Hierarchical and Reliable Multicast Communication for Grid Systems

N. Ranaldo, G. Tretola, E. Zimeo

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 137-144, 2006.

Hierarchical and Reliable Multicast Communication for Grid Systems

Nadia Ranaldo[a], Giancarlo Tretola[a], Eugenio Zimeo[a]

[a]RCOST - University of Sannio - 82100 Benevento, Italy

Grid computing is emerging in recent years as a viable computing paradigm to solve data and compute-intensive distributed applications. Most of these applications need to transfer the same large amount of data to a wide collection of resources, typically shared among multiple organizations. This paper analyzes the performances of a reliable multicast protocol and the impacts derived from its integration in a Grid middleware platform to efficiently implement one-to-many communication mechanisms easily usable at programming level. At this level, object-oriented, typed groups are used by programmers to transparently exploit the features of the underlying Grid middleware. This way, if the middleware takes into account a hierarchical organization of resources, scalable master-slave applications can be easily written and efficiently executed by using hierarchical groups.

1. Introduction

Thanks to the huge amount of resources available across the Internet and to improvements of wide-area network performance, computational, data and service Grids are becoming effective distributed infrastructures to execute high-performance, general purpose, data and compute-intensive applications. Most of these applications (such as simulations applied to scientific and engineering fields, or data acquisition and analysis from distributed measurement instrumentation and sensors, etc.) deal not only with intensive computations, but also with the management of huge amounts of data that often are transferred to a wide collection of resources. The majority of existing middleware platforms (Globus [9], UNICORE [10], etc.) typically adopt unicast communication mechanisms implemented, independently of the underlying physical network, atop unicast reliable protocols (such as TCP) to perform data transmission. We think that better performance could be achieved through more efficient communication mechanisms. The idea is to perform the data transmission leveraging optimized services of underlying Grid middlewares to exploit the potentialities of the network connections actually available. For instance, a transport layer based on IP multicast should be exploited as an alternative to the commonly used unicast transport communication based on TCP to reduce network traffic and communication overheads in some cases: resources connected through bus-based LANs that support broadcast communication at data-link layer, or a set of workstations directly connected to an IP multicast-enabled router, or indirectly connected through the tunnelling technique.

Using reliable multicast protocols in Grid computing is still an open issue and a lot of researches have been made in recent years [15] [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] [13] [16]. 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 [14], 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 the 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.

In this paper, we propose an approach based on the integration of a reliable multicast protocol into a Grid middleware platform (*middleware level*). The paper discusses the impacts of multicast on the applications by using HiMM (*Hierarchical Metacomputer Middleware*) [7], a Java-based middleware for Grid computing, which delivers basic services of computation, information, communication and resource management to write and execute parallel and distributed object-oriented applications leveraging a virtual hierarchical computing environment. However, the problems and solutions proposed can be easily applied to other Grid middleware platforms.

At programming level, the paper focuses on the requirements for easily programming and executing applications based on the master-slave model, known also as task-farming, or master-worker model. Such model is particularly suited to program in heterogeneous Grid environments [3] where the main issues are: (1) easy-to-use and high-level application programming interface and (2) high performances through an efficient implementation on the available computing and network infrastructures.

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.

The rest of the paper is organized as follows. In Section 2, the integration of a reliable multicast protocol in HiMM is described. Section 3 presents the extended ProActive groups to easily write master-slave applications, and in Section 4 some experimental results are reported. Finally, Section 5 concludes the paper and introduces future work.

2. Reliable Multicast for Hierarchical Grid Computing

HiMM implements basic asynchronous point-to-point and point-to-multipoint communication mechanisms which exploit the hierarchical organization of a Grid hardware architecture. HiMM allows an application to asynchronously send a message to (1) a node at the same level of the sender (*send*); (2) all the nodes at the same level of the sender (*limited broadcast*); (3) all the nodes at the same level of the sender and all the other nodes recursively met in each macro-node belonging to the level of the sender (*deep broadcast*).

All these primitives are implemented on top of a transport layer which provides a one-way communication interface for sending objects (in unicast and broadcast mode). This layer permits to transparently use different transport protocols or other connectivity services, on the basis of the availability. As a consequence, a reliable multicast protocol can be integrated in the multi-protocol transport layer of HiMM to efficiently implement the point-to-multipoint communication mechanisms provided by the primitives “limited broadcast” and “deep broadcast”. When a broadcast primitive is invoked on a node, many partial broadcast invocations take place, one for each protocol domain (PD). A PD is the set of nodes that use the same protocol to communicate.

Each transport adapter provides a specific implementation of the broadcast primitives, which performs point-to-multipoint communication as fast as possible for the specific PD. Each PD can manage one of two possible communication approaches, unicast or multicast. With the first approach, the protocol adapter transmits the data to all nodes of a PD using a sequence of unicast messages; with the second approach, the protocol adapter exploits a reliable multicast protocol to send the same data to the nodes of a PD using a single multicast message.

We implemented and integrated an additional protocol adapter for a reliable multicast protocol that exploits native multicast communication over IP. Among the reliable multicast protocols proposed in the literature, we chose to use TRAM (*Tree-based Reliable Multicast*) [5] because of its tree-based organization of nodes that represents the most scalable method for supporting multicast communication on large scale Grid systems. In fact, TRAM was designed to support the transfer of a large quantity of data and to manage a large number of receivers without affecting the sender activity [8]. The tree-based strategy is the most efficient one for implementing reliable multicast protocols for Grid applications based on the hierarchical master-slave pattern.

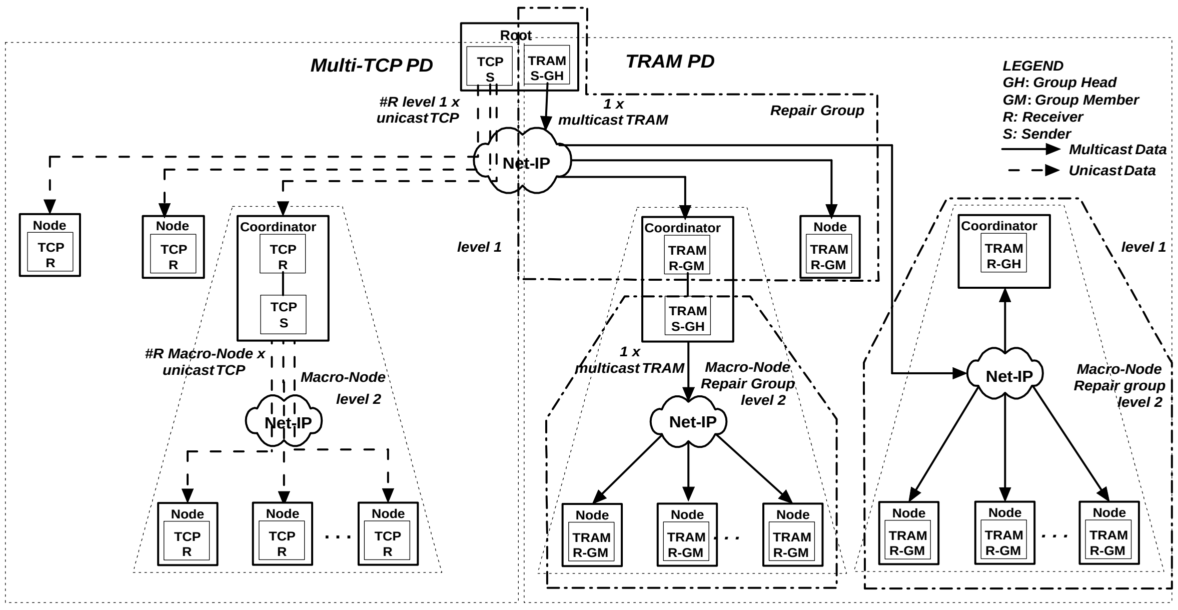

Figure 1. Deep broadcast over a hierarchical architecture with two PDs

According to the hierarchical schema, TRAM ensures reliability by means of a local error recovery mechanism based on repair groups organized according to a tree structure. Each recovery group has a receiver that represents the group head that has to store every message received from the sender in order to allow message recovery when some members of its group notify losses. All the group members receive multicast data from the sender and periodically send ACK messages to their group heads for notifying messages reception or losses. TRAM builds the tree so that group heads are close to their group members. The tree formation is dynamic and is based on TTL values which permits to minimize network bandwidth consumption. On the other hand, this mechanism has some limitations since current multicast routing protocols do not provide accurate TTL values. Moreover, not every group member may be suitable to perform the role of group heads, due to hardware, software or administrative limitations. In HiMM these drawbacks can be overcome if the TRAM repair tree is mapped on a hierarchical structure in which each level includes a special node, the *coordinator*, dedicated to special management functionalities.

Figure 1 shows an example of deep broadcast primitive invoked by the root node of a hierarchical network to send the same data to all the nodes of the system, for example for a master-slave computation. The figure shows two PDs, the Multi-TCP PD and the TRAM PD. The Multi-TCP PD uses the

TCP transport adapter to typically manage communication primitives on the HiMM nodes installed on machines which do not support native IP multicasting. The TRAM PD uses the TRAM transport adapter to efficiently implement the broadcast primitives. Thanks to the mechanisms supported by TRAM for the repair tree formation, each HiMM coordinator can be configured as group head for the nodes of the managed macro-node, while the nodes can be simple group members. The root is also a group head because it is the sender node. On the basis of the effective physical connections among the nodes, the TRAM transport adapter can manage two different cases: (1) a coordinator used as front-end of a pool of machines not directly accessible by the user machine; (2) a coordinator connected to the managed nodes through an IP-multicast enabled network. In the first case, the deep broadcast mechanism requires the coordinator to create its own repair tree in order to reach the nodes of the macro-node. In the second case, the nodes of the macro-node can receive data directly from the sender, and the coordinator, configured as a group head, can be used from the nodes to recover lost data, so reducing the work-load on the sender.

3. Master-slave Computing over Reliable Multicast

The original master-slave pattern [4] may cause scalability problems when it is implemented in a geographically distributed environment and does not give the master the direct visibility and access to the slaves hidden behind front-end computers. A hierarchical version of the master-slave pattern better fits in the architectural features of large Grid systems. In this case, the master at the top controls the overall computation and distributes it among the masters at lower levels, and so on, until the computation is sent to slaves that directly process the request. The collection of results is performed in the reverse order.

To support the master-slave computational model and its hierarchical version, in [1] we proposed to use groups at programming level through an extension of object oriented, typed groups provided by ProActive [6] that allows for the programmability of group behaviors. An extended ProActive group is a group whose behavior can be dynamically configured by means of a set of group semantics, which specify both the behavior of the group and the logical communication models to apply during a method invocation (unicast, multicast and broadcast, that can be unreliable or reliable). Such semantics are (1) *request mapping*, which determines the policy of request dispatching to the group members; (2) *input distribution*, used to determine how to manage the actual input parameters of a method invocation; (3) *synchronization*, used to define the synchronization policy when a method invocation has to return a result; (4) *output collection*, which determines how to reply to the caller the final return value of a group method invocation. The extended group mechanism can be adopted to easily implement the hierarchical master-slave pattern as a generic template to directly use for programming applications. To this end, the four semantics are to be specified as follows: (1) a request has to be dispatched to all the slaves to leverage the overall computational power of the distributed resources; (2) the workload is divided among the slaves by using a scatter policy; (3) the policy has to be “a wait for all the results” since the final result will be the aggregation of all the partial results coming from the members; (4) the final result is built assembling the partial results of the slaves. For the constructor, the input distribution semantic can be implemented with the broadcast policy, since all the members must be the same objects.

4. Experimental Results

To implement multicast communication in HiMM we used the Java-based JRMS framework (*Java Reliable Multicast Service*) [12], which currently provides reliable multicast based on LRMP and

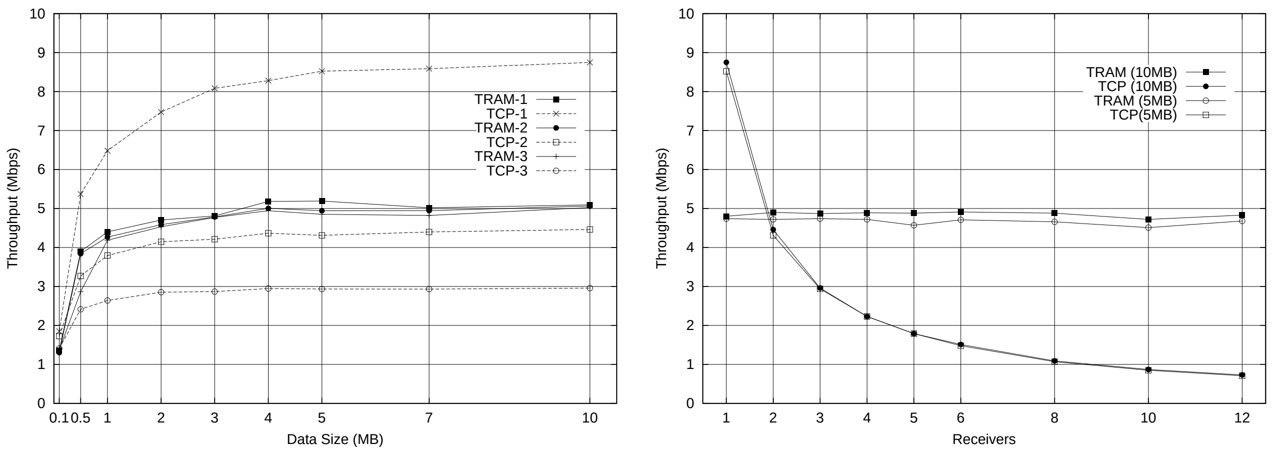

Figure 2. (a) TCP and TRAM throughputs; (b) TRAM scalability

TRAM protocols, but can be easily extended to support other protocols. In this section we analyze the throughput and scalability of TRAM and we show the performance comparison of a master-slave application developed adopting, for the group member creation, the default ProActive group mechanism, based on RMI (which in turn uses the TCP unicast communication mechanism to communicate with each group member), and the extended ProActive group mechanism based on TRAM.

4.1. TRAM throughput and scalability analysis

A first experiment was conducted considering a cluster of eight nodes, each one equipped with two Intel Pentium II 350 MHz, 128 MB RAM and 10/100 Mbps network card, running Microsoft Windows 2000 operating system. The nodes were interconnected through a hub 3COM TP16C at 10 Mbps, and the maximum data rate for the TRAM flow control protocol was fixed to 1.25 MBps, a value that was proven [8] to deliver the best performance. Figure 2.a shows the actual throughput obtained by TRAM and TCP to transfer the same amount of data to one, two and three receivers. This throughput is measured as the data amount divided by the total time to transfer data to all the receivers without losses, where the total time is calculated as the arithmetic mean of multiple measurements. In the case of TCP, the total time is evaluated considering a unicast transmission for each receiver. TRAM delivers a throughput of about 5 Mbps, which is not significantly affected by data dimension and by the number of receivers. TCP delivers a lower throughput starting from two receivers. Figure 2.b shows the throughput obtained by TRAM and TCP to transfer 10 MB and 5 MB of data to a varying number of receivers (from one to twelve). The figure shows a nearly constant trend of the throughput of TRAM with respect to an increasing number of receivers, which proves the good performance of TRAM and its useful adoption as an alternative to TCP for reliable communication mechanisms in Grid computing. Moreover, it is worth noting that the performances of TRAM are not significantly influenced by the message dimension. We stress the fact that the bandwidth efficiency of TRAM is about 50% and it is nearly independent of the number of receivers.

A second experiment was conducted considering the same cluster of nodes interconnected through a switch HP Procurve 2524 at 100 Mbps to evaluate the total transfer times, throughput and scalability of TRAM. In this case the maximum data rate was fixed to 12.5 MBps. Because of the high available physical bandwidth, we pointed up re-transmissions, which instead we did not notice in the previous experiment. Such re-transmissions affect the average total transfer times and as a consequence the throughput. Figure 3.a shows the throughput obtained by TRAM and TCP to transfer the same amount of data to a varying number of receivers: the dotted curve represents the throughput of TRAM obtained in absence of re-transmissions. This curve demonstrates, as for the previous case, a nearly constant trend of the TRAM throughput, which is about 20 Mbps, therefore the bandwidth efficiency is only 20% of the maximum bandwidth. Moreover the figure indicates that in this case TRAM gives better performance than TCP only from five receivers on.

Finally, in a third experiment a network of computers, interconnected through the same switch at 100 Mbps, was used to evaluate the TRAM performance considering heterogeneous and more powerful computers in a non-dedicated environment with a time-varying bandwidth availability. A notebook Pentium M 1.6 GHz with 512 MB RAM was used as sender, and six computers were used as receivers, in particular: a Pentium IV 1.8 GHz with 256 MB RAM, two Pentium IV 2.0 GHz with 512 MB RAM, a Pentium IV 2.4 GHz with 512 MB RAM, two AMD Athlon XP 1800+ 1.5 GHz with 384 MB RAM. All the computers ran Microsoft Windows XP operating system.

Figure 3.b shows the actual throughput obtained by TRAM and TCP to transfer 10 MB of data to a varying number of receivers. It is worth noting that, while TCP is not significantly affected by hardware features, using computers with more CPU speed and RAM capability, TRAM achieves a higher throughput (about 24 Mbps). Also in this case we pointed up re-transmissions in TRAM and, as for the previous case, a nearly constant trend of the TRAM throughput with respect to the number of receivers.

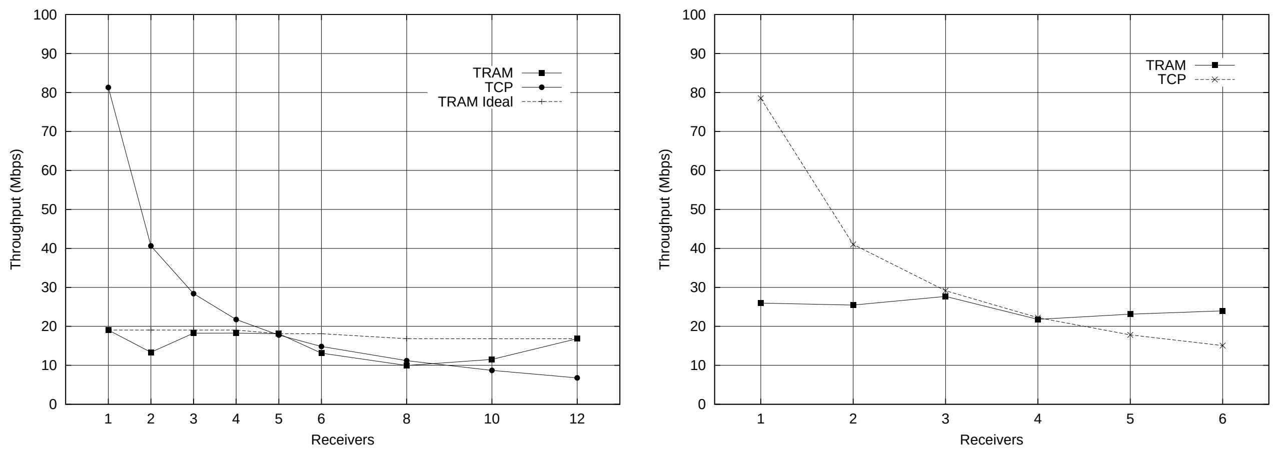

Figure 3. TRAM scalability on a 100 Mbps network of (a) slow homogeneous computers; (b) fast heterogeneous computers

Considering the previous experiment results, we can state that TRAM-based multicast communication can be adopted instead of TCP-based unicast communication over the Internet in order to increase the throughput, whereas for dedicated high-performance networks, the advantage in using TRAM depends on the number of receivers and on the features of adopted network and machines. We are currently studying the parameters and experimental conditions which could affect re-transmissions and the TRAM data flow control in order to improve the TRAM performance and eventually individuate improvements and extensions for specific physical networks and machines.

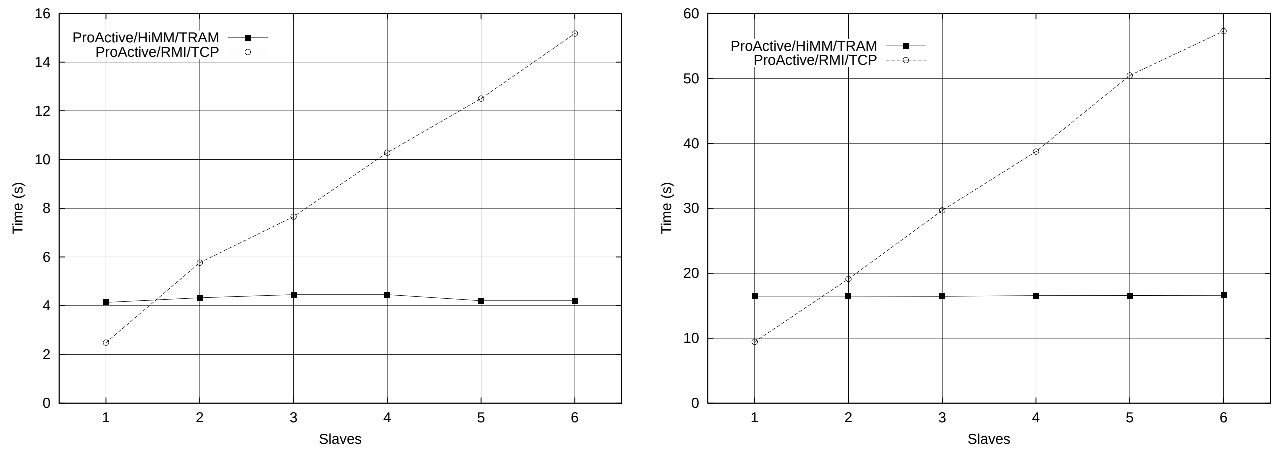

Figure 4. Transfer times for the right matrix of (a) 800x800; (b) 1600x1600

4.2. Master-slave application performance analysis

An experimental analysis of a master-slave application (parallel matrix multiplication) implemented both with extended and native ProActive groups was performed on the first testbed described above and characterized by a 10 Mbps network.

Figure 4 shows the transfer times for the creation of group members, which include the transmission of the right matrix to a varying number of slaves. The native ProActive groups use repeated unicast TCP transmissions whereas the extended Proactive groups leverage the TRAM transport adapter, and so the time for group creation is almost independent of the number of receivers. Figure

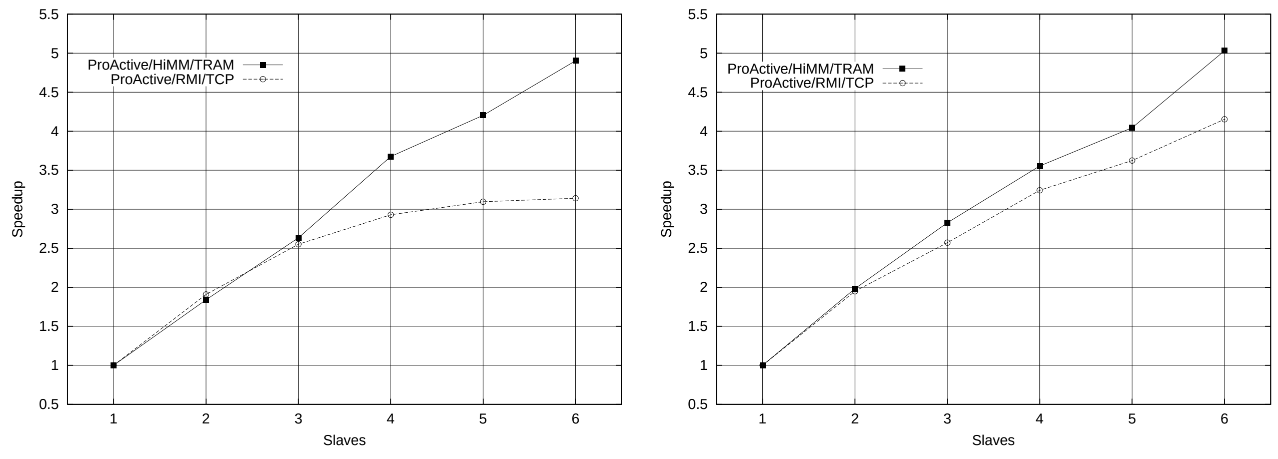

Figure 5. Speedup evaluation for matrices of (a) 800x800; (b) 1600x1600

5 shows relative speedups measured considering the total execution time for the matrix multiplication. The implementation based on reliable multicast exhibits better performances, mainly due to the reduced traffic on the network during the group creation phase. The distribution of the left matrix

and the results collection are performed through unicast communication.

5. Conclusions and Future Work

The paper presented and discussed the impacts derived from using a reliable multicast protocol for the communication in a Grid environment when at application level the master-slave model is adopted. Such model was implemented with the group communication mechanisms provided by an extension of Proactive groups whose performance improvements were clearly demonstrated by the experimental results. An analysis in larger distributed environments will be performed as future work to evaluate the advantage of the hierarchical organization of TRAM on large scale systems. An additional improvement of TRAM will be possible taking into account the structure of the repair-tree, its formation algorithms and other parameters related to the management of the repair buffers.

References

[1] L. Baduel, F. Baude, N. Ranaldo, E. Zimeo. Effective and Efficient Communication in Grid Computing with an Extension of ProActive Groups. Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium, Colorado, 2005.

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

[3] F. Berman. High-Performance Schedulers. in I. Foster and C. Kesselman, editors, The Grid: Blueprint for a New Computing Infrastructure, Chapter 12, Morgan Kaufmann Publishers, July, 1998.

[4] F. Bushmann et al. Pattern-Oriented Software Architecture: A System of Patterns. J. Wiley and Sons, 1996.

[5] D.M. Chiu, S. Hurst, M. Kadansky, J. Wesley. TRAM: A Tree-based Reliable Multicast Protocol. Sun Microsystems Laboratories. SMLI TR-98-68, 1998.

[6] D. Caromel, W. Klauser, J. Vayssiere. Towards Seamless Computing and Metacomputing in Java. Concurrency: Pract Exp., 10(11-13), pp.1043-1061, 1998.

[7] M. Di Santo, N. Ranaldo, E. Zimeo. A Broker Architecture for Object-Oriented Master/Slave Computing in a Hierarchical Grid System. Parallel Computing , Elsevier, Dresda, Germany, September, 2003.

[8] G. Fortino, W. Russo, E. Zimeo. Reliable Multicast Protocols for Java-based Grid Middleware Platforms. IASTED Parallel and Distributed Computing and Systems, California, USA, November, 2003.

[9] I. Foster, C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. International Journal of Supercomputer Applications, Sage Publications, 11(2) pp. 115-128, USA, 1997.

[10] V. Huber. UNICORE: A Grid Computing Environment for Distributed and Parallel Computing. Proceedings of 6th Int. Conf. on Parallel Computing Technologies, Springer, pp. 258-266. Russia, 2001.

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

[12] S. Hanna, M. Kadansky, P. Rosenzweig. Java Reliable Multicast Service Overview. Sun Microsystems Laboratories, SMLI TR-98-68, September, 1998.

[13] S. Liang, D. Cheriton. TCP-SMO: Extending TCP to Support Medium-Scale Multicast Applications, Proceedings of IEEE INFOCOM, pp. 1356-1365, 2002.

[14] G. Rajappan, M. Dalal. Reliable Multicast with Active Filtering for Distributed Simulations. Proceedings of Military Communications Conference (MILCOM 2003), Boston, MA, October, 2003.

[15] B. Van Assche. IP Multicast for PVM on Bus Based Networks. Proceedings of Parallel Computing, Elsevier, pp. 403-410, 1998.

[16] V. Visoottiviseth et al. M/TCP: The Multicast Extension to Transmission Control Protocol. Proceedings of ICACT 2001, Muju, Korea, February, 2001.