ISSN:0975-9646

Efficient Multicast for Grid Environment Using Robber and Bit Torrent Protocol

V.Matheswaran[1], S.Usha [2]
[1]*MCA Department, Gnanamani College of Technology,Tamilnadu,India*
[2]*CSE Department,AnnaUniversity of Technology,Tiruchirappalli(Panruti Campus),Tamilnadu,India*

Abstract- Grid computing can be characterized as distributed infrastructure that is a collection of computing resources within or across locations that are aggregated to act as a unified processing resource. In some of the anticipated future Grid applications, the same data will be transmitted to multiple sites. It is widely accepted that this can be, in theory, best achieved using reliable multicast protocols. In this paper we present Robber, a collective, receiver-initiated, high-throughput multicast approach inspired by the BitTorrent protocol. Unlike BitTorrent, Robber is specifically designed to maximize the throughput between multiple cluster computers. Nodes in the same cluster work together as a collective that tries to steal data from peer clusters. Instead of using potentially outdated monitoring data, Robber automatically adapts to the currently achievable bandwidth ratios. Within a collective, nodes automatically tune the amount of data they steal remotely to their relative performance. Our experimental evaluation compares Robber to BitTorrent, to Balanced Multicasting, and to its predecessor MOB. Balanced Multicasting optimizes multicast trees based on external monitoring data, while MOB uses collective, receiver-initiated multicast with static load balancing. Our experimental evaluation shows that our approach outperforms existing multicast strategies by large margins.

Keywords: High-throughput multicast, Load-balancing, Cluster computing, application layer multicast; grid computing; overlay networks.

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 depend primarily on the bandwidth of the interconnection network.[1]. 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.

Using reliable multicast protocols in Grid computing is still an open issue and a lot of researches have been made in recent years [2]. Most of these have aimed at easily porting existing Grid applications to multi-destination environments by enriching TCP with multicast capabilities (*protocol level*) [11]. However, it is worth noting that typically Grid middleware platforms exhibit their own programming interfaces that hide low-level communication APIs, such as sockets. Therefore, TCP extension is useful when an existing application that directly uses TCP has to be modified in order to deliver data to multiple receivers. Another approach could be based on application-aware components, called Active Routers [24], disseminated in specific points of the Grid infrastructure, which are able to handle application-dependent services on incoming data packets (*infrastructure level*), for example to improve the performances of a multicast communication. This approach has some disadvantage of requiring the deployment of specific routers, with ad-hoc execution environments, that limits its application and widespread in the implementation of real Grid systems.

To tackle the above issues, we think that group communication mechanisms can be adopted to easily write applications according to the hierarchical master-slave model. As a consequence, providing a middleware for Grid computing with an effective and efficient implementation of the group abstraction could simplify software development and reduce the communication overhead both in small scale and in large scale networks. In this paper we have presented Robber a receiver initiated multicast approach that combines collective data stealing with load balancing. In Robber, the amount of data a node steals remotely is treated as "work". Initially, the work is divided equally among all nodes in a collective and its steal from local peers. In this way fast nodes will steal more data from slower nodes. However, the problems and solutions proposed can be easily applied to other Grid platforms.

2. MATERIALS &METHODS

In a *multicast* operation, the *root* node is transmitting data to all other nodes of a given group, like the processes of an application. This is comparable to MPI's broadcast operation. For optimizing multicast, we are minimizing the overall completion time, from the moment the root node starts transmitting until the last receiver has got all data. As we are interested in multicasting large data sets, we optimize for high throughput. Thus report our results as achieved throughput (in MB/s). Before presenting our proposed work, Robber algorithm, we first discuss more traditional approaches to multicasting in grids and Internet-based environments. In this section, we also summarize our previous approaches Balanced Multicasting and MOB, as well as some other receiver-initiated multicast approaches, and discuss their performance limitations. We complete our discussion with some background on random stealing, which is used in our Robber algorithm.

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. Several centralized or distributed algorithms have been proposed to find a single overlay multicast tree with maximum throughput [13], [14]. 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 [14], [15], a single multicast tree is used for this purpose. Split Stream [16] 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 multicast approaches try to use the maximum amount of bandwidth the hosts and networks can deliver.

2.2 Network Performance Modeling

Throughout this work, we assume networks as sketched in Fig. 2.2. Nodes are distributed among clusters. Within each cluster, nodes are connected via some local interconnect. Towards the WAN, each node has a network interface that is connected to a shared access link. All access links end at a gateway router (typically to the Internet). Within the WAN, we assume full connectivity among all clusters. For optimizing multicast operations, we need to efficiently use the available network bandwidth where we distinguish, as outlined in [6]. *Bandwidth Capacity* is the maximum amount of data per time unit that a hop or path can carry. *Achievable Bandwidth* is the maximum amount 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. We are interested in maximizing the achievable bandwidth of all data streams used for a multicast operation.

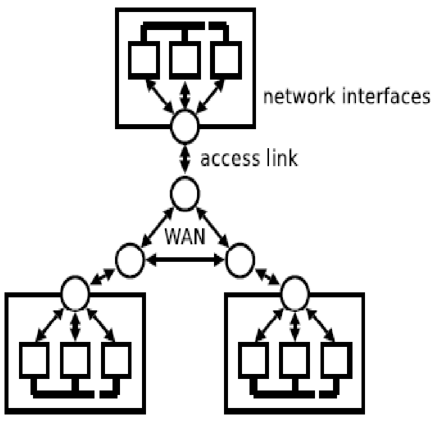


Fig. 1: Network model including clusters

In multicasting, sharing effects can be observed whenever 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 limited either by the network interface (e.g., a Fast Ethernet card, connected to a gigabit network), or by the access link to the Internet that is shared by all machines of a site. In this Section we refer to this setting as a *local bottleneck* environment, dominated by local bandwidth capacity. The opposite situation, where the bottleneck bandwidth is dominated by the achievable bandwidth across the wide-area network, we will call a *global bottleneck* environment.

In order to optimize multicast operations based on the given network characteristics, one has to rely on external network monitoring systems like the Network Weather Service [17], REMOS [18], or Delphoi [19]. Using such tools, however, has its own issues. First of all, the monitoring tools have to be deployed between all clusters in question. Frequently, this is an administrative issue. Second, network bandwidth is measured using active probes (sending measurement traffic) which can take significant amounts of time and scales only poorly to large environments as, for N clusters, O(N2) network paths need to be measured. Consequently, measurements are run in frequencies that are too low to properly follow dynamic bandwidth fluctuations. Finally, network monitoring tools measure the properties of the network paths themselves rather than the properties that are relevant to the applications, namely *achievable bandwidth*. Translating monitoring data to application-level terms is a hard problem [19].

2.3 Optimizing Sender-initiated Multicast

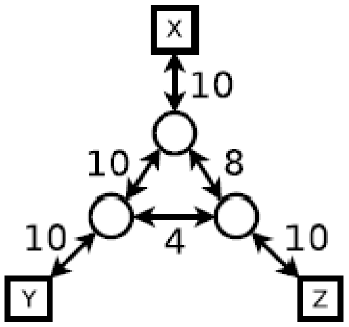
Optimization of multicast communication has been studied extensively within the context of message passing systems and their collective operations. The most basic approach to multicasting is to ignore network information altogether and send directly from the root host to all others. MagPIe [5] used this approach by splitting a multicast in two layers: one within a cluster, and one flat tree between clusters. Such a flat tree multicast puts a high load on the outgoing local capacity of the root node, which often becomes 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 [20] 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 [21]. 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 [22]. As a first optimization for heterogeneous networks, we can take the achievable bandwidth between all hosts into account. The throughput of a multicast tree is then determined by its link

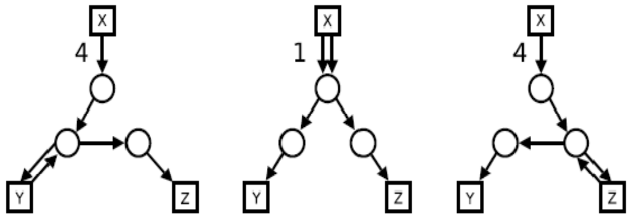
with the least achievable bandwidth. Maximizing this *bottleneck bandwidth* can be done with a variant of Prim's algorithm, which yields the *maximum bottleneck tree* [13].
However, this maximum bottleneck tree is not necessarily optimal because each host also has a certain local capacity. A forwarding 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 NPhardproblem.
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. In theory, this can be reduced to a square number of constraints, but in practice finding the exact solution can be slow and expensive [23]. Any solution thus will have to rely on heuristics to be applicable in real time. 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 [4] is implementing multiple concurrently used multicast trees. FPFR repeatedly uses depth-first search to find a trees panning 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, leading to over subscription of links, forming capacity bottlenecks. In consequence, depending on local capacities, FPFR may perform much worse than expected.

2.4 Balanced Multicasting

In previous work [7], we have presented Balanced Multicasting, improving over FPFR by also taking bandwidth capacity into account. An example is shown in Fig-2(a), consisting 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. (The actual units for bandwidth are not relevant here.)
In this example, Balanced Multicasting creates the three multicast trees shown in Fig. 2(b), with a total achievable bandwidth of 9. Together, these trees maximize the multicast throughput while not oversubscribing individual link capacities. Please note that the individual trees may have different bandwidth, in the example 4, 1, and 4. These different data rates are enforced by traffic shaping at the sender side.



(a) Network example: x sends data to y and z



(b) Balanced multicast trees

Fig. 2: Example of Balanced Multicasting

This process of balancing the bandwidth shares gave the name to the approach. If the sender would not balance the shares of the three trees, then the middle tree (using the LAN at x twice) would consume bandwidth that was intended for the other trees, resulting in a total bandwidth of 3 × 10/4 = 7.5, instead of the anticipated 9. This example shows balanced multicast trees as they are computed by our algorithm published in [7]. Finding the optimal set of balanced multicast trees is an NP-hard problem. For this reason, our implementation is using heuristics to find solutions which we have shown in [7] to be close to the optimum.
When evaluating Robber, we compare to Balanced Multicasting as a (close-to) optimal solution that can be found with complete network performance information. Balanced Multicasting, however, like all other spanning tree based multicasting strategies, is computing its optimized spanning trees based on the monitoring data available at the time when the multicast is started. Later changes in the network will *not* be taken into account.

2.5 Receiver-initiated Multicast

As explained so far, deriving optimized multicast trees is a hard problem. Especially in the case of dynamically changing network performance, carefully computed multicast trees can easily become inefficient. Therefore, several alternatives have been developed based on receiver-initiated communication, in which nodes explicitly request data from each other instead of forwarding it over trees. Bullet [24] takes a hybrid approach to high-throughput multicasting. A Bullet network consists of a tree combined with a mesh overlay. The data is divided in blocks that are further divided in packets. Nodes send a disjoint subset of packets to their children in the tree, and request the remaining pieces from a set of disjoint peers in the system. The selection of those peers is based on random, orthogonal subsets of nodes distributed periodically by the RanSub algorithm. Bullet's additional mesh distribution layer yields significantly better throughput than traditional tree structures. Bit Torrent [8] is a peer-to-peer application, designed to distribute large files efficiently. The data is logically split in P equally-sized pieces, usually a few hundred kilobytes each. Nodes create an overlay mesh by connecting to a few peer nodes chosen at random, and tell each other which pieces they already possess. From then on, nodes constantly inform each other which new pieces they received. Nodes explicitly request pieces from their peers, which are randomly chosen from the reported ones. Each node always has R outstanding requests (we use R = 5) to get the 'pipelining' effect described in [8]. Which peers are allowed to request pieces is decided by the so-called *choking algorithm*. A node 'unchokes' only N peers at the same time (we use N = 5), thereby allowing them to download pieces. The decision to choke or unchoke peers is made every 10 seconds, and is based on the observed download rate. This results in an incentive to upload pieces, since uploading gives a higher chance of being allowed to download.

Chainsaw [9] uses a simplified version of the Bit Torrent protocol, applied to live streaming of data. Nodes have a sliding window of interest, which they advertise to their neighbors. Packets that could not be found in time 'fall off' the trailing edge of the window, and are considered lost.

2.6 Clustering Nodes

Other work has already recognized that grouping receiver-initiated multicast nodes to clusters can increase the overall throughput. *Biased neighbor selection* [25] proposes to group BitTorrent nodes by ISP (Internet Service Provider), which reduces the amount of costly traffic between ISPs. Robber is doing a similar grouping by cluster, but also adds teamwork among the nodes of a cluster to further improve multicast performance. In uncooperative peer-to-peer environments, this improvement would not be possible.

Another approach is followed by Tribler [26], a BitTorrentclient that groups users in social clusters of friends. The amount of trust between nodes is increased using existing relations between people. Users can tag each other as a friend, indicating they are willing to donate upload bandwidth to each other by searching each other's pieces. Robber is essentially an automation of this technique applied to grid clusters, with all nodes in the same cluster being friends. However, the coordination of teamwork in Robber is much more efficient. Robber's predecessor *MOB* [10] is based on the Bit-Torrent protocol. Nodes in the same cluster are grouped to 'mobs'. Each node in a mob steals an equal part of all data from peers in remote clusters, and distributes the stolen pieces locally. This way, each piece is transferred to each cluster only once, which greatly reduces the amount of wide-area traffic compared to BitTorrent. The incoming data is also automatically spread overall nodes in a mob, which works very well when the NICs of the nodes are the overall bandwidth bottleneck instead of the wide-area links. Although MOB achieves good throughput with a small amount of homogeneous clusters, its static load balancing approach fails in larger or more heterogeneous grid environments.

The amount of WAN traffic between clusters of Bit-Torrent nodes can also be decreased using network coding [23]. Nodes then exchange linear combinations of pieces, which increases the probability that a peer in the same cluster has data of interest. However, the complexity and computational overhead of network coding have limited its practical use. The work division in MOB and Robber is much more lightweight and also minimizes the WAN traffic between clusters.

2.7 Random Work Stealing

Random work stealing is a well known load balancing technique used in various distributed computing systems. It can be applied when multiple nodes are solving a computational problem by dividing it into a number of smaller problems. All problems assigned to a node are called its *work*. Each node starts solving all problems assigned to it. When a node becomes idle, it will attempt to *steal* some work from a randomly selected peer, repeating steal attempts until it succeeds. This way, faster nodes will eventually process more work than slower nodes. Robber uses this technique to dynamically spread the bandwidth demand of a multicast operation over nodes in the same cluster.

3. PROPOSED WORK

More recently, receiver-initiated multicast has become popular in peer-to-peer networking. Here, the application nodes are arranged in a random mesh, and explicitly request data from their neighbors. Nodes update each other about which parts of the data they possess, and randomly exchange parts with each other. This way, nodes dynamically route the data over the mesh. The request-reply interaction between nodes automatically adapts the effective throughput to the available bandwidth, which handles heterogeneous and fluctuating WAN bandwidth very well.

Most current receiver-initiated multicast approaches are designed for peer-to-peer systems of individual and uncooperative nodes. In contrast, we apply this approach to grid environments, consisting of multiple clusters of cooperative application nodes.

In such systems, the set of nodes used by a grid application remains constant during a single run, i.e., there is no churn. Any data loss will only affect the achievable bandwidth, which, in turn, is handled by our multicast algorithms.

This paper presents Robber, a successor to MOB that adds dynamic load balancing within a collective. Instead of using a static division of work (the data to request remotely), nodes that have become idle steal work from other nodes in the same collective. As a consequence, each node automatically performs an amount of work proportional to its relative speed in a collective.

This avoids waiting for slow nodes to complete their share of work, greatly enhancing the overall throughput. In large grid environments and with heterogeneous clusters, Robber automatically adjusts the workload of nodes in each cluster to their relative performance, resulting in much better throughput.

To reduce the work load for the users secure data transmission from the server using WLA.

In this section we present *Robber*, a multicast algorithm based on collective data stealing. Robber distributes data over a random mesh by letting nodes 'steal' pieces from other nodes. In addition, nodes in the same cluster team up in collectives that together try to steal pieces from nodes in other clusters as efficiently as possible. The total set of pieces a collective C has to steal from nodes in other clusters is called its work(C). Initially, each node n ε C is assigned an equal share of work, denoted as work (n). Each stolen piece is exchanged locally between members of a collective, such that, at the end, all nodes will have received all data. When a node has no more work left, it attempts to *steal work* from a randomly selected local peer. This way, faster nodes download more pieces to a cluster than slower nodes, which prevents the latter from becoming the overall bandwidth bottleneck.

Fig-3 illustrates the peer selection process. With equally-sized clusters, potential global peers have same collective rank. With different-sized clusters, potential global peers are selected uniformly at random from an equal share of all nodes. This strategy ensures that the connections between nodes in different clusters are well spread out over all clusters and their nodes.

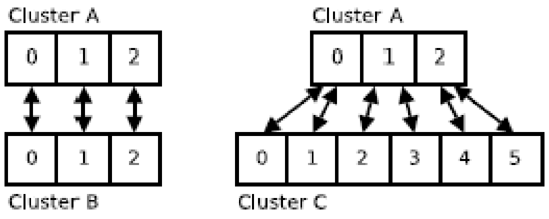


Fig. 3: Examples of peer selection between two clusters of the same size (A and B) and different sizes (A and C). Potential global peers are connected by an arrow.

After all connections are set up, data can be transferred. The data is logically split into P equally sized pieces, numbered 0 to P − 1. At the start of a multicast operation, each node n provides a set of the indices of the pieces it already possesses, denoted as possession(n). In a regular multicast operation, on eroot node possesses everything and the other nodes nothing. Other variations are also possible, like multiroot multicast (where multiple nodes have all data) or striping (where each node in a collective has a part ofall data). As long as there is at least one *'root collective'* in which all nodes together possess all pieces, all nodeswill receive all data. The set of piece indices denoting the pieces a Robber node n wants to steal from a peer p is called its desire(n, p). From local peers, a node desires all pieces it does not have. From global peers, a node n only desires the pieces that are part of its work(n). When a node has received all P pieces, it joins a final synchronization phase. Here, each node keeps serving requests from its peers until this is no longer necessary, so every node is able to finish. A node starts the synchronization phase by sending a *'done'* message to all its peers. Whenever it receives such a message from a peer, it remembers the peer is done. When a node and its peer are both done, they send a final *'stop'* message to each other. This *'stop'* message s the last message sent to a peer in a single multicast operation. When a node receives a *'stop'* message from a peer, it stops listening to it. A node finished a multicast operation once it stopped listening to all its peers. Nodes communicate with their peers using a variant of the BitTorrent protocol [8] using only *bitfield*, *have*, *request* and *piece* messages for stealing data. We add *desire*, *steal*, *work* and *found-work* messages for stealing work. Table 1 summarizes the format of each message.

TABLE 1: Format of messages used by Robber

Message(s)	**Format**
steal, found-work, done, stop	opcode (byte)
have, request	opcode (byte), piece index (integer)
bitfield, desire, work	opcode (byte), piece indices (list of booleans)
piece	opcode (byte), data (list of bytes)

4. Results & Discussion

In Grid environment, this method can be efficient as bandwidth between sites can not vary among network paths. The completion time of large data transfer depends primarily on the bandwidth across network. Fig- 4 shows a high-level view of the software layers in our implementation, and Robber's implementation in more detail. We have implemented Robber, as well as BitTorrent, MOB, and Balanced Multicasting (BM) on top of Ibis, our Java-based Grid programming environment. We used the Smart Sockets library to emulate different clusters inside a single cluster by providing configurable custom routing of data, which we used to evaluate our algorithms. Ibis provides all nodes with the names and ranks of all other nodes and the names of their clusters, which is all the input data Robber and MOB need. Balanced Multicasting also needs network monitoring data information. we use the input for the emulation itself as monitoring data.

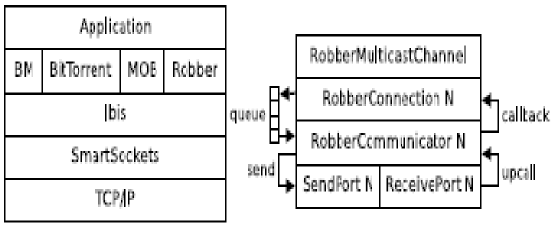


Fig.-4: Software layers and details of Robber's implementation

Each of the four multicast methods is encapsulated in a Multicast Channel object. Fig- 4 shows the implementation of the Robber Multicast Channel object. All connections to peers are created in the constructor. Each connection is abstracted in a Robber Connection object. The underlying Robber Communicator object sends and receives protocol messages using Ibis' *send port* and *receive port* primitives. All received messages are processed in asynchronous up calls provided by Ibis, and translated to individual callback functions in the Robber Connection object on top. Outgoing data is put into a queue and sent in a separate thread per connection to avoid deadlocks.

Data can be multicast by invoking the same method on a Robber Multicast Channel object on each node, which returns when all data has been received. Received data can only be altered after calling Robber Multicast Channel. flush(), which waits until all peers have received all data too.

We emulated the various WAN scenarios in all test cases within one cluster of the Distributed ASCI Supercomputer 3 (DAS-3) [12]. Each node in the DAS-3 is equipped with two 2.4GHz AMD Opterons and a 10Gbit Myrinet network card for fast local communication. Using emulation enabled us to precisely control the environment and subject each multicast method to exactly the same network conditions without any interfering background traffic. This ensured a fair comparison and reproducible results. The emulation only concerns the network performance. All application nodes run the real application code (Ibis and one of the four multicast protocols on top).

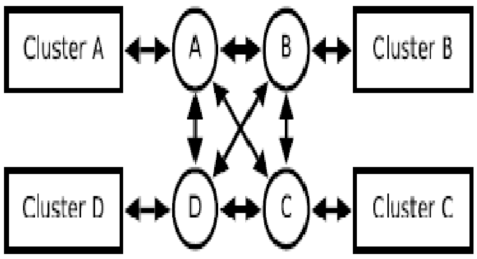


Fig-5: Emulation setup with four clusters. The hub nodes A, B,C, and D route data between clusters and apply traffic control.

We have evaluated Robber by comparing its performance to that of BitTorrent, MOB, and Balanced Multicasting in four test cases. The first two test cases consist of several '*global bottleneck*' scenarios of various dynamics and size. The second two test cases are examples of '*local bottleneck*' scenarios, and show that Robber achieves the same optimized throughput between clusters as Balanced Multicasting, without needing any external monitoring data. Finally, we have analyzed the communication overhead and computational overhead of Robber and MOB. Fig-5 shows the setup of four emulated clusters, as used in the first test case.

The other test cases use an identical setup, except for the amount of clusters. Nodes in the same emulated cluster communicate directly, but traffic between nodes in different emulated clusters is routed via two special *'hub'* nodes using Smart Sockets. Besides routing inter-cluster traffic, the hubs also emulate the wide-area bandwidth and delay between clusters using the Linux Traffic Control (LTC) kernel module to slow down outgoing Myrinet traffic.

Bandwidth is emulated using HTB qdiscs , while one-way delay is emulated using Netem qdiscs. The qdiscs use a default maximum queue length of 1000 packets. The hubs apply simple end-to-end flow control to slow down a source node in case of 'congestion'. All qdiscs together mulate incoming and outgoing capacity of nodes and clusters, and delay and bandwidth between clusters.

The first two test cases evaluate the performance of BitTorrent, MOB, Robber and Balanced Multicasting in various *'global bottleneck'* environments. In the first test case, we compare the performance of all multicast methods under fluctuating wide-area bandwidth. The second test case demonstrates the emergence of slow nodes in larger environments, and shows the benefit of Robber's load balancing strategy.

Test Case 1: *Dynamic Wide-area Bandwidth* The first test case consists of five WAN scenarios with different dynamics:

1) **fast links**: the WAN bandwidth on all links is stable and set.
2) **slow links**: like scenario 1, but with the bandwidth of links A ↔ D and B ↔ C set to 0.8 MB/s.
3) **fast → slow**: like scenario 1 for 30 seconds, then like scenario 2, emulating a drop in throughput on two WAN links.
4) **slow → fast**: like scenario 2 for 30 seconds, then like scenario 1, emulating an increase in throughput on two WAN links
5) **mayhem**: like scenario 1, but every 5 seconds all links randomly change their bandwidth between10% and 100% of their nominal values, emulating heavy background traffic. The random generator is always initialized with the same seed to ensure that this scenario uses identical fluctuations every time.

In all five scenarios, the one-way delay of the wide area links is set to 10 ms. We emulate four clusters of16 nodes each. One root node in cluster A sends 600 MB to all other nodes, using four different multicast methods (BitTorrent, MOB, Robber, and Balanced Multicasting)in the five scenarios described above. In each scenario, Balanced Multicasting uses the exact *initial* emulated bandwidth values as input for its algorithm. We also compute the theoretical maximum throughput in each scenario by converting all 16 possible multicast trees between the four clusters to a linear program. For the maximum throughput in the dynamic scenarios, we assume that the theoretical algorithm can adapt instantly to the optimal strategy in each new situation.

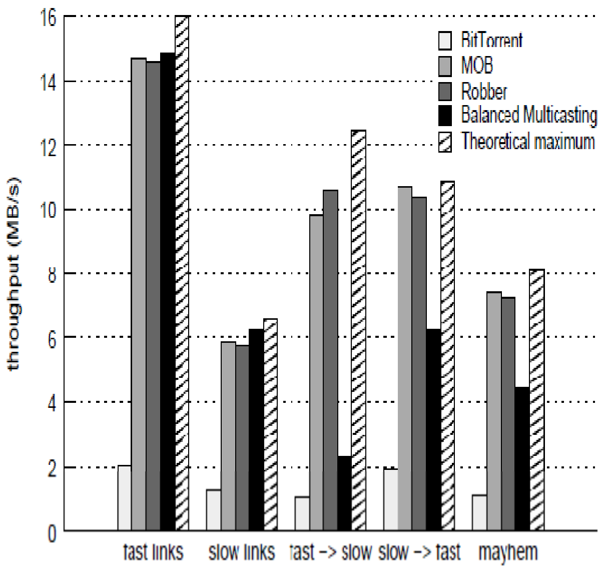


Fig. 6: Multicast throughput between four clusters of 16 nodes each. The root node sends 600MB to all others using four different methods.

Fig.6 shows for each scenario the throughput of each multicast method, calculated as 600 MB divided by the time passed between the moment the root started the multicast and the moment the last node received all data. Table-1 shows these throughput values as a percentage of the theoretical maximum throughput in each WAN scenario. It can be seen that BitTorrent always performs worst, which is caused by the overhead it creates by sending duplicate WAN messages. The difference between MOB and Robber shows the small overhead of Robber's extra load-balancing communication (which is not really needed in this case, since all nodes in each cluster are equally fast). Robber and MOB perform similar to Balanced Multicasting in the static scenarios, and outperform it in all three dynamic ones. In the first dynamic scenario *'fast → slow'*, Balanced Multicasting over uses the links that became slow and does not adapt, for which it is heavily penalized.

In contrast, Robber and MOB adapt and use the other WAN links to distribute the data, resulting in much higher throughput. In the second dynamic scenario *'slow→fast'*, Balanced Multicasting does not use the extra achievable band width that became available on two WAN links due to its sender-side traffic shaping. It therefore achieves the same throughput as in the *'slow links'* scenario, whereas Robber and MOB greedily use the extra bandwidth that became available. In the last dynamic scenario *'mayhem'*, the average throughput of all links is 55% of that in the *'fast links'* scenario. Balanced Multicasting actually achieves 30% of its throughput in the *'fast links'* scenario, since it continuously uses the same WAN links and the throughput of the bottleneck link in each of its multicast trees determines its overall throughput.

TABLE 2: Percentage of the theoretical maximum throughput each multicast method achieves in the five WAN scenarios

Links	BitTorrent	MOB	Robber	BM
Fast links	13%	92%	91%	93%
Slow Links	13%	92%	91%	93%
Fast-> Slow	8%	78%	85%	19%
Slow->Fast	18%	98%	95%	57%
Mayhem	14%	92%	90%	55%

5. Conclusion

In this paper we have presented Robber a receiver initiated multicast approach that combines collective data stealing with load balancing. In Robber, the amount of data a node steals remotely is treated as "work". Initially, the work is divided equally among all nodes in a collective and its steal from local peers. In this way fast nodes will steal more data from slower nodes.

The completion time of large-data multicast transfers depends primarily on the bandwidth an application can achieve across the interconnection network. Traditionally, multicasting is implemented using a sender initiated approach in which the application nodes are arranged in spanning trees over which the data are sent. However, the bandwidth between sites in a grid environment can be significantly different and also dynamically changing. To remain efficient, existing methods compute optimal multicast trees based on network monitoring information. This approach, however, is problematic because it assumes network monitoring systems to be deployed ubiquitously. It also assumes monitored data to be both accurate and stable during a multicast operation, which might not be the case in shared networks with variable background traffic. Our Balanced Multicasting approach indeed suffers from all these problems.

References

[1] CERN, "LHC Computing Grid Project."http://lcg.web.cern.ch/LCG/, July 2004.

[2] M. P. Barcellos, M. Nekovee, M. Daw, J. Brooke, S. Olafsson. Reliable Multicast for the Grid: a Comparisonof Protocol Implementations. Proceedings of the UK E-Science All Hands Meeting, Nottingham(UK), 2004.

[3] O. Beaumont, L. Marchal, and Y. Robert, "Broadcast Trees for Heterogeneous Platforms," in *19th Int. Parallel and Distributed Processing Symposium (IPDPS'05)*, Denver, CO, USA, Apr 3-8 2005.

[4] R. Izmailov, S. Ganguly, and N. Tu, "Fast Parallel File Replication in Data Grid," in *Future of Grid Data Environments workshop (GGF-10)*, Berlin, Germany, Mar 9 2004.

[5] T. Kielmann, R. F. Hofman, H. E. Bal, A. Plaat, and R. A. Bhoedjang,"MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems," *ACM SIGPLAN Symposium onPrinciples and Practice of Parallel Programming (PPoPP)*, pp. 131–140, May 4-6 1999.

[6] 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.

[7] M. den Burger, T. Kielmann, and H. E. Bal, "Balanced Multicasting:High-throughput Communication for Grid Applications," in *Supercomputing 2005 (SC05)*, Seattle, WA, USA, Nov 12-18 2005.

[8] B. Cohen, "Incentives Build Robustness in BitTorrent," in 1st *Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, Jun 5-6 2003.

[9] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr, "Chainsaw: Eliminating Trees from Overlay Multicast," in *4th Int. Workshop on Peer-to-Peer Systems (IPTPS 2005)*, Ithaca, NY, USA, Feb 24-25 2005.

[10] K. Jeacle, J. Crowcroft. Reliable High-speed Grid Data Delivery using IP Multicast. Proceedings of UKE-Science All Hands Meeting, UK, September, 2003

[11] (2006) The Distributed ASCI Supercomputer 3. [Online]. Available: http://www.cs.vu.nl/das3/

[12] R. Cohen and G. Kaempfer, "A Unicast-based Approach for Streaming Multicast," in *20th Annual Joint Conf. of the IEEE Computer and Communications Societies (IEEE INFOCOM 2001)*, Anchorage, AK, USA, Apr 22-26 2001, pp. 440–448.

[13] M. Kim, S. Lam, and D. Lee, "Optimal Distribution Tree for Internet Streaming Media," in *23rd Int. Conf. on Distributed Computing Systems (ICDCS '03)*, Providence, RI, USA, May 19-22 2003.

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

[15] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, andA. Singh, "SplitStream: High-Bandwidth Multicast in Cooperative Environments," in *19th ACM Symposium on Operating System Principles (SOSP-19)*, Bolton Landing, NY, USA, Oct 19-22 2003.

[16] R. Wolski, "Experiences with Predicting Resource Performance On-line in Computational Grid Settings," *ACM SIGMETRICS Perf. Evaluation Review*, vol. 30, no. 4, pp. 41–49, Mar 2003.

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

[18] J. Maassen, R. V. van Nieuwpoort, T. Kielmann, K. Verstoep, and M. den Burger, "Middleware Adaptation with the Delphoi Service," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 13, pp. 1659–1679, Nov 2006.

[19] 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 *14th Int. Parallel and Distributed Processing Symposium (IPDPS '00)*, Cancun, Mexico, May 1-5 2000, pp. 377–384.

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

[21] K. Verstoep, K. Langendoen, and H. Bal, "Efficient Reliable Multicaston Myrinet," in *Int. Conf. on Parallel Processing (ICPP 1996)*, vol. 3, Bloomingdale, IL, USA, Aug 12-16 1996, pp. 156–165. [23] Y. Cui, B. Li, and K. Nahrstedt, "On Achieving Optimized Capacity Utilization in Application Overlay Networks with Multiple Competing Sessions," in *16th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '04)*. Barcelona, Spain:ACM Press, Jun 27-30 2004, pp. 160–169.

[22] D. Kosti´c, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: HighBandwidth Data Dissemination Using an Overlay Mesh," in *19th ACM Symposium on Operating System Principles (SOSP-19)*, Bolton Landing, NY, USA, Oct 19-22 2003.

[23] R. Bindal, P. Cao, W. Chan, J. Medval, G. Suwala, T. Bates, and A. Zhang, "Improving Traffic Locality in BitTorrent via Biased Neighbor Selection," in *26th Int. Conf. on Distributed Computing Systems (ICDCS 2006)*, Lisboa, Portugal, Jul 4-7 2006.

[24] J. Pouwelse, P. Garbacki, J. W. A. Bakker, J. Yang, A. Iosup, D. Epema, M.Reinders, M. van Steen, and H. Sips, "Tribler: A Social-based Peer-to-Peer System," in *5th Int. Workshop on Peer-to- Peer Systems (IPTPS'06)*, Santa Barbara, CA, USA, Feb 27-28 2006.

[25] C. Gkantsidis and P. Rodriguez, "Network Coding for Large ScaleContent Distribution," in *IEEE/INFOCOM'05*, Miami, FL, USA, Mar 13-17 2005.

[26] G. Rajappan, M. Dalal. Reliable Multicast with Active Filtering for istributed Simulations. Proceedingsof Military Communications Conference (MILCOM 2003), Boston, MA, October, 2003.