The following paper was originally published in the

Proceedings of the 3rd USENIX Windows NT Symposium

Seattle, Washington, USA, July 12–13, 1999

THE RECORD-BREAKING TERABYTE SORT ON A COMPAQ CLUSTER

Samuel A. Fineberg and Pankaj Mehra

The Record-Breaking Terabyte Sort on a Compaq Cluster

Samuel A. Fineberg
Sam.Fineberg@Compaq.com
Pankaj Mehra
Pankaj.Mehra@Compaq.com
Compaq Tandem Labs
19333 Vallco Parkway, M/S CAC01-27
Cupertino, CA 95014 USA

Abstract

Sandia National Laboratories (U.S. Department of Energy) and Compaq Computer Corporation built a 72-node Windows NT cluster, which Sandia utilizes for production work contracted by the U.S. government. Recently, Sandia and Compaq's Tandem Division collaborated on a project to run a 1-terabyte commercial-quality scalable sort on this cluster. The audited result was a new world record of 46.9 minutes, three times faster than the previous record held by a 32-processor shared-memory UNIX system. The external sort utilizes a unique, scalable algorithm that allows near-linear cluster scalability. The sort application exploits several key hardware and software technologies; these include dense-racking Pentium II based servers, Windows NT Workstation 4.0, the Virtual Interface Architecture, and the ServerNet I System Area Network (SAN). The sort code was highly CPU-efficient and stressed asynchronous and sequential I/O and IPC performance. The I/O performance when combined with a high-performance SAN, yielded supercomputer-class performance.

1. Introduction

High-performance sorting is important in many commercial applications that require fast search and analysis of large amounts of information (for example, data warehouse and Web searching solutions). We present the design of a cluster and a commercial-quality sorting algorithm that together achieved record-breaking performance for externally sorting a terabyte of data. The audited terabyte-sorting time of 46.9 minutes was three times faster than the previous record [NyK97], which was achieved on a 32-processor SGI Origin 2000.

The cluster consists of 72 Compaq Proliant™ servers running Microsoft Windows NT™, although only 68 of the servers were utilized for the sort. It was built in collaboration with US Department of Energy's Sandia National Laboratories in New Mexico, where it is currently deployed for production work contracted by the U.S. government. The cluster nodes communicate using Compaq's ServerNet-I SAN (System Area Network). The SAN topology uses unique dual-asymmetric network fabric technology for performance and reliability [MeH99].

The external three-pass parallel sorting algorithm is also expected to scale well on larger clusters. In fact, the algorithm was originally developed for Compaq's NonStop™ Kernel (NSK) based servers (and will be productized on both NSK and Windows NT clusters). The program uses an application-specific portability library, libPsrt, and a message-passing library, libVI. Section 4 describes the application software architecture in detail. LibVI builds upon the VI Primitives Library specified by the Virtual Interface Architecture (VIA) Specification [Com97]; it provides a robust, high-throughput, multi-threaded and thread-safe messaging layer with efficient waiting primitives based on NT's IO Completion Ports. LibPsrt builds upon libVI and supports more application-specific operations, such as remote I/O and memory management.

2. Cluster Hardware Configuration

The cluster hardware consisted of rack mounted Proliant servers, a ServerNet™ I SAN, and over 500 disks. The disks were either internal to the nodes or plugged in to Compaq hot-pluggable drive enclosures. The total purchase price for a similarly configured 68-node cluster would be $1.28M.

This system consists of the following hardware:

Item	Quantity
Servers	68 systems, 136 CPUs, 34GB RAM
Disks	269 Wide Ultra SCSI-3 busses, 537 disks, 62 external disk enclosures, 4.5TB storage
ServerNet	68 dual-port ServerNet I NICs, dual-fabric topology utilizing 48 6-port ServerNet I switches
Racking	22 racks, 12 KVM switches, 2 rack mounted flat-panel monitors
Operating System	67 Windows NT Workstation, one 70-license Windows NT Server
Ethernet Network	68 Embedded 100BaseT NICs (included in the servers) attached to a single Cisco Catalyst 5500 switch

The costs break down as shown in Figure 1. As shown in the graph, the largest portion of our system's cost was in disks and external disk enclosures. The second largest portion was the servers (including additional CPUs and memory added onto the base server configuration). The ServerNet network was only 16% of the system cost, including all of the cables, switches, and NICs. The racking gear, KVM (keyboard, video, mouse) switches, and the monitors used as system consoles contributed only 5% to the cost. The 72-port Cisco Catalyst 5500 Ethernet switch made up 3% of the system cost, and finally, the Windows NT operating system made up 2% of the system cost.

2.1 Compaq Proliant 1850R Servers

The full Sandia cluster contained 72 rack-mounted Compaq Proliant 1850R servers, each with:

- Two 400MHz Intel Pentium II Processors
- 512MB, 100Mhz SDRAM
- One integrated and one PCI-card based dual-Ultra-Wide SCSI-3 controller (for a total of 4 SCSI busses)
- One dual-ported ServerNet I NIC

Of these nodes, 68 were actually used for the sort: 67 as sort nodes and one as the sort manager. In addition, a total of over 500 SCSI disks (in varying configurations) were distributed among the 67 sort nodes (nominally, 8 disks per sort node — 7 for sorting and one for the OS). The disks were all striped using Windows NT's ftdisk utility. Each node ran Windows NT Workstation 4.0 Service Pack 3 (except the sort manager node which ran NT Server). All nodes were running a ServerNet-specific software stack comprising Compaq ServerNet I VI (SnVie) version 1.1.10, ServerNet PCI Adapter Driver (SPAD) version 1.4.3, and ServerNet SAN Manager (SANMAN) version 1.1.

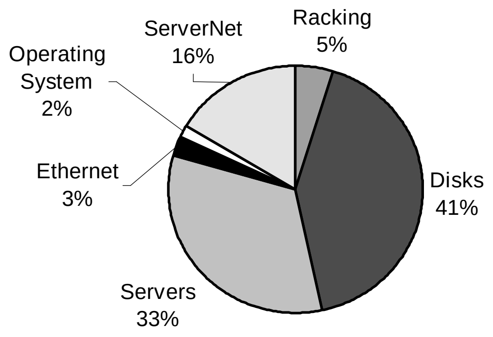

Figure 1: Cluster cost breakdown

The Proliant 1850R nodes were ideal for this cluster because of several key features: high memory bandwidth, small rack footprint, and integrated remote console for system management. Memory bandwidth of 450MB/sec was measured with the STREAMS benchmark on these nodes (using both CPUs). This is vital to achieving good sorting performance because much of the time spent sorting is spent in merges, which tend to stress processor-memory bandwidth in a manner similar to the long-vector accesses modeled by the STREAMS benchmark. The small footprint was important in order to make such a large cluster feasible. Proliant 1850Rs take only 3U (about 5.25 in/13.3 cm) of rack space, and include 3 1-inch hot-pluggable drive bays, room for 2 internal drives, and 4 PCI slots. Finally, the integrated manageability features allowed us to monitor and reboot nodes from a single system console using Compaq Insight Manager software.

2.2 ServerNet I SAN

Each ServerNet I PCI NIC provides two ServerNet ports: an "X" port and a "Y" port. Each port contains a transmitter and a receiver, both of which can operate concurrently, driving a bi-directional parallel LVDS cable in both directions simultaneously. The NICs in our system are interconnected by two fabrics of 6-port ServerNet-I crossbar routers (see Figure 2). An "X fabric" connects all the X ports, and a "Y fabric" connects all the Y ports. The two router fabrics are complete but asymmetric: each node interfaces with both fabrics and each fabric interfaces with every node, but the topologies of the two fabrics are different. Therefore, each fabric provides a path between every source-destination pair, but the path lengths (measured

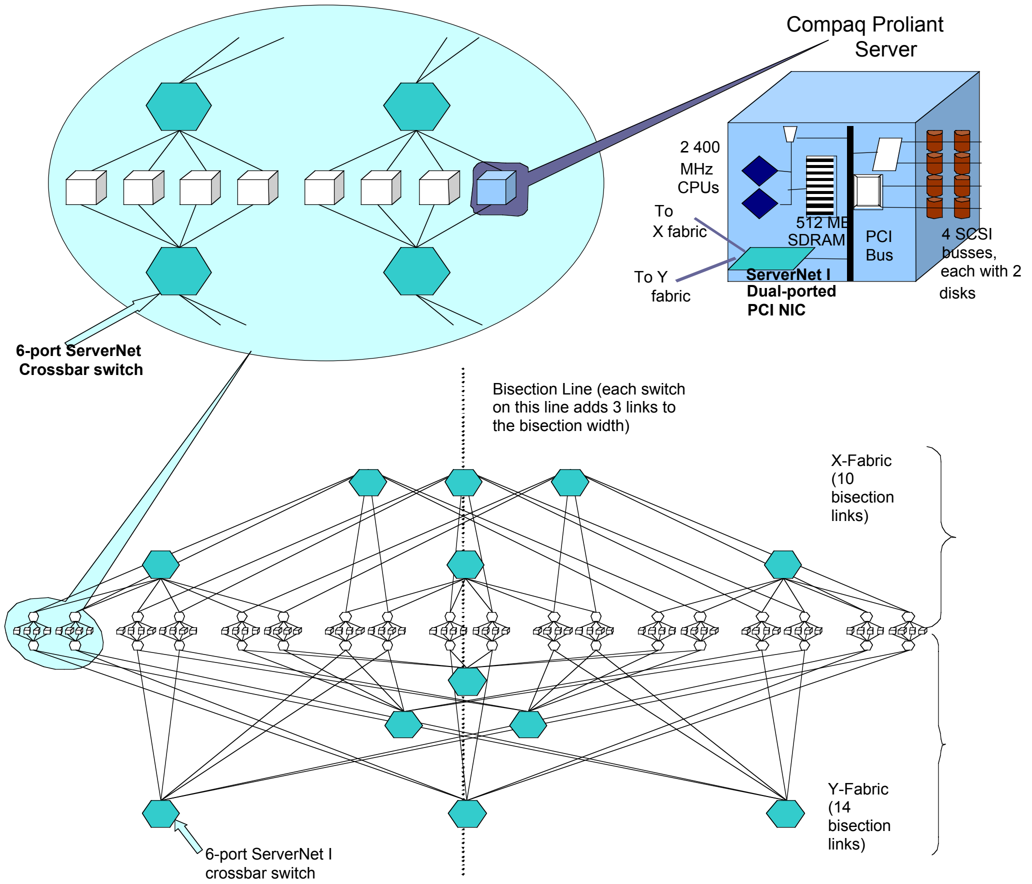
Compaq Proliant Server
2 400 MHz CPUs
To X fabric
To Y fabric
512 MB SDRAM
PCI Bus
4 SCSI busses, each with 2 disks
ServerNet I Dual-ported PCI NIC
6-port ServerNet Crossbar switch
Bisection Line (each switch on this line adds 3 links to the bisection width)
X-Fabric (10 bisection links)
Y-Fabric (14 bisection links)
6-port ServerNet I crossbar switch

Figure 2: Sandia network topology

in router hops) of most paths differ from one fabric to the other.

Traditionally, ServerNet topologies used two identical fabrics for reliability (but not for performance) by switching to a path in the Y fabric when the X fabric path to a destination was down, and *vice versa*. Here, we utilize the two fabrics for reliability *and* performance by configuring SnVie to "prefer" the shorter path when establishing a connection between two processes on different end nodes. This reduces network-routing latency (300 ns pipelined latency per packet per ServerNet I router) by creating and using paths with fewer hops; we also lower the probability of output-port contention (because each fabric handles only half of the total message traffic). Fault tolerance is maintained by this approach because there are at least two possible paths between each node pair, and traffic can always be routed via the alternate path if the preferred path fails.

3. The Sort Algorithm

David Cossock of Tandem Labs developed sort, merge, and parallel selection algorithms [Cos98] that were used. The file to be sorted is distributed across the local filesystems of the sort nodes, and sort processes are assigned to nodes. While the algorithm applies equally well to variable-length records, the results reported here are with files containing fixed-width records (80 bytes long) each containing an 8-byte (uniformly distributed) random integer key. The sort algorithm leaves the sorted data distributed among nodes such that the node-order concatenation of all the files consists of records sorted by the integer key in either ascending or descending order.

The three sort phases include the following:

- Phase 1 (local-sort): each node sorts memory-resident runs of the local partition.
- Phase 2 (local-merge): the local runs are merged so that each node has a completely sorted work file.

- Phase 3a (partition): The sort processes execute a parallel selection protocol over ServerNet, determining (1) the upper and lower bounds of their ultimate sorted output partition, and (2) the relative byte addresses, within each node's work file, of the beginning and end of records containing keys between these bounds. This is the only part of the sort that utilizes the sort monitor process.
- Phase 3b (parallel-merge): Using remote I/O across the ServerNet SAN, nodes merge pieces of each of the other nodes' work files to end up with their portion of the sorted data.

While Phases 1 and 2 stress the system balance within each node, stressing its CPU-memory and disk I/O subsystems, Phase 3b stresses each node's I/O bus and the cluster interconnect.

operation completion. This means that each sorting thread can issue commands, which can be IPC, remote I/O, or inter-thread (within a process) communication, and then it can wait on its own I/O completion port for the next completed operation or incoming request. The shadow thread is essentially a dispatch loop that receives requests and completion notifications on one IOCP and posts completion of local requests on another IOCP to be read by the local sorting thread. It also posts completions of remotely requested operations on VI connections (as "reply" messages).

4.1 LibPsrt

LibPsrt provides an RPC-like model, based on the I/O architecture of Compaq's NSK operating system (note: we could not utilize Windows NT RPC because it did

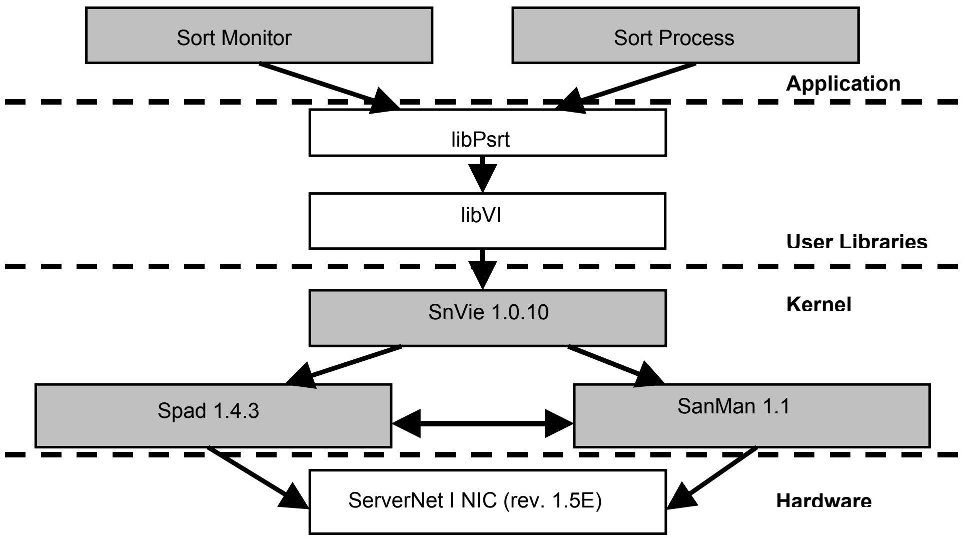

Figure 3: Sort Communication Architecture

4. Communication Software Architecture

The architecture of the sort application is shown in Figure 3. Each sort node runs a "sort process", and the sort manager node runs the "sort monitor" process. Each sort process contains two threads. A "sorting thread" implements the sorting algorithm minus remote I/O and interprocess communication (IPC), and a "shadow thread" performs, in the background, IPC and remote I/O (on behalf of other sort processes). Each of these threads uses a library called libPsrt, which provides mechanisms for both IPC and remote I/O. Local I/O is handled directly by the sort thread itself using NT's asynchronous ReadFile and WriteFile system calls and event-based completion. All remote I/O and IPC is also asynchronous and uses Windows NT I/O Completion Ports (IOCPs) to determine

not support the ServerNet SAN). In this model, sort and shadow threads issue "requests" to other threads that can be the same node or on any other node in the cluster. Each of the requests contains a command and a data portion. When the request is fulfilled, the thread sends a "reply", which also includes a data portion. An example of this is when the sort manager wants to obtain the "key" ranges in order to do partitioning in Phase 3a of sorting. It sends a message to each of the nodes with a "key request" command and a blank data segment. Each node replies to key request with its keys included in the reply's data segment. Because the sort process is not always ready to process requests, all IPC requests are first handled by the shadow thread, which forwards them to the sorting thread (i.e., it posts them to the sort thread's IOCP). Then, when it is ready, the sort thread reads these requests from its IOCP, and replies directly with the requested data. For remote I/O requests, the shadow thread reads the local sort file and responds directly to the requesting remote thread without involving the sort thread.

4.2 LibVI

LibPsrt is built on top of libVI, a message passing library built on the Virtual Interface Architecture (VIA). LibVI provides a traditional send/receive style messaging API. It is thread safe, and has support for asynchronous send/receive operations (each implemented as a synchronous call in its own thread). One key feature of libVI is its ability to post message completions through Windows NT I/O completion ports. For a more detailed description of libVI, see [Fin99].

4.3 ServerNet Drivers

LibVI uses SnVie, a kernel-level implementation of the Virtual Interface Architecture for ServerNet I hardware [Hor95]. It is intended as a porting vehicle for migration to ServerNet II [HeG98][1]. While SnVie provides all of the features of ServerNet II, it is implemented in software. Therefore, it has lower performance than ServerNet II VI, which will be implemented in hardware. In ServerNet II, VI data transfers will occur without any kernel transitions. LibVI will run without modification on ServerNet II when it becomes available. SnVie uses ServerNet I's TNet Services API, which is supported by the ServerNet PCI Adapter driver (SPAD). Name service support for low-level ServerNet Node IDs is provided by the SAN Manager driver (SANMAN).

5. Performance Studies with the Sandia Cluster

5.1 Sequential I/O Performance with Striped Disk Partitions

Windows NT offers several mechanisms for issuing and completing I/O requests. In this section we present some performance measurements that were made during sort application development. These results represent filesystem and disk performance under Windows NT, but not the performance of the sort application code. However, we utilized these measurements to make better choices when implementing the sort.

[1]for more information on ServerNet II see http://www.servernet.com

Riedel, et al [RiV98] report that striping large accesses across multiple disks, using unbuffered I/O, and having many outstanding requests are the ways to optimize sequential disk performance. Benchmark code for their work is also available from Microsoft Research (http://research.microsoft.com/barc/Sequential_IO/).
We augmented their benchmark code with support for IOCPs (I/O Completion Ports), an efficient mechanism for notification of outstanding I/O requests. Riedel, *et al* had found write performance saturating below read performance.

Using this benchmark, we found that:

- The best performance was given by asynchronous unbuffered I/O using either event-based or IOCP signaling;
- At 512KB I/O request size, peak read performance and very nearly the peak write performance could be achieved with just 2 I/O requests outstanding at any given time; otherwise, 8 outstanding I/O requests were needed for full throughput at 128KB I/O size; and
- We needed to stripe I/Os across either 2 10K-rpm or 3 7200-rpm disk drives in order to fully exploit a 40 MB/s Ultra-Wide SCSI bus.

In fact, a single Seagate ST39102LW (9 GB, 10K-rpm) drive was measured at 18.43 MB/s at 128KB I/O size with 8 outstanding requests under an Adaptec SCSI controller.

The Compaq Proliant 1850R nodes have an integrated Symbios 53c875 dual-channel SCSI controller. With 3 10K-rpm drives striped across two SCSI strings, at 512KB I/O size, and 20-deep asynchronous unbuffered I/O issue, we were able to reach a total bandwidth — reading or writing — of 53 MB/s. The deployed system has a second dual-channel SCSI controller in the form of Symbios 53c876 PCI card. At about 110MB/s the 32-bit 33-MHz PCI bus peaks; the four SCSI strings with 3 disks each are therefore quite enough to exploit all available I/O bus bandwidth in the server.

5.2 Communication Performance with LibVI over ServerNet I VI

5.2.1 ServerNet Hardware Performance and Scalability

Each ServerNet I link is rated at 50 MB/s in each direction. ServerNet I hardware-level packets contain up to 64 bytes of payload and 16 bytes of header and CRC. Each packet specifies either a read/write request or a read/write response. Only read responses and write requests carry payload. Requests and responses occur in one-to-one correspondence; *i.e.,* every request must be *acknowledged* by a response. It is easy to see that peak *uni*-directional data bandwidth on a ServerNet I link is at most 40 MB/s when we consider packetization and acknowledgement overheads; likewise, peak *bi*-directional data bandwidth on a ServerNet I link is 33.3 MB/s.

Each node is allowed to have up to 8 request packets outstanding. The SPAD driver configures this limit to 4. The ServerNet I NIC responds to requests received from the network by performing PCI read or write operations. It is important to note that, irrespective of the request type, every request-response sequence involves exactly one PCI (memory space) read transaction and one write transaction. While fewer than 4 requests are outstanding, an end node can continue to generate requests. Once the limit is reached, an end node can only issue responses; the next request must wait for a response.

Data transfer occurs when an initiating node uses its Block Transfer Engine (BTE) to initiate a request; at the target node, the Access Validation and Translation (AVT) logic carries out the specified operation and generates a response. ServerNet routers perform wormhole routing, allowing packet transfer to occur in a pipelined fashion: Each packet travels through the network as a train of symbols; the routing logic is pipelined so that within 300 nanoseconds of arriving at an input FIFO on the switch, the head of the train emerges at an output port with the remaining symbols following one per 50 MHz clock. So long as four or more requests can be completed per round-trip time, the pipeline will continue to run efficiently.

There are three principal sources of "pipeline bubbles" in ServerNet I: (1) PCI bus first-byte read latency, (2) single-threadedness of the BTE engine, and (3) output-port contention in routers.

The first of these reduces the average PCI efficiency of ServerNet I NICs, pushing the 32-bit 33-MHz PCI bus into saturation as soon as a node hits 33 MB/s uni-directional outbound traffic or 19 MB/s bi-directional traffic. PCI saturation can delay generation of response packets, causing packet round-trip times to shoot up.

The second causes the BTE to idle when all 4 requests are outstanding, pushing the network interface into an outstanding request (OR) limited mode, where the peak bandwidth of a connection drops to 4*64/RTT MB/s where RTT is the hardware round-trip time for a request-response pair in microseconds. Another property of a single-threaded BTE is its inability to use both fabrics simultaneously; at best, it can support static load balancing of connections between fabrics.

Output-port contention occurs in ServerNet routers when two packets try to go out of the same router port; one of them is blocked, causing other traffic behind it to back up. Thus, output-port contention causes congestion in the network, lowering the network's link utilization and throughput and increasing the RTT of packets. We hasten to note that ServerNet II has much better performance characteristics on all three counts.

5.2.2 LibVI Performance and Scalability

In this section we present raw performance measurements of the libVI communication library running over ServerNet I VI. We measured both point-to-point and all-to-all performance, though the sort primarily stresses the all-to-all bandwidth or the network.

5.2.2.1 Point-to-point latency and bandwidth

To measure point-to-point performance a simple ping-pong test was utilized. This test consisted of one process sending a message of a given size to another process, then that process sending the message back. The time for this operation was halved to get the one-way send time. For this test, all data were sent synchronously using blocking send and receive functions. In addition, all graphs in this paper use the VIA send/receive-based long message protocol (rather than the VIA remote-DMA based protocol) as described in [Fin99] because it performed better for sort phase 3b.

LibVI's basic message latency was about 138 microseconds. Note that ServerNet I VI is a software implementation, and this latency is system dependent.

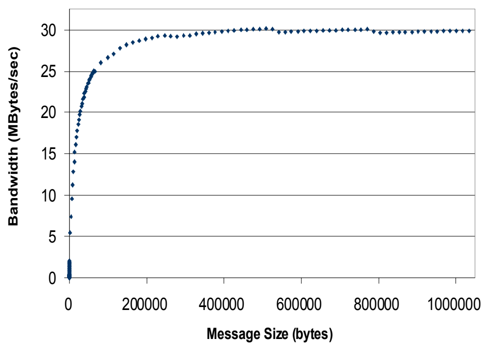

Figure 4: Point-to-point libVI bandwidth

Latencies between 80 and 200 microseconds have been observed with libVI on different systems. While this variation should decrease with ServerNet II's hardware VIA implementation, PCI and memory latency as well as CPU speed will always cause this to vary.

One of the primary advantages of VIA based networks is their ability to transfer long messages using DMA, freeing the processor to perform computation or to service short messages. LibVI was designed to minimize CPU utilization, and maximize the use of DMA. This was done primarily by using "wait" commands to check for message completion, rather than polling for completion. This implementation results in extremely low CPU utilization. For long messages, libVI's CPU utilization was less than 5% running at full bandwidth. Even for short messages, CPU utilization was quite low. However, this was done at the cost of message latency. It is likely that lower latency could be achieved with a polling-based short-message protocol, or a protocol that does not have an additional thread involved in message receipt. However, our protocol seeks to maximize concurrency and message throughput, a feature well suited to data-intensive applications. In fact, this sort was quite insensitive to latency. Sort performance was almost entirely a function of SAN and disk bandwidth. With such a low CPU utilization, it is possible to effectively overlap operations in the CPU with communication to an extent not possible with most message-passing libraries and networks.

Message bandwidth is shown in Figure 4. The point-to-point bandwidth achieved was 30MB/s. This difference between this and the "achievable peak" of 33.3MB/sec was due to the "long-message" protocol used for this measurement.

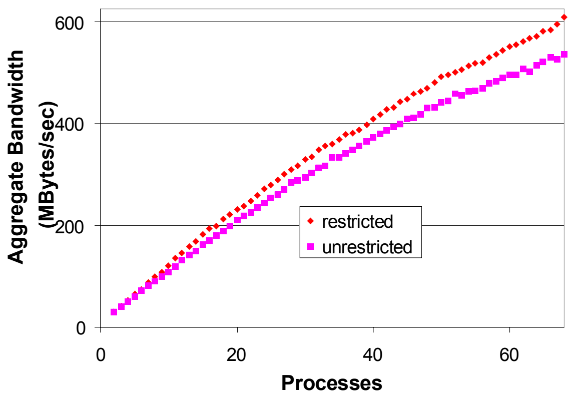

Figure 5: Long message aggregate bandwidth protocol comparison

LibVI internally can utilize either VIA's send/receive mechanism or its remote DMA mechanism for transferring messages larger than 56 bytes (small messages are always transferred with send/receive). In our experiments, VIA send/receive delivered better aggregate bandwidth when many nodes were communicating with many nodes simultaneously. However, point-to-point bandwidth of the send/receive based protocol was about 10% worse than the remote DMA based protocol (with only one pair of nodes communicating across the entire cluster). Because the sort performance was more dependant on aggregate bandwidth, we chose to use libVI's send/receive based long-message protocol for the sort. This protocol is a compile-time internal option for libVI, and does not change libVI's semantics or its API.

5.2.2.2 Aggregate bandwidth

One of the key metrics against which libVI was benchmarked is aggregate bandwidth. This was especially important for the Terabyte Sort code, which included a parallel-merge phase limited by the network's aggregate bandwidth. While these results were limited by the networking hardware utilized in the Sandia cluster, we also found some non-obvious protocol effects.

The aggregate bandwidth test utilizes the IOCP send/receive routines. The test initially issues one 512K byte send to every other process and one 512K byte receive from every process (the 512KB message size was chosen because it was the block size used in the parallel-merge phase of the sort). Then, it sits in a loop waiting on the IOCP for one of the operations to complete. As operations complete, the program re-issues the same operation (either or an asynchronous send or receive) to or from the same process. It continues to re-issue the operations until each specific send and receive has been issued 50 times. This results in the transmission of 25MB*<# of procs> of data both in and out of every process in the cluster. Because messages are issued in completion order, the communication pattern is random and there is no attempt made to alleviate hot-spots. This unstructured communication is very similar to the sort's parallel merge, and was a good predictor of sorting performance.

We ran this test with one process per node across the set of nodes from 0 to N-1 (where N is the total number of processes on which we ran the test). This resulted in the performance shown by the squares in Figure 5 (i.e., the "unrestricted" line). As can be seen from the graph, scaling is relatively linear, but per-node bandwidth is below the peak ServerNet bandwidth of 33MB/sec per direction.

In fact, we were achieving only about 8 MB/sec/node (bi-directional) vs. the "achievable" maximum, 19 MB/sec (bi-directional). The primary cause of this was that the network was not a fully connected crossbar. The topology we utilized was only capable of about _ of the peak bandwidth, so we should only be able to attain about 9.5MB/sec/node.

However, 8MB/sec/node was still slower than what we expected. This turned out to be from node contention. Essentially, if two or more nodes are sending or receiving data from a node in the system, then they will receive it slower than if they had sequentialized their access to that node. This is particularly true for the sort, where every node is likely to have multiple send/receive operations that are ready to begin at any given time. While it is combinatorially unlikely that 3 or more nodes will be reading from or writing to any node's ServerNet NIC, it is quite likely that there will always be some NIC that has at least two nodes contending for it. Unfortunately, it was impossible to orchestrate the communication in the sort code to fully alleviate this problem, but it was possible to impose restrictions on the libVI long-message protocol.

The changes to the libVI communication protocol introduced "restrictions" in different portions of the protocol. First, consider the send/receive protocol. In the standard "unrestricted" protocol, it is possible to be simultaneously receiving a message from and sending a message to every other node in the cluster. However, we can impose a lock that restricts this to a single "long-message" send and a single "long-message" receive active at any given time. The results of these changes are shown in the diamonds in Figure 5. As can be seen, for small numbers of nodes, the unrestricted protocols performed slightly better, but for larger number of nodes, the restricted send/receive performed significantly better. This difference was an increase in the bi-directional bandwidth of about 1MB/sec/node for the full machine, raising performance to about 9MB/sec/node. Therefore, the restricted send/receive protocol was used for the Terabyte Sort. However, for typical applications that do not perform the same amount of unstructured all-to-all communication, the unrestricted protocol is recommended since it performs better in the more common case.

5.3 Sort Application Performance

The following sorting performance was obtained on 68 nodes of our 72-node cluster. One node was dedicated to a sort monitoring process. Each of the remaining 67 nodes owned a partition consisting of 205,132,767 80-byte records (with 8-byte integer keys) for a total of 16,410,621,360 bytes per node. The benchmark sorts a total of 13,743,895,389 records in 46.9 minutes. The breakdown of this sort time is shown in Figure 6 and further described in the following sections.

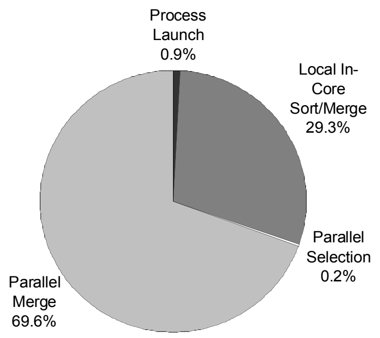

Figure 6: Breakdown of terabyte sort time

5.3.1 Process Launch and Connection Establishment

The initial 25 seconds of the sort were lost in process launching and communication initialization between sort processes. While this time may seem excessive, it was in fact the result of a significant amount of tuning. When the job starts the following must occur:

- the sort monitor must start the sort processes on each of the 67 sort nodes;
- the sort processes must establish VI connections with each of the other sort processes and the sort monitor; and finally,
- the sort monitor must send initial sort parameters to each of the sort processes.

The remote process launch was achieved with a custom RPC server on each of the sort nodes. This RPC server implemented a remote "create process" API, similar to the standard Win32 *CreateProcess* system call. Then, the nodes had to establish VI connections. Because libVI emulates a connectionless model, all VI connections must be established in advance, before the sort algorithm can start. Unfortunately, the connection mechanism in ServerNet I VI is inefficient, and highly timing dependent. While it is expected that the connection mechanism for ServerNet II will be better, connection establishment overhead remains one of the weaknesses of the VI Architecture standard, and it is unlikely it will ever be fast. If this were a significant performance problem for the sort, we could have overlapped part of the connection establishment process with Phase I of sorting. However, that would have required significant modification to libVI, which was undesirable in order to keep the code maintainable.

5.3.2 Local In-Core Sorting and Merging

Each node sorted approximately 16 GB of data in 13:05 to 13:45 minutes. As data were read and written once during sort and once during merge, 64 GB of net disk access was accomplished at a net I/O rate of 5.3 GB/s on 67 nodes, or 78.3 MB/s per node on 7 disks spread across 4 SCSI strings on a single PCI bus. One of the stripe sets consisted of 4 7200-rpm drives on 2 SCSI strings in an external disk enclosure. The other stripe set consisted of two internal and one hot-pluggable 10K-rpm drives installed in the server. These were also spread across two SCSI strings (the drive containing the system's OS shared the SCSI string with the single hot-pluggable 10K disk). In addition to this, a few of the stripe sets used additional disks to make up for underperforming drives. As noted above, this drive count is below what is needed to saturate the SCSI buses. Peak I/O rates observed while sorting were therefore disk-count limited. Performance on this phase during stress periods was also PCI limited; with dual-PCI systems the cluster would have hit an average I/O rate of 7.5 GB/s.

The application uses multi-threading and asynchronous file operations to provide input/output concurrency during the local sort and merge phases. The 16 GB data are treated as 116 runs of about 138 MB each. Each sort process, as discussed above, maintains 30 MB of outstanding requests with 2 MB buffers. Multiple asynchronous bulk reads are issued during the sort phase, while the double-buffered merge input consists of 116 concurrent 1.25 MB requests. The merge phase is seek-count limited (which could be optimized further on large-memory systems by utilizing larger buffers and doing fewer seeks). In spite of multi-threading, the sort phase incurs occasional CPU delays.

Sort Phase	CPU Utilization (out of 2.0)
Local Sort	1.3
Local Merge	0.65
Parallel Merge	0.18-0.22

Table 1: Sort CPU Utilization

CPU utilization for the entire sort was low, and it is likely that we could have achieved similar results with uniprocessor nodes. The results are summarized in Table 1. The "local sort" was the only phase of the sort that had a CPU utilization of more than one, and the "local merge" used only _ as much CPU as the local sort.

With the advent of Ultra2 Wide (i.e., 80 MB/sec or faster) SCSI and Fiber Channel SCSI systems, and dual, wider, and/or faster PCI busses available in many servers, we expect the time for these phases to drop below five minutes on newer systems.

5.3.3 Parallel Selection and Partitioning

Using a highly efficient parallel selection procedure, the sort took only 4-6 seconds to conduct a three round protocol of bounding, k-th record sampling, and merging, to determine what output partition needed which piece of each node's sorted data.

5.3.4 Parallel Merge

Essentially, this phase involves a global data movement in which almost every byte of data moves across the network in a random all-to-all communication pattern. This is the phase where the ServerNet I SAN and each node's PCI bus were stressed. Approximately a terabyte of data moved across the cluster in 32.6 minutes. This means that each node merged at about 8MB/sec for a total rate of 535MB/sec. To achieve this, each node had to simultaneously read its disk at 8MB/sec, write its disk at 8MB/sec, and sustain 8MB/sec of bi-directional bandwidth through its ServerNet NIC. The total bandwidth through the network was 535MB/sec. This was about 88% of the aggregate ServerNet bandwidth measured in Section 5.2, despite the increased load on the PCI bus due to SCSI disk transfers.

Because most of the real work in this phase was I/O or IPC, and both SCSI and ServerNet have very low CPU utilization, the total CPU for this phase was 0.18-0.22 across both CPUs (as shown in Table 1). While this is far less than the 2.0 CPUs we had available, we have observed that libVI's performance does drop substantially in a uniprocessor environment. This is due to the overhead associated with handling I/O and IPC interrupts and executing application code on the same processor.

The performance of this phase was PCI limited; high first-byte latencies on PCI bus drive small-transfer efficiency in to the 25% range ServerNet I's small packet size caused the inevitable PCI reads — 1 per 64-byte packet — to operate at this low efficiency.

The limiting effects of the PCI bus were further magnified by the single-threadedness of BTE. The network operated in outstanding request exhausted mode: requests could not be issued waiting for responses to earlier requests. Simulation of our topology under the Phase 3b workload [ShA99] showed that at 4 KB transfer size, a 50/50 mix of read and write requests, and a request injection rate of 4700 4KB-requests per second, the min, max and mean values of packet RTT were, respectively, 3.34, 682 and 25.8 microseconds. Therefore, whenever ServerNet I NIC went into outstanding request (OR) exhaustion (due to all 4 unacknowledged packets outstanding), performance dropped below 10 MB/sec bi-directional.

Simulations also showed that the network dynamics consisted of periodic phase transitions into and out of this OR-exhausted mode.

In addition, the 4-in 2-out topology at the periphery of the network was necessary to keep the router counts low. Output-port contention caused by traffic concentration further limited the utilization of the bisection links.

Even so, a sustained bisection bandwidth of 535MBps was achieved by the topology primarily because the asymmetric-fabric architecture kept the hop-counts low, thereby reducing the round-trip times. Many of the lessons learned about ServerNet I's large-scale performance influenced the design of ServerNet II.

With a dual-PCI server, and two ServerNet I NICs per node, the time for this phase would have been cut in half, falling to 15 minutes, without requiring any change in network topology. Coupled with 4-way 10K rpm disk striping per partition, estimated total sort time would be below 25 minutes.

With ServerNet II NICs (multi-threaded engine, efficient PCI operation due to 512-byte packet size, faster 125 MB/s links), 64-bit PCI busses, and ServerNet II's 12-port routers (which will drop inter-node distance to 2 hops), we expect Phase 3 time to drop below 10 minutes, to a point where even remote I/O will be limited by disk speeds.

To summarize, while the current result stands at 46.9 minutes to sort a terabyte of data, we envisage a time of about 15 minutes — a three-fold improvement — using commodity hardware available within the next few months. It is important to note that our sort algorithm, libPsrt and libVI are architected for scalability and efficiency; that is why, improvements in price and performance of sorting are expected to closely track forthcoming improvements in cost and performance of hardware.

6. Summary

This paper described the hardware and software used for Compaq's Sandia Terabyte Sort benchmark. The Sort exploited several key technologies including dense-racking Intel Pentium II based Compaq servers, Ultra-Wide SCSI disks and dual-channel controllers, Microsoft Windows NT 4.0, Compaq ServerNet I SAN, and the Virtual Interface Architecture. These technologies combined with a unique scalable sorting algorithm and highly efficient communication software, deliver high performance at a fraction of the cost of other solutions.

7. Acknowledgments

The authors of this paper would like to thank David Cossock, John Peck, and Don Wilson of Compaq Tandem Labs for their contributions to the sort application. Thanks to Jim Hamrick and Jim Lavallee for help with configuring and debugging ServerNet I and SnVie. In addition, we would like to thank Milt Clauser, Carl Diegert, and Kevin Kelsey of Sandia National Laboratories for purchasing the cluster, helping us put it together, and allowing us to use it for this work.

8. References

[Com97] Compaq Computer Corporation, Intel, and Microsoft, *Virtual Interface Architecture Specification, Version 1.0,* December 1997, http://www.viarch.org.

[Cos98] Cossock, D., " Method and Apparatus for Parallel Sorting using Parallel Selection/Partitioning," Compaq Computer Corporation, patent application filed November 1998.

[Fin99] Fineberg, S., Implementing a VIA-based Message Passing Library on Windows NT, Revision 1.0, Compaq Tandem Labs, January 1999.

[RiV98] Riedel, E., van Ingen, C., Gray, J., "Sequential I/O on Windows NT 4.0 — Achieving Top Performance," *2nd USENIX Windows NT Symposium Proceedings*, pp. 1-10, August 1998.

[HeG98] Heirich, A., Garcia, D., Knowles, M., and Horst, W., "ServerNet-II: a Reliable Interconnect for Scalable High Performance Cluster Computing," *Parallel Computing*, submitted for publication.

[Hor95] Horst, R. "TNet: a Reliable System Area Network," *IEEE Micro*, Volume 15, Number 1, pp. 37-45, February 1995.

[MeH99] Mehra, P. and Horst, R.W., "Switch Network using Asymmetric Multi-Switch/Multi-Switch-Group Interconnect," Compaq Computer Corporation, patent application filed March 1999.

[NyK97] Nyberg, C, Koester, C., Gray, J., *Nsort: a Parallel Sorting program for NUMA and SMP Machines*, Ordinal Technologies, Inc. white paper, November 1997, http://www.ordinal.com.

[ShA99] Shurbanov, V., Avresky, D.R., Mehra, P., Horst, R., "Comparative Evaluation and Analysis of System-Area Network Topologies," *1999 International Parallel Processing Symposium*, submitted for review.