Bandwidth and Latency Sensitivity of Parallel Applications in a Wide-Area System

Aske Plaat Henri Bal Rutger Hofman
aske@cs.vu.nl bal@cs.vu.nl rutger@cs.vu.nl
Department of Mathematics and Computer Science, Vrije Universiteit
Amsterdam, The Netherlands
http://www.cs.vu.nl/albatross/

This paper has been submitted for publication

March 16, 1998

Abstract

As parallel architectures become larger, interconnects become increasingly hierarchical, resulting in an increasing gap in memory access times. Exactly how application performance will be affected by this gap is unclear. In this paper we study the effect of a gap of orders of magnitude in bandwidth and latency. Our system consists of local Myrinet clusters of Pentium Pros connected by dedicated wide-area ATM links. We use applications with a medium to high level of communication, that have been optimized for such a two-level system. The results show a surprisingly large tolerance to gaps in bandwidth and latency. Four out of six applications achieve adequate speedup for a wide-area latency up to 30–100 millisecond, implying the feasibility of cluster computing up to a distance of 10,000 miles.

1 Introduction

As parallel computer systems become larger, their interconnects will become more hierarchical, resulting in nonuniform latencies and bandwidths. This trend is already visible in NUMA machines and clusters of SMPs, where local memory accesses are typically a factor of 2–10 faster than remote accesses [28, 29]. The performance gap will be much larger in a meta-computer or a computational grid, which are built by interconnecting multiple parallel computers through wide-area networks. In a meta-computer, the wide-area interconnect can be several orders of magnitude slower than the local network. Given the large interest in meta-computing (with projects such as Legion [20], Globus [18], and others [1, 3, 16, 21, 14, 33, 35, 36, 42, 47]) application performance becomes an important topic. This paper studies the following questions:

- Can adequate performance be achieved on a meta-computer for non-trivial parallel applications?
- What is the impact of a large gap in bandwidth and latency between the fast and slow interconnect?

Performance is studied with several nontrivial parallel applications. We have built an experimental testbed using 64 Pentium Pros, a high-speed network (Myrinet) and an ATM network. The testbed can be configured as multiple Myrinet-clusters that are interconnected by ATM links with different latencies and bandwidths. In this way, we can emulate a variety of meta-computer configurations, where the gap between the fast local network (Myrinet) and the wide-area network (ATM) varies from 0 to 3 orders of magnitude. The measurements have been validated using a real wide-area system.

The goal of this study is to gain insight in application performance under a wide range of gaps in latencies and bandwidths. Most meta-computing projects currently use embarrassingly parallel (job-level) applications

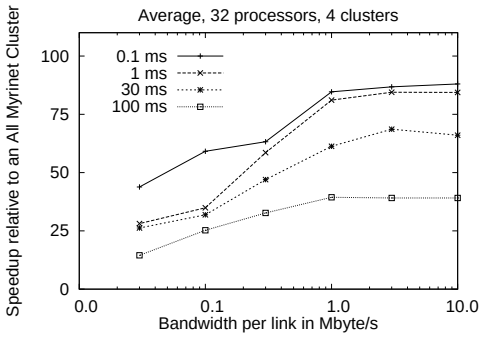

Figure 1: Mean Four-Cluster Performance

that barely communicate. Our study, on the other hand, targets a more challenging set of applications, including medium and fine-grained programs with diverse communication patterns. Furthermore, most existing applications are written for a particular architecture, such as shared or distributed memory. In order to eliminate this bias, and come closer to the inherent limit on speedup of an application, we have carefully hand-optimized each application to take the gap between WAN and LAN speeds into account. Most optimizations are adaptations of well-known ideas to a multi-cluster environment (e.g., cluster caches, message combining, latency hiding). The performance of these optimized programs is discussed and analyzed for bandwidths ranging from 10 to 0.03 MByte/s, and for one-way latencies ranging from 0.1 to 300 milliseconds. For comparison, at application-level, Myrinet has a bandwidth of 50 MByte/s and a one-way latency of 18 µs in our system.

We find that for surprisingly large bandwidth and latency gaps good speedup can be achieved. As a preview, Figure 1 shows the impact of wide area latency and bandwidth, compared to performance of an all-Myrinet cluster. The figure shows the geometric mean speedup of four applications, Barnes-Hut, Water, TSP, and ASP, for 4 clusters of 8 processors, as a percentage of the speedup on a single 32-node Myrinet cluster. For bandwidths as low as 0.1–0.3 MByte/s, and one-way latencies as high as 30–100 ms, multi-cluster speedups of 40–60% of single-cluster speedups can be achieved. This corresponds to a bandwidth gap of around 300, and a latency gap of around 3000. We conclude that in the optimized applications (a) speedup is insensitive to a gap in bandwidth between the local and wide-area network, and (b) speedup is even less sensitive to a gap in latency. On an efficiently routed fiber optics network, 100 ms corresponds to more than 10,000 miles—close to half the circumference of the earth. One implication is that the set of applications that can be run on future large scale, computational-grid like, architectures is larger than assumed so far, and includes medium grain applications.

The rest of the paper is concerned with finding the reason for such a good speedup, and its sensitivity to bandwidth and latency gaps. Section 2 discusses related work. Sections 3 and 4 describe in detail the system and the applications that have been used in this experiment. Section 5 describes the results of our measurements, and analyzes them. Section 6 concludes the paper and discusses implications of this work.

2 Related Work

Other performance evaluations have used either single-tier systems, or multi-tier systems with a qualitatively smaller performance difference. The Splash-2 paper [46] characterizes applications using a simulation of a cache coherent NUMA machine. In NUMAs like the SGI Origin 2000 access times differ typically by a factor of three [26]. In clusters of SMPs differences are up to one order of magnitude. Many groups are extending their software DSM to cover clusters of SMPs. Studies found that achieving high performance may necessitate application rewriting, and that performance can be quite sensitive to variations in the configuration of the machine (such as the number of processors per network interface) [9, 22, 29, 38, 41]. This agrees with our findings. Wide-area systems provide the most challenging environment in terms of latency and bandwidth gaps, but also of fault tolerance and heterogeneity. Meta-computing research focuses on the latter two issues [18, 20]. Because of the high (and non-uniform) latencies, applications are typically embarrassingly parallel. The only other work that we know of that tried medium grain applications on a meta-computer is [6]. They used a set of smaller applications, and did not perform a sensitivity analysis of bandwidth and latency gaps.

Bandwidth and latency sensitivity have been studied in [12, 13, 30]. They focused on sensitivity of communication mechanisms to absolute, single-level, bandwidth and latency, while we target the difference in a multi-tier

system. Related issues are also studied, for example, by [31, 32, 48]. Again, the basic architecture assumptions are based on traditional NUMA access latencies.

This work builds on other research in applications, systems and networking. Compiler research has shown promising results in locality optimizations (see for example [2]), but does not address multi-tier locality. An excellent introduction to algorithms for different physical networking topologies is [27]. Scheduling in heterogeneous networks is discussed in [45]. The multi-level optimizations that we perform build on work in single-level communication patterns that are also used in, for example, [7, 25]. Studies devoted to restructuring applications and communication patterns are [22, 40]. High performance computing recognizes the importance of high level communication patterns. See, for example, BSP [43] and MPI [17]. In ccNUMAs, communication is multi-level by default [28, 39], in contrast to explicit use in our multi-tier optimizations.

3 Experimental Setup

This section describes the system with which we perform our experiments.

We use an experimental cluster-of-clusters system that consists of four local clusters of 200 MHz/64 MByte Pentium Pro machines connected by Myrinet [11], using LANai 4.1 interfaces. The peak bandwidth of Myrinet is 2.4 Gbit/s, and one-way host-to-host latency is at least 5 μs. In our system, application-level bandwidth is 50 MByte/s, one-way latency is 18 μs. The clusters are located at four universities in the Netherlands, which are at most 50 miles apart (it is a small country). They are connected via dedicated gateway machines over ATM by 6 Mbit/s Permanent Virtual Circuits (application-level bandwidth is 0.55 MByte/s over TCP). The round trip latency is 2.5–3 ms between the universities that are the furthest apart, which corresponds to a third of the speed of light through fiber. Three sites have 24 compute nodes each, one site has 64. The wide-area network is fully connected, the local area networks are hypercubes. The operating system is BSD/OS version 3 from BSDI.

There are three distinctive features about the system that make it an interesting research vehicle. First, it has a high speed local area network, with a throughput that is comparable to modern SMP and hardware DSM speeds, and a latency that is one order of magnitude worse than typical SMP speeds. Second, the wide-area network has predictable performance, making it well suited for benchmarks, unlike the traditional Internet. The wide-area bandwidth is higher than what most current Internet links offer, but future networks will provide even higher, Gigabit/s, bandwidths. Third, there is a large difference in bandwidth and latency between the layers of the interconnect, providing the basis for this research. Another important issue is that the system is homogeneous, making it well suited for isolating individual communication aspects (if unsuited for modeling "real life" behavior of heterogeneous meta-computing systems).

The goal of this paper is to gain insight into the effect of the interconnect on application behavior using realistic hard and software. The wide-area ATM links have a fixed latency and bandwidth. To allow for experimentation with different speeds, 8 local ATM links have been installed in the 64 processor cluster, using the same hardware as in the real wide-area system (ForeRunner PCA-200E ATM boards). The latency and bandwidth of the ATM links van be varied by delay loops in the cluster gateway machine. Except for the local ATM links this experimentation system is identical to the real wide-area system; the same binaries are run in both setups, and except for the delay loops, there are no simulated parts. When the delay loops are set to the wide area latency and bandwidth, run times differ on average by 3.6% for our applications. All measurements presented in this paper have been performed on the dedicated experimentation system using the local ATM links.

The system can be programmed through different libraries and languages, from message passing libraries such as MPI and Panda [4] to parallel languages such as Orca [4] and CRL [23]. The Panda messaging layer has been adapted to support communication over both Myrinet (using Illinois Fast Messages [34]) and ATM (using BSD's TCP/IP). Panda hides the multiplexing/de-multiplexing details, allowing the programmer to communicate locally or remotely with the same primitives; the logical destination address determines whether communication will go to a remote cluster. The system exposes the wide-area/local area topology explicitly. On top of the messaging layer the Orca system runs, an efficient, entry consistent [8], parallel language that has been extended to allow inquiries

Program	Speedup 32 p.	Speedup 8 p.	Total Traffic 32 p. MByte/s	Runtime 32 p, in sec
Water	31.2	7.8	3.8	9.1
Barnes	27.2	7.1	17.8	1.8
TSP	31.1	7.9	0.48	5.9
ASP	31.3	7.8	0.75	6.0
Awari	7.8	4.6	4.1	2.3
FFT	26.7	5.3	128.0	0.3

Table 1: Single Cluster Speedup on 8 and 32 processors

about the cluster topology at runtime. Five of our six applications are written in Orca, for ease of use of the wide-area system, and for ease of debugging (Orca is type-safe). For most programs, serial performance is comparable to serial C performance. Barnes-Hut is written in C with calls to the Panda wide-area/local area messaging layer.

4 Applications

This section provides a summary of the applications that are run, the optimizations that were applied to these applications, and a characterization of their communication behavior on a single Myrinet cluster. The applications have diverse communication patterns. More details about the applications can be found elsewhere [4, 5, 46]. Table 1 summarizes the behavior of the applications on a single Myrinet cluster. For all our applications larger problems give better speedups. We use relatively small problem sizes in order to get medium grain communication, so that the speedup measurements show interesting behavior. Medium grain is defined here as a total communication volume of at least 100 KByte/s on a single level cluster of 32 processors. All applications and problem sizes run with a good efficiency on a single Myrinet cluster. Because of the good performance of the other applications, we also included FFT in the test set. This application has high communication requirements that could not be optimized for the multi-cluster, providing a way to explore the limits of our parameter space.

Water The Water program is based on the "n-squared" Water application from the Splash suite [46], rewritten for distributed memory [37]. Related distributed memory optimizations are described by [22]. We report on experiments with a medium sized input set of 1500 particles. The serial speed of the distributed memory program is about ten percent better than the original Splash code.

This version suffers a severe performance degradation on multiple clusters. A form of application-level cluster caching is used for better performance. 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 same data item. In the optimized program, for every processor P in a remote cluster, one of the processors in the local cluster is designated 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. When 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. Table 2 lists the performance impact of the optimization on a 4-cluster system.

Barnes-Hut Barnes-Hut is an $O(n \log n)$ N-body simulation. The version of the Splash-2 suite has a fine coherency unit which causes inefficiencies on coarse grain hardware [4, 22]. In this experiment a new distributed memory code by Blackston and Suel [10] has been used. Instead of finding out at runtime which nodes and bodies are needed to compute an interaction, this code precomputes where nodes and bodies are needed, and sends them in one collective communication phase at the start of each iteration. Stalls are thus eliminated from the computation phase [10]. Related improvements for Barnes-Hut codes for distributed memory have been reported by [19, 22, 44]. Using the same input problem, the serial program runs slightly faster than the Splash code (while still computing the exact same answer). We used a moderate sized input set of 64K particles. The program is written for the BSP communication library [43] in C, with calls to the wide-area/local area messaging layer. It is further optimized with message combining at the BSP and cluster level and overlaps communication with computation to further reduce the impact of wide-area communication.

ASP The All-pairs Shortest Path program is a parallel version of the classic Floyd-Warshall algorithm [15]. It uses a relatively small replicated distance matrix of 1500 by 1500 entries. Each processor iterates over rows in the distance matrix, and broadcasts result rows as they are computed. These have to be received in-order by the other

processors before they can compute their rows. A designated node issues sequence numbers to achieve this ordering. On the wide-area cluster an optimization is performed by taking advantage of the predictable nature of the broadcasts: since the algorithm computes the rows in the matrix in sequence, all sends first originate from the processors in a single cluster. The optimization is to migrate the sequencer to a processor in the cluster of the sending processor. In this way, the sequencer only migrates 3 times with 4 clusters, and senders will get a sequence number quickly, allowing them to continue computing, while in the mean time their rows will get delivered to the other clusters asynchronously.

TSP The Traveling Salesperson Problem computes the length of the shortest path along n cities, by enumerating the number of possible paths. The program, originally written in Orca, uses a centralized job queue which is filled with partial paths, from which workers get jobs. A small 16 city problem is used as input; jobs consist of a partial tour of 6 cities, creating small jobs and a fine communication grain, as the high communication volume in Table 1 shows. Deterministic runs are ensured by using a fixed cutoff bound [4].

Even though the program communicates infrequently on a single-tier cluster, its centralized job queue causes too much wide-area traffic on a multi-tier cluster to achieve good performance. The wide-area optimization is to replicate the queue, by giving each cluster its own queue, and to perform work stealing among the cluster queues to achieve a good load balance. The work stealing mechanism communicates infrequently. Wide-area traffic is solely influenced by the number of clusters, not by the number of processors per cluster.

Awari Awari, or more precisely retrograde analysis, is a symbolic application that computes end game databases, of importance for programs playing games such as checkers. It is based on backwards reasoning and bottom-up search. The program, written in Orca, sends many small, asynchronous, packets of work to other processors [5]. These messages are combined into larger messages for performance reasons. Here we compute a relatively small 9 stone database for the African board game Awari, whose speedup on the single cluster is mediocre. The communication pattern of Awari is random asynchronous point to point messages. The optimization uses message combining at the cluster level, with cross cluster messages first being assembled at a designated local processor, then sent in batch over the wide-area link, and subsequently being distributed by a designated processor at the other cluster to the final destinations. Here wide-area communication is structurally the same as the original pattern, though messages are larger and less frequent. Messages represent work for the other processors. The computation proceeds in phases, and increasing message combining increases load imbalance, since processors are starved of work at the end of a phase.

FFT The FFT application computes a one dimensional Fast Fourier transform, using the transpose algorithm [24]. The program is a rewrite of the Splash-2 code for distributed memory, and achieves an excellent speedup on a single Myrinet cluster. The communication part of this program is very simple: it performs three transposes, interspersed by parallel FFTs. No multi-tier optimization for the transposes has been found. 1D FFT is well known for its high communication volume. It is especially ill-suited for a system with long latencies and low bandwidths, such as a wide-area network. The purpose of this work, however, is to gain insight in the limits of multi-layer wide-area systems, and 1D FFT fits this purpose. Due to Orca's pointer-less array handling the serial code runs about 80 percent more slowly than the original C code. (Speedup is not inflated, however, since a C version of the same rewritten algorithm using the CRL DSM achieved the same single-cluster speedup.) The problem size is 2^{20} complex floating point numbers, the largest that could fit in memory, which still takes only a few seconds to run.

Despite the small input problems, all applications except Awari perform well on a single Myrinet cluster (Table 1). Table 2 compares speedup (relative to a single processor) of the optimized and the unoptimized versions, for a bandwidth of 1 Mbyte/s, a wide-area latency of 3 ms, and 4 clusters of 8 processors each.

For the Barnes-Hut application, the "original" version is the Blackston and Suel code run without our own latency hiding optimizations (e.g., message combining at the BSP and cluster level).

5 Results

The goal of this work is charting the sensitivity of application performance to gaps in bandwidth and latency—

Program	Optimized	Original
Water	27.2	0.2
Barnes-Hut	19.3	0.12
TSP	20.0	5.95
ASP	24.0	6.37
Awari	1.57	0.94
FFT	–	1.16

Table 2: Optimized versus Unoptimized Speedup

the effort invested in restructuring the applications is but a means to achieve this goal. This section discusses the performance measurements.

The speedup of a multi-cluster machine is bounded from above by the speedup of a single-cluster machine with the same number of processors, and the same execution schedule. (To put this differently, when some of the fast Myrinet links of the interconnect are changed into slow wide-area ATM links, our parallel applications will run more slowly.) For multi-cluster runs the wide-area bandwidth and latency are limited by our local OC-3 ATM link of 155 Mbit/s—at application level, over TCP, this comes to about 10 MByte/s and 0.1 ms, substantially worse than the Myrinet speeds of the corresponding single cluster setup. Speedups are shown relative to the all-Myrinet upper bound.

5.1 Relative Speedup

Figure 2 shows speedup graphs for the six applications, for 4 clusters of 8 processors. Speedup is shown relative to the speedup of the 32 processor all-Myrinet cluster, for different wide-area bandwidths and latencies. It is computed as $\frac{T_L}{T_M}$ where T_L is the run time of the single cluster and T_M is the run time of the multi-cluster. The wide-area ATM links run TCP, whose occasional retransmissions cause small perturbations in run time for bursty communication in Barnes-Hut and FFT (as it does in the real wide area system). Startup phases are omitted from time and traffic measurements. Myrinet bandwidth is constant at 50 MByte/s, latency is constant at 18 µs. ATM bandwidth is shown along the *x*-axis for 10, 3, 1, 0.3, 0.1, and 0.03 MByte/s. ATM latency is measured at 0.1, 0.3, 1, 3, 10, 30, 100, and 300 ms. To avoid clutter in the graphs, some latency lines are omitted. All latencies are one-way.

The general shape of the graphs is as can be expected: higher bandwidths and lower latencies improve performance, and multi-cluster (ATM) performance is lower than single-cluster Myrinet performance. For high bandwidth/low latency combinations performance for four applications, Barnes-Hut, Water, ASP, and TSP, is very good. For wide-area latencies of 0.1–3 ms and bandwidths of 0.3–10 MByte/s multi-cluster speedup is well above 50% of single-cluster speedup. For bandwidths better than 1 MByte/s speedup reaches 60% for 30 ms latency, and about 40% for 100 ms latency. (In ASP, communication can be overlapped with computation. Performance drops sharply at 1 MByte/s, since at that point the network becomes saturated.) For extreme bandwidths and latencies (30 KByte/s bandwidth or 300 ms latency) relative speedup drops below 25%, which corresponds to the performance of a single Myrinet cluster of 8 processors. Thus, for these bandwidths and latencies, using extra clusters actually slows down the computation.

Performance for Awari and FFT is significantly lower. For Awari 40% speedup is achieved for bandwidths better than 1.0 MByte/s and latencies better than 0.3 ms. For FFT the 25% point is not even reached. FFT at last shows the performance that one would expect when some of the links are slowed down by an order of magnitude. The reason for the bad performance of Awari and FFT is that these applications have a higher level of wide-area communication. In Awari the multi-level optimizations are less effective because too much cluster combining introduces load imbalance; in FFT no optimization has been implemented.

To summarize, for Barnes-Hut, Water, ASP, and TSP, reasonable speedups start at a bandwidth of 0.1–0.3 MByte/s and a latency of 30–100 ms. Given a Myrinet bandwidth of 50 MByte/s and a latency of 18 µs, this corresponds to a local area/wide-area performance gap of 167–500 for bandwidth and 1667–5556 for latency, depending on whether 40% or 60% of single-cluster speedup is desired. Thus, the applications allow a significantly larger gap for latency than for bandwidth.

In addition to the sensitivity to bandwidth and latency gaps, we have also performed experiments with different cluster structures. (This is a straightforward effect, and graphs are omitted for reasons of space.) Performance increases as there are more, smaller, clusters: a setup of 8 clusters of 4 processors outperforms 4 clusters of 8 processors. This may seem counter-intuitive, since replacing fast links with slow links ought to reduce performance. However, performance is limited by wide-area bandwidth, and our wide-area network is fully connected: in the multi-

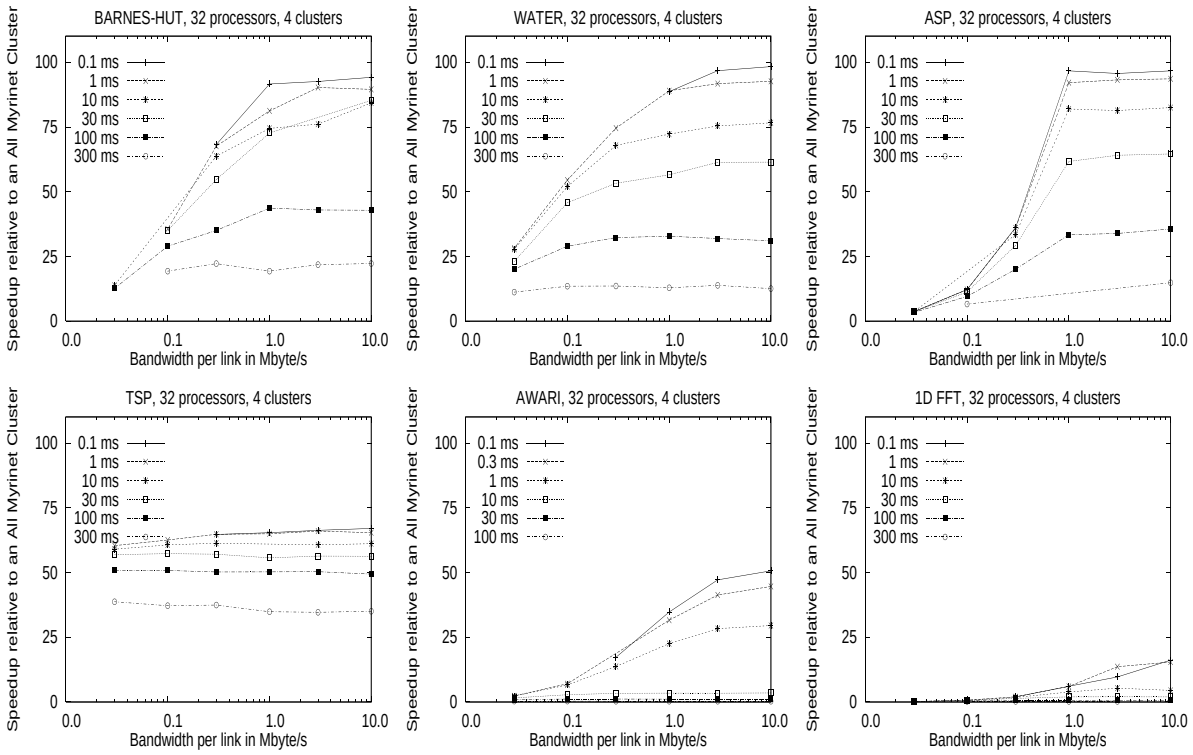

Figure 2: Speedup Relative to an All-Myrinet cluster

cluster, bisection bandwidth actually increases as more slow links are added, despite the loss of fast links. This effect can be traced back to simple bandwidth sensitivity: speedup decreases as more processors compete for the same wide-area links.

In a larger system it is likely that the topology is less perfect. This effect will then diminish, and disappear in star, ring, or bus topologies. Future topologies will in practice be somewhere in between the worst case of a star or ring and the best case of a fully connected network.

5.2 Wide-Area Communication

This subsection analyzes the wide-area traffic in more detail, to complement the speedup picture. The left graph in Figure 3 summarizes wide-area traffic of the applications. It shows data volumes in MByte/s per cluster and numbers of messages per second per cluster (for 3 MByte/s bandwidth per link and 0.3 ms latency, and 4 clusters of 8 processors, a configuration with 12 wide-area links in total). TSP can be found in the origin of the graph; it has extremely low wide-area communication, a few tens of KByte/s and few messages per second. FFT has a high communication volume of more than 3 MByte/s (note that the bandwidth limit in this case is 9 MByte/s per cluster, since with 4 clusters there are 3 links of 3 MByte/s out of each cluster). Awari can be found at the other end of the graph, with a high number of messages over the WAN (more than 4000 per second per cluster). Barnes-Hut, Water and ASP have a modest level of wide area traffic, less than 1000 messages per second and less than 0.5 MByte/s per cluster.

Performance is influenced strongly by intercluster traffic (high traffic applications have a low speedup and vice versa). The speed of the interconnect influences communication and synchronization overhead of the programs.

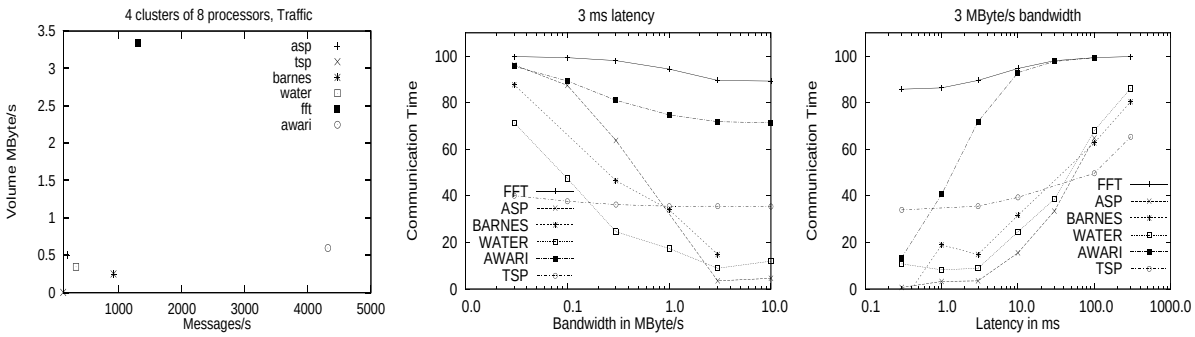

Figure 3: Wide-Area Traffic

The middle graph in Figure 3 shows the percentage of runtime that is spent on communication over the wide-area interconnect, relative to bandwidth, for 4 clusters of size 8 (one-way latency is set at 3 ms). The right graph in Figure 3 shows the wide-area communication time of the interconnect relative to latency (bandwidth is set at 3 MByte/s). The communication time percentage is computed as $\frac{T_M - T_L}{T_M} \cdot 100$, or the difference between multi-cluster run time and single cluster run time as a percentage of multi-cluster run time.

In both the bandwidth and the latency graph communication time for FFT is close to 100%, indicating that run time is almost completely dominated by communication. Awari is a close second, although at latencies lower than 10 ms communication time drops sharply (at 3 MByte/s). For Barnes-Hut, Water, ASP and TSP communication time is significantly less at high bandwidth and low latency.

These graphs show which applications are dominated by synchronous communication, and which by asynchronous communication. Purely asynchronous communication is limited by bandwidth (if we disregard startup time). Purely synchronous communication (i.e., a null-RPC) is limited by latency. Purely asynchronous communication would give a horizontal line in the latency graph (and vice versa). The graphs in Figure 3 show that the actual applications have a mix of both: streaming of asynchronous communication is present, but also request/reply style synchronous communication. (These effects translate for the speedup graphs in Figure 2 to the trivial observation that both lower latency and higher bandwidth give better speedup.)

It is interesting to note the differences among the applications, however. First, FFT is heavily communication bound. Little more can be said about its line, since it is so close to 100%.

Latency: Up to 3 ms Barnes-Hut, Water, and ASP are relatively insensitive to latency; their lines are still rather flat. For longer latencies communication becomes quite sensitive to latency. Apparently, at the 3 ms point the room in the data-dependencies of the programs that hides latency becomes exhausted. TSP's line stays quite flat up to 100 ms.

Bandwidth: For a bandwidth of 10–3 MByte/s, Barnes-Hut, Water, and ASP are relatively insensitive to bandwidth. TSP is almost completely insensitive to bandwidth; its work-stealing communication pattern comes quite close to the null-RPC. Awari is also quite sensitive to latency, more so than to bandwidth. It sends many relatively short messages, to obtain a good load balance.

The previous subsection comes to the general conclusion that Barnes-Hut, Water, ASP and TSP are more sensitive to the bandwidth gap than to the latency gap (by a factor of 167–500 and 1667–5556, respectively), and that Awari and especially FFT have a significantly lower performance. The bandwidth and latency graphs of Figure 3 show in more detail for each application the nature of its sensitivity to wide-area bandwidth and latency.

The reason for the low level of inter cluster communication of Barnes-Hut, Water, ASP, and TSP is that the multi-cluster optimizations work well. For Awari the optimization has limited effectiveness, and for FFT no optimiza-

tion was found. The communication patterns of Awari and FFT are quite simple (asynchronous point-to-point messages and personalized all-to-all), leaving little room for optimizations.

In Barnes-Hut, Water, ASP, and TSP, the single-tier communication patterns have a medium communication load; when, in the single-level situation the capacity of all links is the same, one need not distinguish between communication loads over different links. All communication is lumped together and described as "medium grain." In the two-tier case it pays to be more careful and to look for ways to reduce communication over certain links, creating a (clustered) multi-level communication pattern, to reduce the communication load over the slow links. Calling such a communication pattern "medium grain" is a simplification.

6 Conclusions

We have analyzed the performance of applications with a non-trivial communication load on a cluster of clusters, focusing on the effect of large gaps in bandwidth and latency between the slow and the fast part of the interconnect. The main conclusion is that performance is remarkably high for a bandwidth gap up to a factor of 167–500, and a latency gap up to a factor of 1667–5556. The fact that performance is even less sensitive to latency than to bandwidth gaps is fortunate since in the future bandwidth is expected to improve more than latency. The tolerance to latency is remarkable. Our situation is different from others in that we have a multi-tier system. Most of the links in the interconnect are still fast Myrinet links, and the load over the (few) slow links has been reduced by optimizations. The surprisingly large opportunity for creating clustered communication out of flat communication patterns is the reason for the large tolerance to different latencies in the interconnect.

This work has important implications for related fields. First of all, meta-computing has largely been targeted towards embarrassingly parallel applications. The set of applications that can be run efficiently on a meta-computer is larger than expected (although meta-computers suffer from additional overheads due to non-dedicated wide area links and heterogeneous hardware, whereas our system is homogeneous). Second, the large tolerance to latency gaps suggests ample room for multi-tier prefetching. Third, it implies the feasibility of scaling up current hardware and software DSM designs to larger interconnects with longer latencies. Finding an effective long latency cache coherence protocol, however, is a challenging problem. Another, less general, approach is to embed multi-tier optimizations in a high level communication library such as BSP or the collective communication primitives of MPI.

Future systems will have deeper hierarchical interconnects than today. This paper has given a preview of how good performance can be obtained on such systems.

7 Acknowledgements

This research is supported in part by a PIONIER and a SION grant from the Dutch Organization for Scientific Research (NWO) and a USF grant from the Free University. NWO has further supported the DAS machine, providing a basis for this research. We are grateful to Torsten Suel for providing us with his n-body code. We thank Raoul Bhoedjang for many insightful and inspiring discussions. We thank Andy Tanenbaum for his sound judgment. This work benefited further from discussions with Tom Cormen, Andrew Grimshaw, Arvind Krishnamurthy, Monica Lam, Koen Langendoen, Rich Martin, Sivan Toledo, and Kees Verstoep. Raoul Bhoedjang, Thilo Kielmann, and Andy Tanenbaum provided valuable feedback on the paper. Mirjam Bakker implemented the Awari optimizations. Peter Dozy implemented the Water and TSP optimizations. Kees Verstoep kept the DAS alive.

References

[1] A. D. Alexandrov, M. Ibel, K. E. Schauser, and C. J. Scheiman. SuperWeb: Towards a Global Web-Based Parallel Computing Infrastructure. In *11th International Parallel Processing Symposium*, April 1997.

[2] J. Anderson, S. Amarasinghe, and M. Lam. Data and computation transformations for multiprocessors. In *5th Symposium on Principles and Practice of Parallel Processing*, July 1995.

[3] J. Arabe, A. Beguelin, B. Lowekamp, E. Seligman, M. Starkey, and P. Stephan. Dome: Parallel programming in a heterogeneous multi-user environment. In *10th International Parallel Processing Symposium*, pages 218–224, April 1996.

[4] H. Bal, R. Bhoedjang, R. Hofman, C. Jacobs, K. Langendoen, T. Rühl, and F. Kaashoek. Performance Evaluation of the Orca

Shared Object System. *ACM Transactions on Computer Systems*, 16(1), February 1998.

[5] H.E. Bal and L.V. Allis. Parallel Retrograde Analysis on a Distributed System. In *Supercomputing '95*, December 1995. Online at http://www.supercomp.org/sc95/proceedings/.

[6] H.E. Bal, A. Plaat, M.G. Bakker, P. Dozy, and R.F.H. Hofman. Optimizing Parallel Applications for Wide-Area Clusters. In *IPPS-98 International Parallel Processing Symposium*, April 1998.

[7] J. Bennett, J. Carter, and W. Zwaenepoel. Implementation and performance of munin. In *13th Symposium on Operating Systems Principles*, pages 152–164, October 1991.

[8] B.N. Bershad, M.J. Zekauskas, and W.A. Sawdon. The Midway Distributed Shared Memory System. In *Proc. COMPCON 1993*, pages 528–537, 1993.

[9] 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.

[10] David Blackston and Torsten Suel. Highly portable and efficient implementations of parallel adaptive n-body methods. In *SC'97*, November 1997. online at http://www.supercomp.org /sc97/program/TECH/BLACKSTO/.

[11] 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.

[12] S. Chandra, J. Larus, and A. Rogers. Where is Time Spent in Message-Passing and Shared-Memory Programs. In *ASPLOS-94 Architectural Support for Programming Languages and Operating Systems*, 1994.

[13] F. Chong, R. Barua, F. Dahlgren, J. Kubiatowicz, and A. Agarwal. The Sensitivity of Communication Mechanisms to Bandwidth and Lantency. In *HPCA-4 High Performance Communication Architectures*, pages 37–46, February 1998.

[14] B. Christiansen, P. Cappello, M. F. Ionescu, M. O. Neary, K. E. Schauser, and D. Wu. Javelin: Internet-Based Parallel Computing Using Java. *Concurrency: Practice and Experience*, 1997. to appear.

[15] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1989.

[16] 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.

[17] MPI Forum. Mpi: A message passing interface standard. *Int. J. Supercomputer Applications*, 8(3/4), 1994. Version 1.1 at http://www.mcs.anl.gov/mpi/mpi-report-1.1/mpi-report.html.

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

[19] A. Grama, V. Kumar, and A. Sameh. Scalable parallel formulations of the barnes-hut algorithm for n-body simulations. In *Supercomputing '94*, November 1994.

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

[21] 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.

[22] D. Jiang, G Shan, and J. Singh. Application Restructuring and Performance Portability on Shared Virtual Memory and Hardware-Coherent Multiprocessors. In *PPoPP-97 Symposium on Principles and Practice of Parallel Programming*, June 1997.

[23] K. Johnson, F. Kaashoek, and D. Wallach. Crl: High-performance all-software distributed shared memory. In *Symposium on Operating Systems Principles 15*, pages 213–228, December 1995.

[24] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin Cummings, November 1993.

[25] K. Langendoen, R. Hofman, and H. Bal. Challenging applications on fast networks. In *HPCA-4 High-Performance Computer Architecture*, pages 125–137, February 1998.

[26] 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.

[27] F. Thomson Leighton. *Introduction to parallel algorithms and architectures*. Morgan Kaufmann, 1992.

[28] D. Lenoski, J. Laudon, K. Gharachorloo, W. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M.S. Lam. The Stanford Dash Multiprocessor. *IEEE Computer*, 25(3):63–79, March 1992.

[29] 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/.

[30] 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.

[31] L. Monnerat and R. Bianchini. Efficiently Adapting to Sharing Patterns in Software DSMs. In *HPCA-4 High Performance Communication Architectures*, pages 289–299, February 1998.

[32] T. Mowry, C. Chan, and A. Lo. Comparative Evaluation of Latency Tolerance Techniques for Software Distributed Shared Memory. In *HPCA-4 High Performance Communication Architectures*, pages 300–311, February 1998.

[33] B. J. Overeinder, P. M. A. Sloot, R. N. Heederik, and L. O. Hertzberger. A Dynamic Load Balancing System for Parallel Cluster Computing. *Future Generation Computer Systems*, 12:101–115, May 1996.

[34] 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.

[35] M. Philippsen and M. Zenger. JavaParty—Transparent Remote Objects in Java. In *ACM 1997 PPoPP Workshop on Java for Science and Engineering Computation*, June 1997.

[36] A. Reinefeld, R. Baraglia, T. Decker, J. Gehring, D. Laforenza, J. Simon, T. Rümke, and F. Ramme. The MOL Project: An Open Extensible Metacomputer. In *Heterogenous computing workshop HCW'97 at IPPS'97*, April 1997.

[37] John W. Romein and Henri E. Bal. Parallel n-body simulation on a large-scale homogeneous distributed system. In Seif Haridi, Khayri Ali, and Peter Magnusson, editors, *EURO-PAR'95 Parallel Processing, Lecture Notes in Computer Science, 966*, pages 473–484, Stockholm, Sweden, August 1995. Springer-Verlag.

[38] D. J. Scales, K. Gharachorloo, and A. Aggarwal. Fine-grain software distributed shared memory on SMP clusters. In *HPCA-4 High-Performance Computer Architecture*, pages 125–137, February 1998.

[39] J. Singh, C. Holt, T. Totsuka, A. Gupta, and J. Hennessy. Load balancing and data locality in adaptive hierarchical n-body methods: Barnes-hut, fast multipole and radiosity. *Journal of Parallel and Distributed Computing*, June 1995.

[40] E. Speight and J. Bennett. Using Multicast and Multithreading to Reduce Communication in Software DSM Systems. In *HPCA-4 High Performance Communication Architectures*, pages 312–323, February 1998.

[41] R. Stets, S Dwarkadas, N. Hardavellas, G. Hunt, L. Kontothanassis, S. Parthasarathy, and M. Scott. Cashmere-2L: Software coherent shared memory on a clustered remote-write network. In *Proc. 16th ACM Symp. on Oper. Systems Princ.*, October 1997.

[42] H. Topcuoglu and S. Hariri. A Global Computing Environment for Networked Resources. In *Proc. 1997 Int. Conf. on Parallel Processing*, pages 493–496, Bloomingdale, IL, August 1997.

[43] L. Valiant. A Bridging Model for Parallel Computation. *Comm. ACM*, 33(8):100–108, August 1990.

[44] M. Warren and J. Salmon. A parallel hashed oct-tree n-body algorithm. In *Supercomputing '93*, November 1993.

[45] Jon B. Weissman and Andrew S. Grimshaw. A Federated Model for Schedupling in Wide-Area Systems. In *5th International Symposium on High Performance Distributed Computing (HPDC-5)*, August 1996.

[46] 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.

[47] W. Yu and A. Cox. Java/DSM: A Platform for Heterogeneous Computing. In *ACM 1997 PPoPP Workshop on Java for Science and Engineering Computation*, June 1997.

[48] Y. Zhou, L. Iftode, J. Singh, K. Li, B. Toonen, I. Schoinas, M. Hill, and D. Wood. Relaxed Consistency and Coherence Granularity in DSM Systems: A Performance Evaluation. In *PPoPP-6 Symposium on Principles and Practice of Parallel Programming*, June 1997.