Parallel Computing on Wide-Area Clusters: the Albatross Project

Henri E. Bal Aske Plaat Thilo Kielmann
Jason Maassen Rob van Nieuwpoort Ronald Veldema
Department of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands
{bal,aske,kielmann,jason,rob,rveldema}@cs.vu.nl
http://www.cs.vu.nl/albatross/

Abstract

The aim of the Albatross project is to study applications and programming environments for wide-area cluster computers, which consist of multiple clusters connected by wide-area networks. Parallel processing on such systems is useful but challenging, given the large differences in latency and bandwidth between LANs and WANs. We apply application-level optimizations that exploit the hierarchical structure of wide-area clusters to minimize communication over the WANs. In addition, we use highly efficient local-area communication protocols. We illustrate this approach using a high-performance Java system that is implemented on a collection of four Myrinet-based clusters connected by wide-area ATM networks. The optimized applications obtain high speedups on this wide-area system.

1 Introduction

As cluster computers become more widely available, it becomes feasible to run parallel applications on multiple clusters at different geographic locations. By using several clusters for a single application, computationally challenging problems can be solved and a better usage may be made of the available resources. Wide-area cluster computing thus is a form of metacomputing [7, 14]. To enable wide-area cluster computing, however, many problems have to be solved. Foremost, a suitable software infrastructure has to be built, which deals with issues like security, heterogeneity, fault tolerance, and accounting. Legion and Globus are examples of such infrastructures [6, 8]. In addition, research is required on algorithms, applications, and programming environments for wide-area systems, since their performance model is quite different from local clusters.

The Distributed ASCI Supercomputer (DAS) is an experimental system that was built for doing research on wide-area cluster computing (see Figure 1). It consists of four Myrinet-based cluster computers located at four Dutch universities that participate in the ASCI research school.[1] This paper briefly describes one of the projects being done with the DAS system. The goal of this project, called *Albatross,* is to study applications and programming environments for wide-area cluster computers.

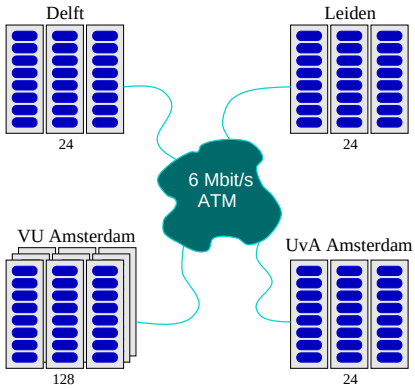

Figure 1: The wide-area DAS system.

An assumption in our project is that wide-area clusters will be structured *hierarchically* and will consist of local clusters connected by wide-area networks. Communication within a cluster is fast, typically with latencies of 1-100 microseconds. Wide-area communication is much slower, with millisecond latencies. The DAS system is one example of such a hierarchical system. Our algorithms and programming systems exploit this hierarchical structure by reducing the amount of communication over the wide-area links. This is similar to locality optimizations for NUMA machines, except that the performance gap between the local and wide-area network is much larger; with a NUMA, the gap typically is

[1] The ASCI research school is unrelated to, and came into existence before, the Accelerated Strategic Computing Initiative.

a factor of 3-5, whereas with wide-area clusters it often is orders of magnitude.

The optimized wide-area applications succeed in minimizing the communication traffic over the wide-area links [2, 10, 13, 15]. As a result, most communication of the programs is local, and it often also is important to optimize intra-cluster communication over the local area network. Our research therefore focuses on two issues:

- efficient communication protocols and runtime systems for local cluster computers, and
- efficient algorithms and programming environments for wide-area cluster computers.

The communication software we use is based on the Panda library [1]. Panda provides multithreading and communication primitives (point-to-point message passing, RPC, and broadcast) for implementing runtime systems of various parallel languages. The programming environments developed in the Albatross project are implemented on top of the Panda interface. Panda is implemented on Myrinet using LFC [3], a highly efficient, user-space communication substrate similar to active messages. For wide-area communication, Panda uses the TCP/IP protocol.

We have implemented several wide-area parallel programming systems on top of Panda. Orca is a parallel language that provides an object-based distributed shared memory model [1]. We have implemented Orca on the wide-area DAS system and we have successfully optimized several Orca applications [2, 13]. MagPIe [10] is an MPI library (based on MPICH [9]) whose collective communication primitives have been optimized for wide-area hierarchical systems. MPI applications that mainly use collective operations can be run efficiently on a wide-area system just by relinking with the MagPIe library. Finally, we are implementing a high-performance wide-area Java system, called Manta [11, 15].

In the rest of this paper, we will use the Manta system to illustrate the research issues addressed in the Albatross project. The advantages of using Java for wide-area parallel programming are its clean, object-oriented programming model, support for distributed polymorphism, security, and garbage collection [5, 17]. Below, we first describe the Manta system. Next, we discuss how we implemented Manta on the DAS and used it for wide-area cluster computing.

2 The Manta system

Manta is a high-performance Java system. Unlike most other Java implementations, it uses a native compiler that generates executable code rather than byte code. An important advantage of Manta is its highly efficient implementation of Remote Method Invocation. Manta's RMI model is similar to that of JavaParty, which is somewhat more flexible for parallel programming and easier to use than the original Java RMI model [12]. We use RMI for communication both within a cluster and between clusters, so its performance is crucial. Other RMI implementations (e.g., the Sun JDK 1.1) have a large software overhead, mainly due to slow serialization and communication protocols. With Manta, all serialization routines are generated by the compiler, so no runtime inspection (reflection) is used. Manta uses its own, light-weight RMI protocol, written in C. Finally, Manta is implemented on top of highly efficient communication layers (Panda and LFC), whereas other RMI implementations use TCP/IP. As a result, the null latency of Manta's RMI over Myrinet is less than 40 microseconds, a factor of 35 improvement over the JDK [11]. Manta obtains a throughput close to 40 Mbyte/sec over Myrinet.

The most difficult issue in the design of Manta is how to interoperate with other Java implementations (JVMs). To solve this problem, a Manta node can also communicate through a JDK-compliant RMI protocol. Thus, two Manta nodes communicate through Manta's own fast RMI protocol, while communication with non-Manta JVMs follows the standard protocol. A related problem is that Manta nodes must be able to exchange byte codes with other Java nodes, because RMIs in Java are polymorphic [16]. (The parameters or result value of an RMI may be of a subclass of the class specified in the declaration, and this subclass may not yet be present at the sending or receiving machine.) To support polymorphic RMIs, Manta is able to accept byte codes from JVMs; this byte code is compiled during runtime to object code, which is linked into the executable program using the `dlopen()` dynamic linking interface. A more detailed description of these techniques is given in [11].

3 Wide-area computing in Java

We have implemented Manta on the wide-area DAS system, to create an environment for experimental research on wide-area parallel programming. DAS consists of four clusters, located at four Dutch universities. The cluster at the Vrije Universiteit has 128 processors, the other clusters have 24 nodes each. Each node contains a 200 MHz Pentium Pro and has 64-128 MByte memory. The machines run RedHat Linux version 2.0.36. The nodes within each cluster are connected by Myrinet [4]. The four clusters are fully connected through dedicated 6 Mbit/sec wide-area ATM networks. The DAS system is described in more detail on *http://www.cs.vu.nl/das/*.

The Manta system on wide-area DAS uses one dedicated gateway machine per cluster. The gateways implement the Panda library, but (unlike normal nodes) support communication over both Myrinet and ATM, using LFC and TCP/IP respectively. Since the Manta RMI protocol is implemented on top of Panda, the RMI protocol does not have to be aware of the different underlying communication protocols. Applications often have to be optimized to take the hierarchical structure of the wide-area system into account, so Manta exposes this structure to the application.

The null latency of Manta RMI over the wide-area ATM network is at most 5.6 msec (between the clusters at VU Amsterdam and TU Delft). The measured throughput is 0.55 MByte/sec. The latency and throughput over ATM are roughly two orders of magnitude worse than those over Myrinet, making parallel processing a challenging task.

So far, we have implemented four parallel applications in Java and optimized them for wide-area systems. The applications are: Successive Overrelaxation (SOR), the All-pairs Shortest Paths problem (ASP), the Traveling Salesperson Problem (TSP), and Iterative Deepening A* (IDA*). The applications and the optimizations are described in [15]. The performance results (taken from [15]) are shown in Figure 2. The figure shows the speedups (relative to a sequential Java program) on 40 processors distributed over 4 clusters. For comparison, it also gives speedups on a single cluster of 10 or 40 nodes. Comparing the second and third bar for each application shows that the speedups on 4 ATM-connected clusters of 10 nodes are close to those on a 40-node Myrinet cluster. Hence, we succeeded in reducing the communication overhead over the wide-area network. Also, the figure shows that significant gains can be obtained by running the applications on multiple 10-node clusters instead of a single 10-node cluster.

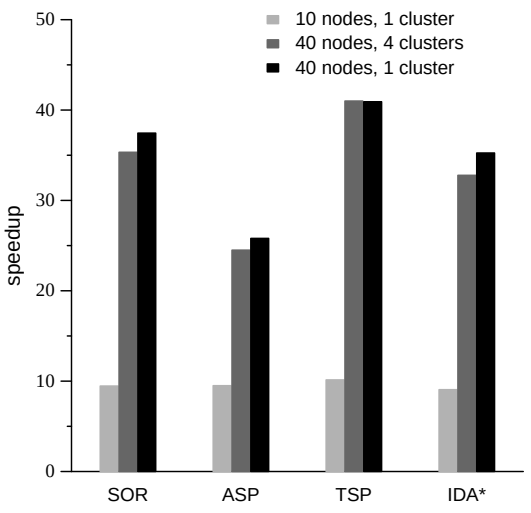

Figure 2: Speedups of four Java applications on three different cluster configurations.

To obtain this good performance, we had to optimize the applications in various ways. Several applications (SOR and ASP) require asynchronous communication, to overlap wide-area communication with computations. Java's RMI, however, is synchronous. To solve this problem, we invoke the wide-area RMIs from a separate thread, allowing the computation thread to continue. For local RMIs over Myrinet, thread-switching overhead outweighs the performance gains. We therefore only do the optimization for inter-cluster RMIs and not for intra-cluster RMIs, although this is awkward to express. Another limitation of RMI is the lack of a broadcast primitive. For performance reasons, ASP requires broadcasting, so we implemented broadcasting on top of RMI; again, this was awkward to express. For TSP and IDA*, the hardest problem was to find a work distribution scheme that minimized wide-area communication while still avoiding load imbalances. The schemes we used (job queues and work stealing) were easy to express in Java.

4 Conclusions

The work in the Albatross project so far has shown that it is feasible to efficiently run parallel applications on multiple cluster computers connected by wide-area networks. An important insight is to exploit the hierarchical structure of such wide-area clusters and minimize the amount of communication over the WAN (or overlap the communication with computation). For many parallel applications, the overhead of wide-area communication can be made sufficiently small. As a result, optimizing local-area communication also becomes important. In our research, we combine efficient local communication software with application-level wide-area optimizations. In this paper, we applied this strategy to Java. We have briefly described an efficient implementation of Java RMI and we have discussed optimizations for several applications. In other papers, we have described similar experiences with different programming systems and their applications [2, 10, 13]. Also, we have performed a sensitivity analysis [13] on a wide-area emulation system, showing that many optimized applications can even tolerate very high latencies and low bandwidths.

The next step in the Albatross project is to develop programming support that eases wide-area parallel programming. Many of the application-level optimizations we implemented for Java and other languages are complicated to express. For MPI, our MagPIe library already is an important step forward, as it hides the wide-area optimization inside a library. For example, a parallel ASP program written in MPI can be run unmodified on the wide-area DAS system and obtain excellent speedups [10]. A restriction of MagPIe, however, is that it is only effective for applications that are dominated by collective operations. In the near future, we will therefore also study other communication paradigms. Finally, the usage of Java for wide-area parallel computing is attractive, given Java's advantages for distributed programming.

Acknowledgements

This work is supported in part by a SION grant from the Dutch research council NWO, and by a USF grant from the Vrije Universiteit. The wide-area DAS system is an initiative of the Advanced School for Computing and Imaging (ASCI). We thank Raoul Bhoedjang, Rutger Hofman, Ceriel Jacobs, and Cees Verstoep for their contributions to this research. We thank Cees Verstoep and John Romein for keeping the DAS in good shape, and Cees de Laat and the University of Utrecht for getting the wide area links of the DAS up and running.

References

[1] 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):1–40, February 1998.

[2] H.E. Bal, A. Plaat, M.G. Bakker, P. Dozy, and R.F.H. Hofman. Optimizing Parallel Applications for Wide-Area Clusters. In *International Parallel Processing Symposium*, pages 784–790, Orlando, FL, April 1998.

[3] R. A. F. Bhoedjang, T. Rühl, and H. E. Bal. User-Level Network Interface Protocols. *IEEE Computer*, 31(11):53–60, November 1998.

[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] 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.

[6] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Int. Journal of Supercomputer Applications*, 11(2):115–128, Summer 1997.

[7] I. Foster and C. Kesselman, editors. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[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] William Gropp, Ewing Lusk, Nathan Doss, and Antony Skjellum. A High-performance, Portable Implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996.

[10] T. Kielmann, R.F.H. Hofman, H.E. Bal, A. Plaat, and R.A.F. Bhoedjang. MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Atlanta, GA, May 1999.

[11] J. Maassen, R. van Nieuwpoort, R. Veldema, H.E. Bal, and A. Plaat. An Efficient Implementation of Java's Remote Method Invocation. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Atlanta, GA, May 1999.

[12] M. Philippsen and M. Zenger. JavaParty—Transparent Remote Objects in Java. *Concurrency: Practice and Experience*, pages 1225–1242, November 1997.

[13] A. Plaat, H. Bal, and R. Hofman. Sensitivity of Parallel Applications to Large Differences in Bandwidth and Latency in Two-Layer Interconnects. In *Fifth International Symposium on High-Performance Computer Architecture*, pages 244–253, Orlando, FL, January 1999. IEEE CS.

[14] L. Smarr and C.E. Catlett. Metacomputing. *Communications of the ACM*, 35(6):44–52, June 1992.

[15] R. van Nieuwpoort, J. Maassen, H.E. Bal, T. Kielmann, and R. Veldema. Wide-Area Parallel Computing in Java. In *ACM 1999 Java Grande Conference*, Palo Alto, CA, June 1999.

[16] J. Waldo. Remote procedure calls and Java Remote Method Invocation. *IEEE Concurrency*, pages 5–7, July–September 1998.

[17] A. Wollrath, J. Waldo, and R. Riggs. Java-Centric Distributed Computing. *IEEE Micro*, 17(3):44–53, May/June 1997.