Optimizing Parallel Applications for Wide-Area Clusters

Henri E. Bal Aske Plaat Mirjam G. Bakker Peter Dozy Rutger F.H. Hofman
Dept. of Mathematics and Computer Science
Vrije Universiteit, Amsterdam, The Netherlands
http://www.cs.vu.nl/albatross/

Abstract

Recent developments in networking technology cause a growing interest in connecting local-area clusters of workstations over wide-area links, creating multilevel clusters, or meta computers. Often, latency and bandwidth of local-area and wide-area networks differ by two orders of magnitude or more. One would expect only very coarse grain applications to achieve good performance. To test this intuition, we analyze the behavior of several existing medium-grain applications on a wide-area multicluster. We find that high performance can be obtained if the programs are optimized to take the multilevel network structure into account. The optimizations reduce intercluster traffic and hide intercluster latency, and substantially improve performance on wide-area multiclusters. As a result, the range of metacomputing applications is larger than previously assumed.

Keywords*: Meta computing, Wide-area networks, Communication patterns, Parallel algorithms*

1 Introduction

One of the visions in the field of parallel computing is to exploit idle workstations to solve compute-intensive problems, such as those found in physics, chemistry, and biology. Today, many individual computers are connected by a local-area network (LAN) into a cluster. Advances in wide-area network technology make it possible to extend this idea to geographically distributed computers. By interconnecting multiple local clusters through a high-speed wide-area network (WAN), very large parallel systems can be built, at low additional cost to the user, creating a large parallel virtual machine. Several so-called meta computing projects (e.g., Legion [8], Condor [6]) try to create infrastructures to support this kind of computing. These projects try to solve the problems that result from integrating distributed resources, such as heterogeneity, fault-tolerance, security, accounting, and load sharing.

The usefulness of a meta computing infrastructure depends on the applications that one can run successfully on them. Since wide-area links are orders of magnitude slower than local-area links, it is reasonable to expect that only applications that hardly communicate at all (i.e., embarrassingly parallel applications) will benefit from multiple WAN-connected clusters. The research question we address here is *how parallel applications perform on a multilevel network structure*, in particular, on systems built out of both LANs and WANs. Existing meta computing projects often use applications with very coarse-grained (job-level) parallelism, which will perform well on any parallel system [8]. We investigate applications with a finer granularity, which were designed originally to run on a local cluster of workstations. In addition, we study optimizations that can be used to improve the performance on multilevel clusters.

The paper presents the following contributions. First, we present performance measurements for eight parallel programs on a wide-area multilevel cluster, and we identify performance problems. Second, we describe optimization techniques that substantially improve performance on a wide area system. Third, we conclude that, with the optimizations in place, many programs obtain good performance, showing that it is beneficial to run parallel programs on multiple WAN-connected clusters. This conclusion is surprising, since our system's WAN is two orders of magnitude slower than its LAN. Finally, since adequate performance can be obtained for a variety of nontrivial applications, our work indicates that meta computing efforts like Legion are all the more worth while.

The testbed we use consists of four cluster computers located at different universities in the Netherlands, connected by an ATM network (see Figure 2). The four clusters use identical processors (a total of 136 Pentium Pros) and local networks (Myrinet [4]). The advantage of this homogeneous setup is that we can study the impact of LAN and WAN speed on application performance without having to be concerned about other factors (e.g., differences in CPU types or speeds). We isolate one important performance factor and study its impact. The experimental testbed is designed specifically to allow this kind of research. In contrast, most meta computing projects use existing workstations, which leads to more heterogeneous testbeds.

Multilevel clusters are somewhat similar to NUMA (Non-Uniform Memory Access) machines, in that the communication latency is non-uniform. However, the relative difference between between sending a message over a LAN or a WAN is much higher than that between accessing local or remote memory in a NUMA (for example, in an SGI Origin2000 it is only a factor of 2–3 [10]).

2 Experimental setup

In this section we first describe the experimental system used for our research, including the hardware and systems software. Also, we give basic performance numbers for the system. (Additional information on the system can be found at *http://www.cs.vu.nl/˜bal/das.html*.)

The distinguishing feature of a wide-area multicluster is the large difference in latency and bandwidth of the communication links. Modern LANs have an application-to-application latency of 10–100 microseconds, whereas a WAN has a latency of several *milli*seconds. The bandwidth of a high-speed LAN is about 10–100 Mbyte/s; for WANs bandwidth varies greatly. Currently, an average sustained bandwidth of 100–5000 kbyte/s may be a reasonable assumption for most academic environments. (Note that these are application-level figures, not hardware-level.) It is expected that the relative differences between LAN and WAN performance will persist, if only for latencies, because the speed of light is becoming a limiting factor.

Our wide-area system consists of four clusters. It has been designed by the Advanced School for Computing and Imaging (ASCI).[1] It is called DAS, for Distributed ASCI Supercomputer. The main goal of the DAS project is to support research on wide-area parallel and distributed computing. The structure of DAS is shown in Figure 2. Each of the four participating universities has a local cluster of 200 MHz Pentium Pros. Three sites have a 24 node cluster; the VU Amsterdam has a 64 node cluster. Each node contains 64 Mbyte of memory, 256 Kbyte L2 cache, and a 2.5 Gbyte local disk. The system has 144 computers in total (136 compute nodes, four file servers, and four gateway servers). It has 10 Gbyte DRAM, 376 Gbyte disk, and 28.8 GFLOPS peak performance. The operating system used on DAS is BSD/OS from BSDI. The nodes of each cluster are connected by Myrinet and Fast Ethernet. Myrinet is used as fast local interconnect, using highly efficient protocols that run entirely in user space. Fast Ethernet is used for operating system traffic (including NFS, remote shell). The four clusters are connected by a wide-area ATM network. The network uses a Permanent Virtual Circuit between every pair of sites. At present, they have a Constant Bit Rate bandwidth of 6 Mbit/sec. Each cluster has one gateway machine, containing a ForeRunner PCA-200E ATM board. All messages for machines in a remote cluster are first sent over Fast Ethernet to the local gateway, which forwards them to the remote gateway, using IP.

Currently, two of the four clusters (at Delft and the VU) are operational and connected by the wide-area ATM link. We have measured that the roundtrip application-level latency over this ATM link is 2.7 milliseconds; application-level bandwidth was measured to be 4.53 Mbit/sec.

In order to run four-cluster experiments the 64 node cluster has been split into four smaller sub-clusters. The four sub-clusters are connected by a local ATM network. Each sub-cluster has one machine that acts as gateway; as in the real wide-area system, the gateway is dedicated and does not run application processes. With four sub-clusters, each sub-cluster thus consists of at most 15 compute nodes and one gateway. Each gateway contains the same ATM interface board as in the real system. The ATM firmware allowed us to limit bandwidth by transmitting extra idle cells for each data cell. This was used to configure the boards to deliver the same bandwidth as measured for the real system. In addition, we have increased the latency for communication over the local ATM links by modifying the low-level communication software running on the gateway.

Except for the ATM link, the real system and the experimentation system are the same; the same executable binaries are used on both systems. The experimentation system has been validated by running all applications on a two-cluster experimentation system and on the wide-area system consisting of the clusters in Delft and VU Amsterdam, using 16 compute nodes per cluster. The average difference in run times is 1.14% (with a standard deviation of 3.62%), showing that the wide-area ATM link can be modeled quite accurately in the way we described above.

The applications used for the performance study are written in Orca, a portable, object-based parallel language, in which processes communicate through shared objects. The Orca system replicates objects that have a high read/write ratio. Invocations on non-replicated objects are implemented using Remote Procedure Calls (RPCs). For replicated objects, read-only operations are executed locally. Write-operations on replicated objects are implemented using a write-update protocol with function shipping: the operation and its parameters are broadcast, and each machine applies the operation to its local copy. To keep replicated objects consistent, a totally-ordered broadcast is used, which guarantees that all messages arrive in the same order on all machines. Broadcasting is implemented using a single sequencer machine ordering messages.

The Orca Runtime System (RTS) on DAS uses Myrinet for *intracluster* communication (communication between processors in the same cluster). The low-level software is based on Illinois Fast Messages [13], which we extended to support fast broadcasting, amongst others [1].

For *intercluster* (wide-area) communication, the Orca system uses the ATM network. RPCs are implemented by first sending the request message over Fast Ethernet to the

[1] Unrelated to, and established before, the Accelerated Strategic Computing Initiative.

Benchmark	latency		bandwidth	
	Myrinet	ATM	Myrinet	ATM
RPC (non-repl.)	40 µs	2.7 ms	208 Mbit/s	4.53 Mbit/s
Broadcast (repl.)	65 µs	3.0 ms	248 Mbit/s	4.53 Mbit/s

Table 1: Low-level Orca performance

local gateway machine (which is not part of the Myrinet network). The gateway routes the request over the ATM link to the remote gateway, using TCP. The remote gateway delivers the message to the destination machine. Reply messages are handled in a similar way. Broadcasts pose some problems. Orca's centralized sequencer works well for the local cluster, but becomes a major performance problem on a WAN. For certain applications a specialized mechanism can be used, see Section 3.

Table 1 gives Orca's low-level performance figures, for intracluster and intercluster communication, for non-replicated objects and replicated objects. Latency is measured using null operations, bandwidth using a sequence of 100 Kbyte messages. The replicated-object invocation benchmark measures the latency to update an object that is replicated on 60 machines, which involves broadcasting the operation to all these machines. The remote object invocation latency involves a single remote object. The performance gap between the LAN and WAN is large; even a low communication volume is expected to cause applications to experience serious slowdowns.

3 Application Performance on DAS

We have selected eight existing Orca applications for our performance study. They were originally developed and tuned for an architecture with a single-level communication network (e.g., a LAN). On the LAN/WAN cluster performance was generally unsatisfactory, and optimizations have been developed for each program. The goal of this study is to see whether medium grain communication can (be made to) work on a wide-area multilevel cluster—not to achieve the best absolute speedup for a particular system or application. Therefore, applications and problem sizes were chosen to have medium grain communication: not trivially parallel, nor too challenging. For our set of input problems the applications obtain an efficiency between 40.5 and 98 percent when run on the local 64-node Myrinet cluster. The programs represent a wide variety of application domains and include numerical, discrete optimization, and symbolic applications. Table 2 lists the eight applications, together with a brief characterization, describing their type, their main communication pattern, and some basic performance data. Most applications primarily use point-to-point communication, except ACP and ASP, which use broadcast. We give both the total number of RPCs or broadcasts per second (issued by all processors together) and the total amount of user data sent per second. Finally, we give the speedup on 64 nodes. Essentially, the table shows that on a single Myrinet cluster all algorithms run reasonably efficiently.

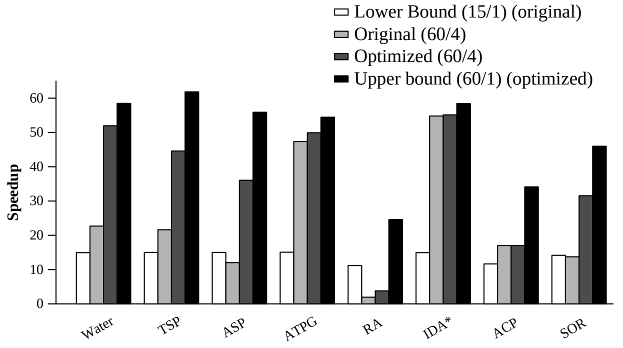

Figure 1: Four-Cluster Performance on 15 and 60 Processors

A performance comparison has been made for the eight applications on the wide-area system. Figure 1 summarizes the results. (Due to space constraints, details about applications, input problems, optimizations, and measurements are to be found in a technical report available from *http://www.cs.vu.nl/albatross/* [2].) To assess the efficiency of an application running on C clusters with P processors each, two metrics are useful. First, the best possible performance is that of the same program running on a single cluster with $C \cdot P$ processors. Second, for wide-area parallel programming to be useful, at the bare minimum one would want the program to run faster on C clusters with P machines each than on one cluster with P machines (i.e., using additional clusters located at remote sites should not hurt performance). These two metrics thus give an upper bound and a lower bound, or optimal versus acceptable performance.

Figure 1 shows the performance of the original and optimized programs for all eight applications, together with the lower and upper bound on acceptable performance. We show results for four clusters with 15 nodes each. For each application, the first bar is the speedup on a single 15-node cluster, which is the lower bound for acceptable speedup. The last bar of each application is the speedup on a single 60-node cluster, which is the upper bound on performance. The second and third bars are the speedups for the original and optimized programs on four 15-node clusters.

As can be seen, for the original programs five applications run faster on four clusters than on one cluster, although for only two applications (IDA* and ATPG) the performance approaches the upper bound; all others perform considerably worse; the slow WAN links caused a significant slow down. Subsequently, for each application optimizations were devised, to adapt the communication patterns to the new environment. For four applications (Water, TSP, ASP, and SOR), the optimizations cause the performance to come close to the upper bound: the optimized versions experience a modest performance degradation from the WAN links. Finally, by comparing the second and third bar for each application we can determine the impact of the

program	type	communication	# RPC/s	kbytes/s	# bcast/s	kbytes/s	speedup
Water	*n*-body	exchange	9,061	18,958	48	1	56.5
TSP	search	work queue	5,692	285	134	11	62.9
ASP	data-parallel	broadcast	3	49	125	721	59.3
ATPG	data-parallel	accumulator	4,508	18	64	0	50.3
RA	data-parallel	irregular	240,297	8,493	296	0	25.9
IDA*	search	work stealing	8,156	202	477	1	62.1
ACP	iterative	broadcast	77	826	1,649	557	37.0
SOR	data-parallel	neighbor	18,811	67,540	326	2	46.3

Table 2: Application characteristics on 64 processors on one local cluster.

performance optimizations. For five applications (Water, TSP, SOR, ASP, and RA) the impact is substantial, with an average speedup increase of 85 percent. These optimizations will be described next.

Water The Water program is based on the “n-squared” Water application from the Splash suite [14]. It suffers from a severe performance degradation when run on multiple clusters, as shown in figure 1. The problem is the exchange of molecule data. With the original program, the data for a given molecule are transferred many times over the same WAN link, since multiple processors in a cluster need the data item. In the optimized program, for every processor *P* in a remote cluster, we designate one of the processors in the local cluster as the *local coordinator* for *P*. If a process needs the molecule data of processor *P*, it does an intra-cluster RPC to its local coordinator, which gets the data over the WAN, forwards it to the requester, and caches it locally. If other processors in the cluster ask for the same data, they are sent the cached copy. A similar optimization is used at the end of the iteration. All updates are first sent to the local coordinator, which does a reduction operation (addition) on the data and transfers only the result over the WAN. (Coherency problems are easily avoided, since the local coordinator knows in advance which processors are going to read and write the data.) The new program achieves a speedup on four 15-node clusters that is close to the single 60-node cluster speedup. The optimization shows that substantial performance improvements are possible when the multilevel network structure is taken into account.

TSP The familiar Traveling Salesman Problem program uses master/worker parallelism, and is used here to study the behavior of a simple dynamic load balancing scheme with a centralized job queue. The overhead of intercluster communication is caused mostly by the work distribution mechanism, which uses a shared job queue object that is stored on one processor. Processors on remote clusters need to do an intercluster RPC each time they need a new job. With four clusters, about 75% of the jobs will be sent over the WAN. As an optimization, we used a static work distribution over the clusters, implemented with a local job queue per cluster. In this way, intercluster communication is reduced substantially, but load imbalance is increased. Nevertheless, this modification improved performance substantially.

ASP The All-pairs Shortest Paths Problem is to find the shortest path between any pair of nodes in a graph. The original program (using an input problem with 3,000 nodes) obtains poor performance on multiple clusters. The program performs 3,000 iterations. At the start of each iteration one of the processors broadcasts a row of the matrix to all other processors. Broadcasts are totally ordered, and the sender has to wait for a sequence number before continuing, which is a problem if the sequencer node is across the WAN link. This mechanism can be optimized by noting that all broadcasts are sent in phases: first processor 1 computes its rows and broadcasts them, then processor 2, etc. The sequencer mechanism of the Orca run-time system has been modified accordingly. The new mechanism migrates the sequencer to the cluster that does the sending, so that the sender receives its sequence number quickly, and can continue. This optimization works well, allowing pipelining of computation and communication.

Automatic Test Pattern Generation ATPG is a problem from electrical engineering. The program is parallelized by statically partitioning the problem among the processors. The program obtains a high efficiency on one cluster. On multiple clusters, the efficiency decreases only slightly, because the program does little communication. On slower networks (e.g., 10 ms latency, 2 Mbit/s bandwidth) the performance of the ATPG program is significantly worse (not shown) and the following optimization has been implemented. In ATPG, each processor updates a shared object (containing the number of test patterns) every time it generates a new pattern. On multiple clusters, many of these RPCs go over the wide-area network. The number of test patterns covered is only needed (and printed) at the end of the program execution. It is therefore possible to let each processor accumulate these numbers locally, to let local sums be computed inside a cluster, and to send the totals in one RPC when the cluster is finished with its part of the work. In this way, intercluster communication is reduced to a single RPC per cluster.

Retrograde Analysis RA is a technique to enumerate end-game databases, of importance in programs playing games such as checkers. The parallel program sends many small messages, in which the sender does not wait for a reply. This allows *message combining*, which greatly improves performance both within and across clusters. In our optimization, cluster-level message combining, a processor that wants to send a message to a machine in a remote cluster first sends the message to a designated machine in its own cluster. This machine accumulates all outgoing messages, and occasionally sends all messages with the same destination cluster in one large intercluster message. Still, the communication volume is too high for multiple clusters to out-perform a single cluster. (Without cluster-level message combining, the speedup on four 15-node clusters is even less than 1.) The communication pattern is highly irregular: every processor sends messages (containing updates for the database) to all other processors in a highly unpredictable way. It is therefore difficult to reduce the intercluster communication volume, making it unsuitable for the wide-area system (at least with our bandwidth and latency parameters).

IDA* Iterative Deepening A* is a combinatorial search algorithm. The program is useful to study another load balancing mechanism: a distributed job queue with work stealing. Each process has a local job queue from which it gets its jobs. When there are no local jobs, the process tries to steal a job from another process. Because of a rather good load balance, the program performs well on multiple clusters. Nevertheless, an optimization was introduced in the work stealing strategy. To steal a job, a worker process asks several other processors in turn for work, until it succeeds, causing a significant number of steal requests to go cross-cluster. We applied two optimizations. First, the optimized program first tries to steal jobs from machines in its own cluster. The second optimization tries to reduce the number of job requests by leveraging off the termination-detection mechanism to maintain load-balancing information, which makes it possible to avoid sending requests to processors known to be idle.

ACP Algorithms for the Arc Consistency Problem can be used as a first step in solving Constraint Satisfaction Problems. The program takes as input a set of *n* variables and a set of binary constraints defined on some pairs of variables, that restrict the values these variables can take. ACP performs many small broadcasts, causing much traffic for cluster gateways. Still, the speedup on two and four clusters exceeds that for a single cluster of the same number of processors. The sender of a broadcast message need not wait until the message has arrived at all processors. Therefore, a possible optimization for ACP is to use asynchronous broadcasts. This idea has not been implemented.

Successive Overrelaxation SOR is an iterative method for solving discretized Laplace equations on a grid. It is used as an example of nearest neighbor parallelization methods. The parallel algorithm we use distributes the grid row-wise among the available processors. The SOR program logically organizes the processes in a linear array. At the beginning of every iteration, each processor (except the first and last one) exchanges a row with both its upper and lower neighbor. This overhead already slows down the program on a single cluster, where all communication goes over the fast Myrinet network. The performance on multiple clusters is much worse, because the first and last processor of each cluster have to send a row to a remote cluster. In (red/black) SOR, the distributed matrix is kept consistent by exchanging all boundary rows after each iteration. As an optimization, at cluster boundaries we skip some row exchanges, reducing the communication overhead substantially, at the cost of slower convergence. Within a cluster all row exchanges proceed as usual. As the number of clusters increases, so does the relative number of row exchanges that are dropped; changes propagate more slowly through the matrix, and convergence becomes slower. The trade-off of intercluster communication versus convergence speed works well for our modest number of clusters. Multicluster speedup has been improved substantially, and with the optimization four 15-processor clusters are now faster than one.

4 Discussion

We will now discuss the communication patterns of the applications and the optimizations (summarized in Table 3). The basic idea is to trade off inter-cluster communication for intra-cluster communication—the goal is not to improve single-level communication, in fact, all optimizations have a negligible effect on single-cluster performance.

Since the performance of the WAN links is much lower than that of the LAN links, one important class of optimizations is reduction of the inter-cluster communication volume. Alternatively, measures to make better use of the available bandwidth can be taken by hiding intercluster latency. The optimizations that were applied to the algorithms either reduce intercluster communication or try to mask its effect.

For five applications (Water, IDA*, TSP, ATPG, SOR), we were able to reduce intercluster traffic. Both job queue optimizations fall into this category. The simplest scheme, a physically centralized work queue, leads to performance problems on a multilevel cluster. The optimization distributes the job queue over the clusters, dividing work statically over cluster queues. This trades off static versus dynamic load balancing, substantially reducing intercluster communication. The resulting increase in load imbalance can be reduced by choosing a smaller grain of work, at the expense of increasing intracluster traffic (TSP). For the fully distributed work-stealing scheme—giving each *processor* its own queue—it is more efficient to look for work in the local cluster before doing an expensive intercluster lookup, and also, to remember which queues were empty previously (IDA*). For associative all-to-one operations across cluster boundaries the optimization is to first perform reductions within the clusters. This occurs when computing statistics (ATPG). Another technique to reduce intercluster traffic is caching at the cluster level. For example, in applications where duplicate data is sent in a (personalized) all-to-all exchange, intercluster exchanges can be coordinated by a sin-

Application	Communication structure	Improvements
Water	All to all exchange	Cluster cache
TSP	Central job queue	Static distribution
ASP	Regular broadcast	Sequencer migration
ATPG	All to one	Cluster-level reduction
RA	Irregular message passing	Mess. comb. per cluster
IDA*	Distributed job queue with work stealing	Steal from local cluster Remember empty
ACP	Irregular broadcast	None implemented
SOR	Nearest neighbor	Chaotic relaxation

Table 3: Patterns of Improvement

gle machine per cluster, making sure that the same data travels over the same WAN link only once. In Water this is used for reading and writing molecule data. Finally, our red/black nearest neighbor algorithm was rewritten to one with a more relaxed consistency, reducing the intercluster communication by dropping some row exchanges (SOR).

For RA and ASP we used forms of latency hiding. Asynchronous point-to-point messages allow message combining, which can be applied at the cluster level. As in the cluster caching scheme, one processor per cluster is designated to handle all intercluster traffic, only now the task is to combine messages to reduce overhead. The combined messages are sent asynchronously over the WAN, allowing the sending processor to overlap computation and communication (RA). As noted in Section 2, totally ordered broadcast performs badly on a multilevel cluster. For certain cases, however, application behavior can be exploited. When several broadcasts are performed in a row by the same machine, they can be pipelined by using a single sequencer that is migrated to the cluster that initiates the broadcasts (ASP). Furthermore, although we did not implement this, asynchronous broadcasts can be pipelined (ACP).

Except for RA, all applications run significantly faster on multiple clusters than on one cluster. It is surprising that what in retrospect looks like a few simple optimizations can be so good at masking two orders of magnitude difference in hardware performance in parts of the interconnect. Apparently, there is enough flexibility in most algorithms to allow a limited number of LAN links to be changed into WAN links. Communication smoothing and intercluster latency hiding overlap computation with WAN communication; reduction of intercluster communication trades off WAN versus LAN communication. As can be expected when WAN links are so slow, intercluster traffic reduction generally achieves better results than latency hiding. It will be interesting to see if other optimizations for these and other algorithms exist.

5 Related Work

In this paper several medium grain algorithms (and optimizations) are studied on a multilevel communication network consisting of LAN and WAN links. The work is of direct relevance to research in meta computing. Some of the numerous projects such as Legion [8], Globe [9], Globus[7], and Condor [6] aim at creating a software infrastructure—addressing such problems as job scheduling, heterogeneity, and fault tolerance—to support computing on a much larger scale than previously thought possible. So far, meta computing has been aimed at coarse grain, embarrassingly parallel jobs. Our work studies applications with more challenging communication behavior.

Several other people have worked on performance analyses of parallel algorithms on specific architectures. Martin et al. [12] perform an analysis of the sensitivity of applications to changes in the LogP parameters in a single-level Myrinet cluster, by putting a delay loop on the Myrinet cards. A number of papers study algorithm behavior on NUMA machines, most notably, the papers on the Splash benchmark suite [14]. NUMA latencies differ typically by a factor of 2 to 3, and directory based invalidation cache coherency achieves adequate performance for a wide range of applications [5]. In wide-area multilevel clusters latency and bandwidth differences are much larger, and we used various forms of algorithm restructuring to achieve adequate performance.

Several researchers are working on coherency models for clustered SMPs or “Clumps” [11]. Here, the latency difference is typically one order of magnitude—larger than NUMA, smaller than a LAN/WAN cluster. Bilas et al. [3] study the behavior of a novel coherency scheme in a cluster of shared memory machines (SMPs). One of their conclusions is that for good performance algorithm restructuring is necessary, as we have done here.

6 Conclusion

Recent technology trends have created a growing interest in wide-area computing. The limited bandwidth and high latency of WANs, however, make it difficult to achieve good application performance. Our work analyzes several nontrivial algorithms on a multilevel communication structure. Both the bandwidth and the latency of the LAN and WAN links differ by almost two orders of magnitude. We have analyzed the applications, and found that, as expected, most existing parallel applications perform worse on multiple clusters connected by a WAN than on a single LAN cluster with the same number of processors. Some applications even perform worse on four clusters of 15 processors than on a single cluster. Since the applications had been designed and tuned for a single cluster, it is not surprising that performance suffers when some of the links become much slower. It is surprising, however, to see that the optimizations that we subsequently implemented worked so well. The optimizations either reduce intercluster traffic or mask the effect of intercluster communication. In some cases the algorithm was restructured, in others the implementation of communication primitives was refined.

It is encouraging, that although each type of optimization is used in only one application, each can be viewed as an adaptation of a general communication reduction and latency hiding technique, such as load balancing, caching, distributed reduction operators, pipelining, and relaxing the data consistency requirements of an algorithm. There is clearly a limit to what can be achieved by trading off WAN versus LAN communication and overlapping WAN communication with computation. Ultimately, the inherent communication of an algorithm and the limited bandwidth and high latencies of wide area links limit performance. Nevertheless, our conclusion is that many medium grain applications can be optimized to run successfully on a multilevel, wide-area cluster. This makes the work on meta computing all the more valuable, since a wider class of algorithms can be run on them than previously thought.

Future research in this area can look for more optimizations, for more applications. Performance was found to be quite sensitive to problem size, number of processors, number of clusters, and latency and bandwidth. This paper has only scratched the surface of these intricate issues, and further sensitivity analysis is part of our future work. Where the optimizations are instances of more general techniques, they can be used in wide-area parallel programming systems.

Acknowledgments

This research is supported in part by a PIONIER and a SION grant from the Netherlands Organization for Scientific Research (NWO). We gratefully acknowledge NWO's additional support and the ASCI research school for making the DAS project possible. We thank Kees Verstoep and Ceriel Jacobs for their work on Orca and DAS. We thank Rien van Veldhuizen for his help with SOR. Dick Grune, Ceriel Jacobs, Koen Langendoen, John Romein, Andy Tanenbaum, and Kees Verstoep gave feedback on the paper.

References

[1] H.E. Bal, R. Bhoedjang, R. Hofman, C. Jacobs, K. Langendoen, T. Rühl, and K. Verstoep. Performance of a High-Level Parallel Language on a High-Speed Network. *Journal of Parallel and Distributed Computing*, 40(1):49–64, February 1997.

[2] H.E. Bal, A. Plaat, M.G. Bakker, P. Dozy, and R.F.H. Hofman. Optimizing Parallel Applications for Wide-Area Clusters. Technical Report IR-430, Vrije Universiteit Amsterdam, September 1997. Available from http://www.cs.vu.nl/albatross/.

[3] A. Bilas, L. Iftode, and J.P. Singh. Shared Virtual Memory across SMP Nodes using Automatic Update: Protocols and Performance. Technical Report TR-96-517, Princeton University, 1996.

[4] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and W. Su. Myrinet: A Gigabit-per-second Local Area Network. *IEEE Micro*, 15(1):29–36, February 1995.

[5] D. Chaiken, C. Fields, K. Kurihara, and A. Agarwal. Directory-Based Cache Coherence in Large-Scale Multiprocessors. *IEEE Computer*, 23(6):49–58, June 1990.

[6] D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A Worldwide Flock of Condors: Load Sharing among Workstation Clusters. *Future Generation Computer Systems*, 12(1):53–66, May 1996.

[7] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Int. Journal of Supercomputer Applications*, 11(2):115–128, Summer 1997.

[8] A.S. Grimshaw and Wm. A. Wulf. The Legion Vision of a Worldwide Virtual Computer. *Comm. ACM*, 40(1):39–45, January 1997.

[9] P. Homburg, M. van Steen, and A.S. Tanenbaum. Communication in GLOBE: An Object-Based Worldwide Operating System. In *Proc. Fifth International Workshop on Object Orientation in Operating Systems*, pages 43–47, October 1996.

[10] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In *24th Ann. Int. Symp. on Computer Architecture*, pages 241–251, June 1997.

[11] S.S. Lumetta, A.M. Mainwaring, and D.E. Culler. Multi-protocol active messages on a cluster of SMP's. In *SC'97*, November 1997. Online at http://www.supercomp.org/sc97/proceedings/.

[12] R.P. Martin, A.M. Vahdat, D.E. Culler, and T.E. Anderson. Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture. In *24th Ann. Int. Symp. on Computer Architecture*, pages 85–97, June 1997.

[13] S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet. In *Supercomputing '95*, San Diego, CA, December 1995.

[14] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture*, pages 24–36, June 1995.

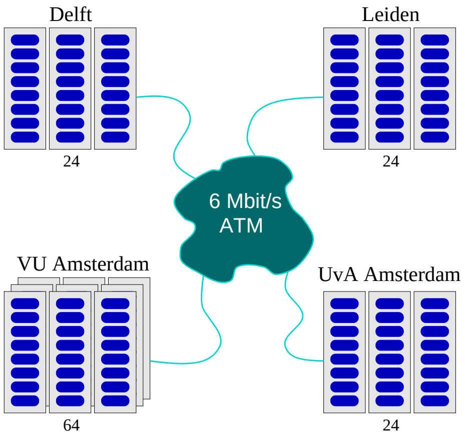

Figure 2: The Distributed ASCI Supercomputer