4 for a high level view of a Beowulf cluster. The processors were too fast for the 10Mbit Ethernet and consequently drivers were developed to support “channel bonding”, i.e.

the use of two or more links, over which traffic could be striped. It was an immediate success, a fairly obvious statement given the fact that Beowulf Cluster soon became an accepted genre in the world of High Performance Computing (HPC).

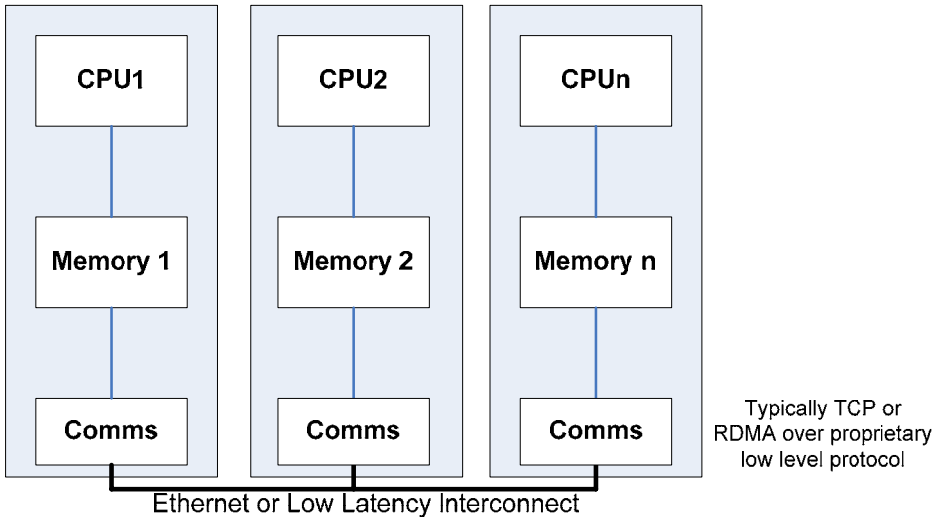

Figure 35-4 Beowulf-Class Computing Cluster

The main elements of a Beowulf cluster include:

- commodity hardware (typically Intel or AMD uniprocessors or dual processors), usually with as much memory as can be afforded
- a commodity operating system (mostly Linux or BSD)
- Message Passing libraries (typically MPI) to allow communication between the cluster nodes
- Cluster interconnect (commonly Ethernet initially)
- Other open source software, e.g. GNU compilers
- Optionally, a scheduling system to distribute the workload.
- A cluster-wide filesystem. This could simply be a vanilla-flavoured NFS system or something richer such as AFS (Andrew File System)
- Some form of cluster management software.

With respect to cluster interconnects, the most common approach was to use Ethernet with TCP/IP running over it. However, the relatively modest performance of Ethernet, particularly the high latency of TCP, and the subsequent growth of the interconnect market with the availability of commodity products at more cost-effective prices, has led to the increased use of faster interconnects with low latency characteristics, e.g. Quadrics, Myrinet, SCI, and latterly Infiniband, particularly for larger clusters.

35.2.3 Cluster-Wide Filesystems and Databases

These topics were initially introduced in the *Back-End Server Clusters* taster. The main issues in both areas are data integrity and cache coherence.

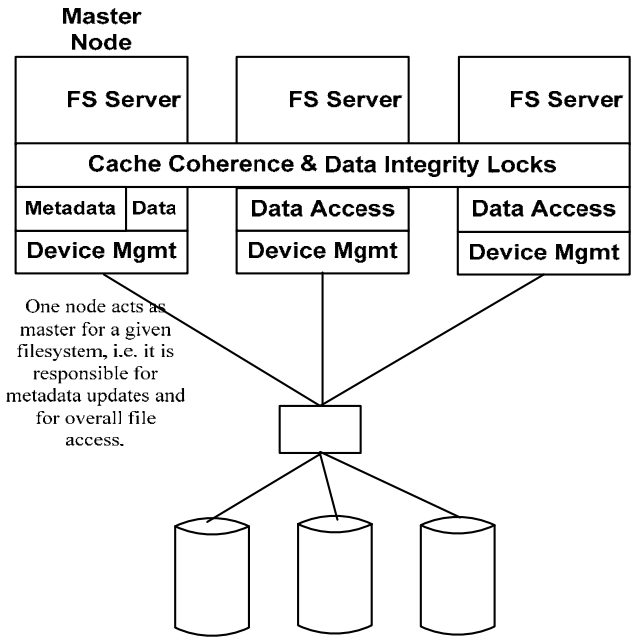

Figure 35-5 Example Of Cluster Filesystem with Master Node For Metadata Changes

Looking at cluster filesystems in the first instance, the simplest solution is to ensure that, while file reads can be done by any node, writes are limited to one node, usually called the master node. A refinement is to allow any node to perform a file write, but any metadata changes, including writes to the journal or intent log, must be done by the master node (see Figure 35-5). A further enhancement is that where there are multiple filesystems there is an ability to specify a different master node for each filesystem so that the update load can be spread across the cluster nodes. The ultimate solution is to allow any node to perform updates, a seeming "free for all". For a cluster-wide database (shared disk – see Figure 35-6) the same issues apply. Here there is arguably a more pressing requirement to support the "any node can update" feature.

Obviously, the more flexible the features are the more complex the underlying protocols must be to support them, and the greater the amount of inter-node traffic that is likely to be generated. Protocol features must include:

- Node membership
- Resilience issues (e.g. what happens if a master node fails)
- Global directory support (probably a "logical" concept, based on the individual physical local node directories)

- The use of locks or tokens to provide data integrity / coherence. For a filesystem the granularity may be at file level (typically) or a contiguous section of a file. For a database it is likely to be at the database page level

- Buffer cache coherence across the cluster (including migrating blocks to the cluster node that requires them)

- (Re)mastering where a node owns a resource and may therefore be involved in any processing – change of ownership may also be supported if data access patterns warrant it, i.e. another node is doing the majority of the accessing).

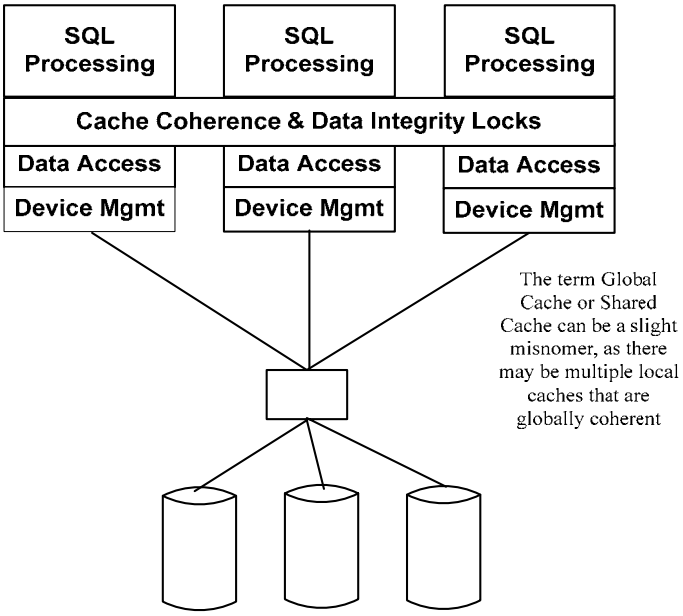

Figure 35-6 Cluster-Wide Database (Share Disk)

The standard communications protocol implementation is TCP/IP over Ethernet. As indicated in the *Cluster Interconnects* taster, Ethernet has general performance and throughput limitations, while TCP, being a fat protocol, imposes a relatively high latency. A first step that has been taken by many vendors has been to develop their own protocols that run directly on top of Ethernet e.g. Veritas's Cluster File System uses LLT (Low Latency Transport). Faster interconnects with lower latency, typically using RDMA, are gradually being supported; the pace of adoption arguably being faster on cluster databases, particularly for Oracle RAC (Real Application Cluster).

35.3 Performance

Check out the performance section of the *Cluster Interconnect Technologies* taster before reading this section. At the risk of sounding repetitive, the choice of cluster interconnect depends on the size of the system, the volume of inter-node traffic, the performance requirements in terms of throughput and latency, and last but not least the cost implications.

Software-based Distributed Shared Memory systems (S-DSMs) are significantly slower than hardware-based DSMs. Richer protocols invariably mean more inter-node traffic. Research is ongoing to improve protocol performance, e.g. reducing the number of explicit

acknowledgements that are required, particularly for home-based implementations where one node owns a page and has to get involved in page migration / replication. An obvious improvement is to support fast, low-latency interconnects which some of the products do.

Computing clusters. Cheaper interconnects (Fast Ethernet and Gigabit) are probably satisfactory for applications that are embarrassingly parallel (a somewhat quaint term that is used in the HPC community to indicate high levels of parallelism – as somebody with a commercial computing background I would be happy to be embarrassed anytime!) or moderately coupled problems where there are modest amounts of inter-node communication. Faster interconnects with low levels of latency are becoming widely used in this area with the growth in the number of larger systems that need to support significant volumes of inter-node traffic. In fact, it can be argued that computing clusters are the major driving forces behind the whole cluster interconnect market.

Cluster Database (Shared Disk). While fast, low latency interconnects have undoubtedly helped to promote this type of product, I question the advisability of placing too much trust in the marketing mantras which encourage you to forget all about the need for partitioning techniques on larger systems as a means of guaranteeing throughput and performance, stating that the product will do it all for you. Explicitly designed partitioning still has a role to play. I came across one very large system where the database requests were load balanced across the nodes in a rudimentary fashion, the objective being simply to spread the load in a non-intelligent way (from an application perspective) across the available nodes. This resulted in blocks being continually being migrated between nodes (ping-pong fashion) to maintain cache coherence.

Cluster Filesystems. Many filesystems have low to medium access rates and therefore the use of 100Mbit Ethernet or Gigabit Ethernet may be sufficient in many cases. It is arguably for this reason that not all products explicitly support faster cluster interconnects.

35.4 General Observations

It is important to check that a given software implementation supports the faster (non-Ethernet-based) cluster interconnects, particularly the low latency features (if they are required). Cluster interconnect vendors are generally moving towards multi-protocol support, and indeed some of them offer TCP, which may offer a half-way house solution for some sites. The emergence of Ethernet-based NICs that support RDMA (Remote Direct Memory Access) over TCP may also be of interest to some sites.

I have not mentioned OpenMP so far (MP stands for Multi Processing). OpenMP is a specification for a set of compiler directives, library routines, and environment variables that can be used to specify shared memory parallelism in Fortran and C/C++ programs. It is available on many platforms including Unix, Linux, and Windows. Jointly defined by a group of major computer hardware and software vendors, OpenMP is a portable, scalable model that provides programmers with a simple and flexible interface for developing parallel applications. It is frequently compared with MPI. It is easier to program than MPI, typically resulting in smaller programs than MPI, and it is more consistent with serial programs. However, MPI is better at parallelisation (this is the usual story about effort versus rewards, the old 80-20 rule – “OpenMP may give you 80% of the rewards for 20% of the effort”. The question is whether you need the extra rewards (in terms of performance). Having said that

they are frequently compared, e.g. OpenMP running in a distributed shared memory environment versus MPI on a Beowulf-class cluster, OpenMP can be run on a Beowulf cluster.

Cluster filesystems can be used to underpin databases. Oracle is a good example. RAC (Real Application Cluster), the cluster-wide database was originally limited to use with Oracle's own file handling, which it has now belatedly termed a cluster filesystem (OCFS). However, it is now possible to plug in specified third-party offerings, the most notable being Veritas's cluster file system.

Grids are worthy of a brief mention. They are one of the "IT flavours of the month" at the moment. Marketing people strive to include the term in their product literature, often when it is not warranted. This has led to a general confusion about what precisely a grid is. In particular, clusters get confused with grids. My definition of a grid is a large, disparate set of computer resources, possibly many thousands of machines, often running different hardware and operating system platforms, typically connected via the internet, that can be used to tackle problems that are too large for a single supercomputer or computing cluster. The problem can be partitioned such that each resource can run small, self-contained, elements. The techniques frequently make use of spare CPU cycles on desktops and workstations. Performance is a much less important issue than the fundamental fact that the problem can be tackled at all.

35.5 Further Reading

Significant material can be located in this field. I have limited the references to a couple of interesting papers and web sites of note.

Tannenbaum, A.S., *Modern Operating Systems*, 2nd ed., Prentice Hall, New Jersey, 2001. See chapter 8 on Multiple Processor Systems.

Van de Steen, A., Dongarra, J.J., *Overview of Recent Supercomputers*. This very useful document is periodically updated. It can be found on the Arcade EU web site http://www.arcade-eu.info

Hu, Y.C., Yu, W., Cox, A., *et al*, *Run-Time Support for Distributed Sharing in Safe Languages*, Purdue and Rice Universities.

Heinrich, M., Speight, E., *Providing Hardware DSM Performance at Software DSM Cost*, Technical Report CSL-TSR-2000-1008, Cornell University.

Barrios, M., Jones, T., Kinnane, S., *et al*, *Sizing and Tuning GPFS*, IBM Redbooks SG24-5610-00, September 1999

Schwarz, B., Ravi, G., *Designing A Balanced Architecture With Oracle RAC and Veritas Software for Linux*, Dell Power Solutions, October 2004.

Oracle 9i Real Application Clusters: Cache Fusion Delivers Scalability, Oracle whitepaper, February 2002

Web sites include:

http://www.beowulf.org Beowulf

http://www.ics.uci.edu/~javid/dsm.html DSM Information site

http://www.openmp.org OpenMP web site

http://lcic.org/ Linux Clustering Information Centre

