MILLENNIUM SORT: A CLUSTER-BASED APPLICATION FOR WINDOWS NT USING DCOM, RIVER PRIMITIVES, AND THE VIRTUAL INTERFACE ARCHITECTURE

Philip Buonadonna, Joshua Coates, Spencer Low, and David E. Culler

Millennium Sort:
A Cluster-Based Application for Windows NT using DCOM, River Primitives and the Virtual Interface Architecture

Philip Buonadonna, Joshua Coates, Spencer Low, David E. Culler
Computer Science Division
University of California, Berkeley
{philipb, jcoates, culler}@cs.berkeley.edu, lowtek@millennium.berkeley.edu

Abstract

We present the design and results of Millennium Sort, a distributed sorting application built using three layers of technology: extensible River System primitives, the Virtual Interface Architecture (VIA) and the Distributed Component Object Model (DCOM). The Millennium Sort application is a vehicle for exploring the issues of commercial cluster technologies and distributed development on commodity node clusters. We discuss the architecture and design of the River System primitives, VIA and DCOM. Performance results are discussed, including the latest Datamation Sort record time of 1.18 seconds achieved by a 16-node Pentium-II cluster.

1. Introduction

One of the most dynamic properties of computing clusters is the set of new technologies and methods for how to build, use and evaluate them. The relentless pursuit to "build a better cluster" offers a large design space of interesting concepts and challenging problems to tackle. A direct method to investigate such issues is to build a cluster-based application and assess the tools and technologies used behind it.

The principal goal of the Millennium Sort project is to study new commercial technologies for utilizing cluster resources and the use of commodity development tools in the context of a database oriented application (sorting). The technologies we employ include the Distributed Component Object Model (DCOM), the River System, and the Virtual Interface Architecture (VIA)[12]. Specifically, we seek to: 1) Explore the feasibility of using DCOM as a parallel remote execution system, 2) Demonstrate the extensibility of River System primitives to data management applications, 3) Investigate the use of the VI Architecture in distributed I/O systems, and 4) Evaluate the use of commodity hardware and software programming tools for distributed application development on PC clusters running Windows NT. In short, we explore the viability of building a high performance distributed system based on emerging commercial technology and available tools. This allows us to focus on component composition rather than just component design. We demonstrate an implementation of a sorting application that breaks the world record for the Datamation sorting benchmark [3] that was previously set by NOWSort [8] on the Berkeley Network of Workstations (NOW) [1].

The remainder of this report is divided into four sections, three of which detail the technical aspects of the overall system; the fourth offers a reflection on the experience as a whole. Section 2 provides background on the Windows NT cluster used for Millennium Sort and outlines the distributed sorting problem. In section 3 we present the individual component technologies of Millennium Sort (DCOM, River, and VIA) and the integration of these technologies to form the application. Section 4 details the performance of the sort in terms of the component technologies. In section 5 we discuss the project in retrospect and comment on our experience with developing a distributed application for an NT cluster. We conclude in section 6 with proposals for future work.

The 16x2 x86 Processor PC Cluster

- Dual 400Mhz Pentium II
- 256 MB SDRAM / 100Mhz Memory Bus
- 33Mhz 32-Bit PCI Bus / Ultra2 LVD SCSI
- 2x9.1GB Disk Storage
- Switched Fast Ethernet / Myrinet M2F
- Windows NT 4.0 Terminal Server Edition

Figure 1: The cluster configuration used for Millennium Sort.

2. Background

The hardware used to run the sort is a homogenous 16-node PC cluster (Figure 1) priced at ~$5800/node, including the Myrinet switch. By comparison, NOWSort used a 32-node Sun Ultrasparc workstation cluster at a cost of approximately $18,000 per-node [8]. The cluster has two principal interconnects: the Myrinet M2F System Area Network

(SAN) [15] and fast Ethernet connected through a Nortel Networks Accelar 1200 series switch. The Myrinet SAN supports the VI Architecture implementation used for the sorting application. It consists of a programmable network interface that allows emulation of VI capable hardware in a flexible system that can be instrumented. The Ethernet was used to evaluate a Winsock based version of the sort as well as provide communication services not available in the VI Architecture implementation. Since bandwidth is a primary concern with data-intensive distributed applications, we assessed the maximum and sustained throughput of various cluster node components (see Table 1). For the VI based sort, the disk is the obvious bottleneck in the system, which prompted us to stripe the two disks into a single ~23MB/sec volume on each node. For the Winsock based sort, the Ethernet becomes the limiting component.

The distributed sorting problem provides an aggressive space to examine and evaluate clusters. The performance of the sorting application depends on the I/O and network as well as the computational limits of the CPU. While our goal was not to conduct distributed sorting research *per se*, we use the sorting task to drive a study of the different technologies implemented around a well-known algorithm, and the cluster development tools used therein.

The Millennium Sort implements a one-pass, disk-to-disk distributed sort (Figure 2). At the start, unsorted data (keys & records) reside on the disk of each of the computing nodes. Upon invocation, each of the nodes simultaneously reads the data from the disk, partitions it according to a predetermined rule, and sends the data to the other computing nodes. The partitioning rule is typically a range of keys to a particular node. At the same time, each node receives data from the other nodes. Once all data has been read and distributed, each node sorts the data and writes the output back to disk. The total elapsed sort time is measured from the beginning of invocation (i.e. includes time to startup the application on each node) until all the data is written back to disk and the application exits. The sort is one-pass: the data set fits into available physical memory on each node of the cluster. To compare our implementation against known results, we use the Datamation sorting benchmark specification. This test measures the performance of a disk-to-disk sort of 1 million 100-byte records each with a random 10-byte key. Past results of this benchmark are available at [19]. The most recent performance record for the Datamation sort was 2.41 seconds on 32 nodes of the U.C. Berkeley NOW [8]. We demonstrate that Millennium Sort achieves twice this performance on the 16-node cluster described.

	MEM Bus	PCI Bus	SCSI Bus	Disk	Network (Ethernet)	Network (Myrinet)
MB/sec Peak	800	133	80	21	12	150
MB/sec Sustained	640*	105*	64	14/23	10	120

Table 1: Bandwidth of cluster node components. Note that the effective disk bandwidth is actually ~23 MB/sec after striping across two disks. (* 80% estimate of peak)

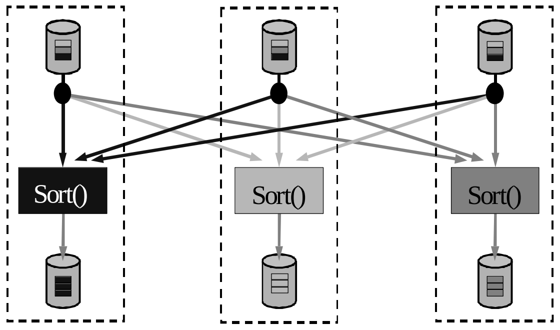

Figure 2: The generalized one-pass distributed sort algorithm.

3. Architecture and Design of Millennium Sort

The Millennium Sort application is developed around three principal technologies: Microsoft DCOM, the River System, and the Virtual Interface Architecture. In this section we describe each of these technologies and how they integrated into the sort program.

3.1. Catapult and DCOM

In order to facilitate the development and use of cluster-based applications, it is important to have some method of remote execution and job control [5]. In line with our goals of surveying industry-developed technology, we chose to use the distributed version of the Microsoft Component Object Model (COM)[4]. COM is a software development specification and a set of binary standards that allow interaction and communication amongst heterogeneous software objects. Distributed COM (DCOM) is the extension of

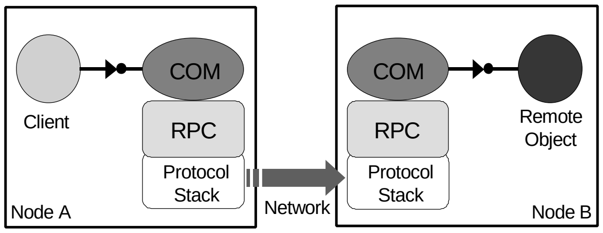

Figure 3: The DCOM Architecture.

the COM paradigm to take advantage of distributed network resources using a single programming interface (Figure 3). While DCOM is not specific to the Windows family of operating systems, Windows NT 4.0 has built in DCOM to support essential system services. For this project, we developed a DCOM based, distributed execution tool that supports invocation of generic command line programs: Catapult.

Catapult is composed of a DCOM object (a COM server executable) and a command line executable. The DCOM object is an agent, which resides on each node of the cluster and acts to service remote execution requests in a manner analogous to the UNIX *rshd*. The Catapult executable acts as the primary tool for invoking and controlling the remote programs. Prior to using Catapult, an administrative install of the DCOM agent on each node must be performed by copying the Catapult agent binary to each local disk and installing the DCOM object ClassID (a 128-bit Globally Unique Identifier) into the registry. The Catapult executable contains a command line option to do this remotely by using NT's default administrative network shares and the Remote Registry API.

To launch a distributed task, the Catapult command is issued on a user's local workstation, passing in the name of the application image and a list of nodes as arguments (e.g. `>catapult -e "hostname.exe" mm1 mm2 mm3 mm4`). For each node, the local Catapult executable spawns a new thread and requests an instance of the remote DCOM object be created. The DCOM instantiation request is received by the always-available NT RPC service running on the remote node. The RPC service references the DCOM object's ClassID in the NT Registry to find the path to the local DCOM object binary. The binary is executed (as the local user's identity) and a DCOM object is instantiated. Note that the DCOM agent binary is not actually running prior to its instantiation. The instance of the Catapult DCOM agent invokes the executable passed in at the command line. Afterwards, the agent executes a series of data methods that redirect the processes' stdout, stdin and stderr back to that node's associated thread on the user's workstation. This allows a certain amount, albeit cluttered, interactivity that is useful with certain types of remote jobs, as well as debugging distributed applications.

By using DCOM to implement Catapult, we are able to run remote executables on standard NT installations with minimal administrative overhead. Instead of requiring a separate dedicated daemon or dedicated resources, we use the NT RPC service as an *inetd*, only creating our process when called upon. Since DCOM is basically object RPC, we are able to avoid writing complex socket and stream parsing code, forgoing it for simple functions and a small amount of IDL (Interface Definition Language).

Since Catapult supports generic command line executables, the Catapult "remote execution platform" had useful applications right from its inception. An early application was to automate the already existing command line process of reloading the NIC firmware on the NT cluster. We did not require special Catapult-enhanced applications (e.g. new DCOM based applications), thus eliminating another variable in debugging sessions. A Catapult execution may be simulated through direct invocation from a system console or remote NT session, removing Catapult from the loop and allowing the use of interactive debuggers.

3.2. The River System

The River System is a distributed dataflow programming model which allows an application to *Put* and *Get* records to and from a distributed data queue. The distributed queue partitions data based on application specific rules. The River programming metaphor treats data as a fluid that flows from Sources to Sinks, which accumulate in the distributed queue of the River (Figure 4). This model is stream-based and allows direct integration with certain types of I/O intensive, distributed applications.

Distributed dataflow models that extend the stream-based metaphor of 'Data Rivers' are not a new

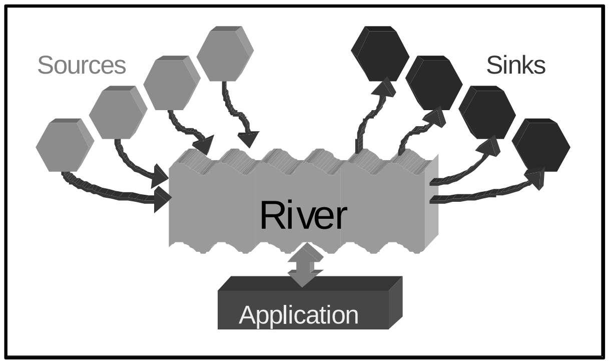

Figure 4: Basic model of the River paradigm. An application *Gets* and *Puts* data to and from a distributed queue (the River) which manages distributed source and sink data streams.

concept [2,7]. Typical distributed dataflow systems read records off of disk, partition them, either statically or dynamically across a cluster, and send the data through a specific network interface. We believe that the type, as well as the transformation applied to the data, should be orthogonal to the dataflow subsystem. This extensibility is realized through River System primitives. These primitives can be composed to describe application specific dataflows.

To understand the construction of an application using River System primitives, it is necessary to define the basic primitive types:

Source – An object that produces and possibly transforms data. Typically associated with a particular sink that the source deposits its data into.

Sink – An object that consumes and possibly transforms data. Typically receives data from a Source object or application via the SinkRecord() or SinkBuffer() methods.

Buffer – A memory object that contains a pointer to heap allocated memory.

MemPool – A data structure that manages Buffer objects, based on a simple queue structure.

These four primitives can be extended to meet specific I/O or application requirements without sacrificing the advantages of the dataflow model. For Millennium Sort these extensions included:

Disk Source & Disk Sink – Sources and sinks tailored for asynchronous disk I/O.

Net Source & Net Sink - Sources and sinks that managed network data transfers. These were implemented for both VI based and Socket based communication.

DiskMemPool – This type of MemPool creates and manages buffers that contain memory allocated in multiples of the disk sector size.

ViaMemPool - A special MemPool that included memory registration management for the VI based sort.

PartitionSink – A sink used to partition data in order to send specific types of data to specific types of sinks. For Millenium Sort, the data was partitioned lexicographically by the key in each record.

RiverSink – A sink that acts as the River System's interface to the application. An application would *Get* or *Put* records to the RiverSink. The RiverSink acts as a central data repository for the other sinks and sources in the system.

Figure 5 illustrates the overall interaction of the extended sources and sinks in Millennium Sort. The arrows indicate the logical path of data in the system. Upon activation, the system begins to first "pump" data from the Disk Sources into the River. As the River receives the data, it flows through the partitioner, which is a component of the River (1). The partitioner sends the data to its destination based on a function of the record key. Records destined for the local node get added to the queue within the River. Records that are destined for a remote node get sent to the appropriate Net Sink (2).

As this process continues, the River queue begins to fill with records from the local Disk Source, as well as incoming Net Sources (3). As the queue fills, the *Get* requests of the Sort Core empty the queue. This process continues until the Disk and Net Sources run dry, and the River queue is empty. Records are sorted and dumped back into the River with a *Put* request, which is then sunk to the Disk Sinks (4). The application waits for the Disk Sink to complete the final disk write, and then exits.

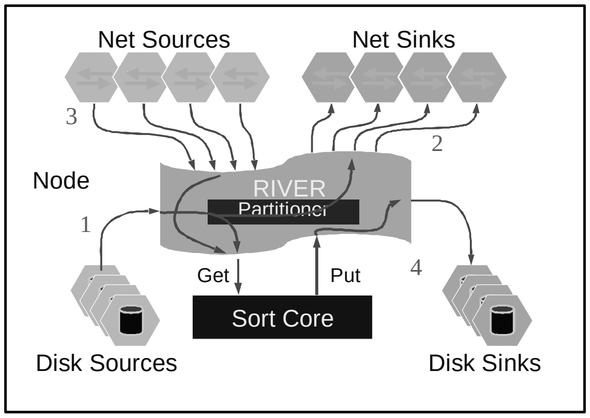

Figure 5: Illustration of dataflow in Millennium Sort.

3.3. VI Architecture

The Virtual Interface (VI) Architecture [12] is a high performance communications infrastructure that supports memory-to-memory network transfers between user processes across a cluster. The architecture is intended as a standard for user-level networking where applications have direct access to network hardware. Network resources are virtualized across user programs at the network interface level and

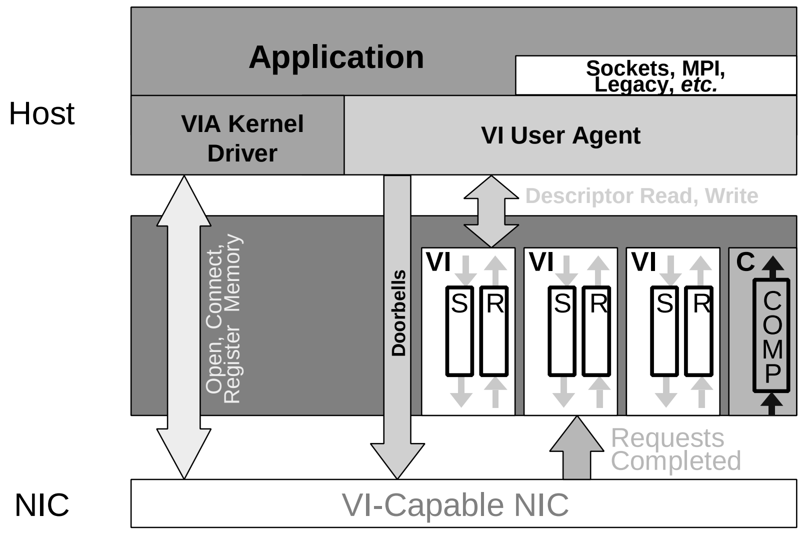

Figure 6: The Virtual Interface Architecture and its components.

OS intervention is eliminated from the critical communications path. The system has four principal components: VI Providers, VI Consumers, Virtual Interfaces, and Completion Queues (Figure 6). The design begins with the VI Provider, which includes a VI capable NIC and an OS Kernel Agent. The principal requirement for a VIA NIC is that it has resources that can be memory mapped directly into a user process address space (the doorbells). The Kernel Agent, essentially a superset of a device driver, performs the command and control functions that require operating system intervention such as device commands, memory registration and connection management. The VI Consumer component consists of the user application(s) and the User Agent, or Virtual Interface Provider Library (VIPL). The User Agent provides the API and necessary user-level support for the VI Provider implementation.

The Virtual Interface itself is the primary abstraction for the users protected, direct channel to the network hardware. Each VI consists of a pair of work queues, send and receive, their associated doorbell resources and the users registered memory regions. The work queues are a FIFO list of descriptors that mark a region of registered memory to transfer data to or from. Network data transfers are initiated by posting a descriptor in the appropriate work queue and writing a token in the queue's associated doorbell (i.e. "ring" the doorbell). The architecture supports both matched send-receive and Remote DMA (RDMA) communication semantics with either unreliable or reliable service models. When the network interface completes an operation, it sets a status mark in the descriptor that can be detected by user polling or through an event.

VI's are connection-oriented and support personalized communication with a single remote VI. This one-to-one connection model may require a large number of VIs per user to achieve full connectivity between nodes in a cluster. To provide scalable performance, Completion Queues maybe used to provide a single monitoring point for network data completions. VIs are assigned to completion queues at the granularity of the individual work queue. A completion of a descriptor places a token in the completion queue, which may be detected by either polling or through an event.

3.3.1. The Berkeley VIA Implementation

To better understand the design internals of the VI Architecture and their impact on application and cluster performance, we use our own implementation instead of an off-the-shelf product. The VIA implementation used in Millennium Sort is an enhancement of the Berkeley VIA implementation [14] that adds memory registration and increased VI/user support. The end goal of these additions is to provide a VI Architecture implementation that may be readily used by a greater variety of applications and that allows a deeper investigation into the transport itself.

The first functional component added was memory registration and virtual address translation. Prior to conducting VI communication, a user process must identify memory segments that will be used for data transfer. Memory registration locks the pages of a virtually contiguous memory region into physical memory, builds the necessary data objects to enable virtual-to-physical translations on the VI NIC, and assigns a name (memory handle) to the registered region. Our challenge was to build a registration/translation mechanism that scaled well with

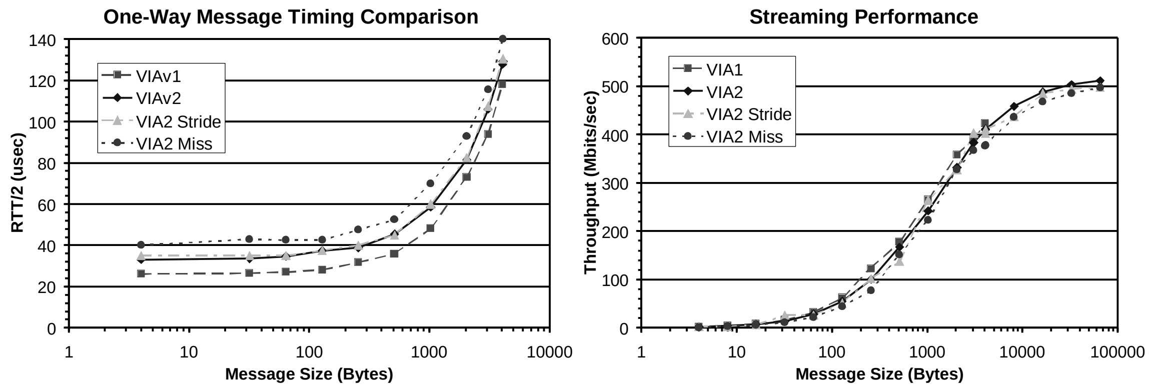

Figure 7 : One-way message timing and streaming performance for VIAv2 in comparison with the previous implementation. 'Stride' refers to the case where the test traverses a data buffer larger than the span of the TLB. 'Miss' refers to a 100% TLB miss rate in all cases.

limited resources and exhibited satisfactory performance. The result was a super-sized (1024 entry) translation lookaside buffer (TLB) on the NIC. At memory registration, the kernel agent pins the memory region into physical RAM, builds a list of the corresponding page frame numbers and passes the physical address of the beginning of the list to the network interface. The NIC stores a collection of these page list pointers in a directory maintained for each VI user process. All subsequent communication operations between the user process and NIC are conducted using user virtual addresses. Data transfers are broken into page-size segments and a TLB lookup performed for each segment. TLB misses are processed using the page directory pointer for that registered region. The benefit of this system is that memory registration scales with available host resources instead of limited NIC resources.

Expanded VI/user support was the next important functional addition to the VIA implementation. In the prototype, a memory allocation equivalent to the host page size is used to hold a single pair of doorbell registers. This mechanism was overhauled to exploit the unused space in the rest of the page. The new implementation provides 256, 128-bit doorbell pairs per page sized unit. Each doorbell page is mapped to the requesting users address space upon creation of the first VI. Since the host page size is the minimum granularity of protection for the memory system, it is possible for interference to occur between an individual user’s VIs. However, any damage will be limited to that user and should be preventable if doorbell access is done through the provided User Agent.

For Millennium Sort, the VI Architecture implementation was only used to support data transfer during the sort. Other inter-node communication (i.e. DCOM calls and barriers) utilized protocols over Fast Ethernet. Other work has been done to implement DCOM over VIA which has shown to improve performance in remote method invocation [13].

4. Performance

The overall performance of Millennium Sort depends on the performance of each component. The application is built upon VIA, extended River System primitives and a parallel remote execution application based on DCOM.

4.1. VIA Performance

Assessing VIAv2 performance is important as a precursor to application performance when layered over the architecture. We found that VIAv2 functioned with only slightly degraded performance relative to the previous implementation (VIAv1) in acute measurements with no noticeable performance loss in the sort application. The performance of VIAv2 is compared against the VIAv1 prototype using one-way message timing and bandwidth benchmarks. The results are presented in Figure 7.

The one way message time is a measure of the average time it takes for a message of a given size to be transmitted from a source node and received by a remote node. It is measured by doing a series of ping-pong tests in which a message of arbitrary size is sent to a remote node that then reflects that same message back to the originator. The resulting round-trip time (RTT) is divided by two to yield the one way time. We performed this test under three principal conditions. The first was where the TLB misses only on the first use of the VI and, thereafter, exhibits a 100% hit rate. The second condition (Stride) involved a host data buffer which is larger than the address space spanned by the TLB. The benchmark traverses this buffer when making network data transfers thus, depending on the message size, forces various miss rates in the TLB. This

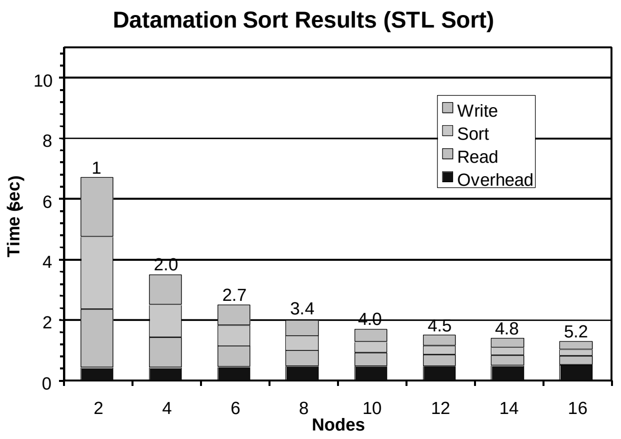

Figure 8: VI based sort performance. The numbers on top of each bar represent relative speedup.

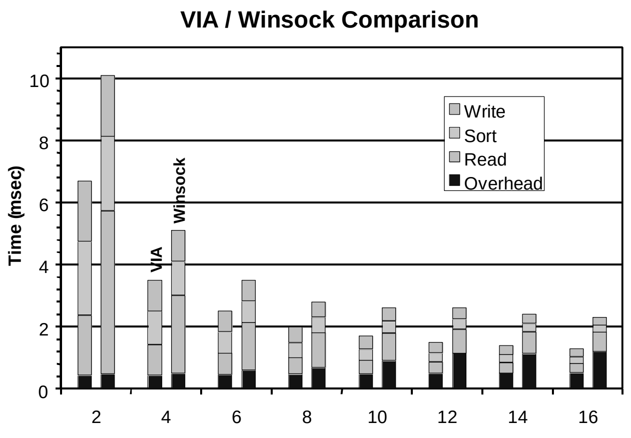

Figure 9: Side-by-side comparison of VI and WinSock based sort performance.

situation is designed to closely approximate the usage pattern of a user process transferring data to/from a large block of registered memory. The last condition (Miss) forced a 100% miss rate in the TLB. It is interesting to note that the performance difference between the large buffer and the near 100% TLB cases is minimal. This suggests that the TLB mechanism extends well to several types of network memory access patterns.

The second metric evaluated for the new implementation was streaming performance. This benchmark measures the net throughput capable when messages are sent successively from a source VI to a sink with no pause in between. We performed this test for the same three conditions as in the one-way timings. For message sizes above approximately 20KB, the throughput achieves the maximum possible value for the network interface, roughly 64MB/sec. (NOTE: while the Myrinet physical layer can achieve 150 MB/sec, the maximum rate is half this value since the VI firmware copies data to its onboard buffer before transmission). Again different TLB performance cases do not significantly impact the bandwidth of the interface.

4.2. River System Performance

The resulting performance of the River System extensions used for Millennium Sort exceeded our expectations for both the VI and Winsock based systems. Figure 8 presents the sort results for the VI Architecture based sort using the Standard Template Library qsort routine. The sort times are broken down into the major stages of the sort application: Overhead, Read and distribute, Sort and Write. The Read sequence consists of reading the data from a Disk Source, partitioning it in the River, and distributing it to the appropriate nodes through Net Sinks while simultaneously receiving data from Net Sources. The Write sequence is simply the time it takes to write data to disk via the Disk Sink. The overhead sequence is derived from the difference of total execution time of the application, from launch to completion, and time spent actually reading, sorting and writing. The overhead operations consist of: application launch via Catapult, blocking on a Winsock based barrier, and data structure/memory initialization.

On 16 nodes, the sort completed the Datamation benchmark in 1.3 seconds and achieved its best time of 1.18 seconds using a hand coded radix sort in place of the STL qsort. This record-breaking time surpassed the previous Datamation record of 2.41 seconds [8].

Figure 9 provides a side-by-side comparison of the VI based sort with the WinSock/Ethernet based sort. The overhead component is consistently higher with the WinSock implementation, increasing from 480ms to 1.2sec. This increase in overhead results from the complex mechanisms required to establish the necessary socket connections. Surprisingly, the Winsock Millennium Sort also broke the previous record with an elapsed time of 2.21 seconds (again using a radix sort core).

4.3. DCOM Performance

In Catapult, DCOM calls to a node are made from within a separate thread created for that node, both in the client and server code. In effect, all DCOM calls between nodes are asynchronous within the application as a whole. Of the overhead, approximately half is incurred by Catapult as the application scales to 16 nodes. Initial tests of a ‘null’ Catapult executions (startup and teardown of a ‘null’ process) showed times of ~180 msecs on a single node, which scaled to ~220 msecs on 16 nodes. The NT Resource Kit contains a DCOM benchmarking program, which shows that a single ‘null’ remote method invocation on our cluster is approximately 400 μsecs. We believe that with further optimizations it is possible to achieve even faster startup times for Catapult processes.

5. Retrospectives

The improvements in the performance of Millennium Sort with respect to previous results are attributable to three principal causes. The first of these lies with the raw hardware resources available. The computing nodes of the cluster are dual processor machines with a CPU clock cycle of 400 MHz as compared to the 167 MHz, uni-processor workstations used to establish the previous Datamation sort. This hardware advantage minimizes time spent in the core of the sort and boosts performance of OS related communications. The second source of improvement relates to the use of the Catapult/DCOM combination as the distributed execution system. Previous distributed sorting systems spent over half the total sort time on remote invocation alone [8]. From our measurements, the DCOM system incurred startup overhead less than 1/5th the total sort time. This improvement in remote execution alone accounts for the majority of the overall performance gain. The last source of improvement results from the use of the VI Architecture based networking. The implementation supports large message sizes (up to 100KB) with zero-copy. Additionally, the low overhead of the VI based sort contributed to better performance scaling as the number of nodes increased.

Aside from the performance results of Millennium Sort, the insight provided into the different technologies and commodity tools is extensive.

Tools. The tools used in the development and evaluation of Millennium Sort included Visual Studio 6.0 and some utilities of the NT Resource Kit. The Visual Studio application provided a clean, straightforward environment for coding and compiling, but lacked in the ability to do distributed debugging. It was not possible to connect to arbitrary remote instances of the DCOM agent or sort executable for debugging purposes. Instead, we were forced to manually debug on a collection of nodes using Terminal Server remote sessions. By contrast, the resource kit utilities were well adapted to cluster-oriented use. Most notable among these was PerfMon, which allows an administrator to view the behavior of objects such as processors, memory, cache, threads and processes. Each of these objects has an associated set of counters that provide information about device usage, queue lengths, delays, and throughput. PerfMon is capable of monitoring a collection of nodes simultaneously from a single workstation. This tool proved to be invaluable for performance analysis and debugging of the system. It is easy to use, and requires no special installation or explicit collaboration from remote nodes.

Operating System. The Windows NT 4.0 Terminal Server Edition (TSE) operating system offered an excellent environment for clustering. TSE is a multi-user version of Windows NT that provides access to a machine through remote windows sessions. With minor exceptions, this allowed us to use and administer the cluster nodes without a continuous local console. The administrative tools included with the TSE are well suited for cluster resource management. The TSE Administration program provides the means to monitor and administer node status and running processes at a cluster-wide level, allowing us to kill deadlocked jobs during debugging. TSE also includes a command line 'kill' utility that can be used in scripts to terminate a series of jobs on the cluster.

DCOM. While Catapult performed well compared to previous cluster execution systems, our overall evaluation is somewhat mixed. The DCOM based environment is robust and possesses built in mechanisms to recover from individual program crashes or global terminations (i.e. Ctrl-C). However, to maintain portability, DCOM defines a wide range of security parameters that are set on a per-machine basis. It is difficult to determine an exact setting of these parameters that would allow a wide variety of distributed applications to run without compromising security. Perhaps the most significant drawback of DCOM over Windows NT RPC is the lack of a full interactive logon to the remote machine. Credentials passed through the NT RPC service permit only a fast network or "null" logon that does not notify the Windows Networking redirector for access to remote file system volumes. There does not appear to be an interactive logon toggle available in the DCOM API. Complete Interactive logons would require the DCOM agent to be designed as an NT service running under the privileged LocalSystem account and for the user's username and password to be separately transported to the DCOM agent. Lack of redirector access requires applications invoked by catapult to be separately installed on the local disk of each node. This complicates the development process in which frequent revisions to the application occur.

VIA. The memory registration system of the VI Architecture required complicated memory management schemes within the user application. Windows NT does not allow large sections of memory to be registered in a single operation due to limited amounts of physically contiguous memory available for address translation structures. Additionally, registering more than 80% of available physical memory yields undesirable and sometime pathologic (i.e. system crashes) operating system response. In Millennium Sort, we designed a simple windowing system that registered smaller amounts ('windows') of memory for receiving data from the network. Our results suggest that this method of registration improves application stability when registering large amounts of address space.

River. We find the most valuable aspect of the River system is the ease of extending River primitives. For instance, in order for Millennium Sort to use the VI Architecture, special operations have to be performed periodically on the memory used for buffering data. By extending the MemPool object, a VIAMemPool object is created which handles memory registration windows. This would have been difficult or impossible to integrate with a non-extensible river system without rewriting large portions of the system. In order to build the Winsock version of the application we only needed to modify the Net Source/Sink classes. These modifications required just a few hours time.

6. Future Work

The obvious next step in the Millennium Sort work is to implement a two-pass sort and continue refining our understanding of the technologies that it is composed from. A two-pass sort will allow us to run sustained sorts that will further stress test the system. Our DCOM remote execution system, Catapult, requires further performance optimizations. The extensibility of the River System through primitives worked well, but it needs to be packaged into a library, perhaps with useful primitive extensions. Lastly, although the addition of virtual memory translation to VIAv2 works and performs well, VIAv2 requires a reexamination of how it extends the memory interface to the programmer.

7. Acknowledgements

Support for this project was provided by the Microsoft Corporation, especially Jim Gray and Joe Barrera of Microsoft Research, and by Intel's Technology 2000 grant program. As well, support was provided by the National Science Foundation SimMillennium Grant (EIA-9802069), the National Science Foundation Infrastructure Grant (CDA 94-01156) and the Defense Advanced Research Projects Administration Grant (F30602-95-C-0014).

References

[1] T. E. Anderson, D. E. Culler, D. A. Patterson. "A case for NOW (Networks of Workstations)." IEEE Micro, vol. 15, (no. 1), February 1995, p. 54-64.

[2] R. H. Arpaci-Dusseau, E. A. Anderson, N. Treuhaft, D. E. Culler, J. M. Hellerstein, D. A. Patterson, K. Yelick, "Cluster I/O with River: Making the Fast Case Common." to appear in *IOPADS '99*, Atlanta, Georgia, May 1999

[3] Anon. Et al. "A Measure of Transaction Processing Power." Datamation, 31(7):112-118, 1985

[4] G. Eddon, H. Eddon. "Inside Distributed COM", *Microsoft Press*, Remond, WA 1998

[5] D. P. Ghormley, D. Petrou, S. H. Rodrigues, A. M. Vahdat, T. E. Anderson. "GLUnix: A Global Layer Unix for a Network of Workstations", *Software Practice and Experience*, vol.28, (no.9), Wiley, 25 July 1998. p.929-61.

[6] E. Riedel, C. van Ingen, J. Gray, "Sequential I/O on Windows NT 4.0 – Achieving Top Performance", *Proceedings of the 2nd USENIX Windows NT Symposium*, 3-5 August 1998, Seattle, WA, pp. 1-10.

[7] T. Barclay, R. Barnes, J. Gray, P. Sundaresan. "Loading Databases Using Dataflow Parallelsim", *SIGMOD RECORD*, Vol 23, (no. 4), December 1994

[8] A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, D. E. Culler, J. M. Hellerstein, D. A. Patterson. "High-Performance Sorting on Networks of Workstations." *SIGMOD '97*, Tucson, Arizona, May 1997

[9] A. C. Dusseau, K. E. Schauser, R. P. Martin, "Fast Parallel Sorting Under LogP: Experience with the CM-5." *IEEE Transaction on Parallel and Distributed Systems*, Vol. 7, (no. 8), August 1996

[10] J. Gray, J. Coates, C. Nyberg. "Performance / Price Sort", http://www.research.microsoft.com/barc July 1998.

[11] The Millennium Project: A Campus-wide cluster of clusters. University of California, Berkeley, Berkeley, CA http://www.millennium.berkeley.edu

[12]"Virtual Interface Architecture Specification. Version 1.0", *Compaq, Intel and Microsoft Corporations*, Dec 16, 1997, available at http://www.viarch.org

[13] R.S. Madukkarumukumana, C. Pu, H.V. Shah, "Harnessing User-Level Networking Architectures for Distributed Object Computing over High-Speed Networks", *Proc. of the 2nd USENIX Windows NT Symposium*, Seattle, WA, August 3-5, 1998, pp. 127-135.

[14] P. Buonadonna, A. Geweke, D. E. Culler. "An Implementation and Analysis of the Virtual Interface Architecture", *Proc. of Supercomputing '98*, Orlando, FL, 7-13 November 1998.

[15] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and Wen-King Su, "Myrinet: A Gigabit-per-Second Local Area Network." *IEEE Micro*, vol. 15, (no. 1), Feb 1995, pp. 29-36

[16] A. Basu, M. Welsh, T. von Eicken. "Incorporating Memory Management in User-Level Network Interfaces", *Hot InterconnectsV*, Stanford, CA, August 1997.

[17] C. Dubnicki, A. Bilas, Y. Chen, S. Damianakis, K. Li. "VMMC-2: Efficient Support for Reliable, Connection-Oriented Communication." *Hot Interconnects V*, Stanford, CA, August 1997

[18] D. Dunning et al., "The Virtual Interface Architecture", *IEEE Micro*, vol. 18, (no. 2), Marcg/April 1998, pp. 66-75

[19] J. Gray, Sort Benchmark Home Page, http://research.microsoft.com/barc/SortBenchmark/