Balanced Multicasting: High-throughput Communication for Grid Applications

Mathijs den Burger, Thilo Kielmann, Henri E. Bal
Dept. of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands
{mathijs, kielmann, bal}@cs.vu.nl

ABSTRACT

Many grid applications need to transfer large amounts of data between the geographically distributed sites of a grid environment. Network heterogeneity between these sites makes throughput optimization of data transfers to multiple sites (multicast) hard or even impossible. We present a technique called *balanced multicasting* that uses monitoring information for both bandwidth capacity and achievable bandwidth to compute balanced multicast trees at runtime that use application-level traffic shaping at the sender side to avoid self-induced congestion. Our experimental evaluation shows that our approach outperforms existing multicast strategies by large margins.

1. INTRODUCTION

A grid consists of multiple sites, ranging from single machines to large clusters, located around the world. Contrary to more traditional computing environments like clusters or super computers, the network characteristics between Grid sites are very heterogeneous. Therefore, communication libraries need to take this heterogeneity into account to maintain efficiency in a world-wide environment.

A typical communication pattern is the transfer of a substantial amount of data from one site to multiple others, also known as *multicast*. The completion time of large data transfers depends primarily on the bandwidth of the interconnection network. Multicasting is usually implemented by arranging the nodes in a certain spanning tree over which the data are sent. This method can be very inefficient in a grid environment, where the differences in bandwidth between sites should be taken into account to achieve high throughput.

In recent years, several network monitoring systems have been developed to provide measurement information to applications to make them 'network-aware'. In this paper we use information about both *achievable bandwidth* and *bandwidth capacity*, as identified in [19]. Based on this information, we construct multicast trees between grid sites aiming at efficiently utilizing the available bandwidth while not over subscribing the bandwidth capacities. Our technique achieves this by employing application-level traffic shaping to build multiple, concurrently used, balanced trees. The resulting multicast trees are computed at runtime and are optimized for throughput.

Our experimental evaluation shows that our approach outperforms existing multicast strategies by large margins. We have evaluated *balanced multicasting* in two settings:

1. For eight sites from the testbed of the European GridLab project[1], we have compared *balanced multicasting* with three other multicasting strategies. Compared to the strongest competitor, the Fast Parallel File Replication tool [14], *balanced multicasting* increased the throughput by up to 50%.

2. For clusters of the Dutch *Distributed ASCI Supercomputer* (DAS)[9], we have applied *balanced multicasting* to overcome the capacity limitations of the individual nodes' network interface cards: We have combined the network interfaces of multiple hosts for WAN communication while locally distributing the data using a separate, high-speed network (Myrinet). Using *balanced multicasting,* we were able to increase the throughput between two clusters up to 550% until all available wide-area bandwidth was used.

The remainder of this paper is structured as follows. In Section 2, we discuss issues in multicasting in grids, as well as existing approaches. Section 3 describes the balanced multicasting algorithm and its implementation. Section 4 describes our experimental evaluation and limitations of our approach, before Section 5 concludes.

2. BACKGROUND AND RELATED WORK

Before presenting *balanced multicasting*, we first discuss more traditional approaches to multicasting in grids and Internet-based environments. In this section, we also outline our network performance model and identify performance shortcomings of existing, optimizing multicasting techniques.

2.1 Overlay multicasting

Multicasting over the Internet started with the development of IP multicast, which uses specialized routers to forward packets. Since IP multicast was never widely deployed, *overlay multicasting* became popular, in which only the end hosts play an active role. Currently, overlay multicasting is also investigated within the high-performance networking community of the Global Grid Forum [12].

Several centralized or distributed algorithms have been proposed to find a single overlay multicast tree with maximum throughput [6, 18]. Splitting the data over multiple trees can increase the throughput even further.

A related topic is the overlay multicast of media streams, in which it is possible for hosts to only receive part of the data (which results in, for instance, lower video quality). In [8, 18], a single multicast tree is used for this purpose. SplitStream [4] uses multiple trees to do distribute streaming media in a P2P context. Depending on the bandwidth each host is willing to donate, the hosts receive a certain amount of the total data stream. The maximum throughput is thus limited to the bandwidth the stream requires. In contrast, our balanced multicasting tries to use the maximum amount of bandwidth the hosts and networks can deliver.

MuniSocket [21] is a middleware layer on top of TCP that transparently splits data over multiple network interfaces much like our optimization for multiple clusters does. However, it can only combine multiple network interfaces within a single machine whereas our multi-cluster optimization can use the network interfaces of all machines in a cluster. MuniSocket's approach could therefore be added to ours to increase the bandwidth capacity of machines in a cluster, provided that they have multiple network interfaces.

2.2 Network performance modeling

Our balanced multicasting algorithm relies on a careful network performance modeling and on the provision of the necessary monitoring data. The network performance model is built according to our goal, which is to minimize the overall completion time of sending a large amount of data from one host to all other hosts of a system. As we focus on large data volumes, the optimization problem is dominated by network bandwidth, so we neglect latency altogether. For network bandwidth, we use the following distinction from [19]:

Bandwidth Capacity is the maximum amount of data per time unit that a hop or path can carry.

Achievable Bandwidth is the maximum amount of data per time unit that a hop or path can provide to an application, given the current utilization, the protocol and operating system used, and the end-host performance capability.

We are interested in maximizing the achievable bandwidth of all data streams used for a multicast operation. For wide-area networks in grids, our own experiments (and those reported on in [4, 8]) indicate that achievable bandwidth is dominated by the end hosts and their capability of efficiently using the TCP protocol suite. For this reason, much work is trying to improve achievable bandwidth by tuning TCP buffers and using parallel TCP streams [10, 22, 26]. Cross traffic by other users is seemingly negligible, as Internet backbone connections provide sufficient bandwidth. With the advent of dedicated, optical wide-area connections, this property will be enforced even more.

In multicasting, sharing effects can be observed whenever a single host is sending to and/or receiving from multiple other hosts. Here, the bandwidth capacity of the local network can become a bottleneck. This local capacity can be caused either by the network interface (e.g., a FastEthernet card, connected to a gigabit network), or by the access link to the Internet that is shared by all machines of a site.

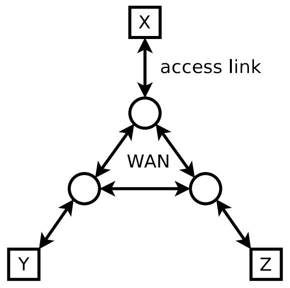


Figure 1: network model

Figure 1 illustrates our network model. Squares represent hosts, arrows represent two unidirectional links (one in each direction) with a certain bandwidth. An arrow connected to a host represents its access link to the Internet, the other arrows symbolize the WAN links. The circles can be seen as the nodes in the network where multiple outgoing and incoming streams from and to each host diverge and converge, respectively. These are typically the border routers, connecting a site to the Internet. Thus, data sent from host x to host y always travels through the outgoing access link of x, the WAN link $x \rightarrow y$ and the incoming access link of y.

For local access links, our model is recording the *local bandwidth capacity*; for WAN paths is is recording the *achievable bandwidth.* We assume that the bandwidth of both the WAN paths and of the local access links is independent in both directions: data sent from x to y does not share bandwidth with data sent from y to x. Furthermore, the bandwidth in both directions can be different. This corresponds to both Internet WAN paths and full-duplex network cards.

To fully utilize our network model, *balanced multicasting* relies on data from an external network monitoring system like the Network Weather Service [27] or Delphoi [20]. The latter uses specialized measurement tools like PathRate [11] and PathChirp [23] to measure the capacity and available bandwidth of WAN links, respectively.

Besides monitoring the network, The REMOS system [13] uses its own measurements to answer flow-related queries from applications. However, it mainly focuses on LAN monitoring and only considers unicast flows. We could therefore evaluate the throughput of a set of multicast trees by modeling them as flows and let REMOS calculate their combined throughput, but we could not find the optimal set of trees that way.

2.3 Optimization of multicast trees

Optimization of multicast communication has been studied extensively within the context of message passing systems and their collective operations. The most basic approach to multicasting is to ignore network information altogether and send directly from the root host to all others. MagPIe [17] used this approach by splitting a multicast into two layers: one within a cluster, and one flat tree between clusters. Such a flat tree multicast will put a high load on the outgoing local capacity of the root node, which will often become the overall bandwidth bottleneck.

As an improvement we can let certain hosts forward received data to other hosts. This allows to arrange all hosts in a directed spanning tree over which the data are sent. MPICH-G2 [15] followed this idea by building a multi-layer multicast to distinguish wide-area, LAN and local communication. As a further improvement for large data sets, the data should be split to small messages that are forwarded by the intermediate hosts as soon as they are received to create a high-throughput pipeline from the root to each leaf in the tree [16].

The problem with this approach is to find the optimal spanning tree. If the bandwidth between all hosts is homogeneous, we can use a fixed tree shape like a chain or binomial tree, which is often used within clusters [25]. As a first optimization for heterogenous networks, we can take the achievable bandwidth between all hosts into account. The throughput of a multicast tree is then determined by its link with the least achievable bandwidth. Maximizing this *bottleneck bandwidth* can be done by using a variant of Prim's algorithm, which yields the *maximum bottleneck tree* [6].

However, this maximum bottleneck tree is not necessarily optimal because each host also has a certain local capacity. A forwarding host should send data to all its n children at a rate at least equal to the overall multicast throughput t. If its outgoing local capacity is less than $n * t$, it cannot fulfill this condition and the actual multicast throughput will be less than expected. Unfortunately, taking this into account generates an NP-complete problem.[1]

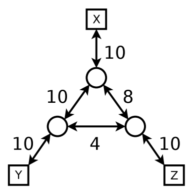
The problem of maximizing the throughput of a set of overlay multicast trees has also been explored theoretically. Finding the optimal solution can be expressed as a linear programming problem, but the number of constraints grows exponentially with the number of hosts. Although, in theory, this can be reduced to a square number of constraints, in practice finding the exact solution can be slow and expensive[7]. Any real-time applicable solution will therefore always have to rely on heuristics.

The multiple tree approach in [3] uses linear programming to determine the maximum multicast throughput given the bandwidth of links between hosts, but requires a very complicated algorithm to derive the set of multicast trees that would achieve that throughput. Therefore, the linear programming solution is only used to optimize the throughput of a single multicast tree.

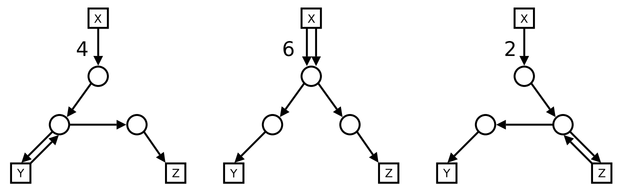
The Fast Parallel File Replication (FPFR) tool [14] is implementing multiple, concurrently used multicast trees. FPFR repeatedly uses depth-first search to find a tree spanning all hosts. For each tree, its bottleneck bandwidth is "reserved" on all links used in the tree. Links with no bandwidth left can no longer be used for new trees. This search for trees continues until no more trees spanning all hosts can be found. The file is then multicast in fixed-size chunks using all trees found.

FPFR does not take the local bandwidth capacity of hosts into account. This is not a problem when using regular TCP streams, since the TCP throughput over long WAN paths

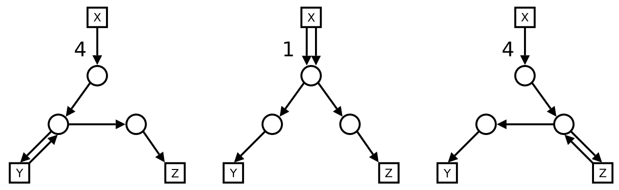
[1]Assuming all links have an achievable bandwidth of either 0 or 1 and each local capacity is also 1, then it can be shown that finding the optimal multicast tree is equivalent to finding a Hamiltonian path in the graph, which is known to be NP-complete.



(a) Network example



(b) FPFR multicast trees



(c) Balanced multicast trees

Figure 2: Example where FPFR overestimates the available bandwidth

is usually rather low. However, by tuning the TCP buffer sizes and using multiple TCP streams in parallel, wide-area throughput can be improved dramatically [22, 26]. As soon as these techniques are applied to all WAN connections (which would be the first step to increase the overall throughput of multicasting), the local capacity of a host can be saturated easily. This causes the throughput of FPFR multicast trees to become less than expected.

We illustrate FPFR's problem using the example network shown in Figure 2a. This network consists of three hosts, each connected to the network by their access line. Routers connect access lines with the WAN. Access lines are annotated with their local capacity, e.g. the capacity of the LAN. Wide-area connections are annotated with their achievable bandwidth. For simplicity of the example, we assume all connections to be symmetrical in both directions.

In this example, FPFR would create the three multicast trees shown in Figure 2b, with an assumed total throughput of 12. However, since all hosts have an incoming and outgoing local capacity of 10, the outgoing local capacity of the root will become a bottleneck. Since four data flows are sharing this local capacity without further coordination, each of them will only get 25% of it. This results in a throughput of 2.5 per tree and a total throughput of 7.5 instead of 12. Yet, when the local capacity would have been taken into account, we could have created the trees shown in Figure 2c, with a total throughput of 9. Enforcing different shares of the outgoing capacity requires traffic-shaping at the sender side, resulting in what we refer to as *balanced multicast trees.*

3. BALANCED MULTICASTING

In this section, we present the algorithm to create balanced multicast trees. We first consider the case of individual hosts before we extend the algorithm to cluster computers. Then we outline our implementation, both for computing the trees and for the runtime system that actually uses them.

3.1 Algorithm

A set of balanced multicast trees can be computed using linear programming (LP). For an exact solution, all possible multicast trees with their achievable bandwidth and the local bandwidth capacities need to be translated to decision variables and constraints of the linear program [7]. Figure 3 shows the translation of the example from Figure 2(a) to a LP problem. For example, the first constraint $a+2b+c \leq 10$ models the outgoing capacity of the root host, which has to transfer one data stream each of the first and third tree and two data streams of the second tree. Solving the LP problem directly yields the optimal throughput per tree. A throughput of zero would mean that a tree could be discarded. The solution for the example can be seen in Figure 2(c).

maximize:

$$throughput = a + b + c$$

subject to:

$$\begin{array}{rcrrcr} a+2b+c & \leq & 10 & a+b & \leq & 10 \\ a & \leq & 10 & b+c & \leq & 8 \\ c & \leq & 10 & a & \leq & 4 \\ a+c & \leq & 10 & c & \leq & 4 \\ b+c & \leq & 10 & & & \end{array}$$

solution:

$$a = 4,\ b = 1,\ c = 4$$

Figure 3: The network in Figure 2(a) translated to a linear programming problem

Since the total number of different trees is n^{n-2} for n given hosts [2], this method is computationally infeasible, even for smaller numbers of hosts. For example, in the experiment in Section 4.1, exactly calculating the optimal set of multicast trees between 8 hosts took about 20 minutes.

Obviously, an approximative solution is required that reduces the linear program significantly. This can be achieved by selecting only a small set of trees, hoping that their combinations will yield throughput results close to the global optimum. For our algorithm, we have chosen to use the set of trees generated by the FPFR heuristics as input to the linear program. FPFR in fact generates a good starting point because of the following properties:

a) When the bottleneck is in the WAN (local bandwidth capacity is much larger than achievable bandwidth), then FPFR generates the optimal set of trees.

b) The opposite case of the problem space is when all local bandwidth capacities and WAN achievable bandwidths are the same. Then, the bottleneck is the local capacity that gets already filled by a single data stream. In this case, FPFR generates a single, linear chain of hosts, which is also optimal.

[2]Cayley's number

c) In the cases in between a) and b) (capacity is somewhat larger than achievable bandwidth), using FPFR's depth-first-search heuristics tends to generate trees with a low average fan out (or: out degree), which lowers the load on a potential capacity bottleneck.

In our experiments, using the set of trees found by FPFR as input, the linear program always took less than one second and resulted in a throughput that was often close to optimal. We can now summarize our algorithm for computing balanced multicast trees as follows:

1) Retrieve the performance monitoring data on bandwidth capacity and achievable bandwidth between the group of hosts, e.g. from a monitoring system like Delphoi [20].

2) Run the FPFR algorithm to generate an initial set of candidate trees, based on the achievable bandwidth information, only.

3) From the result of step 2), construct a linear program for maximizing the overall throughput, which is the sum of the individual throughput values for all trees generated in step 2).

4) Solve the linear program, record the computed throughput values for all trees, and remove those trees with zero throughput from the solution.

Unlike with other approaches (e.g., [3]), it is not necessary to construct trees from the result of our linear program. This is because its input already consists of a set of trees, computed by the FPFR algorithm. We merely use the result of the linear program to determine the optimized send rate of each tree, enforced at runtime by means of traffic shaping, performed by the root node of the multicast operation.

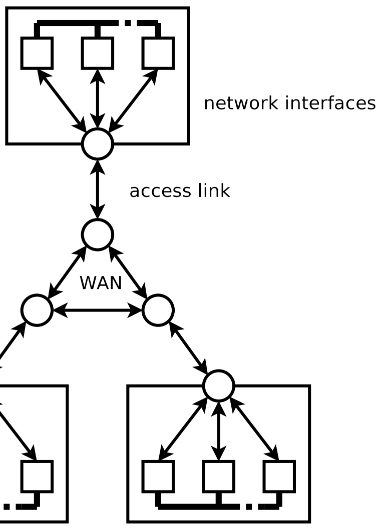


Figure 4: network model including clusters

3.2 Cluster computers

Grid sites are often clusters of hosts, or super computers, with fast local communication capabilities over a separate, high-speed network. Between sites, either regular network interfaces and the Internet or specialized, optical high-performance links are used. The capacity bottleneck between clusters is then easily determined by the individual wide-area network interfaces of the cluster nodes. An example of such an architecture is the Dutch DAS-2 system [9]. However, such a bottleneck can be overcome by dividing the multicast in three steps:

1) Send the data from the root host to all other hosts in its cluster over the fast, local network.

2) Let some of those hosts forward parts of the data to the other clusters using their wide-area network interfaces in parallel.

3) At each destination host, forward the data parts received from the WAN to all other cluster nodes, again over the fast local network.

In the first and last step, we can use the optimal multicast method for the fast local network. The wide-area multicasting can be optimized using *balanced multicasting* by modeling every cluster as a single host. In this way, the local capacity of a cluster is expanded from the capacity of a single network interface to the sum of the capacities of all participating network interfaces, or to the capacity of the shared access link to the WAN, whichever is less.

Figure 4 depicts the network model applied to cluster computers. By simply providing capacity information on cluster basis instead of node basis, we can apply *balanced multicasting* to this case, too.

3.3 Implementation

We implemented *balanced multicasting* on top of Ibis [24], our Java-based Grid programming environment that provides fast local and wide-area communication. Within a cluster, Ibis can use Myrinet to achieve very high throughput, while between clusters it provides multiplexing of data across multiple, parallel TCP streams to boost the achievable wide-area bandwidth [10].

The implementation consists of three logically separated parts that are created in the following order during the run of a program:

1) A `Pool` and a `Gauge` object that provide an abstract interface to information about the environment. The `Pool` object describes which hosts in which clusters are participating in an application run. The `Gauge` object provides a uniform interface to network measurements between those hosts. We implemented several gauges, ranging from reading a static XML description of the environment, via using active probes to measure the network, to retrieving the data from a separate monitoring system. For the experiment in Section 4.1 we used a gauge that obtained measurements from Delphoi [20].

2) A `MulticastMethodFactory` object that implements the tree-generating algorithms. Using the environment information in the `Pool` and `Gauge` object, each algorithm generates a `MulticastMethod` containing one or more multicast trees. For the linear programming part of the balanced multicast trees algorithm we used the QSopt library [2].

3) A `MulticastChannel` object for creating all lower-level communication channels, using the `MulticastMethod` it gets in the constructor.

Ibis' elementary communication primitive is a unidirectional pipe, in which messages are sent from a *send port* to a *receive port*. During connection setup in the multicast channel's constructor, each edge in the trees of the multicast method is translated to such a send/receive port pair. If the edge connects clusters instead of single hosts, multiple send/receive port pairs will be created between the hosts in both clusters. For each WAN connection, TCP is used as the transport protocol.

For the multicast within a cluster, we use a chain connecting all hosts, providing the highest application-level multicast throughput over Myrinet [25]. In the root cluster, we need only one such a chain, but in the other clusters, a chain originating at each host is necessary to locally distribute each of the pieces of data received over the WAN. Figure 5 shows an example connection setup of two clusters with three hosts each; every arrow is a send/receive port pair. The root cluster distributes the data locally over a chain, after which each host sends one-third of the data to the other cluster. Three local chains are then used there to send the data pieces to all other cluster members.

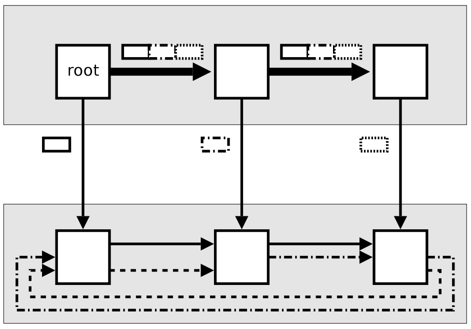


Figure 5: Connections and data flow between two clusters

After all connections are created, data can be multicast by invoking the same method on the `MulticastChannel` object on each host. We only had to implement multicasting of byte arrays, since all Java objects and primitive types have to be serialized to byte streams in the separate serialization layer, before being sent over the network. This allows to split the multicast data array into chunks with sizes proportional to the throughput of each multicast tree.

Each multicast tree then transports its chunk sequentially in small messages, using the zero-copy send method in Ibis. Within a cluster, large messages are used to achieve optimal throughput. In the root's cluster, each host has to receive a steady supply of data from the root to forward across the WAN. The root should therefore iterate round-robin over the chunks that will be forwarded by different hosts when multicasting large messages locally. Each host can then iterate sequentially over its own chunk when sending the smaller WAN messages. Hosts in the second cluster buffer received WAN messages until enough data has been received to fill one large local message, which is then sent to all other hosts. The numbers in Figure 6 show in which order the root host in Figure 5 sends its local and WAN messages.

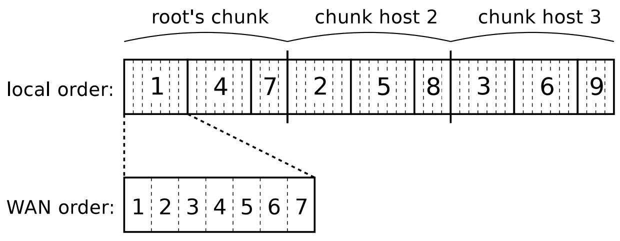


Figure 6: Order of local and WAN messages sent by the root host in Figure 5

Besides splitting the data arrays in chunks of proper sizes, the multicasting also has to ensure that each tree is sending with the precomputed throughput, whereas interferences between multiple streams have to be avoided. The latter is achieved by using one separate thread per outgoing connection. Separate threads also allow us to implement traffic shaping. We implemented the technique from [5] in which threads sleep after sending a message, until it is time for sending the next one. With this technique, we can shape the outgoing traffic precisely to the desired data rates.

4. EVALUATION

We have evaluated *balanced multicasting* using two test cases: using single hosts of the GridLab testbed and using multiple clusters of the Distributed ASCI Supercomputer (DAS). The former compares balanced multicasting with existing approaches, applied to a heterogenous bandwidth environment. The latter examines the added value of the throughput optimization between multiple clusters. We also discuss limitations of our approach.

4.1 Single host test case (GridLab)

The GridLab testbed consists of several sites located in Europe and the US. These sites are shared between multiple users, so we could not get exclusive access to them. To provide a meaningful comparison between multiple multicasting techniques, we decided to emulate the GridLab testbed on one of the DAS clusters. For this purpose, we recorded the network performance information for a given moment in time from the Delphoi system. We used this information for all multicasting techniques while emulating the network performance with an extension of our traffic shaping implementation. This extension simulates both the achievable bandwidth per WAN link and the incoming and outgoing capacity per host. With this setup, we could apply the same network conditions to all multicasting techniques, without interferences by other users.

One observation we made was that, between some GridLab sites, no connection was possible at all, due to misconfigured firewalls. Delphoi reported an achievable bandwidth of zero for such 'dead links', which fitted nicely in our network model (they were simply never chosen to be part of a multicast tree). However, we also wanted to compare our multicast method to a single flat tree, which is the simplest yet widely used implementation. Since such a flat tree can contain dead links, we used semi-flat trees instead. Those are spanning trees with minimum height that do not use dead links. If multiple semi-flat trees existed, we used the one with the highest throughput.

In the experiment, we multicast 200 MB from one of the emulated GridLab sites to all others using four multicast methods: *semi-flat tree*, *maximum bottleneck tree*, *FPFR trees* and *balanced trees.* This was done eight times, once for each site being the root of the multicast. We also calculated the theoretically optimal set of multicast trees for each root by translating all $262,144$ possible multicast trees to a linear program.

Figure 7 shows for each root the throughput of the four different multicast methods and the theoretical maximum throughput. It can be seen that balanced multicasting outperforms the other multicast methods at all root hosts. It also shows the drawback of FPFR, which in some cases performs even worse than using a single multicast tree. Unfortunately, *balanced multicasting* does not always reach the maximum throughput that is theoretically possible. However, calculating the theoretically optimal set of multicast trees took 20 minutes per root, whereas the set of balanced multicast trees was always found in less than a second.

4.2 Cluster test case (DAS)

Our second test case involves multicasting between clusters of the Distributed ASCI Supercomputer (DAS). Those clusters are connected by SURFnet's high-speed backbone of 10 Gb/s. Each compute node is equipped with a Myrinet card for fast local communication and a 100 Mbit FastEthernet card for wide-area communication.

We did two experiments: one with two clusters located in Amsterdam and Leiden, and one with three clusters in Amsterdam, Leiden, and Delft. In the latter case we arranged the clusters in a chain Amsterdam → Leiden → Delft, which is the optimization result when feeding the performance data about the clusters' local bandwidth capacities and the achievable bandwidth into our algorithm. In both experiments, we used *balanced multicast* to transfer 600 MB from the cluster in Amsterdam to the others, using up to eight nodes per cluster.

Figure 8 shows that, in both experiments, the throughput increases almost linearly from one up to six nodes per cluster. With one node per cluster, we reach only 11.2 MB/s due to the FastEthernet cards, but with six nodes per cluster we achieve a total throughput of 62 MB/s from Amsterdam to Leiden: an increase in throughput of 550%. With three clusters, the overhead of forwarding messages causes the throughput to rise a little less quickly, but we still reach 57 MB/s with six nodes per cluster.

Pathchirp reports an available bandwidth of 65 MB/s from Amsterdam to Leiden and 57 MB/s from Leiden to Delft (indicated in Figure 8 as horizontal lines – denoting the upper limits). Deliberately, we extended our tests beyond six nodes, the number of parallel nodes per cluster as proposed by the *balanced multicasting* algorithm, to verify our result. In both the two-cluster and three-cluster cases, the upper limit was almost reached. Using more than six nodes per cluster only created congestion in the bottleneck link, which explains the decrease in throughput, and confirms the results of the *balanced multicasting* algorithm.

Both experiments show that, by combining multiple network interfaces, we can increase the throughput between clusters considerably, while the overhead of forwarding messages remains relatively small. Since this increase in through-

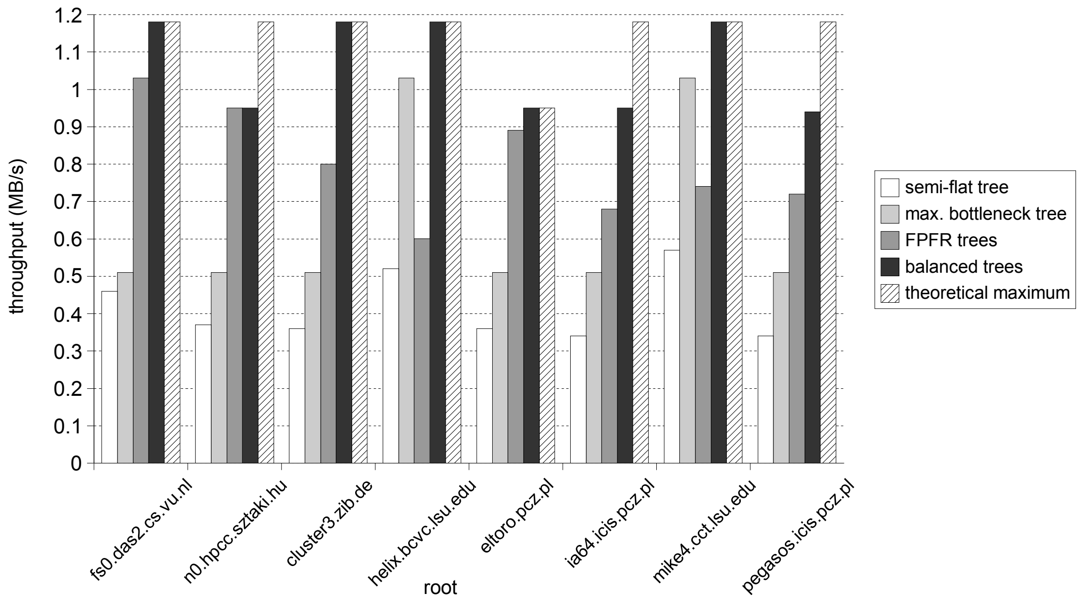


Figure 7: Multicast throughput between eight emulated GridLab sites; each site once multicasts 200MB to all others using four different methods.

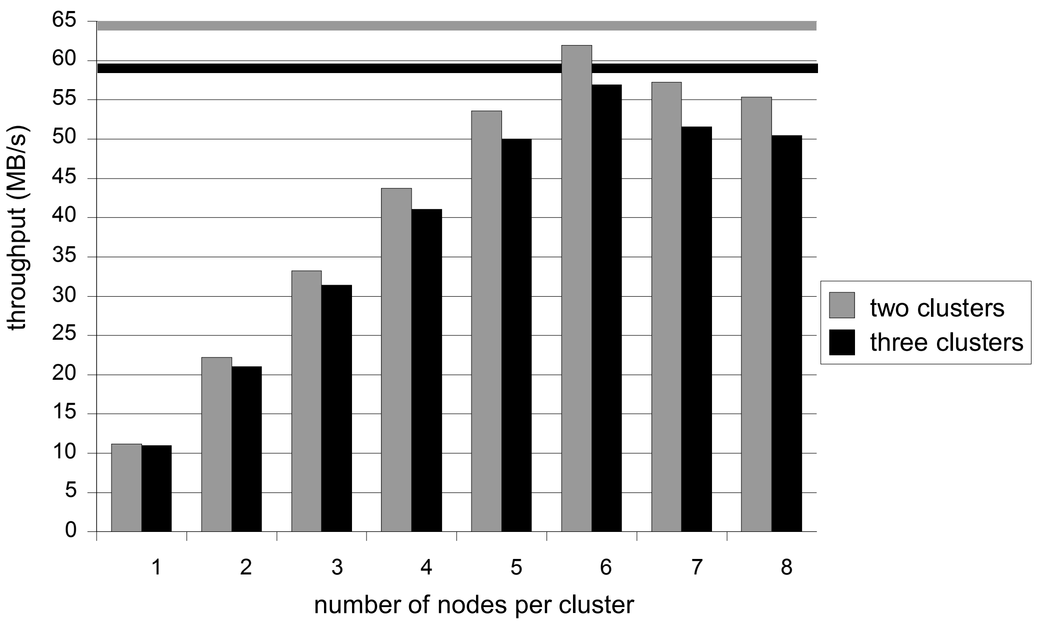


Figure 8: Multicast throughput between two and three DAS clusters.

put starts to expose the limitations of a well provisioned wide-area network, network monitoring information is needed to avoid generating self-induced congestion.

4.3 Limitations

The evaluation of *balanced multicasting* shows promising results. However, it has some limitations concerning scalability and adaptability.

First, with n hosts our network model needs $O(n^2)$ network characteristics. Whether obtaining those characteristics is still feasible when n gets large depends on the scalability of the separate monitoring system.

Second, *balanced multicasting* assumes the environment to be heterogenous, but stable. Adapting to bandwidth fluctuations is done by retrieving a new snapshot of the environment and quickly recalculating the set of multicast trees and their throughput. As our approach is to calculate multicasting trees on the fly, we have to resort to using heuristics instead of aiming at exact solutions.

Since we rely on external monitoring data, the amount of adaptability and the sensitivity to bandwidth fluctuations of *balanced multicasting* depends on the measurement frequency of the separate monitoring system. Between two snapshots, any increase in throughput of some links in the multicasting trees will go unnoticed due to the traffic shaping, but a decrease in throughput will cause the overall throughput of the multicast to drop. Such unexpected decreases in throughput could be a trigger to take a new snapshot of the environment, or in itself be used as a measurement of the achievable bandwidth. Incorporating such adaptability into multicasting is the subject of future work.

Finally, comprehensive analytical results regarding *balanced multicasting* are not available yet. Precise performance bounds relative to the optimal solution are unknown so far. However, it can be argued that the input set of trees used by *balanced multicasting* can always be extended by other, easily computable, candidate trees (like the flat tree or the maximum bottleneck tree). In this sense, the *balanced multicasting* algorithm can always perform at least as well as such other solutions. In practice, however, with our performance experimentation, *balanced multicasting* always outperformed its competitors, and adding other trees was unnecessary.

5. CONCLUSIONS

Because of network heterogeneity, optimization of multicasting communication graphs becomes an NP-hard problem. In this paper, we have proposed *balanced multicasting*, a new heuristic technique for constructing multicasting communication graphs at runtime.

Balanced multicasting combines information about both achievable bandwidth between grid sites and local bandwidth capacity of the individual sites to construct sets of multiple, concurrently used multicasting trees. The balanced multicasting trees efficiently use the achievable bandwidth without suffering from self-induced congestion, as would be caused by over subscribing the local bandwidth capacities. Application-level traffic shaping, done by the multicast root node, enforces the proper balance between the individual multicast trees. Between clusters, throughput can be increased even further by using the fast local network and multiple network interfaces of several cluster nodes in parallel.

Our performance evaluation, both for the testbed of the European GridLab project and between clusters of the Dutch DAS system, shows that *balanced multicasting* outperforms existing approaches by wide margins. We have shown the efficacy of *balanced multicasting* for both the optimization between individual grid nodes and for accumulating the bandwidth capacity of multiple cluster nodes. Combinations of the two cases as well as application of balanced trees to other communication patterns are subject to ongoing work.

Acknowledgements

This work is partially funded by the Dutch National Science Foundation (NWO), grant 631.000.003, "*Network-Robust Grid Applications*." The authors would like to thank Menno Dobber and Evert Wattel for their valuable comments.

6. REFERENCES

[1] G. Allen, K. Davis, K. N. Dolkas, N. D. Doulamis, T. Goodale, T. Kielmann, A. Merzky, J. Nabrzyski, J. Pukacki, T. Radke, M. Russell, E. Seidel, J. Shalf, and I. Taylor, *Enabling Applications on the Grid - A GridLab Overview*, International Journal on High Performance Computing Applications **17** (2003), no. 4, 449–466.

[2] D. Applegate, W. Cook, S. Dash, and M. Mevenkamp, *QSopt Linear Programming Solver*, http://www.isye.gatech.edu/~wcook/qsopt.

[3] O. Beaumont, L. Marchal, and Y. Robert, *Broadcast Trees for Heterogeneous Platforms*, 19th International Parallel and Distributed Processing Symposium (IPDPS'05) (Denver, Colorado), April 3-8 2005.

[4] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, *SplitStream: High-Bandwidth Multicast in Cooperative Environments*, ACM Symposium on Operating System Principles (SOSP) (Lake Bolton, New York), October 2003.

[5] D.M. Chiu, M. Kadansky, J. Provino, and J. Wesley, *Experiences in Programming a Traffic Shaper*, Tech. Report TR-99-77, Sun Microsystems, September 1999.

[6] R. Cohen and G. Kaempfer, *A Unicast-based Approach for Streaming Multicast*, 20th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM 2001) (Anchorage, Alaska), April 22-26 2001, pp. 440–448.

[7] Y. Cui, B. Li, and K. Nahrstedt, *On Achieving Optimized Capacity Utilization in Application Overlay Networks with Multiple Competing Sessions*, 16th annual ACM symposium on parallelism in algorithms and architectures (SPAA '04) (Barcelona, Spain), ACM Press, June 27-30 2004, pp. 160–169.

[8] Y. Cui, Y. Xue, and K. Nahrstedt, *Max-min Overlay Multicast: Rate Allocation and Tree Construction*, 12th IEEE International Workshop on Quality of Service (IwQoS '04) (Montreal, Canada), June 7-9 2004.

[9] *The Distributed ASCI Supercomputer*, http://www.cs.vu.nl/das2/, 2002.

[10] A. Denis, O. Aumage, R. Hofman, K. Verstoep, and T. Kielmann en H.E. Bal, *Wide-Area Communication for Grids: An Integrated Solution to Connectivity,*

Performance and Security Problems, 13th IEEE International Symposium on High-Performance Distributed Computing (HPDC-13) (Honolulu, Hawaii), June 4-6 2004, pp. 97–106.

[11] C. Dovrolis, P. Ramanathan, and D. Moore, *What Do Packet Dispersion Techniques Measure?*, 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2001) (Anchorage, Alaska), April 22-26 2001.

[12] *Grid High-Performance Networking Research Group (GHPN-RG)*, https://forge.gridforum.org/projects/ghpn-rg, Global Grid Forum (GGF).

[13] T. Gross, B. Lowekamp, R. Karrer, N. Miller, and P. Steenkiste, *Design, Implementation and Evaluation of the Remos Network*, Journal of Grid Computing **1** (2003), no. 1, 75–93.

[14] R. Izmailov, S. Ganguly, and N. Tu, *Fast Parallel File Replication in Data Grid*, Future of Grid Data Environments workshop (GGF-10) (Berlin, Germany), March 2004.

[15] 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*, 14th International Parallel and Distributed Processing Symposium (IPDPS '00) (Cancun, Mexico), May 1-5 2000, pp. 377–384.

[16] T. Kielmann, H.E. Bal, S. Gorlatch, K. Verstoep, and R.F.H. Hofman, *Network Performance-aware Collective Communication for Clustered Wide Area Systems*, Parallel Computing **27** (2001), no. 11, 1431–1456.

[17] Thilo Kielmann, Rutger F.H. Hofman, Henri E. Bal, Aske Plaat, and Raoul A.F. Bhoedjang, *MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems*, ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP) (1999), 131–140.

[18] M.S. Kim, S.S. Lam, and D.Y. Lee, *Optimal Distribution Tree for Internet Streaming Media*, 23rd International Conference on Distributed Computing Systems (ICDCS '03) (Providence, Rhode Island), May 19-22 2003.

[19] B. Lowekamp, B. Tierney, L. Cottrell, R. Hughes-Jones, T. Kielmann, and M. Swany, *A Hierarchy of Network Performance Characteristics for Grid Applications and Services*, Proposed Recommendation GFD-R-P.023, Global Grid Forum, 2004.

[20] J. Maassen, R.V. Nieuwpoort, T. Kielmann, and K. Verstoep, *Middleware Adaptation with the Delphoi Service*, AGridM 2004, Workshop on Adaptive Grid Middleware (Antibes Juan-les-Pins, France), September 2004.

[21] M. Nader, J. Al-Jaroodi, H. Jiang, and D. Swanson, *A Middleware-level Parallel Transfer Technique over Multiple Network Interfaces*, ClusterWorld Conference and Expo (CWCE) (San Jose, California), June 23-26 2003.

[22] L. Qiu, Y. Zhang, and S. Keshav, *On Individual and Aggregate TCP Performance*, 7th International Conference on Network Protocols (ICNP '99) (Toronto, Canada), November 1999, pp. 203–212.

[23] V. Ribeiro, R. Reidi, R Baraniuk, J. Navratil, and L. Cottrel, *PathChirp: Efficient Available Bandwidth Estimation for Network Paths*, Passive and Active Measurement workshop (PAM 2003) (La Jolla, California), April 6-8 2003.

[24] R.V. van Nieuwpoort, J. Maassen, G. Wrzesinska, R. Hofman, C. Jacobs, T. Kielmann, and H.E. Bal, *Ibis: A Flexible and Efficient Java-based Grid Programming Environment*, Concurrency & Computation: Practice & Experience **17** (2005), no. 7-8, 1079–1107.

[25] K. Verstoep, K. Langendoen, and H.E. Bal, *Efficient Reliable Multicast on Myrinet*, International Conference on Parallel Processing (Bloomingdale, IL), vol. 3, August 1996, pp. 156–165.

[26] E. Weigle and W. Feng, *A Comparison of TCP Automatic Tuning Techniques for Distributed Computing*, 11th IEEE International Symposium on High-Performance Distributed Computing (HPDC-11) (Edinburgh, Scotland), July 24-26 2002.

[27] R. Wolski, *Experiences with Predicting Resource Performance On-line in Computational Grid Settings*, ACM SIGMETRICS Performance Evaluation Review **30** (2003), no. 4, 41–49.