Sorting on a Cluster Attached to a Storage-Area Network

Jim Wyllie
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120
wyllie@almaden.ibm.com

Abstract

In November 2004, the SAN Cluster Sort program (SCS) set new records for the Indy versions of the Minute and TeraByte Sorts. SCS ran on a cluster of 40 dual-processor Itanium2 nodes on the show floor at the Supercomputing 2004 conference (SC04), performing its data accesses to 240 SAN-attached 8+P RAID5 arrays managed by the IBM General Parallel File System. This hardware and software combination achieved peak data transfer rates of over 14GB/sec, while sorting a 125GB input file in 58.7 seconds, and a 1TB input file in 7 minutes, 17 seconds.

Introduction

In 1985, an article in Datamation magazine proposed that sorting one million records of 100 bytes each, with random 10 bytes keys, would be a useful measure of computer systems I/O performance [1]. The ground rules of that benchmark require that all input must start on disk, all output must end on disk, and that the overhead to start the program and create the output file must be included in the benchmark time. Input and output must use operating system files, not raw disk partitions. Constant improvements in computer hardware and sort algorithms have reduced the time for the Datamation sort from over an hour [15] to less than half a second [12]. Several variations on the basic theme have evolved over the years. "Minute Sort" [3, 8, 11] measures how much can be sorted in one minute, while "TeraByte Sort" measures how fast 1TB of data can be sorted [8, 9, 16]. Jim Gray is the unofficial arbiter and record keeper for sort benchmarks. His web site [5] distinguishes between two categories of sort results. "Daytona" sorts use commercially available sort programs, while "Indy" sorts are programs customized for the specifics of the sort benchmark. The SAN Cluster Sort program (SCS) described in this paper improves substantially upon the best Indy results for the TeraByte and Minute Sorts.

Hardware

SCS runs on a loosely-coupled cluster of nodes, and uses SAN-attached shared disks for its input, output, and work files. The specific record-setting runs described here ran on a cluster consisting of the following hardware components:

- 40 dual-processor Intel Itanium2 nodes using the SR870BN4 server platform [7]. Each node contains 2 64-bit Itanium2 processors clocked at 1.3GHz, 4GiB of RAM, a single gigabit Ethernet interface, a local SCSI disk used by the operating system, and 3 2Gb/sec fibre channel host bus adapters. The nodes have enough PCI busses so that each fibre channel interface card has its own bus. The nodes belong to the San Diego Supercomputing Center (SDSC), and were located in the SDSC booth at SC04.
- 60 IBM TotalStorage DS4300 RAID subsystems [6]. Each RAID subsystem contains two controllers, each with its own fibre channel host interface. Each DS4300 holds 14 73GB 10000RPM fibre channel disk drives, and is connected to two expansion drawers with 14 additional disks each. The 42 total drives assigned to a DS4300 are configured as 4 8+P RAID5 arrays plus two hot spares, with the remaining 4 drives used by another demonstration at SC04. The 15 racks of RAID controllers and their disks were located in the Storcloud booth at the conference.
- 3 Brocade Silkworm 24000 fibre channel switches. Each switch contains 128 2Gb/sec fibre channel ports, of which 120 are used. Each switch is connected to all 40 of the Itanium2 nodes through a single fiber each, and to both controllers in 1/3 of the RAID subsystems with redundant fibers (1/3 * 60 DS4300 * 2 controllers/DS4300 * 2 fibers/controller = 80 fibers). There are no inter-switch links between the fibre channel switches.
- A Force10 model E1200 Ethernet switch. The gigabit Ethernet ports of each node are connected to the Force10 switch, and from there to the Teragrid backbone. During SCS runs, these links carry only control traffic; all data movement uses the fibre channel SAN.

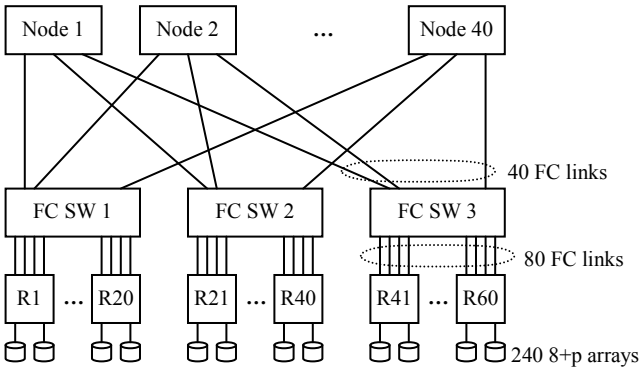

Figure 1. Sort hardware at SC04.

In total, the sort cluster contains 160GiB of RAM and 2560 disk drives (40 internal SCSI + 60*42 external FC). The total formatted capacity of the external RAID used by SCS is over 140TB (60 DS4300 * 4 arrays/DS4300 * 8 data disks/array * 73GB/disk). The aggregate I/O bandwidth of the RAID arrays is over 14GB/sec. Figure 1 shows the hardware available to SCS at SC04.

System Software

The 40 Itanium2 nodes run SUSE LINUX Enterprise Server 8 (SLES8). This distribution is based on version 2.4 of the Linux kernel.

The IBM General Parallel File System version 2.3 (GPFS) [13] manages the 240 RAID arrays as a single mountable file system. GPFS uses wide striping to distribute files across multiple RAID arrays. This allows all I/O subsystem bandwidth to be brought to bear on a single file when necessary. GPFS is a true cluster file system; there is no central server and therefore no single node that can be saturated with data transfer requests. Instead, GPFS accesses data directly over the SAN without intermediate data transfers. No semantic sacrifices have been made in delivering this level of performance; instances of GPFS coordinate their activity such that X/Open file semantics are preserved. All nodes see the same name space and file contents at all times.

In addition to data transfers, many common control functions in GPFS are also distributed in a scalable way. For example, when SCS writes its output, many nodes write into non-overlapping regions of the same file simultaneously, and each of these nodes must be able to allocate disk blocks. The GPFS space allocation maps are organized in such a way that multiple nodes can concurrently allocate space independently of one another, preventing space allocation from becoming a bottleneck.

SCS uses the MPICH [4] implementation of the Message Passing Interface (MPI) [14] to coordinate its operation across multiple nodes. By using the secure server option of MPICH to accelerate program startup, SCS is able to start running on all 40 nodes in about 6 seconds, compared to about 15 seconds using ssh. This program startup time is included in the performance results reported here.

SAN Cluster Sort Program

The SCS program is a custom C program containing approximately 3300 source lines. Since the total amount of main memory available in the cluster is significantly less than 1TB, SCS must use temporary files on disk. This makes SCS into a two-pass sort; each input record is read and written twice, including the traffic to and from the temporary files.

In its first pass, SCS distributes records into *slices* of approximately equal size. A slice consists of records with a contiguous range of keys. In the second pass, nodes buffer entire slices in memory, sort them, and write sorted slices to their correct positions in the output file. The number of slices depends on the amount of buffering available in the second pass. SCS allocates 768MiB of buffers to each slice.

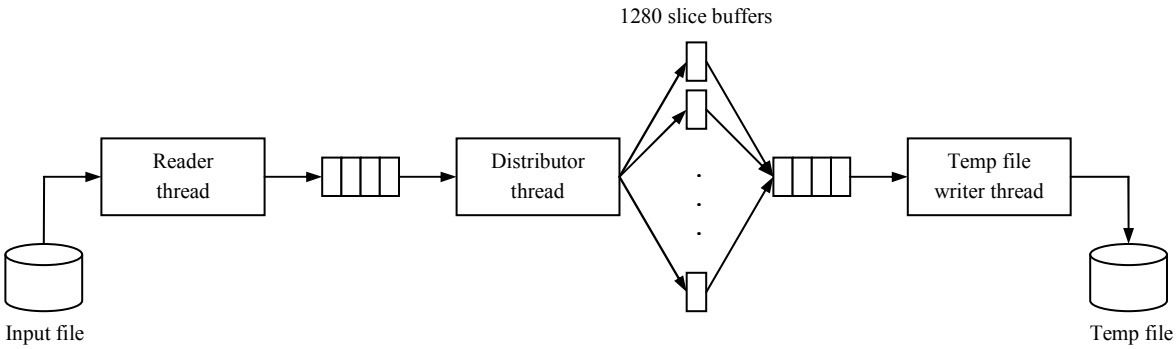

Figure 2. SCS pass 1 processing on each node.

This allows two slices to comfortably fit in memory at once, so SCS can overlap the reading of slice *n+1* with the sorting of slice *n* during pass 2. A 1TB input file and a 768MiB buffer size implies 1242 slices. To improve load-balancing, SCS requires that each node process the same number of slices, so the actual number of slices used is rounded up to 1280. Due to the random nature of the input, slices have approximately, but not exactly, the same number of records. In the TeraByte Sort, the difference between the sizes of the smallest and largest slices was less than 0.3%.

Figure 2 shows the processing done in the first pass of SCS in more detail. There are three threads per node, connected by simple producer-consumer queues. Each reader thread reads 1MiB chunks of the input file sequentially, starting at a file offset that is computed from its node number such that all records in the input file are read by exactly one node. The distributor thread on each node consumes buffers read by the reader thread and moves records into 1MiB buffers according to which slice the record belongs. Since keys are random binary data, assignment to a slice can be done simply by dividing the high-order bits of the key by the number of slices. As slice buffers fill, the distributor thread queues them to another thread that appends buffers to a per-node temporary file in the shared GPFS file system.

Once every node finishes the first pass, all records from the input file have been clustered by slice into blocks in one of the per-node temporary files. Except for partial buffers flushed at the conclusion of pass 1, all slice blocks are aligned on 1MiB boundaries in the temporary files. This size matches the GPFS block size and the RAID full stripe size, so reading slices back during pass 2 can be done efficiently. The order in which slice blocks appear in the temporary files depends on the order in which slice buffers fill, which is random, but each node remembers in memory which slices it wrote at which offsets in its temporary file. Between pass 1 and pass 2, nodes exchange this mapping information using MPI_AllGather. By the time the second pass begins, each node has a complete picture of the layout of every temporary file, as well as counts of the number of records in each slice. MPI is only used to exchange metadata about where blocks of records are located in the temporary files; all actual data movement from node to node occurs through the temporary files on disk.

During the second pass of SCS, nodes are responsible for non-overlapping ranges of the output file. Independently, each node reads entire slices back into memory before sorting them. Since the blocks comprising a slice are scattered in random 1MiB pieces across all of the per-node temporary files, SCS employs multiple parallel threads on each node to read the slice blocks in order to improve its I/O throughput (see Figure 3). To sort a slice once it has been fully buffered, SCS builds an array of <key-prefix, record-pointer> pairs and does a radix sort on the high-order 32 bits of the 10 byte keys. This technique assists in maintaining cache locality, and has been described elsewhere [2, 10, 16]. The pass 2 sort threads break ties in the high-order key bits using bubble sort, then gather records into buffers that they queue to a writer thread. Using the slice size information gathered from all nodes between passes, the writer thread computes where in the output file to begin writing each slice. To overlap sorting one slice with reading the next slice, the second pass of SCS has two sets of reader and sort threads per node, as shown in Figure 3. The two sets of threads are synchronized to insure that only one slice at a time is read, sorted, or written.

SCS makes use of several other techniques to improve its running time:

- SCS does not create its temporary files until after beginning to read and distribute input records.
- Gathering the temporary file mapping information from all nodes can occur before all of the pass 1

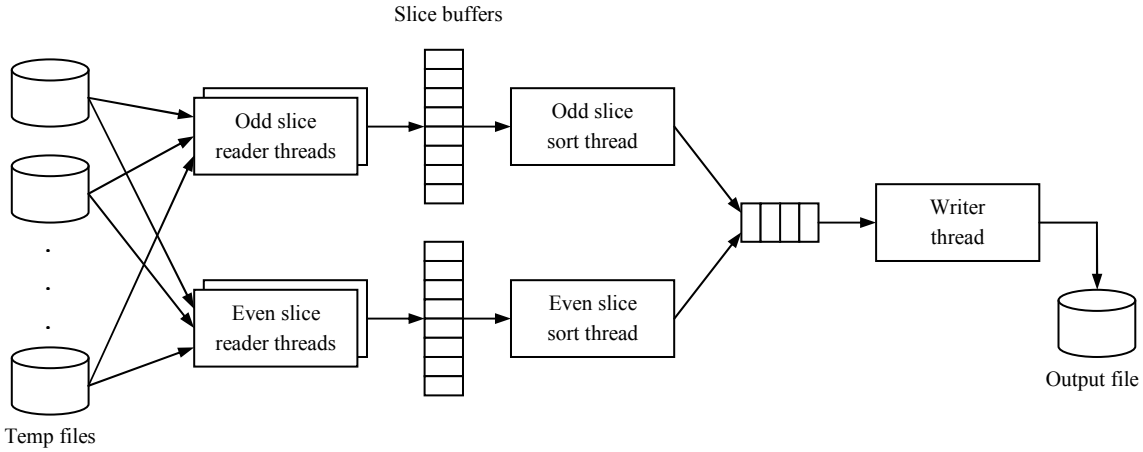

Figure 3. SCS pass 2 processing on each node.

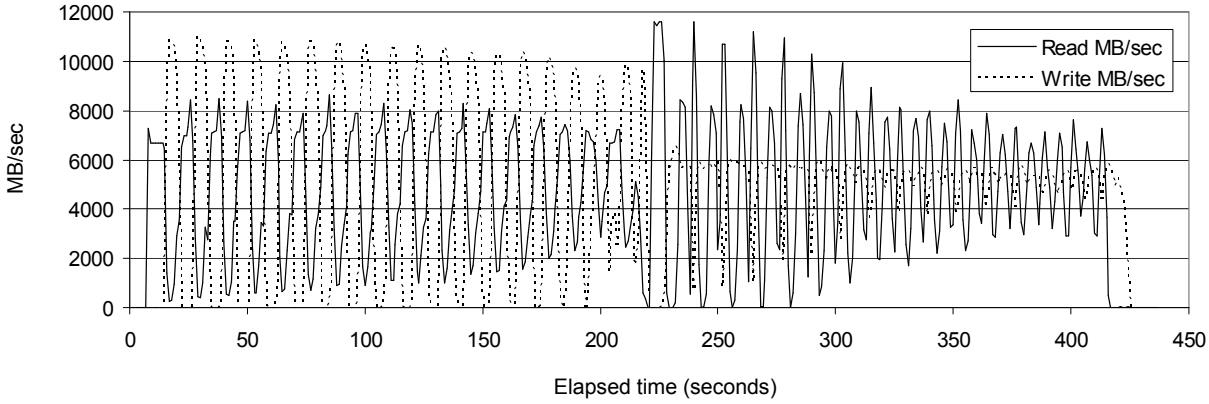

Figure 4. Aggregate cluster I/O throughput during TeraByte Sort.

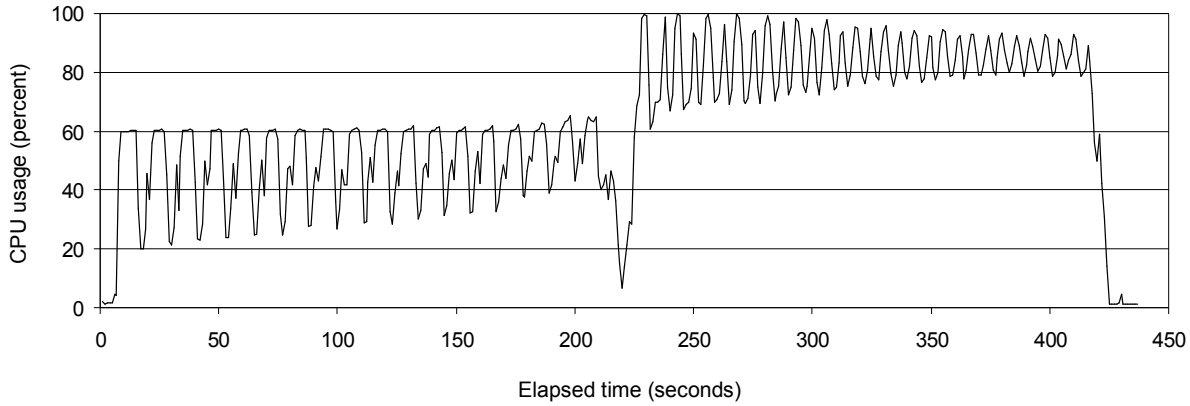

Figure 5. Cluster CPU usage during TeraByte Sort.

threads have exited. The locations of the slice buffers in the temporary files are known as soon as all buffers have been queued to the temporary file writer thread. Thus, SCS begins exchanging mapping information before the writer threads exit.

- GPFS employs distributed byte-range locks to insure cache consistency for files that are accessed from several nodes. Normally, read and write system calls implicitly drive the movement of byte-range locks from one node to another. To avoid unnecessary overhead, SCS explicitly manages these byte-range locks using optional GPFS hint calls. Turning off GPFS hints increases the running time to sort 125GB by 1.8%, based on three runs by each method.
- SCS destroys its temporary files as soon as the last slice has been read by all nodes, in parallel with sorting and outputting the final slice.

SCS contains code to write time-stamped trace records to local files at various points in its execution. These traces indicate that the techniques above are each responsible for no more than 250ms of improvement to the running time, except for GPFS hints as noted.

TeraByte Sort Performance Analysis

SCS sorted a 1TB input file in 437 seconds using the cluster at SC04, improving on the previous record by a factor of 2.4 [16]. Figure 4 shows the aggregate I/O throughput delivered by the GPFS file system across all 40 nodes during the record-setting TeraByte Sort run. Figure 5 shows CPU usage over the same period, where 100% means that all 80 Itanium2 processors in the cluster were completely busy. This data was gathered by summing the output of the Linux vmstat utility from all nodes. The transition from pass 1 to pass 2 at about elapsed time 223 seconds can be seen clearly.

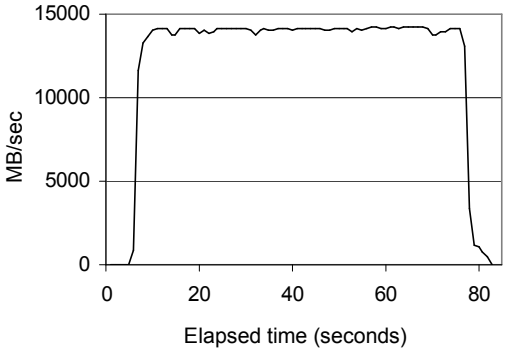

Figure 6. Aggregate read throughput of parallel validation program.

The most striking feature of these graphs is their periodic nature. During the first pass, SCS distributes records into 1280 1MiB buffers, one for each slice. Since the key range spanned by each slice is the same, the uniform distribution of input keys implies that all of the slice buffers fill up at nearly the same time. This leads to bursts of writing every few seconds, along with drops in the read rate and CPU usage until enough buffers have been cleaned to allow the reader and distributor threads to continue. Dividing a 1TB file by 1280*40 1MiB buffers yields 18.6, and indeed there are 18 large peaks and one smaller peak in the throughput graph of pass 1.

In pass 2, there are 31 peaks in the I/O throughput graph of Figure 4, corresponding to the 32 slices sorted by each node. The first read peak is wider than the others because it includes reading the first two slices. The high and low limits of the pass 2 graphs appear to converge towards an intermediate value as time advances. This is an artifact of how data from the nodes are combined. All nodes do approximately the same amount of work at approximately the same rate. Initially, their measured performance variations are in phase. After running for a while, however, the nodes drift out of phase and their aggregate performance measures begin to show interference effects, attenuating the apparent variation in throughput and CPU usage.

To confirm that the 1TB output file was sorted, a parallel validation program read the output and verified that records had non-decreasing keys. The program also verified that no records were corrupted, using record and file checksums. The validation program ran at a sustained rate of over 14GB/sec, as shown in Figure 6. This rate agrees with what was measured independently using raw I/O, establishing that GPFS can drive the disks to their throughput limit.

Minute Sort Performance Analysis

SCS sorted a 125GB input file in 58.7 seconds, establishing a new record for Minute Sort. This file would have fit in the aggregate memory of the 40 nodes in the sort cluster. A single-pass sort program could have avoided disk I/O to temporary files by shipping records by slice directly to buffers in their destination nodes. However, as the hardware was configured, the fastest way to move data between nodes was to write data to disk on the source node and then read it on the target. Although a one-pass algorithm would have been possible for Minute Sort, its performance would have been worse than SCS on the cluster available at SC04. Thus, the same two-pass program configured with the same buffer sizes was used. Sorting 125GB with SCS is a 1/8 scale version of sorting 1TB; instead of 1280 slices, there are 160. SCS was able to sort 12.5% of a terabyte in 13.4% of the time it took to sort the terabyte. The increase is due to the fixed 6 second program startup latency.

Figure 7 shows disk throughput and CPU usage during the

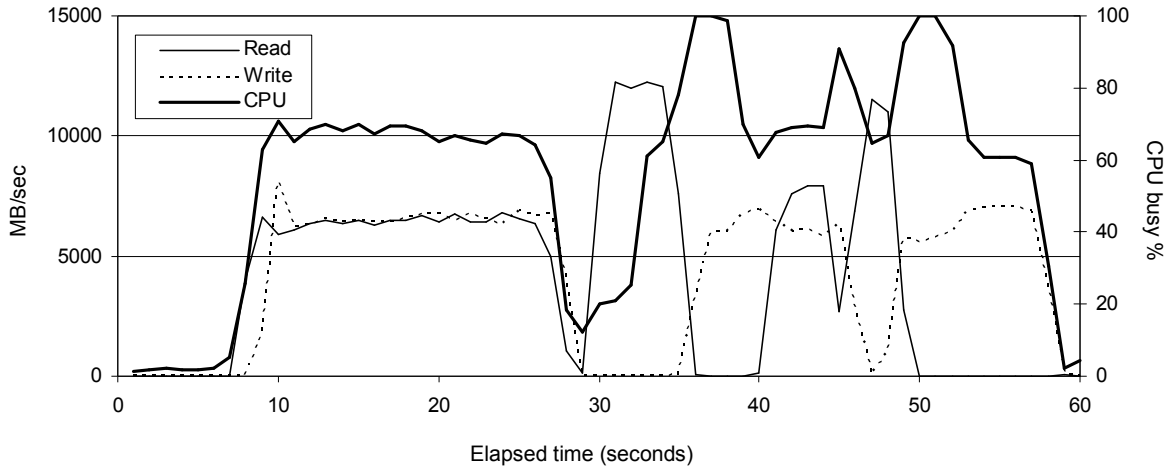

Figure 7. Minute Sort aggregate I/O throughput and cluster CPU usage.

Minute Sort run. Since there are only 160 slices, SCS has enough buffers available to smooth out processing during its first pass. Unlike the TeraByte Sort run, during Minute Sort the reader and distributor threads can continue even while the writer threads write a batch of buffers to the temporary files. Thus, the throughput graphs during pass 1 are very smooth, with none of the periodic oscillations observed during the TeraByte Sort run. The write throughput during pass 1 just looks like a time-shifted copy of the read throughput. However, since there are only 4 slices processed by each node during the second pass, there is significant "lumpiness" in that part of the throughput graph. Performance of pass 2 would have been somewhat better with more and smaller slices, because the unoverlapped writing of the last output slice would have been shorter.

Throughout most of pass 1 of Minute Sort, the sort cluster drove the I/O subsystem to its limiting throughput of 14GB/sec. In pass 2, there were intervals where CPU was the bottleneck, and other intervals where I/O throughput was the limiting resource.

Discussion

As always seems to be the case with large sort benchmarks, the hardware was available for too brief a time to adequately tune the system. In hindsight, making more of the 4GiB of RAM on each node available to SCS undoubtedly would have smoothed out the performance of pass 1 of the TeraByte Sort, and probably would have sped it up as well. The Minute Sort run did have adequate buffering in pass 1, and Figure 7 shows its smooth I/O throughput. Excluding startup latency, pass 1 accounted for 50% of the running time of the TeraByte Sort, but only 42% of the running time of the Minute Sort, suggesting how much the running time of TeraByte Sort might have improved with more buffering.

The startup latency of 6 seconds for 40 nodes accounts for 10% of the time budget of Minute Sort. This is a clear opportunity for optimization. Others have built highly-tuned remote execution services that can start programs in mere fractions of a second [12]. Minute Sort could certainly benefit from the techniques that were vital to sub-second Datamation Sort implementations.

Pass 2 of SCS is frequently CPU-bound. This shows up clearly in Figure 7, and is visible in the detailed per-node CPU measurements for the TeraByte Sort, although not in the aggregated CPU usage graph of Figure 5 due to the interference effects described previously. Part of the reason for this is the use of relatively slow (1.3GHz) processors, but program structure is also partially to blame. SCS uses standard X/Open read and write system calls to interface to the file system; in GPFS these by default copy data between the buffer in the program's address space and I/O buffers in kernel memory. The use of direct I/O would have eliminated these data copies and freed up considerable CPU resources. Direct I/O complicates the structure of the SCS program due to the requirement to handle records whose size does not divide evenly into the file system block size and the need to explicitly manage block prefetching and write behind.

SCS set new records in two sorting categories: Indy Minute Sort and Indy TeraByte Sort. The time for TeraByte Sort was a factor of 2.4 times better than the previous record, set six years earlier [16]. This is a compound growth rate of only 16% per year, much lower than other growth rates typically associated with computers. This reflects more on the difficulty of obtaining a large system for running sort benchmarks than on any fundamental factor. Also, the new TeraByte Sort record used less than one tenth the number of nodes of the former record. The SCS Minute Sort record is a factor of 3.7 better than the prior record, set just last year [11]. Given adequate interconnection bandwidth and appropriate software, clusters are probably easier to scale than large SMP machines.

Aside from increasing the node count, the most straightforward way to extend the sort records described here would be to use a one-pass rather than a two-pass sort algorithm. This requires enough cluster memory to buffer the entire input file, plus enough interconnection bandwidth between nodes so they can transfer slice buffers directly to other nodes. Existing technologies such as Infiniband or 10-gigabit Ethernet are powerful enough to accomplish this; all that is required are the resources to build the right cluster and the will to run sort benchmarks on it.

References

[1] Anon., Et-Al. (1985). "A Measure of Transaction Processing Power." *Datamation*. V.**31**(7): pp. 112-118. Also in *Readings in Database Systems,* M.J. Stonebraker ed., Morgan Kaufmann, San Mateo, 1989.

[2] Agarwal, R.C. "A Super Scalar Sort Algorithm for RISC Processors." ACM *SIGMOD '96*, pp. 240-246, June 1996.

[3] Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H., Culler, D.E., Hellerstein, J.M., and Patterson, D.A. "High-Performance Sorting on Networks of Workstations." ACM *SIGMOD '97*, Tucson, Arizona, May, 1997. Available at http://now.cs.berkeley.edu/NowSort/nowSort.ps.

[4] Gropp, W., Lusk, E., Doss, N., and Skjellum, A. "A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard," *Parallel Computing* V.**22**(6): pp.789-828, Sept. 1996.

[5] http://research.microsoft.com/barc/SortBenchmark/ Sort benchmark home page, maintained by Jim Gray.

[6] http://www-1.ibm.com/servers/storage/disk/ds4000/ds4300/ Description of IBM TotalStorage DS4300 storage subsystem.

[7] http://www.intel.com/design/servers/buildingblocks/SR870BN4/ Description of dual Itanium2 nodes used by SCS.

[8] http://www.ordinal.com/ Home page of Ordinal Corp., whose NSORT program holds the records for the Daytona versions of Minute Sort and Terabyte Sort.

[9] http://www.sandia.gov/LabNews/LN11-20-98/sort_story.htm "Sandia and Compaq Computer Corp. team together to set world record in large database sorting." Description of Terabyte Sort at Sandia Lab on 11/20/98 in "under 50 minutes."

[10] Nyberg, C., Barclay, T., Cvetanovic, Z., Gray, J., and Lomet, D. "AlphaSort: A Cache-Sensitive Parallel External Sort." VLDB Journal **4**(4), pp. 603-627 (1995). Available at http://research.microsoft.com/barc/SortBenchmark/AlphaSort.rtf.

[11] Nyberg, C., Gray, J., Koester, C. "A Minute with Nsort on a 32P NEC Windows Itanium2 Server." Available at http://research.microsoft.com/barc/SortBenchmark/2004_Nsort_Minutesort.pdf.]

[12] Popovici, F., Bent, J., Forney, B., Arpaci-Dusseau, A., and Arpaci-Dusseau, R. "Datamation 2001: A Sorting Odyssey." Technical Report CS-TR-2002-1444, University of Wisconsin, August 2002.

[13] Schmuck, F., and Haskin, R. "GPFS: A Shared-Disk File System for Large Computing Clusters." Proc of the First Conference on File and Storage Technologies (FAST), March 2002.

[14] Snir, M., Otto, S., Huss-Lederman, S., Walker, D., and Dongarra, J. *MPI: The Complete Reference*, The MIT Press, 1995.

[15] Tsukerman, A., "FastSort– An External Sort Using Parallel Processing," *Tandem Systems Review*, **3**(4), Dec. 1986, pp. 57-72.

[16] Wyllie, J. "SPsort: How to Sort a Terabyte Quickly." Available at http://research.microsoft.com/barc/SortBenchmark/spsort.pdf