Bandwidth-efficient Collective Communication for Clustered Wide Area Systems

Thilo Kielmann, Henri E. Bal
Dept. of Mathematics and Computer Science
Vrije Universiteit, Amsterdam, The Netherlands
kielmann@cs.vu.nl, bal@cs.vu.nl

Sergei Gorlatch
Dept. of Mathematics and Computer Science
University of Passau, Germany
gorlatch@fmi.uni-passau.de

Abstract

Metacomputing infrastructures couple multiple clusters (or MPPs) via wide-area networks. A major problem in programming parallel applications for such platforms is their hierarchical network structure: latency and bandwidth of WANs often are orders of magnitude worse than those of local networks. Our goal is to optimize MPI's collective operations for such platforms. In this paper, we focus on optimized utilization of the (scarce) wide-area bandwidth. We use two techniques: selecting suitable communication graph shapes, and splitting messages into multiple segments that are sent in parallel over different WAN links. To determine the best graph shape and segment size, we introduce a performance model called parameterized LogP *(P-LogP), a hierarchical extension of the LogP model that covers messages of arbitrary length. With* P-LogP*, the optimal segment size and the best broadcast tree shape can be determined at runtime. (For conciseness, we restrict our discussion to the broadcast operation.) An experimental performance evaluation shows that the new broadcast has significantly improved performance (for large messages) and that there is a close match between the theoretical model and the measured completion times.*

1 Introduction

Research on global computational infrastructures has raised considerable interest in running parallel applications on wide-area distributed systems, often called metacomputers or computational grids [8, 12]. Writing such applications is much more difficult than programming traditional parallel machines due to the presence of different (local and wide-area) networks. As the wide-area links are orders of magnitude slower than the interconnects within clusters (or MPPs), metacomputers have a *hierarchical* structure.

In earlier work [23, 24], we discussed how collective communication libraries can be used to simplify wide-area parallel programming. We implemented a MPI-compatible library, called MagPIe, which optimizes MPI's collective operations for wide-area systems. MagPIe exploits the hierarchical structure, resulting in much less wide-area communication than other MPI libraries [24]. MagPIe's existing implementation efficiently deals with the high WAN latency but has less than optimal performance with long messages which are more sensitive to WAN bandwidth.

Collective communication with long messages needs a more detailed modeling, including latency and bandwidth of the local and wide-area networks, the number of clusters, the number of processors in each cluster, and the length of the messages. Therefore, we introduce a new performance model for wide-area collective communication, called the *parameterized LogP* model (*P-LogP*), which extends the LogP model [9]. We apply this model to optimizing collective communication using message segmentation and tree shape determination. Due to space limitations, we restrict ourselves to the discussion of MPI's broadcast operation. We further describe heuristics with which the optimizations can be performed dynamically (at runtime), based on measured model parameters of the computing platform.

We currently make several simplifying assumptions about the networks, as we use regular topologies and constant latencies and bandwidths. We have evaluated the model and the optimizations on an experimental wide-area testbed for which these assumptions are reasonable. Our ultimate goal, however, is to develop a MPI library that does not have these limitations and that adapts dynamically to changing network conditions. We intend to use our performance model in combination with dynamic information [28] for selecting the optimal algorithm at any point in time. To allow such runtime decisions, our optimization algorithms themselves also are efficient and are executed on-line, as part of the MagPIe library.

The paper is organized as follows. In Section 2, we introduce our *parameterized LogP* model. In Section 3, we apply it to optimize MPI's broadcast operation in hierarchical systems. Related work will be briefly discussed in Section 4. Finally, Section 5 presents our conclusions.

2 Performance modeling

To motivate our performance model, we first analyze the communication behavior of the MPI implementation on our experimentation platform, called the DAS system. DAS consists of four cluster computers, each containing Pentium Pros that are locally connected by Myrinet [6]. The clusters are located at four Dutch universities and are connected by dedicated 6 Mbit/s ATM networks (using a fully connected graph). The system is described in more detail in [24] (and on *http://www.cs.vu.nl/das/*). Figure 1 illustrates the general system structure we assume throughout this paper, consisting of multiple clusters with fully connected local networks and a fully connected WAN. Each cluster has a gateway that is connected both to the LAN and to the WAN.

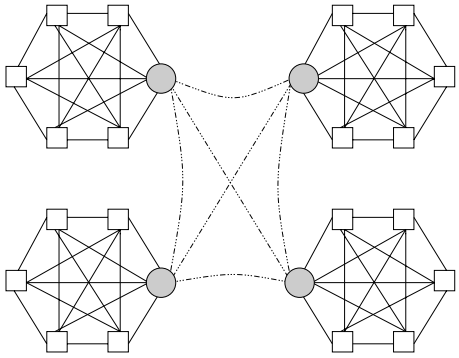

Figure 1. Structure of a clustered wide area system. The circles represent gateways.

We have ported MPICH [18], a widely used public MPI implementation, to the wide-area DAS system. Our MPICH port uses the Panda communication substrate [2] and implements a Panda-specific MPICH device. Panda gives access to IP and Myrinet. On Myrinet, Panda uses the LFC [5] control program. One of the DAS clusters has 128 CPUs, and has been set up to allow easy experimentation with different WAN latencies and bandwidths, by adding delay loops to the networking subsystem [24]. This wide-area emulation is part of Panda and thus is transparent to software layers on top of it, like MPI. We use this emulation system for the performance experiments reported in this paper. We use a varying number of clusters, a wide-area latency of 10 ms, and a wide-area bandwidth of 1 MB/sec. In comparison, the latency over Myrinet is about $20\mu s$ and the (MPI-level) throughput is about 50 MB/sec, so there is almost two orders of magnitude difference between the local and wide-area network of DAS. We emphasize that all our results have been measured on a real parallel machine, using the same software on the compute nodes as in the real wide-area system. Only the wide-area links are simulated over some of the Myrinet links. To achieve this, the software on the gateway uses a different option of Panda. Using the WAN emulator has the additional benefit of highly reproducible results.

Our MagPIe library uses point-to-point messages as provided by the MPI standard. Figure 2 shows *send overhead* and *gap*, measured for the *MPI_Isend* routine on both networks, Myrinet and WAN. In analogy to LogP, the *send overhead* o is the completion time of *MPI_Isend* for a given message size, while the *gap* g is the minimum time interval between consecutive calls to *MPI_Isend*. We measured those parameters with the method described in [22].

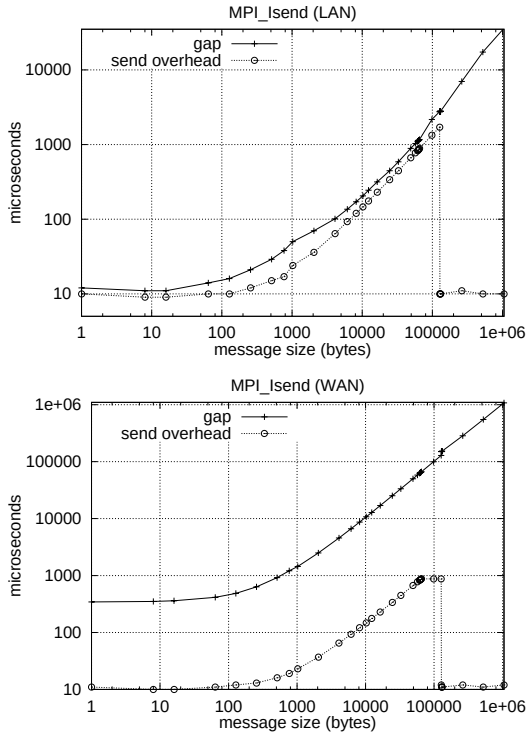

Figure 2. Send overhead and gap for MPI_Isend, measured on the Myrinet LAN (top) and an emulated WAN (bottom)

The MPI standard prescribes that the non-blocking *MPI_Isend* only initiates the send operation. The actual implementation is free to perform as much of the sending task as is convenient, as long as it guarantees that it will never wait for the receiving process. Our MPICH port to Panda sends until it reaches a point at which it would have to block, and then it returns. (The application can later check whether the transfer has actually been completed.) For short messages, this usually means that the entire message has been sent when *MPI_Isend* returns. In fact, Figure 2 shows that

over Myrinet, *MPI_Isend* behaves exactly like this up to a message size of 128 KB. At this size, MPICH switches to rendezvous mode; as *MPI_Isend* would have to wait for a reply message from the receiver, it returns immediately in this case. Using a slow wide-area network instead of the fast Myrinet, the behavior is similar, except that between 64 KB and 128 KB, *MPI_Isend* returns as soon as it would block waiting for flow-control information from the receiver. The corresponding gap values are as expected. They start rather "flat" for short messages and get into a linear increase for sufficiently large messages. On the WAN, there is another non-linearity at 128 KB when the send mode changes.

We can draw several conclusions from these measurements, which are useful for developing a realistic performance model. On Myrinet, the difference between overhead o and gap g is small (up to 128 KB), indicating that the end-to-end bandwidth is limited by the host computers and not by the network. On the wide-area link, on the other hand, the difference is two orders of magnitude for small messages. With this huge difference, LogP's assumption that a sender can not transmit a message faster than g time units of a preceding message is much too pessimistic and yields misleading results for collective communication. This assumption is only true if the next message follows the same network links. With collective operations, a sender typically sends messages to different destinations in a row, so the next send can already start after o time units, but not earlier than g time units as constrained by the local area network. Another important observation from Figure 2 is that o and g depend not only on message size and network bandwidth, but also on the underlying MPI implementation. Collective operations thus have to be optimized carefully. Neither the assumption that *MPI_Isend* will always return "immediately" nor linear approximations for o and g in general give realistic performance models.

2.1 Parameterized LogP

Based on these observations, we now present the *parameterized LogP* model (*P-LogP*). For a single-layer network, it defines five parameters. P is the number of processors. L is the end-to-end latency from process to process, combining all contributing factors such as copying data to and from network interfaces and the transfer over the physical network. $os(m)$, $or(m)$, and $g(m)$ are send overhead, receive overhead, and gap. They are defined as functions of the message size m. $os(m)$ and $or(m)$ are the times the CPUs on both sides are busy sending and receiving a message of size m. For sufficiently long messages, receiving may already start while the sender is still busy, so os and or may overlap. The gap $g(m)$ is the minimum time interval between consecutive message transmissions or receptions. It is the reciprocal value of the end-to-end bandwidth from process to process for messages of a given size m. Like L, $g(m)$ covers all contributing factors. From $g(m)$ covering $os(m)$ and $or(m)$, follows $g(m) \geq os(m)$ and $g(m) \geq or(m)$. A network N is characterized as $N = (L, os, or, g, P)$.

To illustrate how the parameters are used, we introduce $s(m)$ and $r(m)$, the times for sending and receiving a message of size m when both sender and receiver simultaneously start their operations. $s(m) = g(m)$ is the time at which the sender is ready to send the next message. Whenever the network itself is the transmission bottleneck, $os(m) < g(m)$, and the sender may continue computing after $os(m)$ time. But because $g(m)$ models the time a message "occupies" the network, the next message cannot be sent before $g(m)$. $r(m) = L + g(m)$ is the time at which the receiver has received the message. The latency L can be seen as the time it takes for the first bit of a message to travel from sender to receiver. The message gap adds the time after the first bit has been received until the last bit of the message has been received. Figure 3 illustrates this modeling. When a sender transmits several messages in a row, the latency will contribute only once to the receiver completion time but the gap values of all messages sum up. This can be expressed as $r(m_1, \ldots, m_n) = L + g(m_1) + \ldots + g(m_n)$.

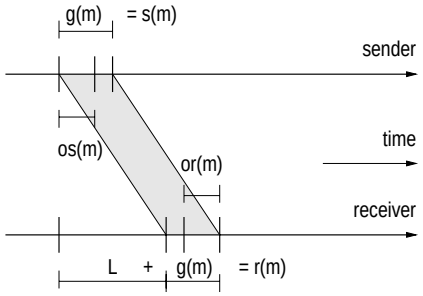

Figure 3. Message transmission as modeled by parameterized LogP

For clustered wide area systems, we use two parameter sets, identified by a subscript l for the LAN and w for the WAN. For example, L_l denotes the latency within a cluster and L_w is the latency when the sender and receiver are in different clusters. For a local network we get:

$$\begin{aligned} s_l(m) &= g_l(m) \\ r_l(m) &= L_l + g_l(m) \end{aligned}$$

A wide area transmission takes three steps: the sender forwards the message to its local gateway, which in turn sends the message to the gateway of the receiver's cluster, which finally forwards the message to the receiving node. r_w always depends on the wide area bandwidth and can be

expressed in analogy to r_l. s_w may either be determined by wide-area overhead or local-area gap, whichever is higher. This gives us the following equations for the wide area case:

$$\begin{aligned} s_w(m) &= \max(g_l(m), os_w(m)) \\ r_w(m) &= L_w + g_w(m) \end{aligned}$$

3 Broadcast

We now illustrate how the new P-LogP model can be used to optimize collective communication. We focus on one important example, broadcast. With broadcast, a single process (called the *root*) sends a message of size M to all other $(P-1)$ processes. Optimal broadcast algorithms use tree shaped communication graphs, so every process receives the message exactly once [20]. MagPIe's original broadcast algorithm is optimized for wide-area networks by sending the message only once to each cluster and by avoiding transmission paths that contain more than one wide-area link.

In MagPIe's algorithm, a so-called *coordinator* node is identified in each cluster. The root process acts as coordinator of its own cluster. First, the root sends the message to the coordinator nodes, forming a flat tree in the WAN. As soon as a coordinator receives the message, it forwards it to the other nodes of its cluster, using a binomial tree shape in the LANs.

A disadvantage of MagPIe's original algorithm is that it forwards complete messages down the spanning tree [24]. For large messages, this leads to poor link utilization. As large messages have a high send overhead, the root can only send over one WAN link at a time. Our goal for the optimized algorithms is to use the accumulated bandwidth of several (or all) available WAN links. With a given LAN bandwidth, the root may in theory simultaneously send over up to $n = g_w(m)/g_l(m)$ WAN links. We use message segmentation to achieve such a better link utilization. Instead of forwarding complete messages down the spanning tree, we split each message into k *segments* of size m (where $k = \lceil M/m \rceil \geq 1$) and forward each incoming segment down all links. In this way, there will be much more overlap in communication over different links. In addition to message segmentation, we optimize the shape of the spanning tree by trading off send overhead for latency.

We try to minimize the total completion time of the broadcast (i.e., the time when the last receiver has obtained the complete message). This optimization problem can be stated more accurately as follows:

> Given a network N and the message size M, our goal is to find a tree shape and a segment size m that together minimize the completion time.

We approach this problem in two steps. First, we develop a *single-layer* broadcast algorithm and its performance model, assuming that all links have the same speed. This algorithm can be used to optimize either communication within a cluster (i.e., all links are fast local networks) or communication between nodes in different clusters (i.e., all links are slow wide-area networks). Next, we develop a two-layer algorithm for the hierarchical model of metacomputers described in the introduction. Two-layer algorithms are more complicated to model, but are more efficient for broadcast operations.

3.1 Single-layer broadcast

The optimal tree shape depends on the network parameters, as well as on M and m. In general, we characterize a tree by two parameters, its height h and its degree d. h is the longest path from the root to any of the other nodes, determined by counting edges. d is the maximum number of successor nodes of any node in the graph. For example, MagPIe's flat WAN tree has $d = P - 1$ and $h = 1$, where P in this case denotes P_w, the number of clusters. Depending on the actual network parameters, the optimal tree shape (yielding the minimal completion time) can have any $d, h \in [1 \ldots P-1]$. Fortunately, d and h of an optimal tree are related to each other, and we can compute the minimal height of a tree with P nodes and degree d as $\min h : \sum_{i=0}^{h} d^i \geq P$.

Similar to the parameters for latency L and gap g for a single message send, we define latency λ and gap γ of a broadcast tree. Here, λ denotes the time at which a message segment has been received by *all* nodes, after the root process started sending it. γ is the time interval between the sending of two consecutive segments. (γ hence indicates the throughput of a broadcast tree.) We compute the completion time T of a broadcast algorithm with message segmentation as:

$$T = (k-1) \cdot \gamma + \lambda$$

To illustrate this formula, we can express the values for γ and λ for MagPIe's flat WAN tree as:

$$\begin{aligned} \gamma &= \max(g(m), (P-1) \cdot s(m)) \\ \lambda &= (P-2) \cdot s(m) + r(m) \end{aligned}$$

Here, γ is the maximum of the gap between two segments of size m sent on the same link and the time the root needs for sending $(P-1)$ times the same segment on disjoint links. The corresponding value for λ is the time at which a message segment is sent to the last node, plus the time until it is received.

For a general tree shape, both parameters can be expressed depending on the degree d and height h of a broadcast tree:

$$\begin{aligned}\gamma &\leq \max\left(g(m), or(m) + d \cdot s(m)\right)\\ \lambda &\leq h \cdot \left((d-1) \cdot s(m) + r(m)\right)\end{aligned}$$

γ is the maximum of the gap caused by the network, and the time a node needs to process the message. For intermediate nodes, this is the time to receive the message plus the time to forward it to d successor nodes. (For the root and for leaf nodes, it is either one of both.) The exact value of λ depends on the order in which the root process and all intermediate nodes send to their successor nodes and which path leads to the node that receives the message last. For the rest of this paper, we approximate γ and λ by their upper bounds, given by the inequalities.

The optimal broadcast tree depends on the number of processors P and the message size M. Both values may change dynamically. This implies that the optimal broadcast tree and segment size have to be computed at runtime for each invocation of *MPI_Bcast*. Optimization at runtime has to be very fast, so it does not outweigh the performance improvement of applying the optimized algorithm. Therefore, we avoid communication between processes and we avoid doing an exhaustive search over the complete search space with $m \in [1 \ldots M]$ and $d \in [1 \ldots P-1]$. Communication is avoided by replicating the network performance information over all application processes. When calling *MPI_Bcast*, all processes simultaneously compute the optimal tree and segment size.

We apply heuristics to reduce the number of segment sizes and tree shapes to be investigated. In general, several segment sizes are tried out, and for each size the optimal tree shape is computed. For a given segment size m, we investigate only the following tree shapes:

1. $d \in [g(m)/s(m) \ldots P-1]$. Lower values for d can only lead to higher completion times, because less accumulated bandwidth would be used.
2. Starting with $d = g(m)/s(m)$, we increase d while only investigating those values of d for which the height h will be reduced. Values for d in between can not improve completion time.

For the segment size, we only evaluate "useful" values. The segment size must be a multiple of the size of the basic data type to be transmitted (in MPI terms, the extent) and it must split the initial message into k *equal* segments. The segment size m is determined in two steps:

1. In a binary search, segment sizes $m = M/2^i, i \in [0 \ldots \log_2 M]$ are investigated.
2. Starting from the best value m' found in step 1, a local hill-climbing strategy is performed. $k' = \lceil M/m' \rceil$ is the so-far best known number of segments. The completion times are computed for $k'-5$, $k'-1$, $k'+1$, and $k'+5$. We then replace k' by the value with the best completion time. This is performed until no further improvement can be found. The simultaneous look-ahead of 1 and 5 steps helps overcoming local minima. Furthermore, it speeds up the process when a minimum is far away from the starting point computed in step 1.

Having identical links inside a network, the actual broadcast tree can be constructed from d in $O(P)$ time by keeping track of each node's degree while assigning receivers to senders.

3.2 Two-layer broadcast

So far, we have optimized broadcast for single-layer networks. Given optimized broadcast trees for the WAN and for the LANs, a simple way to obtain a two-layer algorithm is the sequential composition, as performed in the original MagPIe library. Here, the coordinator nodes first participate in the wide-area broadcast. Then, they forward the message inside their clusters. This leads to a total completion time of:

$$\begin{aligned}T &= T_w + T_l\\ &= (k_w - 1) \cdot \gamma_w + \lambda_w + (k_l - 1) \cdot \gamma_l + \lambda_l\end{aligned}$$

A better integration into a two-layer broadcast can be achieved by *pipelining* both phases. Here, the coordinator nodes immediately forward segments, first to other coordinators, and then to their successor nodes in the LAN tree structure. We use the same segment size for the WAN and for the LANs, so coordinators do not need to re-assemble segments. This leads to:

$$T = (k-1) \cdot \gamma + \lambda_w + \lambda_l$$

To reflect the double forwarding in the coordinator nodes, we use:

$$\gamma = \max\left(g_w(m), or_w(m) + d_w \cdot s_w(m) + d_l \cdot s_l(m)\right)$$

The optimization of the two-layer broadcast is almost as simple as in the single-layer case. The only addition is the optimization of λ_l for given m and γ' which denotes the fraction of γ that can be used for the LAN without slowing down the WAN. We then only have to investigate the LAN trees with $d \in [1 \ldots \gamma'/s_l(m)]$ with:

$$\gamma' = \max\left(g_w(m) - or_w(m) - d_w \cdot s_w(m), s_l(m)\right)$$

3.3 Performance evaluation

We have implemented a two-layer broadcast as described so far as part of the MagPIe library. Figure 4 compares the completion times of this new algorithm with MagPIe's original broadcast, measured on the DAS experimentation system described in Section 2. The new algorithm does message segmentation, and pipelines WAN and LAN forwarding. The original algorithm sends complete messages, and sequentially combines WAN and LAN forwarding. We measured configurations with 4 clusters (with 1 or 16 CPUs) and 8 clusters (with 1 or 8 CPUs).

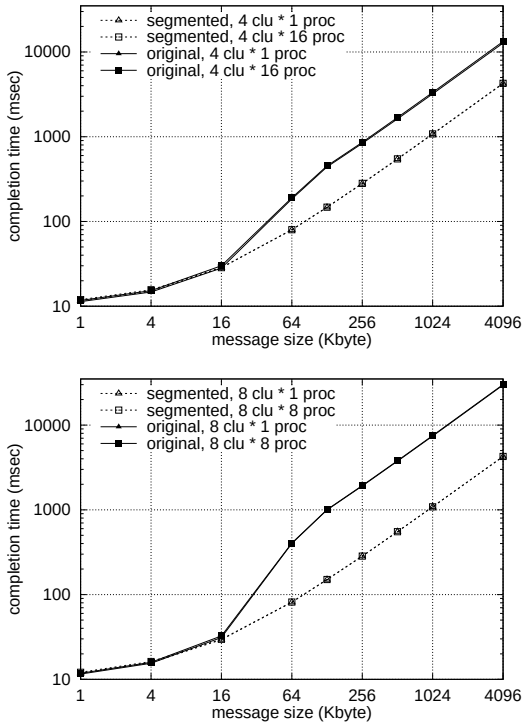

Figure 4. Completion time of MPI_Bcast using 4 clusters (top) and 8 clusters (bottom)

The number of CPUs per cluster has hardly any effect on the overall completion time, causing the respective pairs of lines in Figure 4 to be almost the same. This, and the fact that for short messages the original algorithm performs approximately as fast as the segmented broadcast, both support our original design goals for MagPIe. For larger messages, however, the new algorithm is much faster than the original one and achieves much higher aggregate wide-area bandwidth. The completion times for 4 and 8 clusters are about the same, whereas the original algorithm takes much longer for 8 clusters than for 4 clusters.

Although hard to see from the graphs, the pipelined message forwarding further reduces the overall completion time. With the original algorithm, the sequential combination of WAN and LAN broadcast adds some additional time. With 4 MB messages, for example, the forwarding takes $13270.4 - 12723.4 = 547$ ms. This overhead disappears with pipelining.

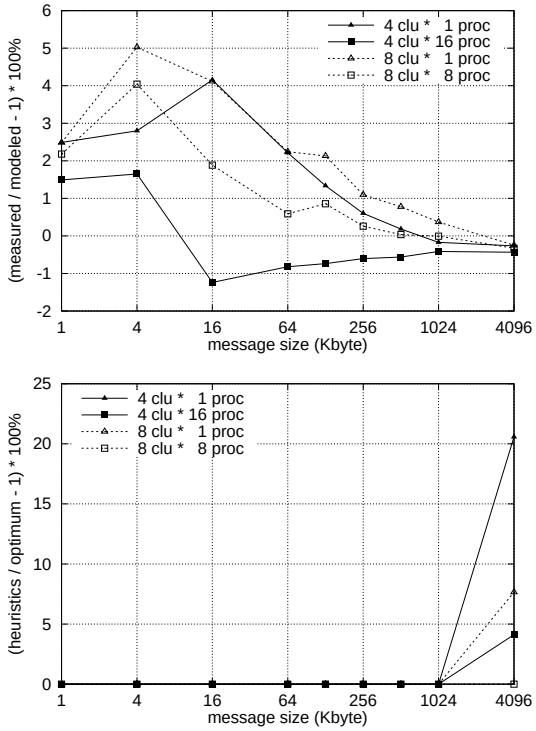

Figure 5. MPI_Bcast: estimation error (top) and heuristics deviation (bottom)

We also investigated the quality of our theoretical estimates. Figure 5 (top) shows the estimation error, namely the difference between estimated and measured completion times. For short messages, it is up to 5%, whereas for large messages the deviation is less than 1%. In general, this is a very close match between our performance model and the implementation. The slightly larger difference for short messages is partly due to the fact that our measurements also include the time for computing the optimized tree. Figure 5 (bottom) shows the deviation of the best heuristically found completion times from the global optima, which were computed offline by an exhaustive search. Our heuristics find the global optimum in almost all test cases, except for $M = 4\text{MB}$, where the optimum with $k = 3$ is isolated and is missed by the hill climbing algorithm.

4 Related work

LogP [9] and LogGP [1] are direct precursors of *parameterized LogP*. Having constant values for *overhead* and *gap*, LogP is restricted to short messages whereas LogGP adds the *gap per byte* for long messages, assuming linear behavior. Neither of them handles *overhead* for medium sized to long messages correctly, nor do they model hierarchical networks. Within their limitations, they have been used to study collective communication [1, 4, 9, 20].

The *parameterized communication model* [26] has two parameters which also depend on message size, resembling *P-LogP*'s sender and receiver completion times. Unfortunately, the model fails to adequately capture receiver overhead, as needed in pipelining broadcast as well as in other collective operations.

Bruck et al. study broadcast and allreduce in homogeneous networks using measured machine parameters and the postal model [7]. Santos studies optimality of k-item broadcast algorithms in a theoretical, simplified LogP variant [27]. The problem of a k-item broadcast comes close to broadcasting large messages split into k smaller segments as presented in this paper. Multicast based on packetization is studied in [21] where the underlying network determines the size of the data items being delivered to multiple receivers. Some work has been performed on optimizing single collective operations (e.g. broadcast) for clusters of SMPs which (like wide-area networks) also exhibit hierarchical structures [15, 19]. None of these studied the optimization of message segments to accommodate hierarchical networks. Others study collective operations in networks of heterogeneous workstations rather than hierarchically structured systems [3, 25]. Compositions of collective operations are optimized in [16], but performance is studied only in the homogeneous case.

Several metacomputing projects are currently building the infrastructure on top of which our MagPIe library may utilize distributed computing capacity [10, 11, 13, 17]. The Interoperable MPI Protocol (IMPI) [14] specifies collective communication algorithms for clustered systems while focusing on interoperability rather than performance.

5 Conclusions

Earlier research has shown that many parallel applications can be optimized to run efficiently on a hierarchical wide-area system and that collective communication is a useful abstraction to do some of these optimizations transparently. In this paper, we described two important optimizations for such wide-area collective operations. First, we use message segmentation to split messages into smaller units that can be sent concurrently over different wide-area links, resulting in better link utilization. Second, we determine the tree shape for the collective operations based on properties of the underlying system (such as the LAN and WAN performance and the processor configuration), instead of using a fixed shape. Both optimizations aim to reduce the completion time for large messages.

For both optimizations, we need a performance model that can accurately estimate the completion time of collective operations. Most existing (LogP based) models are inaccurate for collective operations on hierarchical systems with fast local networks and slow wide-area networks. We described a new model, the *P-LogP (parameterized LogP)* model, which has different sets of LogP parameters for both networks. Also, our model makes these parameters a function of the message size, and uses measured values as input.

We have used the *P-LogP* model to optimize the broadcast operation in our MagPIe library. The new library computes a near-optimal segment size and tree shape for each operation at runtime, based on various system properties. The optimization algorithms use heuristics to prune the search space, so they are efficient and could be used with dynamic information (such as produced by NWS [28]) as input, although our current implementation uses static network performance data. We have empirically validated our approach. Experiments show that the new algorithms significantly improve broadcast performance for large messages. Also, they show that the performance model accurately predicts the completion times.

We think the new algorithms are an important step towards a convenient, easy-to-use infrastructure for parallel programming on wide-area systems like computational grids. The current library can be used for programming hierarchical systems with known (and constant) network performance, like the DAS. The techniques developed here also are a good basis for an MPI library that adapts itself dynamically to changing conditions in the networks. Building such a library is the next step in our research.

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 Rutger Hofman and Aske Plaat for their contributions to this research. Peter Merz (University of Siegen) gave valuable advice on fast optimization techniques. We thank Kees Verstoep and John Romein for keeping the DAS in good shape.

References

[1] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: Incorporating Long Messages into the LogP Model — One Step Closer Towards a Realistic Model

for Parallel Computation. In *Proc. Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 95–105, Santa Barbara, CA, July 1995.

[2] 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.

[3] M. Banikazemi, V. Moorthy, and D. Panda. Efficient Collective Communication on Heterogeneous Networks of Workstations. In *International Conference on Parallel Processing*, pages 460–467, Minneapolis, MN, Aug. 1998.

[4] M. Bernaschi and G. Iannello. Collective Communication Operations: Experimental Results vs. Theory. *Concurrency: Practice and Experience*, 10(5):359–386, April 1998.

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

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

[7] J. Bruck, L. D. Coster, C.-T. Ho, and R. Lauwereins. On the Design and Implementation of Broadcast and Global Combine Operations Using the Postal Model. *IEEE Transactions on Parallel and Distributed Systems*, 7(3):256–265, Mar. 1996.

[8] C. Catlett and L. Smarr. Metacomputing. *Commun. ACM*, 35:44–52, 1992.

[9] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Proc. Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 1–12, San Diego, CA, May 1993.

[10] G. E. Fagg, K. S. London, and J. J. Dongarra. MPI_Connect: Managing Heterogeneous MPI Applications Interoperation and Process Control. In *Proc. 5th European PVM/MPI Users' Group Meeting*, number 1497 in LNCS, pages 93–96, Liverpool, UK, 1998.

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

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

[13] E. Gabriel, M. Resch, T. Beisel, and R. Keller. Distributed Computing in a Heterogeneous Computing Environment. In *Proc. 5th European PVM/MPI Users' Group Meeting*, number 1497 in LNCS, pages 180–187, Liverpool, UK, 1998.

[14] W. George, J. Hagedorn, and J. Devaney. Status Report on the Development of the Interoperable MPI Protocol. In *Proc. MPIDC'99, Message Passing Interface Developer's and User's Conference*, pages 7–13, Atlanta, GA, March 1999.

[15] M. Gołebiewski, R. Hempel, and J. L. Träff. Algorithms for Collective Communication Operations on SMP Clusters. In *The 1999 Workshop on Cluster-Based Computing, held in conjunction with 13th ACM-SIGARCH International Conference on Supercomputing (ICS'99)*, pages 11–15, 1999.

[16] S. Gorlatch, C. Wedler, and C. Lengauer. Optimization Rules for Programming with Collective Operations. In *13th Int. Parallel Processing Symp. & 10th Symp. on Parallel and Distributed Processing (IPPS/SPDP'99)*, pages 492–499, 1999.

[17] A. S. Grimshaw, W. A. Wulf, and the Legion team. The Legion Vision of a Worldwide Virtual Computer. *Commun. ACM*, 40(1), January 1997.

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

[19] P. Husbands and J. C. Hoe. MPI-StarT: Delivering Network Performance to Numerical Applications. In *SC'98*, Nov. 1998. Online at http://www.supercomp.org/sc98/proceedings/.

[20] R. M. Karp, A. Sahay, E. E. Santos, and K. E. Schauser. Optimal Broadcast and Summation in the LogP model. In *Proc. Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 142–153, Velen, Germany, June 1993.

[21] R. Kesavan and D. K. Panda. Optimal Multicast with Packetization and Network Interface Support. In *Proc. Int. Conf. on Parallel Processing*, pages 370–377. IEEE, Aug. 1997.

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

[23] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. 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.

[24] 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 *Proc. Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 131–140, Atlanta, GA, May 1999.

[25] B. Lowekamp and A. Beguelin. ECO: Efficient Collective Operations for Communication on Heterogeneous Networks. In *International Parallel Processing Symposium (IPPS)*, pages 399–405, Honolulu, HI, 1996.

[26] J.-Y. L. Park, H.-A. Choi, N. Nupairoj, and L. M. Ni. Construction of Optimal Multicast Trees Based on the Parameterized Communication Model. In *Proc. Int. Conference on Parallel Processing (ICPP)*, volume I, pages 180–187, 1996.

[27] E. E. Santos. Optimal and Near-Optimal Algorithms for k-Item Broadcast. *Journal of Parallel and Distributed Computing*, 57:121–139, 1999.

[28] R. 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, Aug. 1997. The network weather service is at http://nws.npaci.edu/.