Towards an Application-aware Multicast Communication Framework for Computational Grids*

M. Maimour and C. Pham

RESO/LIP - UCB Lyon - ENS Lyon
ENS, Bât. LR5, 46 allée d'Italie
69364 Lyon Cedex 07, France
Congduc.Pham@ens-lyon.fr

Abstract. With a logical view closer to a distributed operating system than a pure communication infrastructure, one might consider extending for computational grids the basic functionalities found in the *commodity Internet*'s network infrastructure to a higher level, thus offering higher value functionalities than the standard IP routing functionality that has basically remained unchanged for 2 decades. In this paper, we report on our early experiences in building application-aware components for a multicast communication protocol targeted for providing low latencies and low overheads for applications running on a computational grid. Performance results from both simulations and implementation prototypes confirm that introducing application-aware components at specific location in the network infrastructure can succeed in providing not only performances for the end-users but also new perspectives in building a communication framework for computational grids.

1 Introduction

The simplest perception one has of a computational grid [4] is a pool of geographically distributed computers that can be accessed and used in a structured manner to solve a given problem. Such grid infrastructures are foreseen to be one of the most critical yet challenging technologies to meet the exponentially growing demands for high-performance computing in a large variety of scientific disciplines: high energy physics, weather prediction, mathematics and combinatorial problems, genome exploration. . . . In the past few years, many software environments for gaining access to very large distributed computing resources have been made available (e.g. Condor [10], Globus [5], Legion [7] to name a few). National and international projects have been launched all around the world to investigate the potential of grid computing: DataGrid (www.eu-datagrid.org), EuroGrid (www.eurogrid.org), GriPhyn (www.gryphyn.org), PPDG (www.ppdg.net) to name some of them.

* This work is supported in part by the french ACI Grid program and by the E-Toile project.

There are many possible utilizations of a grid, not only intensive computing but also on-line access to expensive scientific instruments or huge storage capacity for instance. However the very basic nature of a grid is to allow a large community of people to share information and resources across a network infrastructure. Moving, distributing data from one location to some other locations are operations common to most of grid applications. Today, most of these applications imply a rather small number of participants and it is not clear whether there is a real need for very large groups of users. However, even with a small number of participants, the amount of data can be so huge that the time to complete the transfers can rapidly become unmanageable! More complex, fine-grained applications could have complex message exchange patterns such as collective operations and synchronization barriers.

With a logical view closer to a distributed operating system than a pure communication infrastructure, one might consider extending for computational grids the basic functionalities found in the *commodity Internet*'s network infrastructure to a higher level, thus offering higher value functionalities than the standard IP routing functionality that has basically remained unchanged for 2 decades. In that sense, these ideas are very similar to those of the peer-to-peer (P2P) and overlay community and those of emerging P2P applications (such as Napster or Gnutella) and web services. Going a step further, we want to embed more generally in the network infrastructure some well identified, generic and basic functions for computational grids in the same manner IP routing is a well identified, generic and basic functionality in the commodity Internet. Therefore application-aware components (AAC) are introduced as opposed to traditional Internet routers although the practical realization of such components would certainly be based on "enhanced routers". We think this approach is highly justifiable since a grid, being a specialized subset of the Internet, should have a specialized set of network services for the data transported in the grid. At the moment, it is highly difficult, perhaps impossible, to know exactly what is a typical grid application, if any, but one might be able to exhibit high-level services that could be common to a large number of distributed applications or communication protocols: caching, filtering, interest management, reduction, election... The general description of such a programmable, configurable grid, view as a distributed system with AACs can be found in [9]. More information on P2P for the grids can be found at `http://www.peer-to-peerwg.org` and `http://openp2p.com`.

In this paper, we report on our early experiences in building such application-aware components for a multicast communication protocol targeted for providing low latencies and low overheads for applications running on a computational grid. The AACs are practically realized with the active networking technology as it will be described later on. The rest of the paper is organized as follows. Section 2 presents the AAC technologies and the active networking approach. Section 3 describes the reliable multicast case and Section 4 presents the performance results. Section 5 concludes.

2 Application-aware component technology

The AACs concept calls for an infrastructure of network elements capable of executing specific processing functions, mostly on-the-fly, on data flows. It is possible to build such an infrastructure by involving only end-hosts: it is the peer-to-peer paradigm. However, although this scheme may work, even on large scale systems such as demonstrated by the recently general public P2P applications, there are a number of drawbacks to this approach: redundancy of functionalities, high end-host overhead, limited range of applications...

Recently, a disruptive technology called "active networking" proposes a new vision for the Internet that involves routers (so-called active routers) in the processing of data packets. In active networking [15], routers can execute, through an execution environment similar to an operating system (ANTS [16] for instance), application-dependent functions on incoming packet. With new perspectives both for telco operators and end-users, the use of active network concepts has been proposed in many research areas including multicast protocols [17, 8, 2], distributed interactive simulations [18] and network management [14]. There are many difficulties, however, for deploying in a large scale an active networking infrastructure. Security and performance are two main difficulties that are usually raised. However, active networking has the ability to provide a very general and flexible framework for customizing network functionalities in order to gracefully handle heterogeneity and dynamicity, key points in a computational grid.

In order to implement the AAC concepts, there is no need to go for a complete active networking scheme with an open architecture and the many security pitfalls. It is possible to adopt an operator's perspective which consists in deploying some well identified services (similar to any protocol supported by a router nowadays), chosen for their genericity. However this approach needs a high amount of standardization thus only very generic and core services are likely to be deployed.

3 Case study: reliable multicast on a grid

As a first illustration of a programmable grid infrastructure, we present the work we have done in providing an efficient framework for reliable multicast transfers on a computational grid. Multicast [3] is the process of sending every single packet from the source to multiple destinations in the same logical multicast group. Since most of communications occurring on a grid imply many participants that can be geographically spread over the entire planet, these data transfers could be gracefully and efficiently handled by multicast protocols provided that these protocols are well-designed to suit the grid requirements. Motivations behind multicast are to handle one-to-many communications in a wide-area network with the lowest network and end-system overheads while achieving scalability (end-system or application-multicast solutions can succeed in providing low complexity but lack scalability). In contrast to best-effort multicast, that typically tolerates some data losses and is more suited for real-time audio or video for instance, reliable multicast requires that all packets are safely delivered to the destinations. Desirable features of reliable multicast include, in addition to reliability, low end-to-end delays, high throughput and scalability. These characteristics fit perfectly the need of the grid computing and distributed computing communities. Embedding multicast support in a grid infrastructure would not only optimize the network resources in term of bandwidth saving, but mainly increase both performances for applications, and interactivity for end-users, thus bringing the usage of grids to a higher level than it is at the moment (mainly batch job submission).

3.1 The DyRAM framework

DyRAM [11] is a reliable multicast protocol suite that uses active network technology to off-load basic functionalities traditionally performed by the end-hosts in the commodity Internet. Several high-level mechanisms have thus been identified to improve the performances (latency, scalability) of a multicast communication and have been separated as application-aware services (AAS) to be off-loaded in the network infrastructure (in active routers for the practical realization). Therefore the end-host part of the protocol is quite light and the main AAS consist in:

1. the early detection of packet losses and the emission of the corresponding NACKs.
2. the suppression of duplicated NACKs (from end-hosts) in order to limit the NACK implosion problem.
3. the subcast of the repair packets only to the relevant set of receivers that have experienced a loss. This service limits the scope of the repairs to the affected subtree.
4. the dynamic replier election which consists in choosing a link/host as a replier one to perform local recoveries. Dynamic election provides robustness to host and link failures.

DyRAM has been designed with the following motivations in mind: (*i*) to minimize AAC's load to make them supporting more sessions (mainly in unloading them from the cache of data) and (*ii*) to reduce the recovery latency. Avoiding cache in AACs is performed by a dynamic replier election service. The elected replier in our case is dynamically determined at each lost packet, and not determined at the beginning of the multicast session as in [13] which is only updated when the group membership changes, thus justifying the "dynamic replier" property of the protocol.

3.2 Application-aware components and services

There can be a large variety of grid infrastructures but in most cases they use a similar network architecture: local computing resources are connected together using any kind of local/system area networks (Fast/Giga Ethernet, Fiber Channel, SCI, Myrinet...) and gain access to the rest of the world through one or more routers. So we will mainly assume that the computing resources are distributed across an Internet-based network with a high-speed backbone network in the core (typically the one provided by the telecommunication companies) and several lower-speed (up to 1Gbits/s) access networks at the edge, with respect to the throughput range found in the backbone. Figure 1 depicts such a typical grid infrastructure. For simplicity we represented an access network by a router but practically such networks would contain several routers.

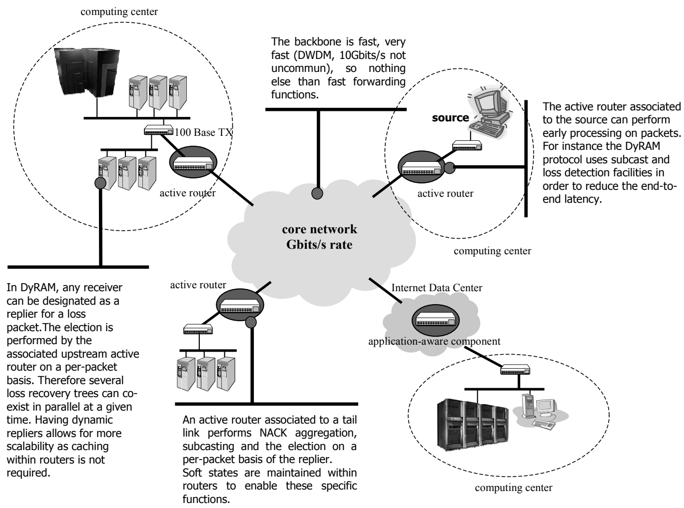

Fig. 1. DyRAM overview.

AACs could be hosted in Internet Data Center (deployed by an operator, difficult at this time) or in private domain, at the initiative of a computing center or a campus for example (very possible now, with AACs based on dedicated hardware such as powerful PC-based active routers). Within AACs, the early loss detection, the NACK suppression, the subcast and the replier election services can be implemented simply by maintaining information about the data packets and NACKs received. This set of information is uniquely identified by the session source and the multicast address. In principle, the nearest router to the backbone is a good location for installing the AAS. Also, in order to take advantage of the fast local recoveries with the replier election mechanism, it would be beneficial to have AACs close to the resource pools.

In the context of the DyRAM framework, all network elements at the edge are assumed to perform at least the NACK suppression and the subcast application-aware services. The AAC located at the sender's side, just before the core network, is called the *source AAC*. A preliminary study, detailed in [12], has shown that the loss detection service is only beneficial if the loss detection capable

AAC is close enough to the source. Consequently the source AAC is the best candidate to perform the loss detection service in addition to the two previous services. The other AACs perform the replier election service.

NACK and data services For each NACK received, the first upstream AAC (either in Internet Data Center or in the private network) creates or simply updates an existing NACK state (NS) structure which has a limited life time. This structure maintains for every "nacked" packet, a subcast list (*subList*) that contains the IP addresses of the receivers that experienced a loss.

On receipt of the first NACK packet for a data packet, the AAC would create a corresponding NS structure, initialize a timer noted DTD (Delay To Decide) that will trigger the election of a replier to whom this first NACK will be sent. All subsequent NACK packets received during the timeout interval are used to properly update the subcast list and are dropped afterward.

DyRAM also requires AAC to keep track of the sequence of received data packets in order to quickly detect packet losses occurring from upstream and affecting all its subtree. This can be done simply by maintaining a track list structure (TL) for each multicast session handled by the AAC. A TL has three components:

- *lastOrdered* is the sequence number of the last data packet received in order.
- *lastReceived* is the sequence number of the last received data packet.
- *lostList* contains the list of data packets not received by the router.

An AAC would detect a loss when a gap occurs in the data packet sequence. The data service, in case no error occurred, then simply consists in updating the track list. On a loss detection, the AAC would immediately generate a NACK packet towards the source. If the AAC has already sent a similar NACK for a lost packet then it would not send a NACK for a given amount of time (based on a timer).

Dynamic replier election Local recoveries, when possible, are performed by elected receivers (repliers) that have correctly received the lost packet. The replier election process executed by the first upstream AAC uses the *subList* in the NS structure to determine which receiver can potentially be elected. The algorithm basically works by comparing in the list of receivers which one does not appear in the *subList* (there is no message exchanges during the election process). However, the AAC does an on-the-fly election that consists in selecting a default replier (the first receiver in the receiver list which is obtained during the initialization session) when the first NACK arrives and updating the choice of the replier at each NACK reception. The advantage of this approach is to perform most of the election processing within the timer duration for gathering NACK information. Once the election is completed, the AAC will forward the NACK downstream toward the elected replier. This latter would then unicast the repair packet to its upstream AAC which would in turn subcast the repair packet on all the links in the subcast list.

4 Performance study

We implemented a simulation model of DyRAM (in the PARSEC language developed at UCLA [1]) and evaluated its performances on a network model. The network model considers one source that multicasts data packets to R receivers through a packet network composed of a fast core network and several slower edge access networks. AACs are located at the edge of the core network as described previously.

4.1 Local recoveries and replier election

Figure 2a and 2b plot for DyRAM the number of retransmissions (M_S) from the source as a function of number the receivers and the completion time[1]. These results have been obtained from simulations of a multicast session with 48 receivers distributed among 12 local groups. The curves show that local recoveries decreases the load at the source as the number of receivers increases. This is especially true for high loss rates.

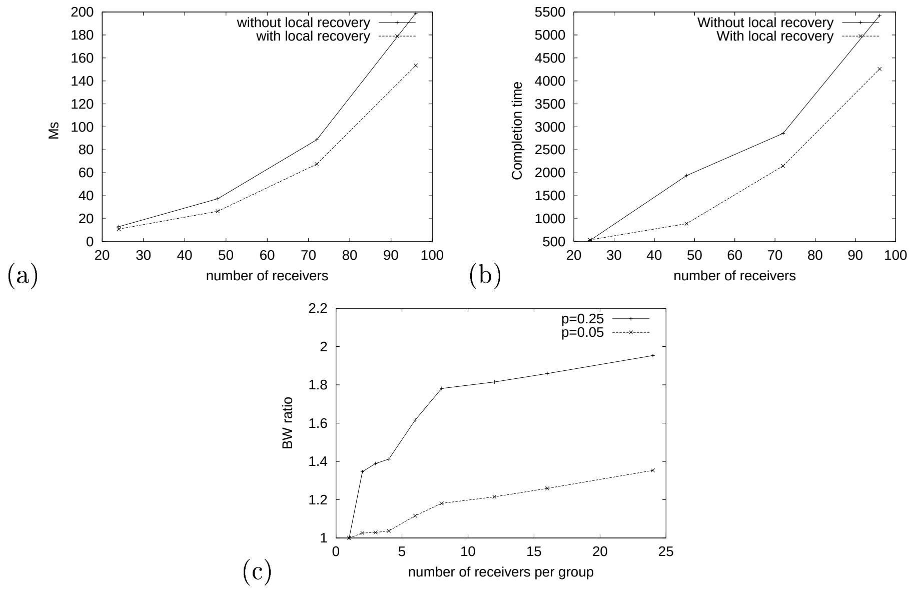

Fig. 2. (a) Load at the source, $p = 0.25$. (b) Completion time, $p = 0.25$. (c) Consumed bandwidth ratio.

Putting the recovery process in the receivers requires at least 2 receivers under a given AAC otherwise local recoveries can not be realized. Therefore the

[1] These 3 curves can be found in [11]. They are provided here for a better understanding of the paper.

local group size (B) is an important parameter. In order to study the impact of B, simulations are performed with the 48 receivers distributed among groups of different sizes and figure 2c shows how much bandwidth (in ratio) can be saved with the local recovery mechanism. As the number of receivers per group increases, the consumed bandwidth is reduced by larger ratios for large loss rates. In fact, when the group size is larger it is more likely that the members of the group can recover from each other. For instance, figure 2c shows that with only 6 receivers per group and local recovery we can achieve a gain of 80%. This result is particularly interesting for distributed applications with many collective and reduction operations.

4.2 Early loss packet detection

The next experiments show how the early loss detection service could decrease the delay of recovery. To do so, two cases noted DyRAM- and DyRAM+ are simulated. DyRAM- has no loss detection services whereas DyRAM+ benefits from the loss detection service in the source AAC. Figure 3 plots the recovery delay (normalized to the RTT) for the 2 cases as a function of the number of receivers. In general, the loss detection service allows DyRAM+ to complete the transfer faster than DyRAM-. For $p = 0.25$ and 96 receivers for instance, DyRAM+ can be 4 times faster.

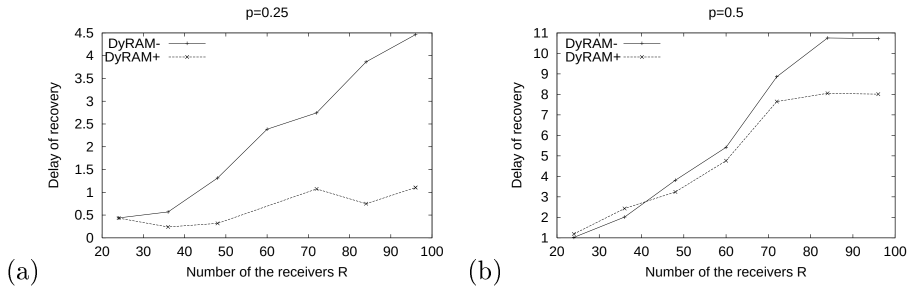

Fig. 3. The normalized recovery delay with (a) $p = 0.25$ and (b) $p = 0.5$

5 Implementation issues

To investigate the cost of introducing AAS within the network infrastructure, a test-bed consisting in a core protocol library for sender and receiver applications, along with the previously described AAS has been developed in Java. The execution environment is Tamanoir [6]. Tamanoir can be installed on a linux-based PC and configured to act like a active router with dynamic download of user code into the execution environment.

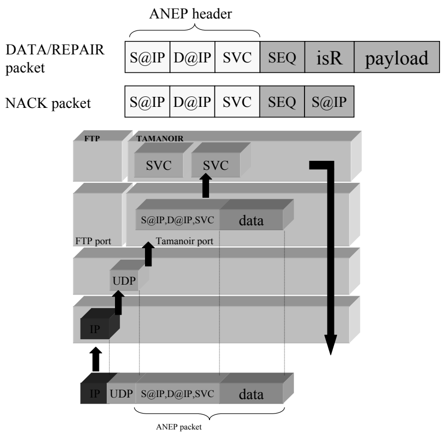

Fig. 4. Packet format and Tamanoir architecture.

5.1 Algorithms, data structures and packet format

The track list (TL) is implemented as a double-linked list of bits (based on array implementation) that allows fast insertion/suppression of bits at the beginning/end of the list. The NS structure is handled with a hash table with the sequence number of the lost paper as the hash key. Each entry referred to a double-linked list of IP addresses (*subList*). In the current implementation, timers are handled by thread and a hash table of timer threads is used to maintain the list of timers. We will see later on that this approach may not be the best design choice.

Figure 4 depicts the data path of incoming messages. DyRAM packets used the ANEP [16] format and UDP is used for the transport as reliability is directly handle by DyRAM. We can see in the figure that incoming DyRAM packets are given to the Tamanoir execution environment (through the port number) which in turn selects the appropriate active service based on the `SVC` (service identifier) field.

5.2 Testbed and scenario

The testbed consists in 2 PC-based AAC (Pentium II 400MHz, 512KByte cache, 128MByte RAM) running a Linux 2.4 kernel and a set of receivers. The Java version is 1.3.1.

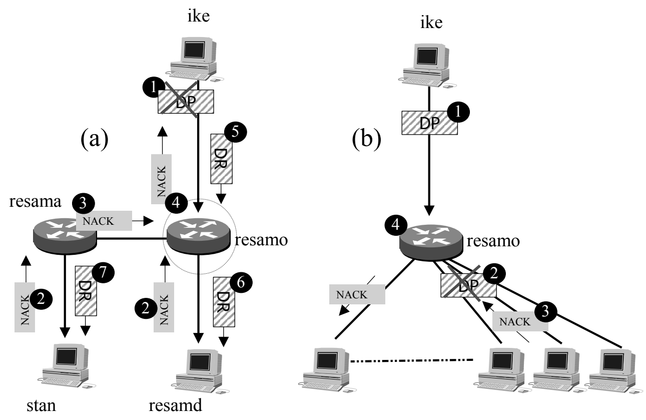

Fig. 5. (a) Topology 1: DP, NACK services and DR service. (b) Topology 2: replier election

2 configurations are used: topology 1, figure 5a, is used to measure the cost of packet services while topology 2, figure 5b is used to measure the cost of the dynamic replier election service. In the first configuration, the source `ike` multicasts packets to `stan` and `resamd` through 2 AACs. In order to measure the data services in this topology, one data packet is lost every 25 packets between the source and the first AAC `resamo`. Figure 5a shows the steps of the recovery process where the source is the replier: DP and DR refer respectively to Data Packet and Data Repair. The cost of the data packet service represents the processing time required to forward the packet to the destinations using the underlying IP multicast functionalities. Therefore, this cost is mainly an update of the track list when there is no sequence gap. In case of a gap sequence indicating a packet lost, the data service consists additionally in setting a timer for ignoring subsequent similar NACKs. The cost of the NACK service represents the processing time to update or create the NS. The cost of the data repair service represents the processing time of scanning the *subList* and to perform the subcast.

In topology 2, the source `ike` multicasts packets to a set of receivers through the AAC `resamo`. The cost of the replier election represents the processing time of comparing the *subList* list against the receiver list (obtained during the initialization session) in order to determine a replier for the lost data packet.

5.3 Cost of active services

Preliminary results on our testbed are illustrated in figure 6 and 7. Figure 6 shows the processing time in μs of a data packet, a repair packet and a NACK packet at the `resamo` AAC. The x-axis shows the packet sequence number while the y-axis shows the processing time. Data packet size is initially set to 4KByte.

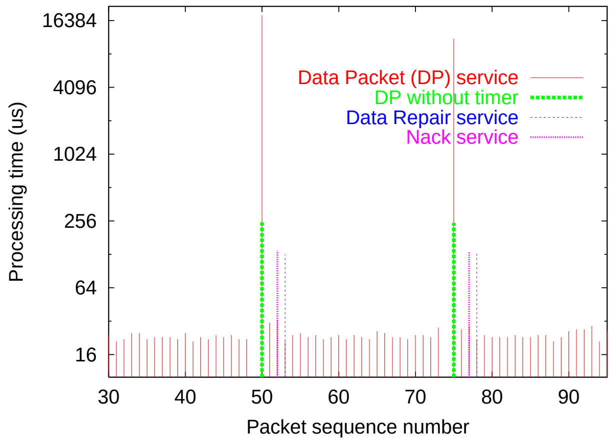

Fig. 6. Cost of packet services

We can see that, in the absence of packet losses, the processing time of a data packet varies from $20\mu s$ to $40\mu s$. A gap in the x-axis represents a packet loss. For instance, in our test scenario packet 49 is first lost thus making the processing of packet 50 longer because of the loss detection and associated data structure updates (track list, NS structures...). Figure 6 shows that each packet loss incur an additional cost of about 12ms to 17ms for the next packet that corresponds to the processing time for the data packet and the time for setting up a timer thread for NACK discarding. It is worth to mention that the first overhead is only about $250\mu s$! Several optimizations are possible for timer management (reuse of old threads, only 1 thread with hash table for timer...) and we expect much lower overheads in next implementations. The NACK packet and the repair packet that follow the packet loss are processed in approximately $135\mu s$ and $123\mu s$ respectively (for these packets, the x-axis indicates the current data packet sequence number at the moment they are processed in order to respect the chronological order). Varying the message size from 1KByte to 32KByte does not change the results. To obtain the exact cost of introducing AAS, we must add the cost for passing packet from the IP layer to the Tamanoir environment. This cost is approximately of $460\mu s$ for a NACK and $640\mu s$ for a 1024-bytes data packet. Finally, the data services on the test-bed introduce less than 1ms of processing time per-packet.

Figure 7 shows the processing time to dynamically determine the replier. The number of receivers under the AAC ranges from 5 to 25. The x-axis indicates how many receivers are involved in the replier election process. For instance, if we look at the 5-receivers curve, a number of 5 at the x-axis means that the correct replier is found after having scanned 4 receivers. It must be noted that the replier election is performed on-the-fly during the time interval for gathering NACK information. In practice, this timer should be set to at least the propagation time to the farthest receiver. Therefore, it is possible that the election cost is masked by the timer duration, resulting in no cost from a protocol end-to-end performance perspective.

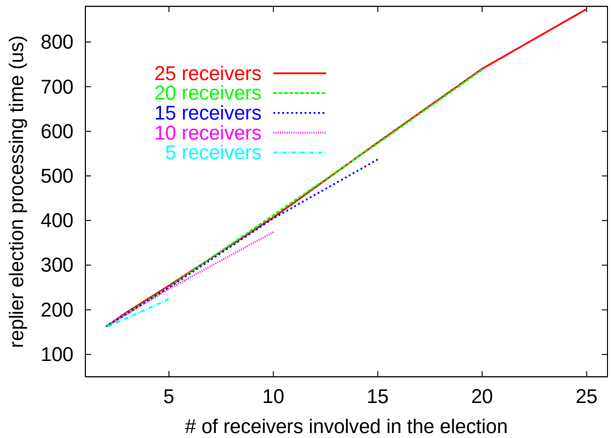

Fig. 7. Cost of replier election.

6 Conclusions

This paper reported on our experiments in implementing application-aware services for the grids, with a case study on the reliable multicast feature. Beyond the DyRAM protocol, the main contribution of this work is to show that application-aware components and services are viable concepts and bring more performance for the end-users. Enabling and generalizing the use of AAS on a grid infrastructure will provide a level of flexibility that have never been achieved in the Internet before and will certainly help for a massive usage of distributed computing.

The last part of the paper addresses the performance issues by measuring the processing cost of the AAS proposed in the DyRAM framework. The results are very encouraging as most processing costs range from tens of μs to hundreds of μs on a Pentium II 400MHz PC-based AAC. Given these results, we believe it is very possible to build AACs from regular PCs (using the most up-to-date processor and clock rate) and still get performances at high bit rates. This possibility, along with the possibility to perform application-level multicast within the DyRAM framework, would certainly help to disseminate the concept of application-aware components for computational grids by allowing local deployment of AACs, independently of carriers and ISP.

References

1. R. B. et al. Parsec: A parallel simulation environment for complex systems. *Computer Magazine, 31(10), October 1998, pp77-85.*

2. M. Calderón, M. Sedano, A. Azcorra, C. Alonso. Active Networks Support for Multicast Applications. *IEEE Networks, May/June 1998.*
3. S. E. Deering and D. R. Cheriton. Multicast Routing in Datagram Internetworks and Extended LANs newblock In *ACM SIGCOMM'88* and *ACM Trans. on Comp. Sys., Vol. 8, No. 2.*
4. Foster, I., Kesselman, C. and Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications, 15 (3). 200-222. 2001.*
5. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl J. Supercomputing Applications, 11(2):115-128, 1997*
6. J.P. Gelas and L. Lefèvre. TAMANOIR : A High Performance Active Network Framework. *Workshop on Active Middleware Services 2000, 9th IEEE International HPDC, Pittsburgh.*
7. A. Grimshaw, A. Ferrari, F. Knabe and M. Humphrey. Legion: An Operating System for Wide-area computing. *IEEE Computer, 32(5):29-37, May 1999*
8. S. K. Kasera et al. Scalable fair reliable multicast using active services. *IEEE Networks, Special Issue on Multicast*, 2000.
9. L. Lefèvre, C. Pham, P. Primet, B. Tourancheau, B. Gaidioz, J. P. Gelas, M. Maimour. Active Networking Support for The Grid. *Proceedings of the third International Working Conference on Active Networks (IWAN'01)*, 2001.
10. M. Litzkow and M. Livny. Experience With The Condor Distributed Batch System. In *IEEE Workshop on Experimental Distributed Systems, October 1990.*
11. M. Maimour and C. Pham. Dynamic Replier Active Reliable Multicast (DyRAM) *Proceedings of the 7th IEEE Symposium on Computers and Communications (ISCC 2002).*
12. M. Maimour and C. Pham. An analysis of a router-based loss detection service for active reliable multicast protocols. *Proceedings of the International Conference On Networks (ICON 2002), Singapore.*
13. Christos Papadopoulos, Guru M. Parulkar, and George Varghese. An error control scheme for large-scale multicast applications. In *IEEE INFOCOM'98*, pp1188–1996.
14. R. State, O. Festor, E. Nataf. A Programmable Network Based Approach for Managing Dynamic Virtual Private Networks In *Proceedings of PDPTA 2000*, Las Vegas, June 26-29.
15. D. L. Tennehouse et al. A survey of active network research. *IEEE Comm. Mag.*, pp80–86, January 1997.
16. D.J. Wetherall, J.V. Guttag and D.L. Tennehouse. ANTS: a Toolkit for Building and Dynamically Deploying Network Protocols. In *IEEE OPENARCH'98, San Francisco, April 1998.*
17. L. Wei, H. Lehman, S. J. Garland, and D. L. Tennenhouse. Active reliable multicast. In *IEEE INFOCOM'98.*
18. S. Zabele et al. Improving Distributed Simulation Performance Using Active Networks. In *World Multi Conference, 2000.*