Fast and Efficient Total Exchange on Two Clusters

Emmanuel Jeannot and Luiz Angelo Steffenel

AlGorille Team, LORIA*
LORIA - Campus Scientifique - BP 239
F-54506 Vandoeuvre-lès-Nancy Cedex, France
Emmanuel.Jeannot@loria.fr, Luiz-Angelo.Steffenel@univ-nancy2.fr

Abstract. Total Exchange is one of the most important collective communication patterns for scientific applications. In this paper we propose an algorithm called $\mathcal{LG}$ for the total exchange redistribution problem between two clusters. In our approach we perform communications in two different phases, aiming to minimize the number of communication steps through the wide-area network. Therefore, we are able to reduce the number of messages exchanged through the backbone to only $2 \times \max(n_1, n_2)$ against $2 \times n_1 \times n_2$ messages with the traditional strategy (where n_1 and n_2 are the number of nodes of each clusters). Experimental results show that we reach over than 50% of performance improvement comparing to the traditional strategies.

1 Introduction

In this paper we address the problem of efficiently perform the *alltoall* (or Total Exchange) communication on two parallel clusters. Total Exchange [1] is one of the most important collective communication patterns for scientific applications, in which each process holds n different data items that should be distributed among the n processes, including itself. An important example of this communication pattern is the `MPI_AlltoAll` operation, where all messages have the same size m.

Although efficient *alltoall* algorithms have been studied for specific networks structures like meshes, hypercubes, tori and circuit-switched butterflies [1,2,3,4], most of the algorithms currently used rely on homogeneous network solutions. These algorithms follow uniform communication patterns between all nodes that prevent an efficient use of both local and distant resources.

In our case, we assume that the parallel application is executed on two different clusters connected by a backbone. This is the case, when for scalability or lack of memory reasons, the application needs to be executed on more than one cluster. In this case, as our experiments will show, standard solutions such as the one implemented in MPI libraries are sub-optimal because they do not

* UMR 7503 CNRS-INPL-INRIA-Nancy2-UHP.

use the fact that the performance of the backbone (latency and bandwidth) is lower than the performance of the interconnection network of the cluster.

The main contribution of this paper is to propose an efficient algorithm for executing a Total Exchange communication pattern on two clusters where the latency of the backbone is a performance constraint. Our strategy consists in exploiting the local-area network performance to reduce the number of inter-cluster communication steps. We have compared this algorithm with the standard *alltoall* implementation from the OpenMPI library [5], and the results show that not only we outperform the traditional algorithm in a grid environment but also that our algorithm is more scalable.

This paper is organized as follows: Section 2 introduces the problem of total exchange between nodes in two different clusters. The related works are presented in Section 3. The algorithm we propose is described in Section 4, while the results of the experiments are given in Section 5. Finally, Section 6 presents some conclusions and the future directions of our work.

2 Problem of Total Exchange between Two Clusters

We consider the following architecture (see Figure 1). Let there be two clusters $\mathcal{C}_1$ and $\mathcal{C}_2$ with respectively n_1 nodes and n_2 nodes. A network, called a backbone, interconnects the two clusters. We assume that a cluster use the same network card to communicate to one of its node or to a node of another cluster. Based on that topology inter cluster communications are never faster than communication within a cluster.

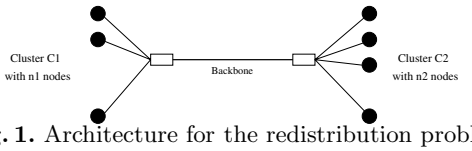

Fig. 1. Architecture for the redistribution problem

Let us suppose that an application is running and using both clusters (for example, a code coupling application). One part of the computation is performed on cluster $\mathcal{C}_1$ and the other part on cluster $\mathcal{C}_2$. During the application, data must be exchanged from $\mathcal{C}_1$ to $\mathcal{C}_2$ using the *alltoall* pattern. *Alltoall* (also called total exchange) is defined in the MPI standard. It means that every node has to send some of its data to all the other nodes. Here we assume that the data to be transfer is different for each receiving node (if the data is the same, the routine is called an *allgather* and is less general that the studied case). Moreover we assume that the size of the data to exchange is the same for every pair of nodes (the case where the size is different is implemented by the *alltoallv* routine: it is more general than our case and will be studied in a future work). Altogether, this means that we will have to transfer $(n_1 + n_2)^2$ messages. The data of all these messages are different but the size of the messages are the same and is given and called m (in bytes).

The question is: how to perform the *alltoall* operation as fast as possible?

Several MPI libraries (OpenMPI, MPICH2, etc.) implement the *allltoall* routine (see Figure 2 for an example). However, these implementations assume that all the nodes are on the same clusters, which means that all the messages have the same importance. However, in our case, some messages are transferred within a cluster (from a node of $\mathcal{C}_1$ to a node of $\mathcal{C}_1$ or form $\mathcal{C}_2$ to $\mathcal{C}_2$) or between the two clusters. In the first case, bandwidth and latency are faster than in the second case. Therefore, there is room for optimizing the transfer time.

```
inbuffer = (void*) calloc (sendcount*size,sizeof(int));
outbuffer = (void*) calloc (recvcount*size,sizeof(int));
(...)
retval = MPI_Alltoall(outbuffer,sendcount,MPI_INT,inbuffer,recvcount,
             MPI_INT,MPI_COMM_WORLD);
(...)
```

Fig. 2. MPI Code extract with an *alltoall* call

3 Related Works

A number of efficient algorithms have been developed for homogeneous clusters with specific network architectures [1,2,3,4]. Generic solutions however rely on direct connections among all nodes, differing only in the communication schedule they use. Indeed, OpenMPI uses a single algorithm that posts all communications and then waits their completion. This algorithm scatters the order of sources and destinations among the processes, so that all processes do not try to send/recv to/from the same process at the same time. At the other hand, MPICH-2 [6] relies on four different algorithms according to the message size and the number of nodes involved in the operation. Indeed, it is worthy of note that the use of the store-and-forward algorithm from Bruck *et al.* [7] for small messages ($m < 256$ *bytes*), which behaves well on situations where the latency dominates the bandwidth [8].

Faraj *et al.*[9], on the other side, propose a scheduling algorithm for switched Ethernet clusters. Their approach consists on scheduling communications to maximize simultaneous connections while avoiding collisions at the network bottlenecks. One drawback of this approach, however, is that it supposes uniform communication steps, which does not hold in the case of heterogeneous networks. A similar approach was presented by Sanders *et al.* [10], who proposed a hierarchical factor algorithm to schedule communications in a cluster of multiprocessor machines where each node can only participate in one communication operation with another node at a time.

Another possibility resides on generating communication schedules according to the communication time between each pair of nodes. Different works propose scheduling heuristics for heterogeneous networks [11,12,13], while Goldman *et al.* [8] studies similar heuristics in the context of homogeneous networks with

irregular messages (*alltoallv*). While these techniques may provide efficient communication schedules, they suffer from two major drawbacks: first, the complexity of the proposed scheduling heuristics - at least $O(n^3)$ - induces an important extra cost to the operation; second, these heuristics require a relatively high heterogeneity among the communications to improve the performance of the *alltoall* operation. Once the bandwidth dominates the latency (as in the case of large messages), the heterogeneity levels are too small to contribute with the scheduling algorithm.

It is important to note that MagPIe [14] does not implement a grid-aware *alltoall*, preferring the standard approach from MPI. Several reasons contribute to this decision, the first one being that the *alltoall* operation does not fit well MagPIe's hierarchical structure. Indeed, as each process should receive a different set of messages, it becomes too expensive to relay these messages through a single cluster coordinator, a problem similar to that of Bruck's algorithm [7].

Hence, to the best of our knowledge, there is no heuristics that efficiently tackles the problem of the Total Exchange on two clusters.

4 Algorithm

4.1 Design principles

In order to construct our *alltoall* algorithm we considered the following design principles:

Multi-level collective algorithms are better suited for grids: The recent efforts for grids divide the set of nodes in different clusters organized in a hierarchical structure, following different communication strategies according to the hierarchical level [15]. Indeed, the local area network is usually faster than the wide-area network, and a careful design allow the algorithms to avoid transmitting data through the slow link connecting two clusters. For instance, a hierarchical approach is essential for ensuring wide area optimality for the collective communication algorithms while performing efficiently on the local network [16].

Wide-area links support simultaneous transfers without performance degradation: Popular algorithms for collective communications on grids (such as the ones implemented in PACX MPI [17] and MagPIe [16]) define a single coordinator in every cluster, which participates in the inter-cluster data transfers across the wide-area backbone. However, this approach is neither optimal concerning the usage of the wide-area bandwidth, nor well adapted to the *alltoall* problem (gathering data from an entire cluster and sending it through the coordinator becomes too expensive [7] and represents a bottleneck in communications). Actually, simultaneous transfers on these links can help in effectively use the WAN bandwidth [18] while reducing the number of communication steps over the slower link. Moreover as experimentally shown in [19], avoiding contention very seldom improves the transfer time.

Avoiding centralization ensures scalability: If we want to target very large scale environment we have to avoid as much as possible the centralization of information such as message size, schedule pattern, etc. That's why we decided to design a fully distributed algorithm that uses only local information and does not synchronize with other nodes.

Actually, most of the complexity of the All-to-All problem resides on the need to exchange *different* messages with each other process. Indeed, the traditional approach consists in establish connections to each other process in the network (local and distant). However, if we assume that the latency between clusters is higher than intra-clusters ones, it might be useful to send data that has to go from one cluster to the other in one single message. Our propose solution is based on this idea and therefore has two phases. In the first phase only local communications are performed. During this phase the total exchange is performed on local nodes on both cluster and extra buffers are prepared for the second (inter-cluster) phase. During the second phase data are exchanged between the clusters. Buffers that have been prepared during the first phase are sent directly to the corresponding nodes in order to complete the total exchange.

More precisely, our algorithm called *Local Group* or simply $\mathcal{LG}$ works as follow. Without loss of generality, let us assume that cluster $\mathcal{C}_1$ has less nodes than $\mathcal{C}_2$ ($n_1 \leq n_2$).

Nodes are numbered from 0 to $n_1 + n_2 - 1$, with nodes from 0 to $n_1 - 1$ being on $\mathcal{C}_1$ and nodes from n_1 to $n_1 + n_2 - 1$ being on cluster $\mathcal{C}_2$. We call $\mathcal{M}_{i,j}$ the message (data) that has to be send form node i to node j. The phases are sum-up in Algorithm 1.

First phase During the first phase, we perform the local exchange: Process i sends $\mathcal{M}_{i,j}$ to process j, if i and j are on the same cluster. Then it prepares the buffers for the remote communications. On $\mathcal{C}_1$ data that have to be send to node j on $\mathcal{C}_2$ is first stored to node $j \bmod n_1$. Data to be sent from node i on $\mathcal{C}_2$ to node j on $\mathcal{C}_1$ is stored on node $\lfloor i/n_1 \rfloor \times n_1 + j$.

Second phase During the second phase only n_2 inter-cluster communications occurs. This phase is decomposed in $\lceil n_2/n_1 \rceil$ steps with at most n_1 communications each. Steps are numbered from 1 to $\lceil n_2/n_1 \rceil$ During step s node i of $\mathcal{C}_1$ exchange data stored in its local buffer with node $j = i + n_1 \times s$ on $\mathcal{C}_2$ (if $j < n_1 + n_2$). More precisely i sends $\mathcal{M}_{k,j}$ to j where $k \in [0, n_1]$ and j sends $\mathcal{M}_{k,i}$ to i where $k \in [n_1 \times s, n_1 \times s + n_1 - 1]$.

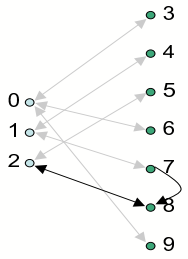

Fig. 3. Example of the 2 phases of the algorithms

Example Suppose that $n_1 = 3$ and $n_2 = 7$. What happens to the message $\mathcal{M}_{7,2}$ (*i.e* the messages that goes from node 7 on cluster $\mathcal{C}_2$ to node 2 on $\mathcal{C}_1$)? This is

Algorithm 1 The $\mathcal{LG}$ (*Local Group*) algorithm when $n_1 \leq n_2$

```
// Local Phase
  for i = {0, ..., (n1 + n2) − 1} do in parallel
    for j = {0, ..., (n1 + n2) − 1} do
      if i < n1 // the sender is on C1
          send M_{i,j} to j mod n1
      else // the sender is on C2
          if j ≥ n1 // the receiver is on C2
              send M_{i,j} to j
          else // the receiver is on C1
              send M_{i,j} to ⌊i/n1⌋ × n1 + j
// Inter-cluster Phase
for s = {1, ..., ⌈n2/n1⌉}
    for i = {1, ..., n1 − 1} do in parallel
      if (i + s × n1 < n1 + n2)
          exchange messages between i and j = i + e × n1
```

illustrated in Figure 3. During the first phase it is stored on node $\lfloor 7/3 \rfloor \times 3+2 = 8$ on $\mathcal{C}_2$. Then during the second phase it is sent to node 2 during the step $s = 2$: node 8 sends $\mathcal{M}_{6,2}$, $\mathcal{M}_{7,2}$ and $\mathcal{M}_{8,2}$ to node 2. During this step nodes 1 and 7 and node 0 and 6 exchange data as well, while in the previous step node 0 and 3, 4 and 1 and 5 and 2 exchange data. Finally, only 0 and 9 exchange data in the last step.

4.2 Comparison with the standard Total Exchange algorithm

As our algorithm tries to minimize the number of inter-cluster communications between the clusters, we need only $2 \times \max(n_1, n_2)$ messages in both directions against $2 \times n_1 \times n_2$ messages in the traditional algorithm. For instance, the exchange of data between two clusters with the same number of process will proceed in one single communication step of the second phase. At the other hand, if $n_2 \gg n_1$, the total number of communication steps will be similar to the traditional algorithm.

5 Experimental validation

To validate the algorithm we propose in this paper, this section presents our experiments to evaluate the performance of the `MPI_Alltoall` operation with two clusters connected through a backbone. These experiments were conducted over two clusters of the Grid'5000 platform [1].

[1] Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see https://www.grid5000.fr).

We used two clusters, one located in Nancy and one located in Rennes, approximately 1000 Km from each other. Both clusters are composed of HP ProLiant DL145G2 nodes (dual Opteron 246, 2 GHz) and are connected by a private backbone of 10 Gbps. All nodes run Linux, with kernel version 2.6.13.

Two different scenarios have been studied. First, we evaluate the performance of the algorithm in a fixed-size grid for different message sizes. In this approach we are able to evaluate the trade-off between local and wide-area communications according to the amount of data to be exchanged. The second scenario considers fixed message sizes while varying the number of processes. Therefore, we are able to study the scalability aspects of the algorithm. Both scenarios were implemented using Open-MPI 1.1.2 [5] in which we implemented our algorithm.

5.1 Communication between two clusters varying the message size

In the first scenario, we compare the completion time of both standard and $\mathcal{LG}$ algorithms when varying the message size. We conducted experiments for messages of size $m = 2^k$ with $k \leq 24$ (16 MB) in two different environments:

In the first experiment, both clusters $\mathcal{C}_1$ and $\mathcal{C}_2$ have the same number of processes (30 processes each). As this situation corresponds to a 1:1 mapping between processes in both clusters, each process will perform at most one message exchange through the backbone.

In the second experiment we consider $\mathcal{C}_1$ and $\mathcal{C}_2$ having different numbers of processes. Hence, we consider $n_1 = 20$ while $n_2 = 40$. In this scenario, the inter-cluster exchange will proceed in two steps, with processes from $\mathcal{C}_1$ exchanging messages with two processes from $\mathcal{C}_2$.

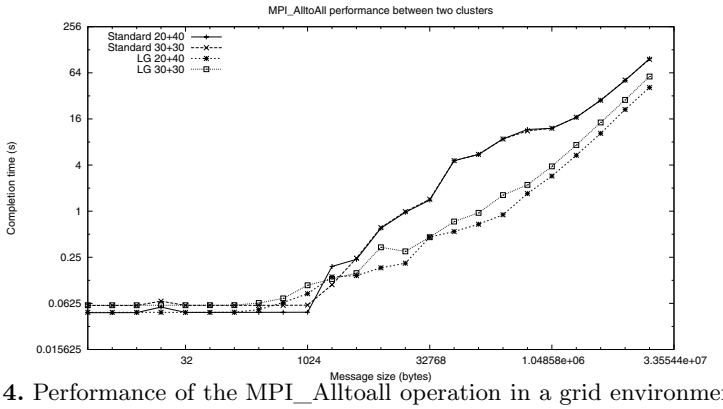

Fig. 4. Performance of the MPI_Alltoall operation in a grid environment

Therefore, in Figure 4 we plot the total communication time obtained for both situations when comparing the performance of the standard `MPI_Alltoall` implementation with the $\mathcal{LG}$ algorithm. Several observations can be made:

- We perform up to 8 times faster than the traditional algorithm according to the message size. Even with large messages we achieve 40% to 60% reduction of communication costs.

– The extra-cost due to message packing observed with small messages is rapidly compensate by the gain on the inter-cluster latency. Indeed, as soon as the traditional algorithm is forced to use more than one single datagram, our strategy presents better results.
– In spite of the different processes distributions, the standard algorithm from OpenMPI does perform almost identically. This evidences the fact that the standard algorithm does not adapts to the network characteristics. At the opposite side we observe that the $\mathcal{LG}$ algorithm performs differently according to the processes distribution, adapting to both scenarios.

5.2 Fixed-size messages varying the number of nodes

While the previous experiment demonstrated that our algorithm performs better than the traditional one due to the reduction of inter-cluster communication steps, we are also interested in the scalability behavior of our algorithm. Indeed, the rational usage of both local and remote links should reduce the network contention that usually characterizes an *alltoall* exchange.

Therefore, this experiment compares the performance of both algorithms when we increase the number of interconnected nodes. We evaluate the overall execution time for two different message sizes, 64kB and 512kB under different processes distributions. Therefore, Figure 5 represents a scenario where $n_1 = n_2$, while Figure 6 represents the performance of the algorithms when $n_2 = 2 \times n_1$.

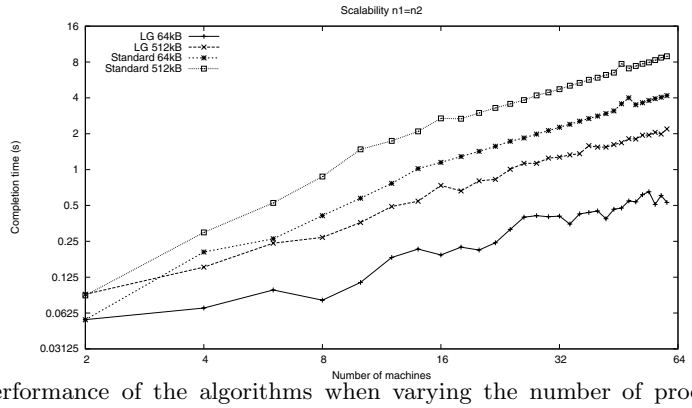

Fig. 5. Performance of the algorithms when varying the number of processes with $n_1 = n_2$

In both experiments we observe that $\mathcal{LG}$ outperforms the standard *alltoall* implemented in Open-MPI. In both graphs we see that the time to perform the *alltoall* with messages of 512kB with $\mathcal{LG}$ is faster than the time to perform the *alltoall* with messages of 64kB with OpenMPI. We see that the slopes of both lines of the $\mathcal{LG}$ algorithm are lower than the slopes of the OpenMPI implementation. This means that, since the y-axis uses a logarithmic scale, the $\mathcal{LG}$ algorithm is more scalable than the OpenMPI implementation.

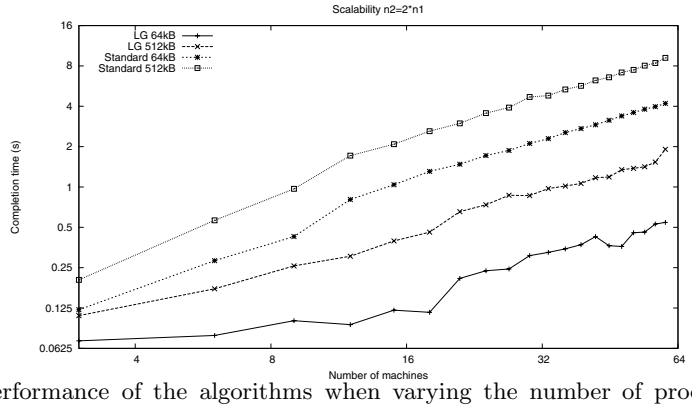

Fig. 6. Performance of the algorithms when varying the number of processes with $n_2 = 2 \times n_1$

6 Conclusions and Future Works

In this paper we have studied and proposed an algorithm called $\mathcal{LG}$ for the total exchange redistribution problem. In our approach we perform communications in two different phases, aiming to minimize the number of communication steps through the wide-area network. Indeed, our algorithm achieves better performances than traditional algorithm on grid environments as it exploits the network heterogeneity to improve the bandwidth utilization in both local and remote networks. Therefore, we are able to reduce the number of messages exchanged through the backbone to only $2 \times \max(n_1, n_2)$ against $2 \times n_1 \times n_2$ messages in the traditional strategy. Further, experiments show a performance improvement of over than 50% comparing to the traditional strategies.

In our future works we plan to extend the model to handle more complex distributions. First, we would like to consider achieving efficient *alltoall* communications with more than two clusters. This would allow efficient communications on general grid environments. Second, we would like to explore the problem of total exchange redistribution when messages have different sizes. This problem, represented by the *alltoallv* routine, is more general than our case and does requires adaptive scheduling techniques.

References

1. Christara, C., Ding, X., Jackson, K.: An efficient transposition algorithm for distributed memory computers. In: Proceedings of the High Performance Computing Systems and Applications. (1999) 349–368
2. Calvin, C., Perennes, S., Trystram, D.: All-to-all broadcast in torus with wormhole-like routing. In: Proceedings of the IEEE Symposium on Parallel and Distributeed Processing. (1995) 130–137
3. Yang, Y., Wang, J.: Optimal all-to-all personalized exchange in multistage networks. In: Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS'00). (2000) 229–236
4. Kalé, L.V., Kumar, S., Varadarajan, K.: A framework for collective personalized communication. In: Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03). (2003)

5. Gabriel, E., Fagg, G.E., Bosilca, G., Angskun, T., Dongarra, 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: Proceedings of the Euro PVM/MPI 2004. LNCS 3241, Springer-Verlag (2004) 97–104
6. Gropp, W.: Mpich2: A new start for mpi implementations. In: Proceedings of the 9th European PVM/MPI Users' Group Meeting, London, UK, Springer-Verlag (2002) 7
7. Bruck, J., Ho, C.T., Kipnis, S., Upfal, E., Weathersby, D.: Efficient algorithms for all-to-all communications in multiport message-passing systems. IEEE Transactions on Parallel and Distributed Systems **8**(11) (November 1997) 1143–1156
8. Goldman, A., Trystram, D., Peters, J.G.: Exchange of messages of different sizes. Journal of Parallel and Distributed Computing **66**(1) (2006) 1–18
9. Faraj, A., Yuan, X.: Message scheduling for all-to-all personalized communication on ethernet switched clusters. In: Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS). (April 2005)
10. Sanders, P., Traff, J.L.: The hierarchical factor algorithm for all-to-all communication. In: Proceedings of the Euro-Par 2002. LNCS 2400, Springer-Verlag (2002) 799–803
11. Bhat, P., Prasanna, V., Raghavendra, C.: Adaptive communication algorithms for distributed heterogeneous systems. In: Proceedings of the IEEE International Symposium on High Performance Distributed Computing (HPDC 1998). (1998)
12. Liu, W., Wang, C.L., Prasanna, V.K.: Portable and scalable algorithms for irregular all-to-all communication. In: Proceedings of the 16th ICDCS. (1996) 428–435
13. Chun, A.T.T., Wang, C.L.: Contention-aware communication schedule for high-speed communication. Cluster Computing: The Journal of Networks, Software Tools and Application **6**(4) (2003) 337–351
14. Kielmann, T., Bal, H., Gorlatch, S., Verstoep, K., Hofman, R.: Network performance-aware collective communication for clustered wide area systems. Parallel Computing **27**(11) (2001) 1431–1456
15. Steffenel, L.A., Mounie, G.: Scheduling heuristics for efficient broadcast operations on grid environments. In: Proceedings of the Performance Modeling, Evaluation and Optimization of Parallel and Distributed Systems Workshop - PMEO'06 (associated to IPDPS'06), Rhodes Island, Greece, IEEE Computer Society (April 2006)
16. Kielmann, T., Hofman, R., Bal, H., Plaat, A., Bhoedjang, R.: Magpie: MPI's collective communication operations for clustered wide area systems. In: Proceedings of the 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. (1999) 131–140
17. Gabriel, E., Resch, M., Beisel, T., Keller, R.: Distributed computing in a heterogenous computing environment. In: Proceedings of the Euro PVM/MPI 1998. LNCS 1497, Springer-Verlag (1998) 180–187
18. Casanova, H.: Network modeling issues for grid application scheduling. International Journal of Foundations of Computer Science **16**(2) (2005) 145–162
19. Jeannot, E., Wagner, F.: Scheduling messages for data redistribution: an experimental study. International Journal of High Performance Computing Applications **20**(4) (November 2006) 443–454