# The Albatross Project: Parallel Application Support for Computational Grids

Thilo Kielmann Henri E. Bal
Jason Maassen Rob van Nieuwpoort Ronald Veldema
Rutger Hofman Ceriel Jacobs Kees Verstoep
Department of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands
{kielmann,bal,jason,rob,rveldema,rutger,ceriel,versto}@cs.vu.nl
http://www.cs.vu.nl/albatross/

**Abstract**

The aim of the Albatross project is to study applications and programming environments for computational grids consisting of multiple clusters that are 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 provide efficient algorithms and programming environments 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 the Manta high-performance Java system and the MagPIe MPI library, both of which are implemented on a collection of four Myrinet-based clusters connected by wide-area ATM networks. Our sample applications obtain high speedups on this wide-area system.

## 1 Introduction

As computational grids 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 the available resources can be used more efficiently. Wide-area cluster computing thus is a form of metacomputing [6, 10]. 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, Globus, and SNIPE are examples of such infrastructures [8, 9, 11]. In addition, research is required on algorithms, applications, and programming environments for wide-area systems, since their performance model is quite different from models for 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

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

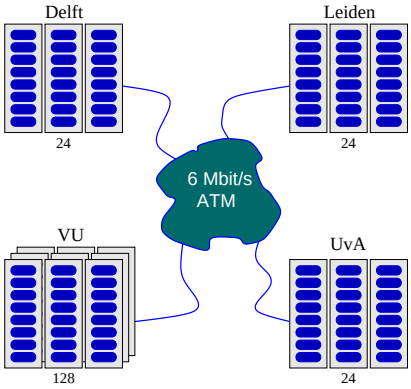



Figure 1: The wide-area DAS system.

goal of this project, called *Albatross*, is to study applications and programming environments for wide-area cluster computers.

An assumption in our project is that computational grids 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 machine, the gap typically is a factor of 3–5, whereas with wide-area clusters it often is in orders of magnitude.

Our optimized wide-area applications succeed in minimizing the communication traffic over the wide-area links [2, 15, 20, 21, 22]. As a result, most communication of the programs is local. Thus, it 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. Panda is implemented on Myrinet using LFC [4], a highly efficient, user-space communication substrate similar to Active Messages. For wide-area communication, Panda uses the TCP/IP protocol.

In the Albatross project, we have implemented several wide-area parallel programming systems on top of Panda. Figure 2 shows the structure of our Panda-based wide-area software. 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, 20]. In order to provide the MPI message passing standard, we have ported MPICH [12] to Panda. Our MagPIe [15] library optimizes MPI's collective communication operations for wide-area hierarchical systems. MPI applications that mainly use col-

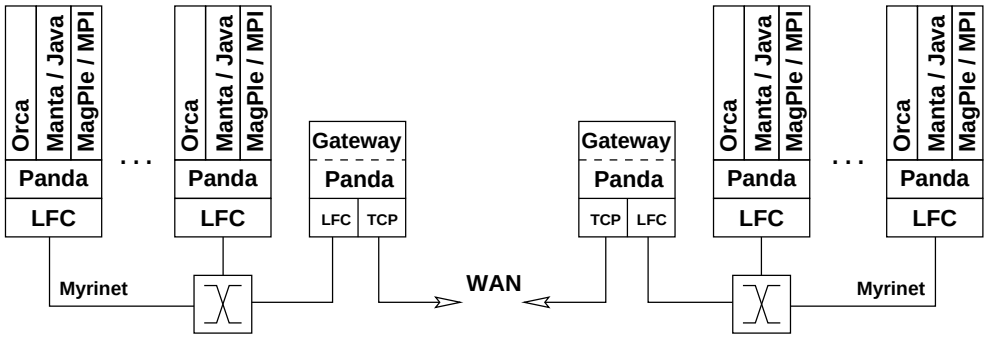



Figure 2: Local and wide-area communication based on Panda

lective operations can be run efficiently on a wide-area system just by relinking with the MagPIe library. Finally, we have implemented a high-performance wide-area Java system, called Manta [17, 21, 22]. 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 [7, 24]. In a previous status report, we focused on our Manta system [3].

In this paper, we will first use the Manta system to illustrate the research issues addressed in the Albatross project. In Section 2, we describe the Manta system and its use for wide-area cluster computing. In Section 3, we present the current state of our MagPIe library that allows existing applications written in C and in Fortran using the MPI message passing interface [18] to efficiently run on wide-area clusters.

# 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 implements both Sun's original RMI model and the JavaParty model [19] which is somewhat more flexible and easier to use for parallel programming. 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) have large software overheads, mainly due to slow serialization and communication protocols. With Manta, all serialization routines are generated by the compiler, so no runtime inspection (reflection) is necessary. 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, 35 times faster than the JDK [17]. Manta obtains a throughput close to 50 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 [23]. (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 [17].

## 2.1 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 5.2.The nodes within each cluster are connected by Myrinet [5]. The four clusters are fully connected through dedicated 6 Mbit/sec wide-area ATM links (cf. Fig. 1). 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 (see Fig. 2). 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. Manta exposes the hierarchical structure of the wide-area system to the application in order to allow application-level optimizations.

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.

## 2.2 Application experience

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 [21, 22]. The performance results (taken from [22]) are shown in Figure 3. For each application, we implemented two versions: one targeted at single-cluster systems (called "non-optimized" below), and one specially optimized for multiple clusters. The figure shows the speedups (relative to a sequential Java program) from left to right, on a single cluster of 16 nodes, on four clusters with 16 nodes each non-optimized and optimized, and on a single cluster of 64 nodes. A comparison of the third and fourth bar for each application (except for ASP) shows that the speedups on 4 ATM-connected clusters of 16 nodes are close to those on a 64-node Myrinet cluster. With ASP, the optimization still improved the speedup substantially. As Figure 3 shows, we succeeded in reducing the impact of communication overhead over the wide-area network. The figure also shows that significant gains can be obtained by running the applications on multiple 16-node clusters instead on a single 16-node cluster.

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 had to invoke the wide-area RMIs from a separate thread, allowing the computation

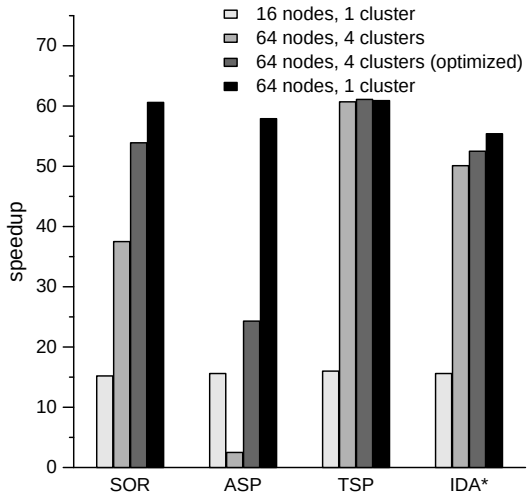



Figure 3: Speedups of four Java applications on different cluster configurations.

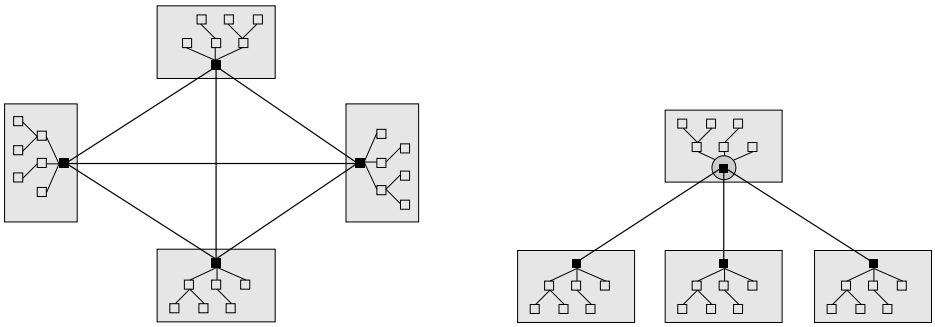
thread to continue. For local RMIs over Myrinet, thread-switching overhead outweighs the performance gains. We therefore only optimized the inter-cluster RMIs and not the intra-cluster RMIs, although this was awkward to express in the program. Another limitation of RMI is the lack of a broadcast primitive. 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.

# 3 The MagPIe library

The collective operations as defined by the MPI standard [18] describe an important set of communication patterns occuring between groups of processes. Frequently used examples are the broadcast, barrier, and reduce operations. Our MagPIe library [15, 16] implements MPI's collective operations with optimizations for wide area systems (grids). Existing parallel MPI applications can be run on grid platforms using MagPIe by relinking the programs with our library. No change in application code is necessary. MagPIe is independent of the underlying MPI platform. Its source code can be downloaded from our project WWW site.

MagPIe's basic idea is to adapt MPI's collective algorithms to the hierarchical shape of grid-based systems. Our hierarchical collective algorithms speed up collective completion time by reducing the utilization of the slow wide-area links to the necessary minimum. MagPIe has a simple API through which the underlying grid computing platform (Panda, in our case) provides the information about the number of clusters in use, and which process is located in which cluster.

MagPIe deals with two basic cases, namely *asymmetrical operations* having a dedicated *root*



(a) symmetrical operation (b) asymmetrical operation

Figure 4: MagPIe's wide-area optimized communication graphs

process, and *symmetrical operations* without a root. Broadcast is an example of an asymmetrical operation, whereas barrier is symmetrical. Figure 4 shows MagPIe's communication graphs for both cases. In each cluster, a so-called *coordinator* node is identified that communicates with its cluster-local peer processes, as well as with the other coordinators. In the case of asymmetrical operations, the root process acts as coordinator of its cluster. MagPIe ensures that each sender-receiver path contains at most one wide-area link and that each data item is sent at most once to each receiving cluster. This is achieved by restricting wide-area communication to coordinator nodes which are communicating using flat tree/graph shapes. Inside clusters, MagPIe uses binomial trees to and from the coordinator nodes. We have shown in [15, 16] that MagPIe significantly reduces the completion times of individual collective operations as well as that of parallel applications, compared to grid-unaware collective algorithms. Actual performance improvements depend on the number of clusters and on WAN latency/bandwidth. In our experiments, MagPIe's operations completed up to 8 times faster than the ones implemented by MPICH [12].

MagPIe's algorithms as described so far work quite well with short and medium sized messages. However, with long messages, utilization of the available wide-area bandwidth needs to be improved. The reason is that while sending a long message over a low-bandwidth link, the other wide-area links are idle. A solution to this problem is to split long messages into small segments which can be sent in parallel over multiple wide-area links. We have shown in [13] that this technique significantly improves MagPIe's algorithms.

Optimizing message segment sizes and communication tree shapes for collective operations needs detailed performance data for sending and receiving individual messages. In order to collect this data, we have developed the *MPI LogP Benchmark* [14]. Its source code is also available from our project WWW site.

# 4 Conclusions

The work on the Albatross project has shown that it is feasible to run parallel applications efficiently on multiple cluster computers connected by wide-area networks as they are found in computational grids. It is important to exploit the hierarchical structure of such wide-area clusters and to minimize the amount of communication over the WAN (or to overlap the commu-

nication 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 first applied this strategy to Java. We briefly described an efficient implementation of Java RMI and we discussed optimizations for several applications. In other papers, we described similar experiences with different programming systems and their applications [2, 15, 20]. Also, we performed a sensitivity analysis [20] on a wide-area emulation system, showing that many optimized applications can even tolerate very high latencies and low bandwidths.

Our current work in the Albatross project continues the development of programming support that eases wide-area parallel programming. However, 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. We currently investigate the adaptation of MagPIe to dynamically changing performance data [25]. We furthermore study the integration of MPI's collective operations into Java's object-oriented model. Finally, we are extending our work to other communication paradigms.

# Acknowledgements

This work is supported in part 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 and Aske Plaat for their contributions to this research, and Sören Johnson for his helpful comments on an earlier version of this report. We thank 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, 1998.

[2] Henri E. Bal, Aske Plaat, Mirjam G. Bakker, Peter Dozy, and Rutger F.H. Hofman. Optimizing Parallel Applications for Wide-Area Clusters. In *Proc. 12th International Parallel Processing Symposium IPPS'98*, pages 784–790, 1998.

[3] Henri E. Bal, Aske Plaat, Thilo Kielmann, Jason Maassen, Rob van Nieuwpoort, and Ronald Veldema. Parallel Computing on Wide-Area Clusters: the Albatross Project. In *Extreme Linux Workshop*, Monterey, CA, June 1999.

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

[5] 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, 1995.

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

[7] 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.

[8] Graham E. Fagg, Keith Moore, Jack J. Dongarra, and Al Geist. Scalable Network Information Processing Environment (SNIPE). In *SC'97*, November 1997. Online at http://www.supercomp.org/sc97/proceedings/.

[9] Ian Foster and Carl Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Int. Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[10] Ian Foster and Carl Kesselman, editors. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[11] Andrew S. Grimshaw, Wm. A. Wulf, and the Legion team. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40(1), January 1997.

[12] 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.

[13] Thilo Kielmann, Henri E. Bal, and Sergei Gorlatch. Bandwidth-efficient Collective Communication for Clustered Wide Area Systems. In *Proc. International Parallel and Distributed Processing Symposium (IPDPS 2000)*, Cancun, Mexico, May 2000.

[14] Thilo Kielmann, Henri E. Bal, and Kees Verstoep. Fast Measurement of LogP Parameters for Message Passing Platforms. In *4th Workshop on Runtime Systems for Parallel Programming (RTSPP)*, Cancun, Mexico, May 2000.

[15] Thilo Kielmann, Rutger F. H. Hofman, Henri E. Bal, Aske Plaat, and Raoul A. F. Bhoedjang. MAGPIE: MPI's Collective Communication Operations for Clustered Wide Area Systems. In *Proc. Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 131–140, Atlanta, GA, May 1999.

[16] Thilo Kielmann, Rutger F. H. Hofman, Henri E. Bal, Aske Plaat, and Raoul A. F. Bhoedjang. MPI's Reduction Operations in Clustered Wide Area Systems. In *Proc. MPIDC'99, Message Passing Interface Developer's and User's Conference*, pages 43–52, Atlanta, GA, March 1999.

[17] Jason Maassen, Rob van Nieuwpoort, Ronald Veldema, Henri E. Bal, and Aske Plaat. An Efficient Implementation of Java's Remote Method Invocation. In *Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'99)*, pages 173–182, Atlanta, GA, May 1999.

[18] Message Passing Interface Forum. MPI: A Message Passing Interface Standard. *International Journal of Supercomputing Applications*, 8(3/4), 1994.

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

[20] Aske Plaat, Henri E. Bal, and Rutger F. H. Hofman. Sensitivity of Parallel Applications to Large Differences in Bandwidth and Latency in Two-Layer Interconnects. In *High Performance Computer Architecture HPCA-5*, pages 244–253, Orlando, FL, January 1999.

[21] Rob van Nieuwpoort, Jason Maassen, Henri E. Bal, Thilo Kielmann, and Ronald Veldema. Wide-area parallel computing in Java. In *ACM 1999 Java Grande Conference*, pages 8–14, San Francisco, CA, June 1999.

[22] Rob van Nieuwpoort, Jason Maassen, Henri E. Bal, Thilo Kielmann, and Ronald Veldema. Wide-Area Parallel Programming using the Remote Method Invocation Model. *Concurrency: Practice and Experience*, 2000. Java Grande Special Issue.

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

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

[25] Rich Wolski. Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service. In *Proc. High-Performance Distributed Computing (HPDC-6)*, pages 316–325, Portland, OR, August 1997. The network weather service is at http://nws.npaci.edu/.