

Network Performance-aware Collective Communication for Clustered Wide Area Systems

Thilo Kielmann^a, Henri E. Bal^a, Sergei Gorlatch^b,
Kees Verstoep^a, Rutger F.H. Hofman^a

^a*Division of Mathematics and Computer Science,
Vrije Universiteit, Amsterdam, The Netherlands*
{kielmann,bal,versto,rutger}@cs.vu.nl

^b*Department of Computer Science, Technical University of Berlin, Germany*
gorlatch@cs.tu-berlin.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. 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 optimize graph shape and segment size at runtime, we introduce a performance model called *Parameterized LogP* (P-LogP), a hierarchical extension of the LogP model that covers messages of arbitrary length. An experimental performance evaluation shows that the newly implemented collective operations have 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 [8,13]. Communication insensitive applications like the ones based on the master-worker paradigm can easily be deployed in grid environments [18]. However, writing applications with frequently communicating processes is much more difficult when targeting at grids rather than 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 [29,30], we discussed how collective communication libraries can be used to simplify wide-area parallel programming. We implemented an 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 [30]. MagPIe's existing implementation efficiently deals with the high WAN latency but has suboptimal performance with long messages which are more sensitive to WAN bandwidth.

Collective communication with long messages needs a more detailed model, 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 use this model to optimize collective communication using message segmentation and tree shape determination. We optimized four important collective operations: broadcast, scatter, gather, and allgather. 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 some simplifying assumptions about the networks, as we use regular topologies and constant latencies and bandwidths. Assuming latency and bandwidth to be constant is certainly realistic for the duration of a single collective operation. In general, MagPIe's network performance data can be updated regularly during an application run. Our fast, heuristic optimizations are designed to cope with such dynamically changing information.

Assuming a regular wide-area topology allows us to focus on the impact of network performance on the design of optimized collective operations. We have evaluated our model and the optimizations on an experimental wide-area testbed, both with simulated WAN links and with a real WAN, using clusters located at four universities in The Netherlands. Our wide-area simulator allows us to study the impact of wide-area performance quantitatively and in detail. Our experiments on the real system confirmed the simulator results qualitatively. Our experimental results are encouraging to further pursue our approach. For example, while using the simulator-based system for broadcasting messages to 7 remote clusters with a 1 MB/s link each, we achieved an aggregate bandwidth of 6.65 MB/s, compared to 0.96 MB/s without message segmentation. Furthermore, we compared theoretically estimated and actually measured completion times and found close matches; for example, with broadcast, the difference is between 1 % and 4 %. Comparing our optimization heuristics with offline, exhaustive searching revealed that the heuristics missed the global optima only in a few exceptional cases. In the broadcast example, differences were always below 1 %.

Our ultimate goal, however, is to develop an MPI library that does not have the mentioned limitations and that adapts at runtime to changing network conditions. We intend to use our performance model in combination with dynamic information about topology and network performance, the latter as provided by the Network Weather Service [38]. 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 collective operations in hierarchical systems. We experimentally verify our approach using a real wide-area system in Section 4. Related work will be discussed in Section 5. Section 6 presents our conclusions. Appendix A summarizes the symbols used throughout the paper.

2 Modeling message-passing performance

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 the Dutch academic Internet backbone, SURFnet. The system is described in detail on <http://www.cs.vu.nl/das/> and in [30]. 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.

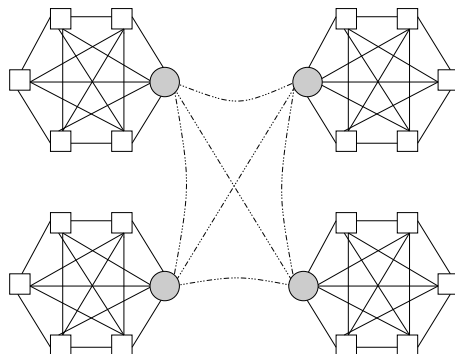


Figure 1. Structure of a clustered wide area system. The circles represent gateways.

MagPIe re-implements the collective operations of a given MPI implementation on top of MPI's point-to-point communication. We run MagPIe on top of MPICH [20], a widely used public MPI implementation, which we have ported to the wide-area DAS system. Our MPICH port uses the Panda communi-

cation sublayer [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 [30]. This wide-area simulation is part of Panda and thus is transparent to software layers on top of it, like MPI. We use this simulation system for the performance experiments reported throughout Sections 2 and 3. We use a varying number of clusters, and simulate realistic values for wide-area latency (10 ms), and for wide-area bandwidth (1 MB/s). In comparison, the latency over Myrinet is about $20 \mu s$ and the MPI-level throughput is about 50 MB/s, so there is almost two orders of magnitude difference between the local and the simulated wide-area network. We emphasize that our simulation 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. In Section 4, we verify our simulator-based results with experiments performed on the real wide-area DAS system.

Throughout this paper, we assume all message data to be in contiguous data buffers. The MPI standard also allows so-called *derived* data types, which may have non-contiguous memory layout, possibly causing higher processing overhead for sender and receiver. However, this additional overhead depends on the data types in use, preventing a generally applicable performance model. The influence of derived data types on communication performance is a topic of its own [21] and not covered here.

The granularity of message segmentation is affected by the layering of MagPIe on top of MPI’s point-to-point communication. According to MPI’s philosophy, MagPIe treats messages as vectors of elements of a base type. MagPIe constructs message segments from multiple vector elements. Hence, the granularity of message segments is the size of a single vector element. This is sufficient for common-case applications using data types of small size, like vectors of floating point numbers. However, *derived* data types might exceptionally reach sizes larger than a suitable message segment. In such a case, MagPIe has to use suboptimal segment sizes. This problem could only be resolved by implementing MagPIe’s algorithms as an integral part of an MPI library. In that case, however, MagPIe could no longer be used across multiple heterogeneous systems using their respective native MPI implementations.

Our MagPIe library uses point-to-point messages provided by the underlying MPI implementation. Thus, to obtain a realistic performance model for collective operations on a clustered wide-area system, we first study the performance of point-to-point communication by looking at the LogP parameters for a local network and a wide-area network. Figure 2 shows *send overhead*

and gap , measured for the MPI_Isend routine for various message sizes on both networks, Myrinet and WAN. In analogy to the LogP model, 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 Section 2.2.

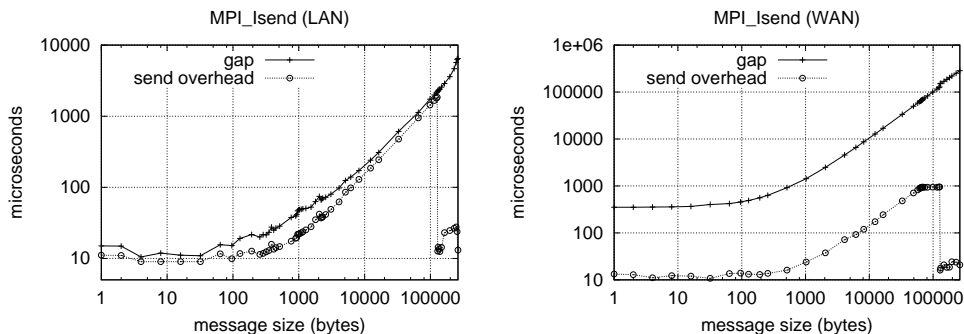


Figure 2. Send overhead and gap for MPI_Isend , measured on the Myrinet LAN (left) and a simulated WAN (right)

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. The application can later check whether the transfer has actually been completed. Our MPICH port to Panda sends until it reaches a point at which it would have to block, and then it returns. 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 now have to wait for a reply message from the receiver, it returns immediately in this case. When a slow wide-area network is used instead of the fast Myrinet, the behavior is similar, except that between 64KB 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 between o and g 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 after 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 behavior of 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 (*P-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 along the same link or connection. 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 $g(m)$ time. $r(m) = L + g(m)$ 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 model. 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 $r(m_1, \dots, m_n) = L + g(m_1) + \dots + g(m_n)$.

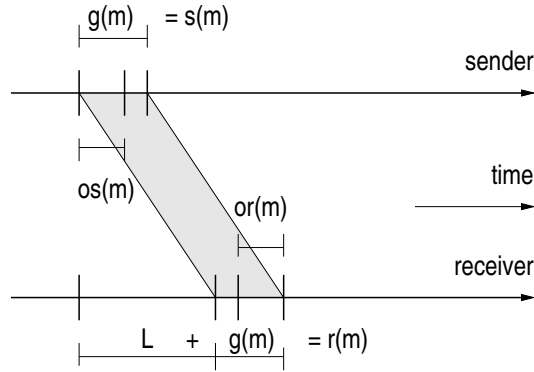


Figure 3. Message transmission as modeled by parameterized LogP

For completeness, we show that *parameterized LogP* subsumes the original model LogP [9] and its version for long messages, LogGP [1]. In Table 1, LogGP’s parameters are expressed in terms of *parameterized LogP*. We use 1 byte as the size for short messages; any other reasonable “short” size may as well be used instead. Note that neither LogP nor LogGP distinguishes between *os* and *or*. For short messages, they use $r = o + L + o$ to relate the L parameter to receiver completion time which gives L a slightly different meaning compared to *parameterized LogP*. We use this equation to derive LogP’s L from our own parameters.

Table 1

LogGP’s parameters expressed in terms of *parameterized LogP*

LogP/LogGP		parameterized LogP
L	=	$L + g(1) - os(1) - or(1)$
o	=	$(os(1) + or(1))/2$
g	=	$g(1)$
G	=	$g(m)/m$, for a sufficiently large m
P	=	P

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:

$$s_l(m) = g_l(m)$$

$$r_l(m) = L_l + g_l(m)$$

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.

The value of r_w always depends on the wide area bandwidth and can be expressed in analogy to r_l . The value of s_w may either be determined by wide-area overhead $os_w(m)$ or local-area gap $g_l(m)$, whichever is higher: the sender can not send the next message before $g_l(m)$ time, but it might have to wait even longer, for example while waiting for flow-control information from its wide-area peer. Unlike the local-area case, the sender is decoupled from the wide-area gap. This gives us the following equations for the wide-area case:

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

2.2 Efficient parameter measurement for P-LogP

An important issue is how to measure the P-LogP parameters described above on an actual wide-area system. Previous LogP micro benchmarks [10,23] measure the gap values by saturating the link for each message size. However, this approach is too intrusive to be feasible for wide-area links which are typically shared with many other users. Our method has to use saturation only for obtaining $g(0)$, which can be done fast and thus is hardly intrusive. In [28] we showed that our measurement procedure, compared to the saturation-based method, yields the same results while reducing the link utilization by 90–95%. Below, we summarize this method.

As we use $g(0)$ for deriving other values, we measure it first using two processes, *measure* and *mirror* (see Figure 4). We measure the time $RTT_n(0)$ for a roundtrip consisting of n messages sent in a row by *measure*, and a single, empty reply message sent back by *mirror*. We take the time measured (after subtracting $RTT_1(0)/2$ for the reply) as $n \cdot g(0)$. The procedure first uses $n = 1$ and then $n = 10$. From this value on n is doubled until the gap per message changes only by $\epsilon = 1\%$. At this point, saturation is assumed to be reached and we compute the gap from the measurement with the so-far largest n .

We start with a small value for n (to speed up the measurement) and double it until the roundtrip time is dominated by bandwidth rather than latency, namely until $RTT_1(0) < \epsilon \cdot RTT_n(0)$ holds in addition to the saturation test. By waiting for a reply we enforce that the messages are really sent to *mirror* instead of just being buffered locally.

All other parameters can be determined by the procedure shown in Figure 4. It starts with a synchronization message by which the *mirror* process indicates being ready. For each size m , two message roundtrips are necessary from *measure* to *mirror* and back. (We use $RTT(m) = RTT_1(m)$.) In the first roundtrip, *measure* sends an m -bytes message and in turn receives a zero-bytes message.

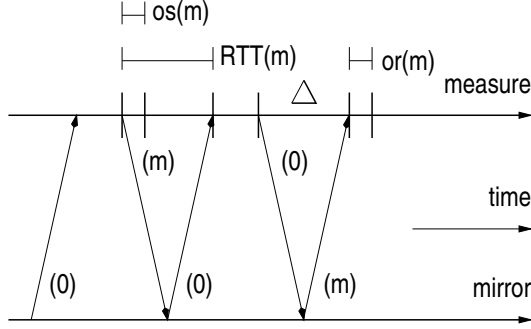


Figure 4. Efficient measurement procedure for P-LogP parameters

We measure the time for just sending and for the complete roundtrip. The send time directly yields $o_s(m)$. $g(m)$ and L can be determined by solving the equations for $RTT(0)$ and $RTT(m)$, according to the timing breakdown in Figure 3:

$$\begin{aligned}
 RTT(0) &= 2(L + g(0)) \\
 RTT(m) &= L + g(m) + L + g(0) \\
 g(m) &= RTT(m) - RTT(0) + g(0) \\
 L &= (RTT(0) - 2g(0))/2
 \end{aligned}$$

In the second roundtrip, the *measure* process sends a zero-bytes message, waits for $\Delta > RTT(m)$ time, and then receives an m -bytes message. Measuring the receive operation now yields $o_r(m)$, because after $\Delta > RTT(m)$ time, the message from *mirror* is available at *measure* for immediate receiving.

For each message size, the roundtrip tests are initially run a small number of times. As long as the variance of measurements is too high, we successively increase the number of roundtrips until a sufficiently small (90%) confidence interval is obtained, or until an upper bound on the total number of iterations is reached (60 for small messages, 15 for large messages). In the latter case, we trade accuracy for non-intrusiveness.

Initially, measurements are performed for all sizes $m = 2^k$ with $k \in [0, k_m]$. The value of k_m has to be chosen large enough to cover any non-linearity caused by the tested software layer. In our experiments, we used $k_m = 18$ to cover all changes in send modes of the assessed MPI implementation (MPICH).

After measuring the initial set of message sizes, we check whether the gap per byte ($g(m)/m$) has stabilized for large m . If this is not the case, sending larger messages may achieve lower gaps (and hence higher throughput). So k_m is incremented and the next message size is tested. This process is performed until $g(2^{k_m})$ is close (within ϵ) to the value linearly extrapolated from $g(2^{k_m-2})$ and $g(2^{k_m-1})$.

So far, the “interesting” range of message sizes has been determined. Finally, possible non-linear behavior remains to be detected. For any size m_k , we check whether the measured values for $o_s(m_k)$, $o_r(m_k)$, and $g(m_k)$ are consistent with the corresponding predicted values for size m_k , extrapolated from the measurements of the previous two (smaller) message sizes, m_{k-1} and m_{k-2} . If the difference is larger than ϵ , we do new measurements for $m = (m_{k-1} + m_k)/2$, and repeat halving the intervals until either the extrapolation matches the measurements, or until $m_k - m_{k-1} \leq \max(32 \text{ bytes}, \epsilon \cdot m_k)$.

The measurement procedure described above assumes that network links are symmetrical, such that sending from *measure* to *mirror* has the same parameters as for the reverse direction. However, this assumption may not always be true. On wide area networks, for example, the achievable bandwidth (the gap) and/or the network latency may be different in both directions, due to possibly asymmetric routing behavior or link speed. Furthermore, if the machines running the *measure* and *mirror* processes are different (like a fast and a slow workstation), then also the overhead for sending and receiving may depend on the direction in which the message is sent. In such cases, the parameters o_s , o_r , and g may be measured by performing our procedure twice, while switching the roles of *measure* and *mirror* in between. Asymmetric latency can only be measured by sending a message with a timestamp t_s , and letting the receiver derive the latency from $t_r - t_s$, where t_r is the receive time. This requires clock synchronization between sender and receiver. Without external clock synchronization (like using GPS receivers or specialized software like the *network time protocol*, NTP), clocks can be synchronized either by statistical time estimation or by simple message exchange protocols. Statistical time estimation [32] is highly intrusive to the network and thus not feasible for WANs. Simple message exchange protocols allow synchronization only up to a granularity of the roundtrip time between two hosts [34], which is useless for measuring network latency. Unfortunately, as we can not generally assume the clocks of (possibly widely) distributed hosts to be tightly synchronized, asymmetric network latencies can not be measured, neither with our framework nor with previous benchmarks [10,23].

3 Performance-aware Collective Communication

We now illustrate how the P-LogP model can be used to optimize collective communication. We discuss four important operations: broadcast, scatter, gather, and allgather. With broadcast, a single process (called the *root*) sends a message to all other processes. Scatter is also known as personalized broadcast. Here, the root splits a large message vector and sends individual messages to the other processes. Gather is the inverse operation of scatter. Here, the root collects messages from its peer processes into a large message

vector. Finally, *allgather* is defined like a *gather* operation, except that all processes receive the result, instead of just the root. For the exchange of data this implies that for *allgather*, all processes conceptually have to broadcast a message.

3.1 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 [25]. MagPIe’s original broadcast algorithm was 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, one of the application processes is selected in each cluster to act as a so-called *coordinator* node. The root process acts as coordinator of its own cluster. First, the root sends the message to the other 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 [30]. 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. Because a cluster gateway decouples LAN and WAN packets, the root can quasi-simultaneously send small packets over up to $n = \lfloor g_w(m)/g_l(m) \rfloor$ 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 $k = \lceil M/m \rceil$) 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 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 meta-computers described in the introduction. Two-layer algorithms are more complicated to model, but are more efficient for broadcast operations.

3.1.1 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 $d, h \in [1 \dots P - 1]$. Fortunately, d and h of the optimal tree are related to each other, and we can compute the minimal height of a tree with P nodes and degree d as the smallest $h \geq 1$ for which $\sum_{i=0}^h d^i \geq P$ is true.

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

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

To illustrate this formula, we can express the values for $\gamma(m)$ and $\lambda(m)$ for MagPIe’s flat WAN tree as:

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

Here, $\gamma(m)$ 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 $\lambda(m)$ 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, upper bounds for both parameters can be expressed depending on the degree d and height h of a broadcast tree:

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

$\gamma(m)$ 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 $\lambda(m)$ 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 $\gamma(m)$ and $\lambda(m)$ 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 are parameters of *MPI_Bcast*. 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 $m \in [1 \dots M]$ and $d \in [1 \dots P - 1]$. 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) $d \in [\lfloor g(m)/s(m) \rfloor \dots P - 1]$. Lower values for d can only lead to higher completion times, because less accumulated bandwidth would be used.
- (2) Starting with $d = \lfloor g(m)/s(m) \rfloor$, we increase d while only investigating those values of d for which the height h will be reduced. Values for d in between would increase $\gamma(m)$ but would not decrease $\lambda(m)$, and could thus 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 $m = M/2^i, i \in [0 \dots \log_2 M]$ are investigated. For each value for m , the degree d with minimal completion time is determined.
- (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. With the respectively best value for d , 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.

3.1.2 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:

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

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:

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

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

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

The optimization of the two-layer broadcast for a given m additionally requires taking care of $\lambda_l(m)$ and $\gamma'(m)$, the latter denoting the fraction of $\gamma(m)$ that can be used for the LAN without slowing down the WAN. We then have to investigate the LAN trees with $d \in [1 \dots \lfloor \gamma'(m)/s_l(m) \rfloor]$ with:

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

3.1.3 Performance evaluation

We have implemented a two-layer broadcast as described so far as part of the MagPIe library. Figure 5 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).

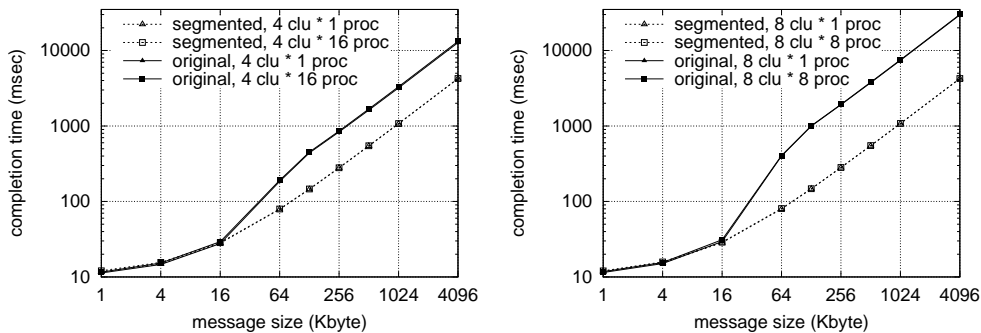


Figure 5. MPI_Bcast: completion time on 4 clusters (left) and 8 clusters (right)

The number of CPUs per cluster has hardly any effect on the overall completion time, causing the respective pairs of lines in Figure 5 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. For example, with 8 clusters of 1 CPU each, broadcasting a 1 MB message takes 1079 ms with segmentation, and 7425 ms without. This corresponds to an aggregate bandwidth of 6.65 MB/s for segmentation. This is 95% of the actually available bandwidth which is 7×1 MB/s. Without segmentation, the wide-area links are used sequentially, resulting in an aggregate bandwidth of only 0.96 MB/s, or 14% of what is available. With fragmentation, the completion times for 4 and 8 clusters are about the same (for 1 MB messages, 1072ms with 4 clusters and 1079 ms with 8 clusters), whereas the original algorithm takes much longer for 8 clusters than for 4 clusters (for

1 MB messages, 3200 ms with 4 clusters and 7425 ms with 8 clusters). This clearly shows that segmentation allows us to use all available wide-area links in parallel as long as the local-area network has enough bandwidth to feed them all.

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 and 4 clusters, for example, the forwarding takes $13224.3 - 12723.3 = 501$ ms. This overhead disappears with pipelining.

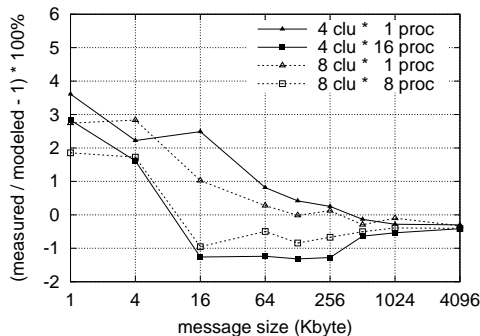


Figure 6. MPI_Bcast: estimation error

We also investigated the quality of our theoretical estimates. Figure 6 shows the estimation error, namely the difference between estimated and measured completion times. For short messages it is up to 4%, 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 mostly due to the fact that our measurements also include the time for computing the optimized tree. We also compared the difference between the best heuristically found completion times and the global optima, which were computed offline by an exhaustive search. Our heuristics found the global optimum in almost all test cases, in the few exceptional cases differences were always below 1%.

3.2 Scatter

The second collective operation we discuss is scatter, which is also called personalized broadcast. With scatter, the root process holds $M \cdot P$ data items which it equally distributes across the P processes, including itself. With homogeneous networks, optimal scatter algorithms use flat trees as communication graphs. (Because each processor has to receive a personalized message, forwarding does not help improving the completion time.) Only for very small messages, message combining and forwarding via coordinator nodes

may improve performance. However, we do not investigate message combining here, because we focus on large messages. Adding message combining to our performance model would be straightforward, by using segment sizes $m \in [M \dots M \cdot P]$ over the WAN and a two-level algorithm, as for broadcast.

With a fixed communication graph (a flat tree encompassing all nodes in all clusters), our analytical model combines both WAN and LAN parameters. We optimize scatter algorithms by finding a message segment size that yields the shortest overall completion time. P_w denotes the number of clusters and P_l the (maximum) number of processes per cluster. Analogous to broadcast, we model $T = (k - 1) \cdot \gamma(m) + \lambda(m)$ with:

$$\begin{aligned} \gamma(m) &\leq P_l \cdot \max(g_w(m), (P_w - 1) \cdot s_w(m) + s_l(m)) + or_l(m) \\ \lambda(m) &\leq (P_l - 1) \cdot \max(g_w(m), (P_w - 1) \cdot s_w(m) + s_l(m)) \\ &\quad + (P_w - 1) \cdot s_w(m) + r_w(m) \end{aligned}$$

$\gamma(m)$ models the time spent at the root process, because for other processes, the gap per segment is much smaller, namely $or(m)$. The model assumes that the message segments are first sent to the first processor in each cluster, then to the second processor in each cluster, etc., using the wide-area links in a round-robin fashion. Before the root process can send out the next segment, it has to receive the message segment it just sent to itself. $\lambda(m)$ denotes the time at which the last message of a segment round is sent, plus its receiving time. This sending time adds the time for sending the segment to all but one processors in each cluster, and the time for sending to the last processor in all but one clusters. $\lambda(m)$ equals its upper bound when the last message is sent to a process in a remote cluster. Otherwise it is somewhat less, depending on which message is actually received last. As with broadcast, we use the upper bounds for optimizing T and use the binary search with subsequent hill climbing for finding a near-optimal segment size.

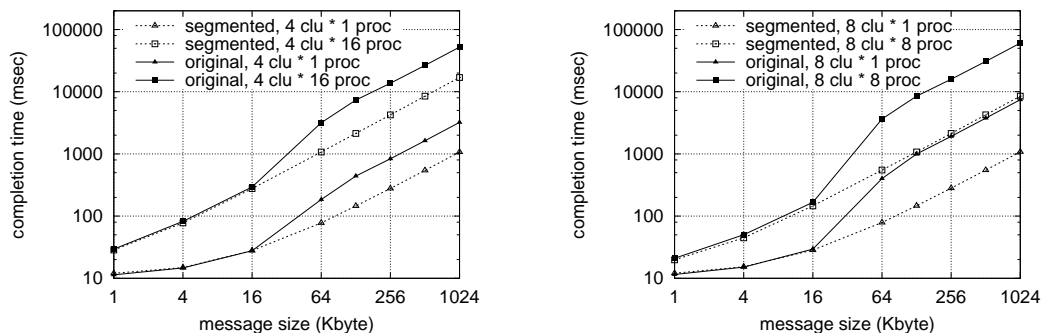


Figure 7. MPI_Scatter: completion time on 4 clusters (left) and 8 clusters (right)

Figure 7 compares the completion times of the new scatter algorithm with MagPIe’s original scatter. The new algorithm does message segmentation

whereas the original algorithm sends complete messages. We measured the same configurations as with broadcast. The new algorithm performs much better, because it achieves higher aggregate WAN bandwidth. For example, with 8 clusters of 1 CPU each, sending a 1 MB message to each receiver takes 1078 ms with segmentation, and 7437 ms without. The completion times for large messages are proportional to $g_w(M) \cdot P_l$, which indicates that almost all available WAN bandwidth can be utilized.

Figure 8 compares the theoretically estimated completion times with the ones actually measured on our system. For small messages, the estimation error is slightly worse than for broadcast (up to 14%), but for large messages it is only about 1%, denoting a very close match between theory and practice. As with broadcast, our search heuristics find the global optimum in almost all test cases, with a few negligible deviations (smaller than 1%).

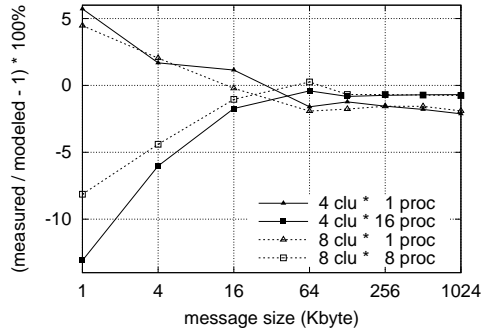


Figure 8. MPLScatter: estimation error

3.3 Gather

Gather is the inverse operation of scatter. Here, each process (including the root) holds M data items which are sent to the root. Analogous to scatter, optimal algorithms use flat trees. The completion time can be modeled as $T = (k - 1) \cdot \gamma(m) + \lambda(m)$ with:

$$\begin{aligned} \gamma(m) &= \max(P_l \cdot g_w(m), P_l \cdot or_l(m) + (P_w - 1) \cdot P_l \cdot or_w(m) + os_l(m)) \\ \lambda(m) &= \max(L_w + P_l \cdot g_w(m), \\ &\quad \max(L_l, os_l(m)) + P_l \cdot or_l(m) + (P_w - 1) \cdot P_l \cdot or_w(m)) \end{aligned}$$

$\gamma(m)$ and $\lambda(m)$ model the time at which the root process completes receiving the message segments. $\gamma(m)$ is the maximum of the time the segments need to cross the WAN links, and the time the root needs to receive all messages of a segment round, plus sending its own segment to itself. $\lambda(m)$ is similar, but models the overlap of latency and receiving at the root.

Figure 9 shows the measured completion times for the same configurations as with broadcast and scatter. We compare MagPIe’s original algorithm with the new segmenting variant. The results are similar to the ones achieved with MPI_Scatter. This time, message segmentation helps simultaneously receiving on all connections, leading to better WAN bandwidth utilization. Figure 10 compares the theoretically estimated completion times with the ones actually measured on our system. As with broadcast and scatter, the estimation errors are mostly negligible, except for a few cases with large messages and many simultaneously sending nodes. Here, some effects related to contention inside the receiving node cause somewhat higher completion times than expected by the theoretical model. As with broadcast and scatter, the search heuristics hardly ever miss the globally optimal configurations.

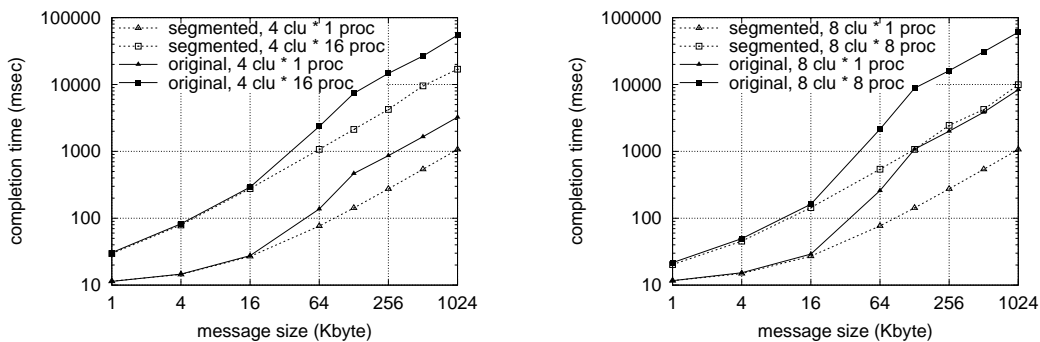


Figure 9. MPI_Gather: completion time on 4 clusters (left) and 8 clusters (right)

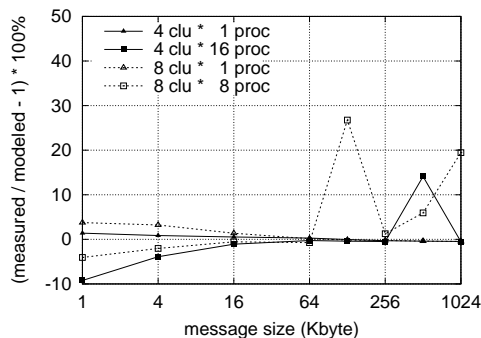


Figure 10. MPI_Gather: estimation error

3.4 Allgather

Allgather is like the gather operation, except that all data is delivered at every node, and not just at a single root. One simple way to implement allgather is as a sequence of gather operations, using every node as root. However, in this way the scarce bandwidth available at the WAN links is not used efficiently.

An alternative, which is currently implemented in MagPIe, is to first let the coordinator of every cluster gather the data of all local nodes; next exchange

that data with all other cluster coordinators; and finally let the coordinators broadcast the whole data vector locally in their clusters. This approach works particularly well when there is a big difference in performance in the LAN and WAN networks, since pipelining issues (as encountered in the optimization of the broadcast implementation) are much less important than the efficient use of the slow WAN links. The completion time of this implementation can be modeled as the sum of local gather, wide-area allgather (between the coordinator nodes), and the local broadcast. The time for gather and broadcast has been outlined above; in the following we only deal with the allgather operation between the coordinator nodes. Its completion time is $T = (k - 1) \cdot \gamma(m) + \lambda(m)$ with:

$$\begin{aligned}\gamma(m) &= \max((P_w - 1) \cdot (s_w(m) + \max(g_l(m), or_w(m))), g_w(m)) \\ \lambda(m) &= (P_w - 2) * s_w(m) + g_w(m) + L_w\end{aligned}$$

The segmenting allgather consists of k rounds in which each node sends a segment to all its peers and receives one segment per peer. $\gamma(m)$ thus is the maximum of the wide-area gap and the processing time per segment, the latter is the number of peers multiplied by the sum of the completion time for send and receive. The receive completion time is the maximum of the local gap and the receive overhead. $\lambda(m)$ is, as usual, the time when a node sends the last message, plus the time to deliver it to the respective receiver.

We compared MagPIe’s original implementation of MPI_Allgather with one that uses segmentation in the central exchange phase between the cluster coordinators. The results for 4 and 8 clusters are shown in Figure 11. For 8 clusters, and using a segment size smaller than 64K, the optimized version avoids performance loss due to flow control issues for message sizes between 64K and 128K. However, for most other message sizes, and for 4 clusters, the results are very similar, since the original MagPIe implementation is already keeping the WAN links well occupied. Finally, Figure 12 compares (for the central phase between the coordinator nodes) the theoretically estimated completion times with the ones actually measured on our system. For a few configurations, the estimation error is up to 16 % which can be explained by contention inside the nodes due to simultaneously receiving from multiple peers.

4 Experimental Results on the Real Wide-Area System

The results presented so far have been obtained using the wide-area simulator of our Panda communication sublayer. The simulator allows to perform measurements on a real parallel machine with only the wide-area links being

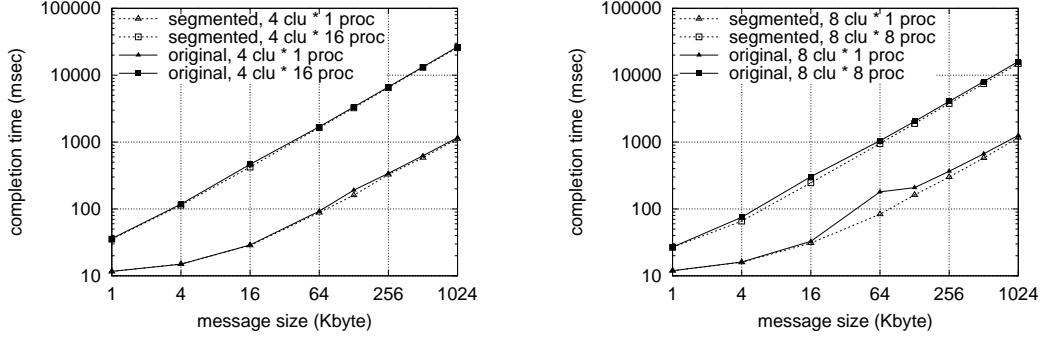


Figure 11. MPI_Allgather: completion time on 4 clusters (left) and 8 clusters (right)

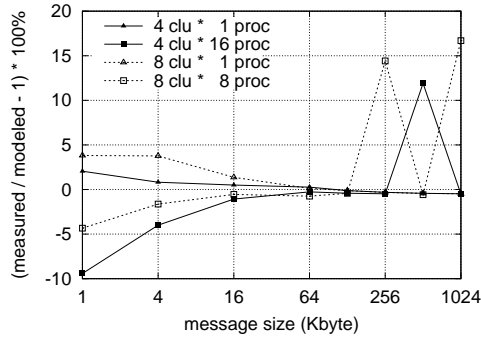


Figure 12. MPI_Allgather: estimation error

simulated, by adding delay loops in the Panda gateway nodes. With the simulator, we were able to investigate our collective communication operations in a clean environment without interference of network traffic caused by other users. This allowed us to *quantitatively* analyze our results.

In this section, we present experimental results from the real system, the four DAS clusters located at Vrije Universiteit Amsterdam (VU), at Universiteit Leiden, the University of Amsterdam (UvA), and Delft University of Technology. The clusters are connected via the Dutch academic Internet backbone (SURFnet). Table 2 summarizes the average TCP-level bandwidth (in Mb/s) and one-way latency (in ms) between the DAS clusters. As can be seen in the table, the latency between the clusters varies from 1.5 ms to 4.0 ms. Bandwidth ranges from 3.0 Mb/s to 28.0 Mb/s. These values for bandwidth and latency indicate that, with the DAS, wide-area communication is somewhat faster than in our simulated environment. This can be attributed to the rather small physical distances (up to 50 km) between the clusters. Additionally, the network parameters vary over time due to concurrent network traffic. Furthermore, the wide-area links are asymmetric due to the routing setup. Therefore, the purpose of the experiments presented in this section is to *qualitatively* verify our results from Section 3 on a real wide-area system.

The collective communication operations presented in Section 3 are based on P-LogP parameters for both local-area and wide-area networks. Figure 13

Table 2

Average TCP-level latency and bandwidth between the four DAS clusters

from	Latency (ms) to				Bandwidth (Mb/s) to			
	VU	Leiden	UvA	Delft	VU	Leiden	UvA	Delft
VU		3.5	1.5	2.5		5.0	26.0	7.0
Leiden	3.5		3.0	4.0	3.5		4.0	3.0
UvA	1.5	3.0		2.0	28.0	6.0		7.5
Delft	2.5	4.0	2.0		6.0	4.5	6.5	

shows send overhead and gap on four of the 12 links between the DAS clusters. The two graphs on the right side of Figure 13 (VU to UvA and VU to Delft) closely resemble the measurements on the simulated WAN in Figure 2. However, the sender can transmit messages larger than 64KB without stalling while waiting for flow-control information from the receiver. This is due to the lower wide-area latency. Unlike our clean simulated environment, the Leiden cluster is connected to the rest of the DAS via a very lossy link. The frequent packet losses not only degrade bandwidth, they also lead to various artefacts with the measured gap values. But even with such a lossy link, the shape of the measured curves qualitatively matches the results from the simulated environment.

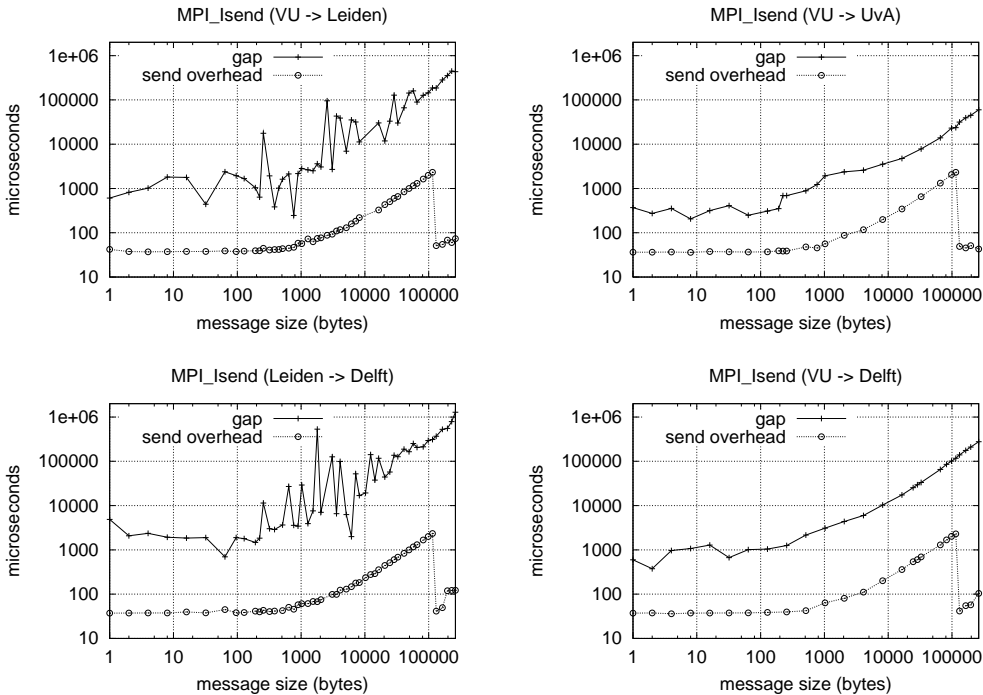


Figure 13. Send overhead and gap for MPI_Isend between the four DAS clusters

The collective algorithms presented in Section 3 assume a homogeneous wide-area network with identical P-LogP parameters for all links. However, as shown

in Table 2, the wide-area links of the DAS perform somewhat differently from each other. For measuring the collective operations on the real wide-area system, we have used P-LogP parameters according to the slowest available link (from Leiden to Delft). With this conservative approach, message segmentation and tree shape are determined according to the “bottleneck” WAN link.

Figure 14 shows the completion times of the four collective operations (broadcast, scatter, gather, and allgather) across the four DAS clusters. As in Section 3, we measured completion times for four clusters with 1 and with 16 processors each. The results qualitatively confirm the simulator measurements. Due to the somewhat faster wide-area links, however, the effects are less pronounced. But still, starting with a message size between 16 KB and 64KB, the implementation based on message segmentation shows better performance than MagPIe’s original implementation.

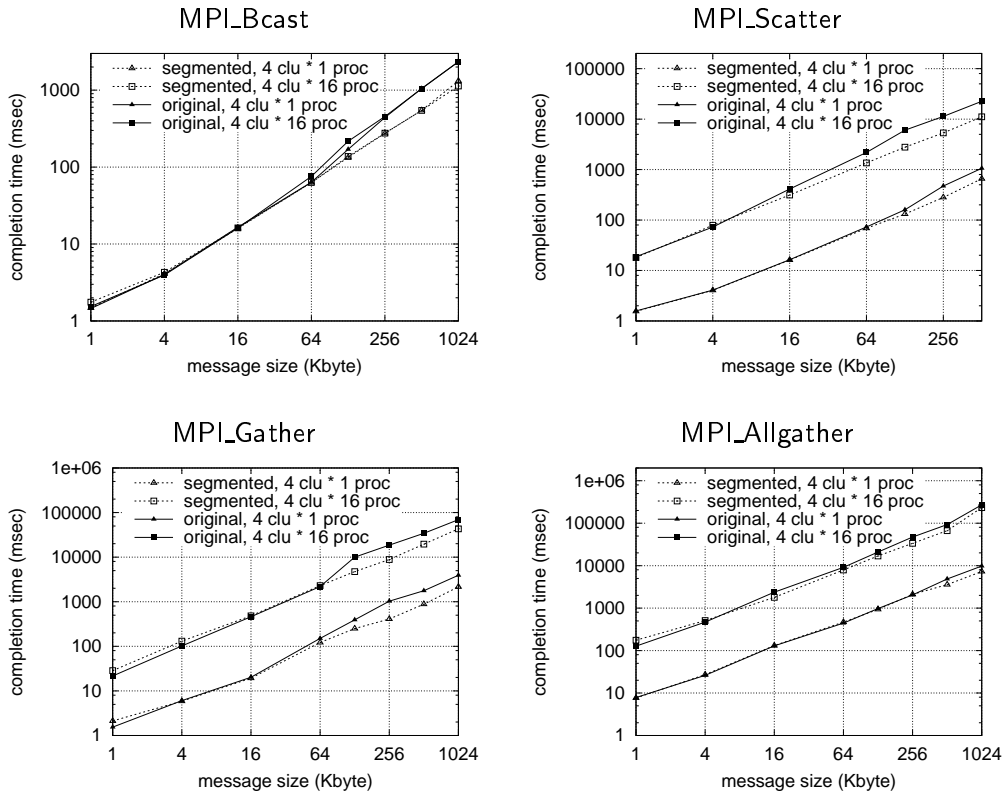


Figure 14. Completion times on the four DAS clusters

5 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 [1,4,9,25].

The *parameterized communication model* [33] 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 by the collective operations discussed in this paper.

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 [35]. 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 [26] where the underlying network determines the size of the data items being delivered to multiple receivers. In a previous paper, we focussed on optimizing MPI's broadcast operation by message segmentation and tree shape optimization [27].

Van de Geijn et al. study efficient pipelined implementations of broadcast [37] and reduction [36] on homogeneous one-level networks, like meshes and hypercubes. 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 [16,22], however without using message segmentation to accommodate hierarchical networks. Others study collective operations in networks of heterogeneous workstations rather than hierarchically structured systems [3,31]. Compositions of collective operations are optimized in [17], but performance is studied only in the homogeneous case.

Our work currently makes simplifying assumptions about the wide-area networks. Karonis et al. apply optimizations to MPI's broadcast for hierarchical systems with more than two layers [24]. The authors show performance improvements compared to the non-segmenting version of MagPIe for a system with three hierarchy layers. However, identifying a hierarchical system representation seems to be a challenging problem for the general case of Internet-based metacomputing platforms.

Several metacomputing projects are currently building the infrastructure on top of which our MagPIe library may utilize distributed computing capacity [11,12,14,19]. The Interoperable MPI Protocol (IMPI) [15] specifies collective communication algorithms for clustered systems while focusing on interoperability rather than performance.

6 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 to cluster mapping), instead of using a fixed shape. Both optimizations 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 four important operations in our MagPIe library (broadcast, scatter, gather, and allgather). The new library computes a near-optimal segment size and tree shape 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 [38]) 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 collective performance for large messages. In the controlled simulator environment, our experiments show that the performance model accurately predicts the completion times. Our algorithms improve performance also on the real wide-area system where performance information is imprecise.

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 similar network performance on the wide-area links, like our DAS system. Other possible target platforms are clusters of SMP nodes that have a similar network hierarchy. 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 Peter Merz (University of Siegen) for his valuable advice on fast optimization techniques and Grégory Mounié for his contributions to this paper. Finally, we thank 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] D. E. Culler, L. T. Liu, R. P. Martin, and C. O. Yoshikawa. Assessing Fast Network Interfaces. *IEEE Micro*, 16(1):35–43, Feb. 1996.
- [11] 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.
- [12] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Int. Journal of Supercomputer Applications*, 11(2):115–128, 1997.
- [13] I. Foster and C. Kesselman, editors. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.
- [14] 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.
- [15] 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.
- [16] M. Golebiewski, 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.
- [17] 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.
- [18] J.-P. Goux, S. Kulkarni, J. Linderoth, and M. Yoder. An Enabling Framework for Master-Worker Applications on the Computational Grid. In *Proc. High Performance Distributed Computing (HPDC 2000)*, pages 43–50, Pittsburgh, PA, Aug. 2000.
- [19] A. S. Grimshaw, W. A. Wulf, and the Legion team. The Legion Vision of a Worldwide Virtual Computer. *Commun. ACM*, 40(1):39–45, January 1997.
- [20] 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.
- [21] W. D. Gropp, E. Lusk, and D. Swider. Improving the Performance of MPI Derived Datatypes. In *Proc. MPIDC'99, Message Passing Interface Developer's and User's Conference*, pages 25–30, Atlanta, GA, March 1999.
- [22] 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/>.

- [23] G. Iannello, M. Lauria, and S. Mercolino. Cross-Platform Analysis of Fast Messages for Myrinet. In *Proc. Workshop CANPC'98*, number 1362 in Lecture Notes in Computer Science, pages 217–231, Las Vegas, Nevada, January 1998. Springer.
- [24] N. T. Karonis, B. R. de Supinski, I. Foster, W. Gropp, E. Lusk, and J. Bresnahan. Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance. In *Proc. International Parallel and Distributed Processing Symposium (IPDPS 2000)*, pages 377–384, Cancun, Mexico, May 2000. IEEE.
- [25] R. M. Karp, A. Sahay, E. E. Santos, and K. E. Schauer. Optimal Broadcast and Summation in the LogP model. In *Proc. Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 142–153, Velen, Germany, June 1993.
- [26] 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.
- [27] T. Kielmann, H. E. Bal, and S. Gorlatch. Bandwidth-efficient Collective Communication for Clustered Wide Area Systems. In *Proc. International Parallel and Distributed Processing Symposium (IPDPS 2000)*, pages 492–499, Cancun, Mexico, May 2000. IEEE.
- [28] 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)*, number 1800 in Lecture Notes in Computer Science, pages 1176–1183, Cancun, Mexico, May 2000. Springer.
- [29] 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.
- [30] 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.
- [31] 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.
- [32] E. Maillet and C. Tron. On Efficiently Implementing Global Time for Performance Evaluation on Multiprocessor Systems. *Journal of Parallel and Distributed Computing*, 28:84–93, 1995.
- [33] 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.

- [34] V. Paxson. On Calibrating Measurements of Packet Transit Times. In *Proc. SIGMETRICS'98/PERFORMANCE'98*, pages 11–21, Madison, Wisconsin, June 1998.
- [35] E. E. Santos. Optimal and Near-Optimal Algorithms for k -Item Broadcast. *Journal of Parallel and Distributed Computing*, 57:121–139, 1999.
- [36] R. van de Geijn. On global combine operations. *J. Parallel and Distributed Computing*, 22:324–328, 1994.
- [37] J. Watts and R. van de Geijn. A pipelined broadcast for multidimensional meshes. *Parallel Processing Letters*, 5(2):281–292, 1995.
- [38] 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/>.

Appendix A: Symbols used in analytical modeling

M	total message size
m	message segment size
k	number of message segments, $k = \lceil M/m \rceil$
L	network latency
$os(m)$	send overhead for message of size m
$or(m)$	receive overhead for message of size m
$g(m)$	gap between two messages of size m
P	number of processors
N	network, $N = (L, os, or, g, P)$
T	completion time of a collective operation
$s(m)$	sender completion time for message of size m
$r(m)$	receiver completion time for message of size m
$RTT(m)$	round-trip time for message of size m (empty reply message)
h	height of a broadcast tree
d	degree (fan out) of a broadcast tree
$\lambda(m)$	latency of a broadcast tree with message of size m
$\gamma(m)$	gap of a broadcast tree with message of size m
l	subscript, used to identify LAN parameters
w	subscript, used to identify WAN parameters