MPI vs. BitTorrent: Switching between Large-Message Broadcast Algorithms in the Presence of Bottleneck Links

Kiril Dichev and Alexey Lastovetsky

Heterogeneous Computing Laboratory
University College Dublin
Dublin, Ireland
Kiril.Dichev@ucdconnect.ie, Alexey.Lastovetsky@ucd.ie

Abstract. Collective communication in high-performance computing is traditionally implemented as a sequence of point-to-point communication operations. For example, in MPI a broadcast is often implemented using a linear or binomial tree algorithm. These algorithms are inherently unaware of any underlying network heterogeneity. Integrating topology awareness into the algorithms is the traditional way to address this heterogeneity, and it has been demonstrated to greatly optimize tree-based collectives. However, recent research in distributed computing shows that in highly heterogeneous networks an alternative class of collective algorithms - BitTorrent-based multicasts - has the potential to outperform topology-aware tree-based collective algorithms. In this work, we experimentally compare the performance of BitTorrent and tree-based large-message broadcast algorithms in a typical heterogeneous computational cluster. We address the following question: Can the dynamic data exchange in BitTorrent be faster than the static data distribution via trees even in the context of high-performance computing? We find that both classes of algorithms have a justification of use for different settings. While on single switch clusters linear tree algorithms are optimal, once multiple switches and a bottleneck link are introduced, BitTorrent broadcasts – which utilize the network in a more adaptive way – outperform the tree-based MPI implementations.

Keywords: BitTorrent, MPI, Broadcast, Bandwidth, Bottleneck Link.

1 Introduction and Related Work

The broadcast operation distributing data from a root process to all other processes is one of the fundamental collective operations in high-performance computing. Numerous research has been done to develop broadcast algorithms for various platforms over the last 20 years. This work focuses on Ethernet-switched networks – an overview of the most common MPI broadcasts and their performance for Ethernet networks can be found in [15]. The basic existing algorithms in the state-of-the-art MPI implementations are the flat tree (also called round-robin), linear tree (also called daisy chain or pipeline) and binomial tree algorithms.

I. Caragiannis et al. (Eds.): Euro-Par 2012 Workshops, LNCS 7640, pp. 185–195, 2013.

For clusters connected through heterogeneous networks, very different strategies have emerged for large-message broadcasts. Tree-based algorithms can be optimized through topology awareness – however, computation of optimal communication trees is expensive to compute (NP-complete). Heuristics can alleviate this problem, and early work computing communication trees in polynomial time includes [9,2,1]. More recently, we have used models to also generate efficient communication trees [7]. As an alternative, simpler broadcast algorithms can be devised to reflect e.g. a two-layer hierarchy [12].

In the area of peer-to-peer computing, Burger [3,4] has recently demonstrated that receiver-initiated multicasts (of which BitTorrent is an example) can be very effcient. Experiments have shown that this class of algorithms has the potential to outperform even optimized MPI libraries on emulated heterogeneous networks.

Independently from this work, we have recently addressed a related problem in high-performance computing – why try to minimize the network utilization on clusters through complex tree-based algorithms instead of maximizing it? Our objective is the minimization of total communication time, and not the minimization of network utilization. We decided to experiment with the BitTorrent [5] protocol and use it in an unusual setting – for cluster communication. Our research in large message broadcast algorithms on hierarchical and heterogeneous networks shows an interesting and surprising way forward. We show that the protocol has a number of properties which contribute to maximizing the network utilization. We experimentally verify that BitTorrent broadcasts often outperform optimized tree-based broadcasts for different cluster settings. In addition, the original implementation of BitTorrent is oblivious of the network topology – which makes it trivial to deploy anywhere.

Table 1. Optimal large-message broadcast algorithms for the two extremes of homogeneous or heterogeneous networks according to recent research

Underlying network	**Optimal broadcast algorithm**
Very homogeneous	Linear tree algorithm (static)
Very heterogeneous	topology-aware pipelined trees (static) or receiver-initiated multicasts (dynamic)
Some level of heterogeneity	Unknown

Inspired by these developments – summarized in Table 1 – this work builds a bridge between the algorithms used in the high-performance computing domain and the algorithms used in the distributed computing domain for large message broadcasts. In particular, we examine if the described algorithms both have their justification when using settings which are neither fully homogeneous nor very heterogeneous. Some of the questions we answer in this work are:

1. *How sensitive is the performance of operations like linear tree and BitTorrent broadcasts to the heterogeneity of the underlying network? When does it make sense to switch between algorithms?*

This question is not addressed in any previous work to the best of our knowledge. The high-performance computing domain usually assumes a single switched cluster or a hierarchy without any underlying heterogeneity [13]. On the other hand, [3] assumes a high level of heterogeneity, with emulated cluster scenarios differing in the available link bandwidth around 10 times.

2. *Are the results of the related work on large-message broadcasts as shown in Tab. 1 in agreement with experiments on modern grid clusters with a moderate level of heterogeneity?*
   This is an important experimental validation, since this study overlaps with existing research on broadcast operations performed in very different research domains and using different settings.

The paper is structured as follows - in section 2 we highlight the principle differences between the BitTorrent version and the main MPI versions of performing a broadcast operation. In section 3 we present our main experimental settings – with or without a bottleneck link – and our benchmark results. Section 4 concludes the paper.

2 Broadcast in BitTorrent and MPI

2.1 Complexity for Large Message Broadcasts

The classic three algorithms for broadcast operations in MPI – linear, binomial and flat tree algorithm – are described in detail in previous work [15]. A number of variations based on these algorithms exist – for example the scatter/allgather version used in MPICH2 for large messages, where the scatter operation uses a binomial tree, and allgather has a number of variants (e.g. ring implementation).

For large messages, fragmentation and pipelining always yield better performance and variations on that include pipelined linear tree algorithm, pipelined binomial tree algorithm and scatter/allgather-based broadcast. [13] describes pipelined algorithms and observes that on Ethernet switched clusters and for sufficiently large messages, the message size dominates the runtime (rather than the number of processes). Then the theoretical lower limit for a broadcast of a message is the transfer time of this message only between two nodes, since in a pipeline many nodes can overlap their message transfer to each other. The authors perform some simple analysis finding that a pipelined linear tree broadcast without contention and with good segment size comes close to the theoretical lower limit for large messages on single-switch clusters as well as on multi-switch clusters with fully homogeneous network. This is also experimentally confirmed.

In summary: When broadcasting very large messages (Megabytes) across 10s to 100s of processes, efficient algorithms like the linear tree algorithm in MPI have a time complexity of $O(M)$ with M being the message size. With the BitTorrent-based approach, we can only hope to reduce this complexity by a constant factor.

2.2 Algorithm Differences between MPI and BitTorrent Broadcasts

In the following, we focus on the significant differences between the data distribution in BitTorrent as compared to the MPI broadcast algorithms.

- **Pipelining**
  Fragmentation and pipelining always lead to higher parallelism for large messages, and MPI and BitTorrent both use these techniques. The fragment size respectively is best determined at runtime. Our profiling shows that BitTorrent uses fragment size of 16 KB, and Open MPI uses fragment size of 128 KB.
- **Parallel point-to-point transfers at a time**
  Another important parallelization aspect is how different point-to-point calls are parallelized, i.e. how many point-to-point transfers can be performed in parallel through the network. In MPI, the pipelined linear tree algorithm can theoretically provide a parallelism of (p-1) point-to-point calls when the pipeline is full. In BitTorrent, a much denser (but not complete) communication graph can be used. The protocol can provide for 2 * p * c parallel point-to-point calls with c being the allowed active downloads or uploads at a time. For the original client we use, 4 parallel uploads are allowed (more downloads are allowed, but only the lower bound is relevant), i.e. $c = 4$. The constant 2 denotes that BitTorrent allows bidirectional message transfers. In theory, this means that BitTorrent could utilize the network better, particularly parts of the network in the presence of bottlenecks. In practice, 100 % parallelization of all point-to-point transfers can not be achieved, since a node can not physically send or receive data in parallel to all peers - the Ethernet adapter is one example of a serialization point. Furthermore, network saturation and even congestion are a possibility.
- **Dynamic Parallel Access**
  Dynamic parallel access is a significant feature of BitTorrent not present in any of the MPI-based collective algorithms. In a BitTorrent protocol, a process can listen on a socket for a data chunk from several peers in parallel (see Fig. 1). Indeed, this is a very powerful feature which allows for dynamicity and adaptivity in communication as demonstrated for wide-area networks, e.g. in [14]. This is not possible in the sender-initiated MPI collectives where the schedule allows each process to receive data from only one peer.
- **Tree vs. Complete Graph**
  All modern broadcast algorithms in MPI follow a communication tree. The flat tree and the linear tree algorithm can be seen as two opposite extremes of communication trees (with minimal and maximal depth), the binomial tree is another option. On the other hand, the BitTorrent algorithm builds a dense communication graph, in which every process can potentially exchange data with every other process (but only keeps a record of 35 peers in the used implementation).

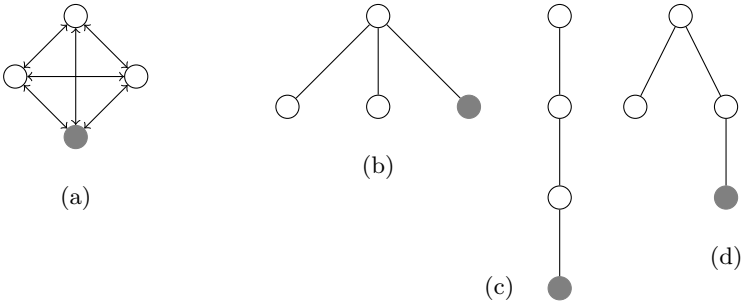


Fig. 1. Point-to-point communication in BitTorrent (a) and MPI broadcasts with flat tree (b), linear tree (c) or binomial tree (d). Only BitTorrent allows a random process (colored in gray) to receive fragments from any other process

3 Experimental Setup and Results

3.1 Bordeaux Site in Grid'5000 as Experimental Platform

We used the 3 clusters Bordereau, Borderline and Bordeplage in Bordeaux in Grid'5000 as experimental platform. Figure 2 shows the network between the clusters. Here, isolated point-to-point bandwidth between any two nodes across the clusters is 1 Gbps (limited by the network interface). However, when intense collective communication is used, the point-to-point throughput decreases significantly across the Dell ProConnect – Cisco connection , because only a single 1 Gbit link connects the two switches. In addition, we measure an increased latency along this link – easily explained with the traversal of more switches. This is the main potential bottleneck. To a lesser extent, the Nortel-HP link also can turn into a bottleneck link for collective communication.

The used setting has significantly less heterogeneous network properties than settings usually used in the distributed computing domain, but we do not consider this a disadvantage. On the contrary, this moderately heterogeneous setting is more typical for high-performance computing.

3.2 Modifications for Profiling BitTorrent and MPI Libraries

We use the original BitTorrent client written by Bram Cohen [5] for our experiments. It has Open Source License and is written in Python. While probably not as efficient as C/C++ compiled binaries, the Python scripts are convenient as proof of concept, and debugging and modifications are comparatively easy. We mostly used the source code itself as well as [10] as a reference for details on the algorithm. The software is available as a package in most modern Linux-based distributions.

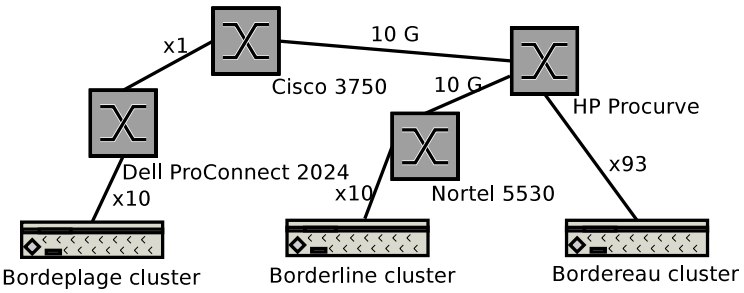


Fig. 2. Ethernet network on Bordeaux site. 10G denotes a single 10 Gigabit link; x1, x10 and x93 denotes Ethernet bonding of 1, 10 or 93 1 Gigabit connections. The Dell–Cisco link is a major bottleneck during intense collective communication between Bordeplage and Borderline or Bordeplage and Bordereau

The following modifications were introduced into the original BitTorrent client:

- File I/O was removed. Instead, dummy data strings are generated on-the-fly and transferred over the network.
- The wall clock time is taken at initiation of the class StorageWrapper and at download completion in the same class. The time difference is used as reference.

For discovering the runtime algorithm and fragment size for pipelining in Open MPI [8] we used PERUSE [11]. PERUSE is an event-driven tracing library for internal MPI events. We used it because the decision making process in MPI collectives is very complex and runtime checks with PERUSE are a reliable way to find which algorithm is used for particular process number and message size. For MPICH2, we use related work and the source code to find which algorithm is being used.

3.3 Timing Mechanism Used in BitTorrent and MPI

In this section, we give a detailed explanation of the timing methodology used consistently both in BitTorrent and MPI experiments. This is important since BitTorrent originally does not provide such timing, while MPI supports timing and logical operations on timings for communication calls.

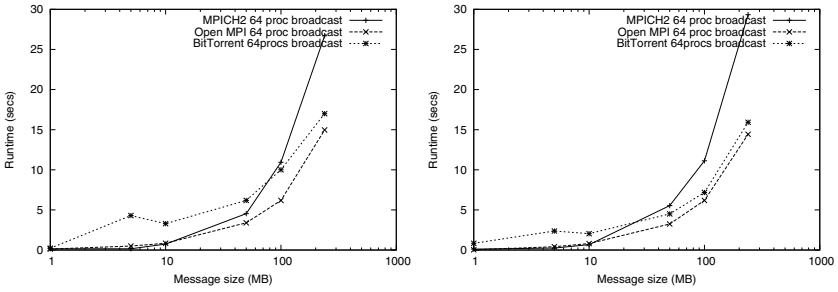
The runtime setup for BitTorrent involves starting a BitTorrent tracker and then launching BitTorrent clients simultaneously identically to MPI program startup. The execution time of a BitTorrent program is then taken as the wall clock time between the start of a StorageWrapper instance and the moment download completion is registered. A BitTorrent client then has to be explicitly terminated since it has no concept of completing a collective communication. In MPI, a barrier call is made, and then the wall clock time is taken before and after the broadcast operation. Then, timing mechanism is as follows:

- In each run and for both types of broadcasts, 64 processes are run, and each of them provides a different wall clock time to finish. As a reference, we take the maximum time between all processes both for BitTorrent and MPI.
- We also perform a number of iterations for each run. As a reference, we take the average of all iterations - again, both for BitTorrent and MPI.

For each message size and each setting, we perform 5 iterations for MPI and BitTorrent.

3.4 Benchmarks of MPI and BitTorrent Broadcasts on a Homogeneous Setting

In the first setting, we test the performance of the presented broadcast algorithms without involving the main bottleneck link. We first use 64 nodes on the Ethernet cluster Bordereau (Fig. 3(a)). Then we use 55 nodes on Bordereau and 9 nodes on Borderline (Fig. 3(b)). We benchmark three broadcast versions - MPICH2, Open MPI and BitTorrent. The used message sizes are 1, 5, 10, 50, 100 and 239 MB. The results for both runs are similar. They demonstrate that for the message sizes 5 MB and 10 MB, MPICH2 marginally outperforms Open MPI, but Open MPI and the linear tree algorithm is most efficient for the rest of message sizes. However, the broadcasts with BitTorrent come very close to the Open MPI broadcasts and even outperform MPICH2 for large messages. This result is unexpected, since the initial assumption is that BitTorrent-based broadcasts are not suitable for homogeneous clusters – however, BitTorrent performs excellently for both settings. The difference to the linear tree broadcast is even minimal for the second run. We interpret this with the fact that this run involves the Nortel-HP link as well and this introduces an increase in network heterogeneity.



(a) (b)

Fig. 3. 64 node broadcasts without the main bottleneck link: using single cluster Bordereau (a) or clusters Bordereau and Borderline (b). On this homogeneous setting, the linear tree algorithm performs best as expected for very large messages, but BitTorrent also performs well.

3.5 Benchmarks of MPI and BitTorrent Broadcasts on More Heterogeneous Settings

In the second setting, we involve the main bottleneck link (Fig. 2) in two different runs. First, we use 32 Bordeplage nodes and 32 Bordereau nodes (Fig. 4(a)). Then, we involve 32 Bordeplage nodes, 25 Bordereau nodes and 7 Borderline nodes (Fig. 4(b)). For the MPI runs, processes are started in the way they are listed. We consider this the most efficient process-to-node assignment for the linear tree algorithm. Inter-cluster communication is minimized, and intra-cluster communication has been proved to be efficient anyway. For MPICH2, there is no simple solution for providing an optimal file for the scatter/allgather algorithm, and we provide the same process-to-node mapping. We use the same broadcast implementations and message sizes as before. The benchmarks show that on both settings, for message sizes of 50 MB and larger BitTorrent (which is oblivious of the topology) outperforms both MPICH2 and Open MPI. A secondary result is that for the same large message range, the scatter-allgather algorithm used by MPICH2 outperforms the linear tree algorithm used by Open MPI.

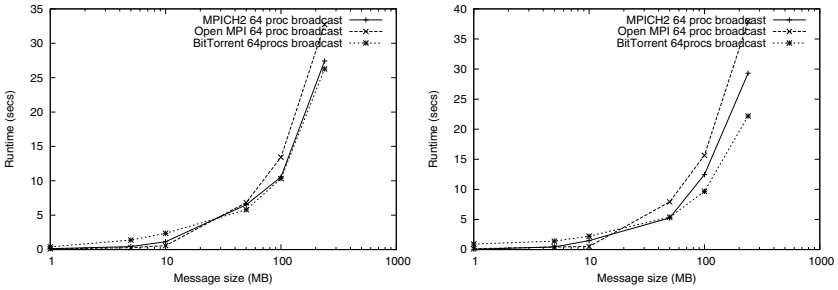


Fig. 4. 64 node broadcasts with main bottleneck link: using clusters Bordeplage and Bordereau (a) or all clusters (Bordeplage, Bordereau and Borderline) (b). This setting has some level of heterogeneity, and BitTorrent performs better than MPI for large messages.

3.6 Interpreting the Results

The performance of the linear tree algorithm of Open MPI decreases significantly with the introduction of a bottleneck link; the throughput decreases by around 50 %. On the other hand, the binomial tree and particularly the BitTorrent algorithm perform excellent. The total amount of received data at all processes does not differ between the different broadcast algorithms. However, the flow of data differs between the three algorithms. Tree-based algorithms statically schedule the transfer of data – e.g. the linear tree algorithm only transfers the exact message size once across the bottleneck link. On the other hand, as described in Sec. 2.2, BitTorrent can use a larger number of parallel point-to-point connections to dynamically schedule the broadcast. This allows the protocol to

utilize the network better. A more detailed analysis confirms that data is broadcast in different ways for BitTorrent and MPI. In Fig. 5, we display a "view" of a 239 MB broadcast with Open MPI and BitTorrent. The dynamics of data movement in a broadcast is difficult to visualize. We choose a view representing the amount of data passing through all switches in any direction during the broadcast. For this purpose we do not use monitoring on the switch level, but a different approach depending on the library we use. For MPI, we know the underlying broadcast algorithm (linear tree) and placement of processes, and we can determine the data movement a-priori. For BitTorrent, as explained in this work, this is not possible, so we measure traffic through additional profiling at each peer instead. Fig. 5 displays the BitTorrent switch traffic in rectangles, and the MPI switch traffic in ellipses. It reveals that inter-cluster communication is more intense with BitTorrent than with MPI – a typical 239 MB broadcast with the setting of Fig. 4(b) transfers around 3,4 GB across the bottleneck switch (in any direction) with BitTorrent, and only the minimal 239 MB with Open MPI. Within clusters however, data exchange with BitTorrent is less intense than with the linear tree algorithm. This behaviour can be partially explained with the fact that BitTorrent is topology-unaware. However, the protocol observes topology to some extent, since the intra-cluster communication is still more intense than inter-cluster communication. Fig. 5 suggests that rather than ignore topology, BitTorrent uses a different ratio between intra- and inter-cluster communication. We conclude that in the presence of bottleneck links, BitTorrent dynamically finds a more efficient schedule for a broadcast than a good tree-based algorithm.

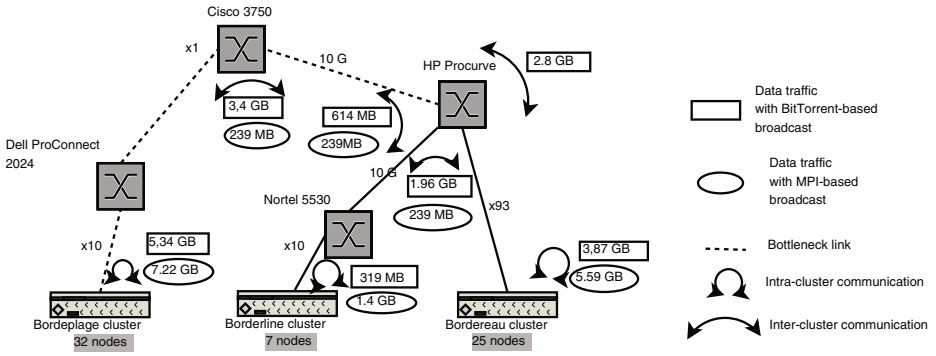


Fig. 5. Data flow in a 239 MB broadcast with BitTorrent. Traffic along switches is shown in rectangles for BitTorrent, and in ellipses for Open MPI.

4 Conclusion

In this work, we compared a BitTorrent broadcast with state-of-the-art MPI broadcasts on switched Ethernet clusters which are either homogeneous or introduce some level of heterogeneity through bottleneck links. The results demonstrate that while in the former setting the linear tree broadcast of Open MPI is

more efficient, for the somewhat heterogeneous setting with a bottleneck link the BitTorrent broadcast outperforms both MPI implementations for large enough messages. We concluded that the intense point-to-point communication in BitTorrent and the resulting adaptive and dynamic schedule of a broadcast can lead to a better performance.

The setting we use is significantly closer to the high-performance computing domain than any previous experimental work using BitTorrent. This means that large-message broadcasts with BitTorrent should be considered in this domain even for moderately heterogeneous networks, and should be preferred for more heterogeneous networks.

Furthermore, the BitTorrent protocol in its tested version is oblivious of the network topology. In a common case in grid and cloud infrastructures, where the network topology is unknown a priori to the user, the BitTorrent protocol is a more sensible choice than MPI for large message broadcasts. If we run applications broadcasting large messages on homogeneous clusters, BitTorrent will be nearly as efficient as modern MPI implementations. If such applications are run on clusters with some level of network heterogeneity, the BitTorrent broadcast will outperform its MPI counterparts.

We refer to our recent work [6], which describes in further detail how BitTorrent's dynamics and adaptivity can be used also in another context – namely for network discovery in grids and clouds.

Acknowledgment. We are thankful to the technical staff at the Bordeaux site of Grid'5000 for the information on the underlying network.

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant Number 08/IN.1/I2054.

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

References

1. Beaumont, O., Marchal, L., Robert, Y.: Broadcast trees for heterogeneous platforms. In: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, p. 80b (April 2005)
2. Bhat, P., Raghavendra, C., Prasanna, V.: Efficient collective communication in distributed heterogeneous systems. In: Proceedings of the 19th IEEE International Conference on Distributed Computing Systems, pp. 15–24 (1999)
3. den Burger, M.: High-throughput multicast communication for grid applications. Ph.D. thesis, Vrije Universiteit Amsterdam (2009)
4. den Burger, M., Kielmann, T.: Collective receiver-initiated multicast for grid applications. IEEE Transactions on Parallel and Distributed Systems 22(2), 231–244 (2011)
5. Cohen, B.: Incentives build robustness in BitTorrent (2003)
6. Dichev, K., Reid, F., Lastovetsky, A.: Efficient and reliable network tomography in heterogeneous networks using BitTorrent broadcasts and clustering algorithms. In: SC 2012 (2012)

7. Dichev, K., Rychkov, V.M., Lastovetsky, A.: Two Algorithms of Irregular Scatter/Gather Operations for Heterogeneous Platforms. In: Keller, R., Gabriel, E., Resch, M., Dongarra, J. (eds.) EuroMPI 2010. LNCS (LNAI), vol. 2536, pp. 289–293. Springer, Heidelberg (2010)
8. Gabriel, E., Fagg, G.E., Bosilca, G., Angskun, T., Dongarra, J.J., Squyres, J.M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R.H., Daniel, D.J., Graham, R.L., Woodall, T.S.: Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In: Kranzlmüller, D., Kacsuk, P., Dongarra, J. (eds.) EuroPVM/MPI 2004. LNCS, vol. 3241, pp. 97–104. Springer, Heidelberg (2004)
9. Hatta, J., Shibusawa, S.: Scheduling algorithms for efficient gather operations in distributed heterogeneous systems. In: Proc. Int Parallel Processing Workshops, pp. 173–180 (2000)
10. Izal, M., Urvoy-Keller, G., Biersack, E.W., Felber, P., Al Hamra, A., Garcés-Erice, L.: Dissecting BitTorrent: Five Months in a Torrent's Lifetime. In: Barakat, C., Pratt, I. (eds.) PAM 2004. LNCS, vol. 3015, pp. 1–11. Springer, Heidelberg (2004), http://dx.doi.org/10.1007/978-3-540-24668-8_1
11. Keller, R., Bosilca, G., Fagg, G., Resch, M., Dongarra, J.: Implementation and Usage of the PERUSE-Interface in Open MPI. In: Mohr, B., Träff, J.L., Worringen, J., Dongarra, J. (eds.) PVM/MPI 2006. LNCS, vol. 4192, pp. 347–355. Springer, Heidelberg (2006), http://dx.doi.org/10.1007/11846802_48
12. Kielmann, T., Bal, H.E., Gorlatch, S.: Bandwidth-efficient collective communication for clustered wide area systems. In: Proc. 14th Int. Parallel and Distributed Processing Symp., IPDPS 2000, pp. 492–499 (2000)
13. Patarasuk, P., Faraj, A., Yuan, X.: Pipelined broadcast on Ethernet switched clusters. In: 20th International Parallel and Distributed Processing Symposium, IPDPS 2006, p. 10 (April 2006)
14. Rodriguez, P., Biersack, E.W.: Dynamic parallel access to replicated content in the Internet. IEEE/ACM Trans. Netw. 10, 455–465 (2002), http://dx.doi.org/10.1109/TNET.2002.801413
15. Wadsworth, D.M., Chen, Z.: Performance of MPI broadcast algorithms. In: Proc. IEEE Int. Symp. Parallel and Distributed Processing, IPDPS 2008, pp. 1–7 (2008)