Experimenting Active Reliable Multicast on Application-Aware Grids

M. Maimour (moufida.maimour@ens-lyon.fr) and C. Pham (congduc.pham@ens-lyon.fr)
RESO/LIP

Abstract. The ever growing needs for computation power and accesses to critical resources have launched in a very short time a large number of grid projects and many realizations have been done on dedicated network infrastructures. On Internet-based infrastructures, however, there are very few distributed or interactive applications (MPI, DIS, HLA, remote visualization) because of insufficient end-to-end performances (bandwidth, latency for example) to support such an interactivity. For the moment, computing resources and network resources are viewed separately in the grid architecture and we believe this is the main bottleneck for achieving end-to-end performances. In this paper, we promote the idea of a grid infrastructure able to adapt to the application's needs and thus define the idea of application-aware grid infrastructures where the network infrastructure is tightly involved in both the communication and processing process. We report on our early experiences in building application-aware components based on active networking technologies for providing a low latency and a low overhead multicast framework 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.

Keywords: Programmable Grid, reliable multicast, Active network

1. Introduction

The simplest perception one has of a computational grid [7] 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, etc. In the past few years, many software environments for gaining access to very large distributed computing resources have been made available (e.g. Condor [16], Globus [8], Legion [12] 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) and its successor EGEE (public.eu-egee.org), EuroGrid (www.eurogrid.org), GriPhyn (www.gryphyn.org) and PPDG (www.ppdg.net) to name some of them.

There are many possible utilizations of a grid, not only intensive computing but also on-line accesses to expensive scientific instruments, interactive simulations/computations/data manipulation and analysis, collaborative visualization, or huge storage capacity as demonstrated by the iGrid 2002 exhibition [13]. However, these experiences were mainly done on an optical bandwidth-abundant infrastructure where end-to-end performances are guaranteed, especially with predictable latencies. On Internet-based infrastructures, there are very few distributed or interactive applications (MPI, DIS, HLA, remote visualization) because of insufficient end-to-end performances (bandwidth, latency for example) to support such an interactivity. For the moment, computing resources and network resources are viewed separately in the grid infrastructure and we believe this separation is the main bottleneck for achieving end-to-end performances.

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, Gnutella or FreeNet). Going a step further, we want to merge computing and network resources in an infrastructure that would allow a tight cooperation for the benefit of the applications. We introduce application-aware components (AAC), as opposed to traditional Internet routers, that would allow the network to work together with the applications to provide in-network and customized processing functionalities. Given the large variety of grid applications, it is highly difficult to predict what will be is a typical application in a future, if any, and its requirements. We believe the concept of application-aware components provide a higher level of adaptability and scalability to support a larger number of distributed applications or new communication protocols: interest management, filtering, collective/gather operations, on-the-fly flow adaptation for remote displays, intelligent directory services, multicast, distributed and hierarchical security systems, distributed logistical storage, customized QoS policy, etc. These new features can be decomposed in a set of grid services–in a way similar to the OGSA proposal [6]–, or more generally in a set of application-aware services (AAS) that would be deployed–possibly on-the-fly–in the network infrastructure.

In this paper, we report on our early experiences in building such application-aware components for one to many (multi-point or multicast) communications on a grid infrastructure. One of the main functions of a grid is to allow a large community of people to share information and resources across a network infrastructure. Therefore, 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.

The paper is organized as follows. Section 2 presents the application-aware grid infrastructure and the AAC technologies using the active networking approach. Section 3 describes the reliable multicast case and presents our protocol with specialized services targeted for providing low latencies on an Internet-based computational grid. Section 4 and 5 present some performance results. Section 6 concludes.

2. Application-aware infrastructures

2.1. Application-aware components

The AACs concept calls for an infrastructure of network elements capable of executing specific processing functions (called services) on data flows, and this mostly on-the-fly. 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, etc.

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 [23], routers can execute, through an execution environment similar to an operating system (ANTS [24] 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 [25, 14, 3], distributed interactive simulations [28] and network management [22]. 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.

2.2. Application-aware services

AAS, as the name indicates, depends on the grid application under study. Such services are intended to be off-loaded in the network infrastructure (into AACs) to operate on the application's data flows. In [9], the authors have listed 5 major classes of grid applications: Distributed Supercomputing, High-Throughput Computing, On-Demand Computing, Data Intensive and Collaborative. The application's properties that are interesting from an application-aware infrastructure perspective could be enumerated as a combination of the following behaviors:

Remote Distribution (RD) : distribution of data and programs, job submissions.

Interactive (I) : interactive manipulation of data and parameters.

Display (D) : remote display of results, rendering, high-resolution images.

Computation/Processing (CP) : all kind of computation and processing tasks performed by the computing resources.

Data Transfer (DT) : indicates an intensive usage of the network for the transfer of large datasets.

Collaborative (CO) : indicates the collaboration of several end-users/programs.

For example, the ATLAS LightPath Transfer application described in [13] is mainly a DT application, whereas the Collaborative Acces Grid could be described as an CO,I,CP,DT,D application and the APBioGrid as an RD,C,D application.

For each of these behaviors, it is possible to exhibit specific AAS that could enhance the application's performance if deployed on an application-aware infrastructure. For instance, CO applications could make use of specific in-network data filters on the requests from the end-users in order to get only the most useful or up-to-date requests. These data filters depend on the application and also possibly on the set of end-users. For I,D applications it is possible to deploy AAS that would decide how and when heterogeneous video compression will be used. Figure 1 illustrates an other example with a network-supported max operation where AACs are used in the network infrastructure to assist for the computation of the max value by hierarchically computing/aggregating the values sent from end-hosts. This functionality could be found in many I,CO grid applications.

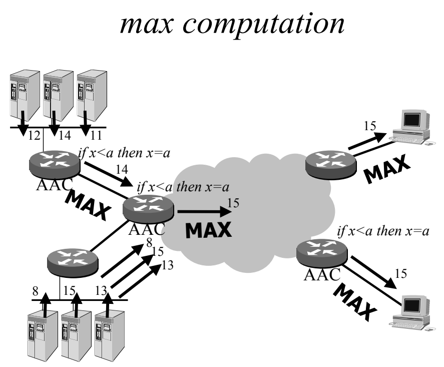


Figure 1. Example of a max operation with network support.

In this paper, we present and study a number of lightweight AAS in the context of a multicast protocol that could be downloaded/selected/enabled by a typical RD,DT,I,CP grid application.

2.3. Application-aware protocols

Using the AAS concepts, protocols can be much more flexible than there are now. With the possibility for the application to specify customized processing functions to be applied on specific data flows, a protocol should be viewed as a flexible framework limited to the core services where additional functions could be plugged, possibly on-the-fly, and deployed along the data path.

In this case, an application-aware protocols could either go for a fully active networking scheme with an open architecture. Applications could then provide their own code to modify/increase the protocol functionalities with AACs providing the ability for grids to dynamically adapt to new usages, and especially to use the network infrastructure as an efficient computing/processing system. This solution is the most flexible but its realization has to face many security problems: is the code valid? is the code harmful? is the code too time-consuming? etc.

An other solution is to adopt an operator-like perspective which consists of deploying beforehand 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 services are likely to be deployed.

In between these 2 end-points, one can also propose protocols with a set of lighweight services to be dynamically deployed depending on the application's needs. In this paper, we illustrate such a solution with a reliable multicast protocol.

2.4. Application-aware grids

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, etc.) 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 1Gb/s) access networks at the edge, with respect to the throughput range found in the backbone. Figure 2 depicts such a typical grid infrastructure. For simplicity we represented an access network by a router but practically such networks would contain several routers.

AACs could be hosted in Internet Data Center (deployed by an operator) but this solution is very difficult at this time because grid computing is not yet the main concern of telecommunication operators or Internet Service Providers. The other solution is to deploy AACs in private domains. There is a great difference between grids and the Internet upon which our work is based: it is much easier to deploy customized solutions on a grid architecture than on the Internet because the number of cooperating sites is much lower. The idea of a global, unique grid on the scale of the planet is a nice idea but is usually impossible to deploy and manage: practically, several grids would co-exist, especially if private companies are involved. Therefore, it is possible to deploy on a small scale, within a well-identified grid user community the application-aware idea. In this case, AACs could be deployed at the initiative of a computing center or an internal enterprise IT infrastructure or a campus for example. This solution is very possible now, with AACs based on dedicated hardware such as powerful PC-based active routers.

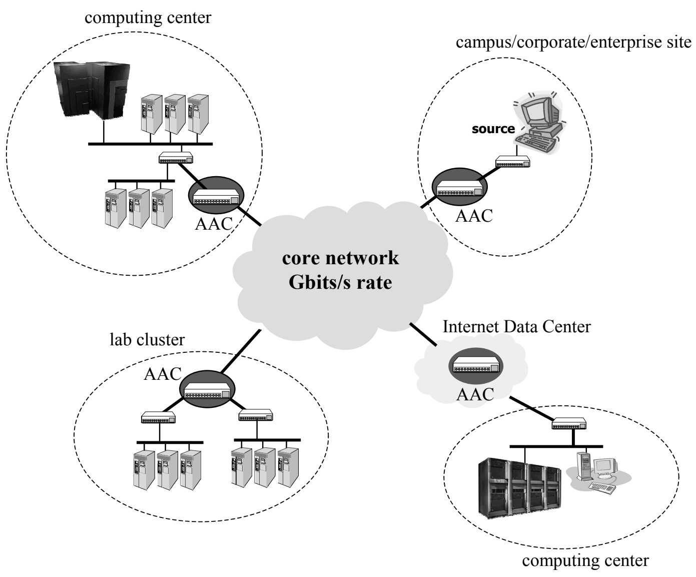


Figure 2. An application-aware grid infrastructure.

As shown in Figure 2, AACs are typically put at the edges of the core network. Since the core network is reliable and a very high-speed network (several Gb/s), adding complex processing functions inside the core network will certainly slow down the packet forwarding functions. The ideas of active elements and application-aware components is not new and some ideas have been taken from other areas. For example, using edge nodes to improve bandwidth and resiliancy has been previously proposed in [21, 1]. However, the application in grids is both novel and practically useful.

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 [4] 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-level 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.

There has been huge amount of work in the multicast area with contributions in both the end-to-end category [26, 5, 20, 27] and the router-assisted category [25, 14, 19, 3, 11]. For instance, ARM (Active Reliable Multicast) [25] and AER (Active Error Recovery) [14] are two protocols that use a *best-effort* cache of data packets to permit local recoveries. ARM adopts a *global* suppression strategy: a receiver experiencing a packet loss sends immediately a NACK to the source. Active services in routers then consist in the aggregation of the multiple NACKs. In contrast, AER uses a *local* suppression strategy inspired from the one used by SRM [5] and based on local timers at the receivers. In addition, an active router in ARM would send the repair packet only to the set of receivers that have sent a NACK packet (subcast). In AER, the active router simply multicasts the repair packet to all its associated receivers. Our proposition takes a slightly different path with a higher number of lightweight services as described in the next section.

3.1. The DyRAM framework

DyRAM [17] is a reliable multicast protocol suite with a recovery strategy based on a tree structure constructed on a per-packet basis. The protocol uses a NACK-based scheme with receiver-based local recoveries where receivers are responsible for both the loss detection and the retransmission of repair packets. DyRAM uses the 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 dynamically off-loaded in the network infrastructure. Therefore the end-host part of the protocol is quite light and the main AAS consist in:

1. **NACK and data services to handle control and data packets**: for instance, duplicated NACKs (from end-hosts) are globally suppressed in order to limit the NACK implosion problem. For each NACK received, the first upstream router from the receivers creates or simply updates an existing structure, that we call Nack State (NS structure), which has a limited life time. It also initialize a timer that will trigger the election of a replier to whom this first NACK will be sent (the election is delayed by the timer to reduce the probability of electing a wrong receiver, i.e. a receiver which has also not received the packet). The NS structure maintains for every "nacked" packet, a subcast list ($subList$) that contains the IP addresses of the receivers that experienced a loss.

2. **the early detection of packet losses** and the emission of the corresponding NACKs. Earlier reliable multicast protocols put the burden of both the loss detection and the recovery at the sender side. The use of NACKs instead of ACKs moves the loss detection responsibility to the receivers. The early detection of packet losses service allows routers to be capable of detecting packet losses and generating corresponding NACKs towards the source. This is done by detecting gaps in the data packet sequence (or possibly on a timeout expiration) by maintaining a track list (TL). When a loss occurs, the router immediately generates a NACK packet towards the source. If the router 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). During this period, all NACK packets received for this data packet from the downstream links are ignored.

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. This service uses the subcast list in the NS structure to keep the set of links (downstream or upstream) from which a NACK has been received. When a data packet arrives at a router, it will simply be forwarded on all the downstream links if it is an original transmission. If the data packet is a repair packet the router searches for a corresponding NS structure and would send the repair packet on all the links that appear in the subcast list.

4. **the dynamic replier election which consists of choosing a link/host as a replier one** (the one which will send again the missing packet) to perform local recoveries. The replier election service is

executed by the first upstream router from the receivers. It 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). Once the election is completed, the router will forward the NACK downstream towards the elected replier. This latter would then unicast the repair packet to its upstream router which would in turn subcast the repair packet on all the links in the subcast list.

5. **providing a very small buffer ring** for caching the most recently received packets. This feature can be dynamically uploaded within routers at any time during the multicast session.

6. **an accurate, on a per-hop basis, RTT computation** for congestion and rate adaptation purposes for interoperability with unicast TCP flows.

DyRAM and its associated services have been designed with the following motivations in mind: (i) to minimize router's load to make them supporting more sessions and (ii) to reduce the recovery latency. The normal operational mode of DyRAM is to avoid cache in router, as opposed to ARM or AER. In this case, a dynamic replier election service is performed by the router. Since a specific replier can be chosen for each loss packet, it is therefore possible to have several logical subtrees at the same time for the recovery process. In addition, this very dynamic choice of the replier provides load balance features that decreases the end-to-end latency and reduces the receiver's overhead (as well as increasing robustness to replier and link failures). However a very small and limited caching system service is provided to further decrease the recovery latency for interactive applications. Reducing the latency recovery is performed by several mechanisms (actually all services contribute directly or indirectly to reduce the recovery latency; for instance AAS number 1 avoids NACK implosion at the source therefore reducing the retransmission time from the source) but when the loss probability is high AAS 2, 4 and 5 are very efficient.

3.2. Application-aware components and services deployment

The additional services proposed within the DyRAM framework could be composed differently to meet the requirements of a grid application (bulk data transfer vs distributed interactive simulation for instance). This ability to adapt to the application's needs is the key idea of an application-aware grid infrastructure. It is of course possible to incrementally compose services from various packages (multicast is for example one package, data confidentiality would be another one and so on) to built incrementally complex services. In what follows, we will only describe how these services could be composed to meet the application's needs for multi-point communications.

Not all grid applications need the whole set of DyRAM's services. For instance, if bulk data transfer is considered, then AAS 1 to 4 can be used. Additionally, if interoperability with other TCP flows is needed then one can add AAS 6 to equally share the bandwidth on an Internet-based grid. When it comes to distributed or interactive applications that put stronger requirements on latencies than in bandwidth, then small caches in routers could be beneficial and AAS 5 downloaded and activated. Figure 3 illustrates on the previous grid architecture a bulk data transfer scenario. In this case, all AACs at the edge should perform at least the NACK suppression and the subcast application-aware services. Therefore these services should be downloaded into the router prior to the data transfer. Regarding the loss detection service, a preliminary study detailed in [18] has shown that this 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 in the grid infrastructure perform the replier election service.

The DyRAM framework for application-aware grids is in its early stage of development. We are currently validating every steps and services (written in Java) in order to demonstrate the end-to-end performances. For the moment which services are downloaded into AAC is simply determined by the class of the application: interactive or file transfer. In the future, the set of services could be determined

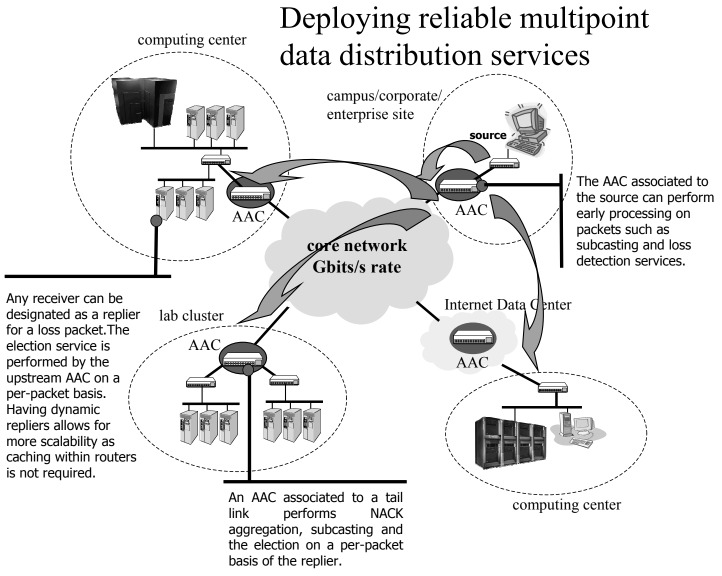

Deploying reliable multipoint data distribution services
computing center
campus/corporate/ enterprise site
source
AAC
AAC
core network Gbits/s rate
The AAC associated to the source can perform early processing on packets such as subcasting and loss detection services.
Any receiver can be designated as a replier for a loss packet.The election service is performed by the upstream AAC on a per-packet basis. Having dynamic repliers allows for more scalability as caching within routers is not required.
lab cluster
AAC
Internet Data Center
AAC
An AAC associated to a tail link performs NACK aggregation, subcasting and the election on a per-packet basis of the replier.
computing center

Figure 3. Deploying the DyRAM framework on a grid infrastructure.

by indicating targeted performance parameters (this is very similar to quality of service selection). The way the various services are deployed is based on the deployment features of the Tamanoir active environment which can upload java code either from the first AACs nearest to the source (that acts like a code repository) or from the source, to be executed on the AACs on the path to the destinations. As explained previously, the set of services to be deployed can vary according to the application's needs. The rest of the paper will present some simulation and implementation results.

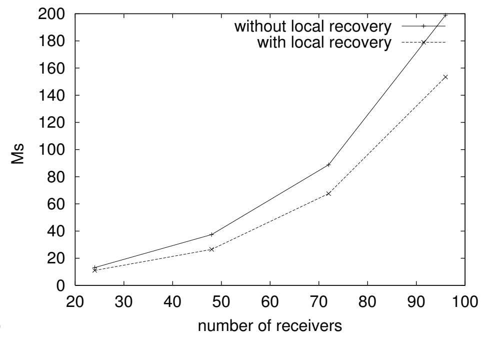
4. Performance study

The performances of the DyRAM framework have been investigated both by simulations and by real implementation and deployment. This section presents the simulation results using the PARSEC simulation language developed at UCLA [2]. 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. This scenario corresponds to a 10Mbytes file transfer from a source to several computing sites. AACs are located at the edges of the core network as described previously.

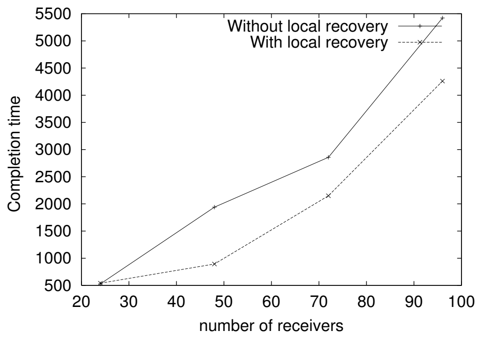
4.1. Local recoveries and replier election

Figure 4a and 4b plot the number of retransmissions (M_S) from the source and the completion time as a function of the number the receivers[1]. p is the end-to-end probability of a packet loss perceived by a receiver. 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 probability.

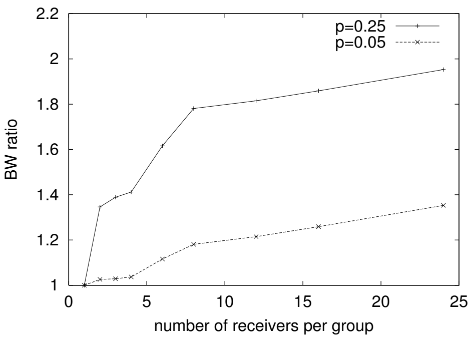
[1] These 3 curves can be found in [17]. They are provided here for a better understanding of the paper.



(a)



(b)



(c)

Figure 4. (a) Load at the source, p D 0:25. (b) Completion time, p D 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 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 4c 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 4c 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 lost packet detection

The next experiments show how the early loss detection service when downloaded in AACs could decrease the delay of recovery perceived by an interactive application. To do so, two cases noted DyRAM- and DyRAM+ are simulated. DyRAM- uses no loss detection services whereas DyRAM+ benefits from the loss detection service in the source AAC. Figure 5 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 for faster transfers: for p=0.25 and 96 receivers for instance, the transfer can be 4 times faster.

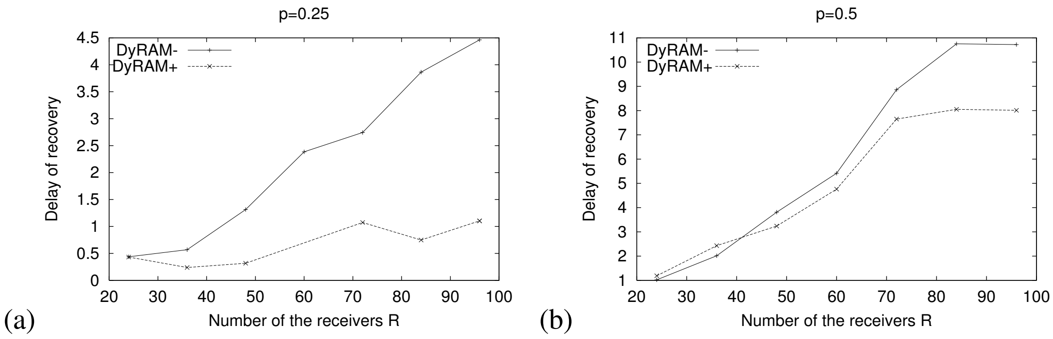


Figure 5. The normalized recovery delay with (a) p=0.25 and (b) p=0.5

4.3. Small cache in routers

In order to support distributed application such as interactive simulations, it is possible to add a small caching system to further reduce the recovery latency. This feature can be dynamically uploaded within AACs at any time during the multicast session. The results we are presenting now assume that some cache amount (memory or disks) are available in routers. Figure 6 shows the gain in completion time compared to a traditional recovery approach from the source (no local recovery) with 48 receivers.

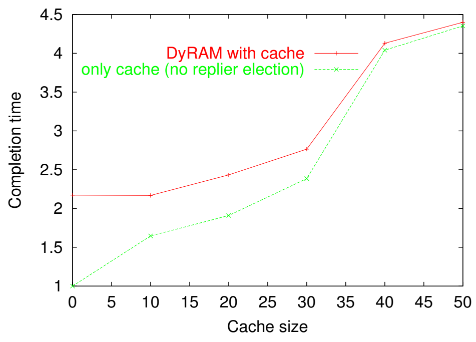


Figure 6. Completion time gain when DyRAM benefits from cache in the routers, $p = 0.25$.

From 0 to 10% (of the entire file) of cache amount, the gain in completion time is similar to the no-cache case (comparison with Figure 4b with 48 receivers) because no repair packets could be found in the cache so replier election is always performed. When the cache size increases, the completion time decreases. With 20% to 30% of cache, we can see that the recovery latency is further reduced compared to the case when replier election is the only recovery method (comparison with figure 4b with 48 receivers). To see the impact of the combination of router cache and replier election, Figure 6 also shows the gain in completion time when only router cache is used. We can see that 20% to 30% of cache is the optimal tradeoff between completion time and cache amount in this experiment.

4.4. Comparison with an end-to-end solution: SRM

SRM [5] is one of the first protocols that moves the responsibility of losses recovery to the receivers. The protocol has been proposed and tested for collaborative white-board. In this section, we are evaluating the scalability of the DyRAM framework and its performance compared to SRM. In order to be able to have the largest multicast group, the number of data packets sent by the source is limited to 200 packets instead an unlimited number for the previous simulations. Losses occur during all the simulation time and the simulation ends when all the receivers receives all the 200 packets sent by the source. The number of the receivers per router is chosen randomly from a uniform distribution of parameters 1 and 20. A

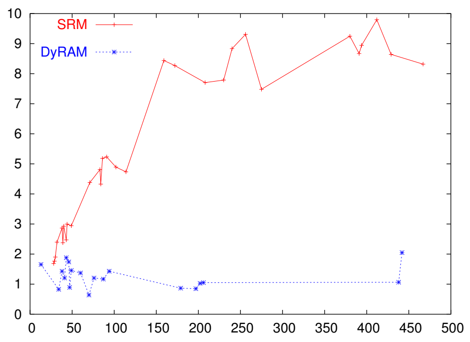


Figure 7. Repair latency with losses at the tail links, p=0.25.

number of simulations are performed and a mean is computed from experiences with the same number of receivers. Figure 7 plots for p=0.25, the repair latency (normalized to the RTT to the source) achieved by the two protocols as a function of the number of the receivers. We observe that DyRAM achieves a repair latency less than the RTT to the source and is more scalable than SRM since this is still true even when increasing the number of receivers. With SRM, the repair latency increases with the number of receivers and is at least about 2 RTTs and may be 10 times the repair latency of DyRAM when the number of receivers exceeds 200.

5. Implementation and deployment issues

A test-bed with a core protocol library for sender and receiver applications, along with the previously described AAS has been developed in Java to both study the cost of introducing AAS within the network infrastructure and as a starting point for further large scale deployment. The execution environment is Tamanoir [10]. Tamanoir can be installed on a linux-based PC and configured to act as an active router with dynamic upload of user's code into the execution environment.

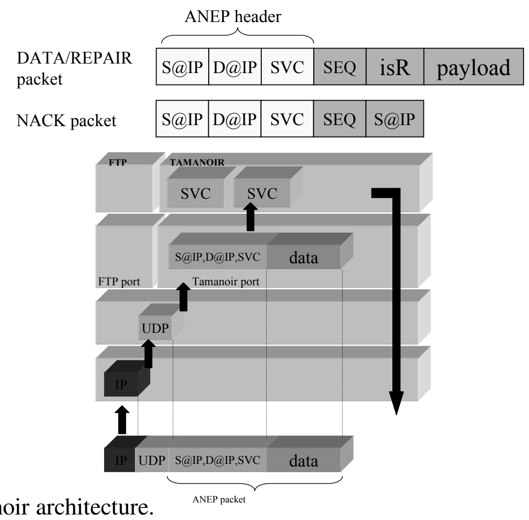


Figure 8. Packet format and Tamanoir architecture.

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 threads 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 8 depicts the data path of incoming messages at end-hosts and AACs. At the application level, the ANEP format [24] is used on top of UDP 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. Test-bed and scenario

The test-bed consists of 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.

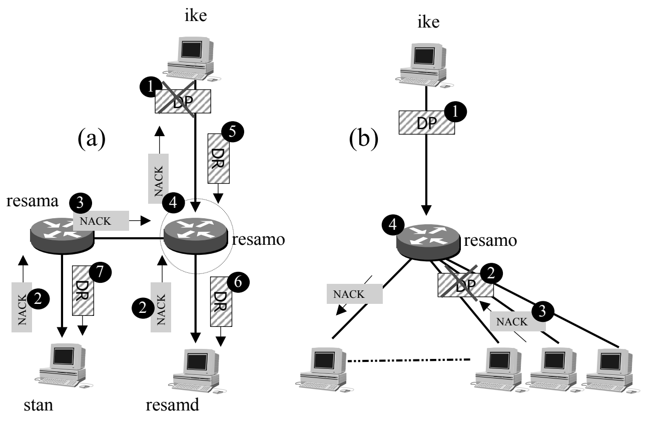


Figure 9. (a) Topology 1: DP, NACK services and DR service. (b) Topology 2: replier election

2 configurations are used: topology 1, Figure 9a, is used to measure the cost of packet services while topology 2, Figure 9b 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 9a 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 of 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 test-bed are illustrated in Figure 10 and 11. Figure 10 shows the processing time in us 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.

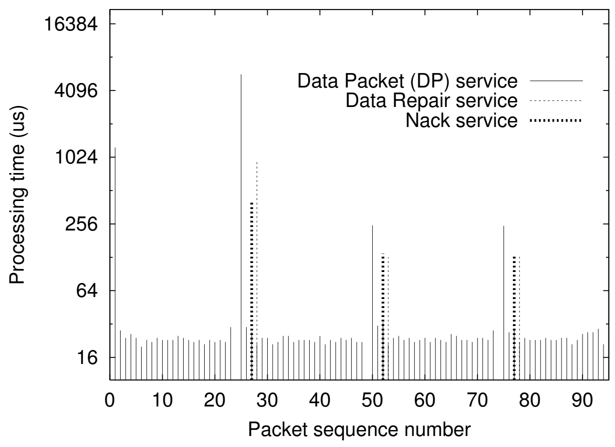


Figure 10. Cost of packet services

We can see that, in the absence of packet losses, the processing time of a data packet varies from 20us to 40us. 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 10 shows that each packet loss incur an additional cost of about 210us for the next packet. The NACK packet and the repair packet that follow the packet loss are processed in approximately 135us and 123us 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 460us for a NACK and 640us for a 1024-bytes data packet. Finally, the data services on the test-bed introduce less than 1ms of processing time per-packet.

Figure 11 shows the processing time to dynamically determine the replier. This election is performed on-the-fly election and works by selecting a default replier (the first receiver in the receiver list which is obtained during the initialization session) when the first NACK arrives and by 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. In Figure 11 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. Since the replier election is performed on-the-fly during the time interval for gathering NACK information, the election cost is masked most of the time by the timer duration, resulting in no cost from the protocol's end-to-end performance perspective.

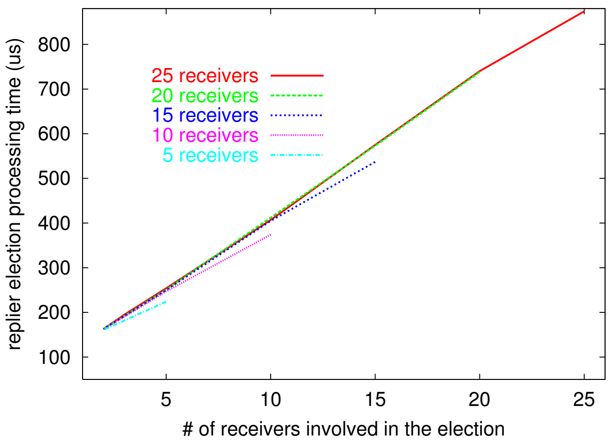


Figure 11. Cost of replier election.

5.4. Test and deployment on a real grid infrastructure

The DyRAM framework has been tested and the active services validated on a real grid infrastructure connecting 4 computing sites. Figure 12 illustrates the test topology where a source, located at CERN in Switzerland, multicasts a data file needed by a biology application to 7 receivers in 3 different sites. At initialization of the multicast transfer, active services are deployed on the path from the source to the destinations using the Tamanoir deployment facilities. AACs located at the egde of the backbone provides the local recovery mechanism with the replier election service. In this experience, the loss detection service is uploaded in the AAC at the ENS site. Figure 13 is a snapshot of the test realized where the various steps of the local recovery are indicated. In this scenario, one computer in ENS lost packet 28 which is repaired by the other computer elected by the AAC located at the ENS site.

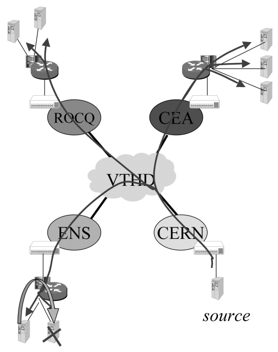
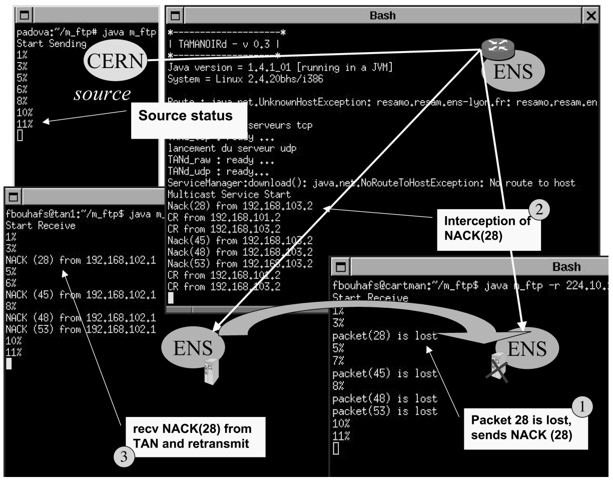


Figure 12. Testbed topology.



```
padova:~/m_ftp# java m_ftp
Start Sending
1%
3%
5%
6%
8%
10%
11%
```

```
*--------------------*
| TAMANOIRd - v 0.3 |
*--------------------*
Java version = 1.4.1_01 [running in a JVM]
System = Linux 2.4.20bhs/i386

Route : java.net.UnknownHostException: resamo.resam.ens-lyon.fr: resamo.resam.en

serveurs tcp
ready ...
lancement du serveur udp
TANd_raw : ready ...
TANd_udp : ready...
ServiceManager:download(): java.net.NoRouteToHostException: No route to host
Multicast Service Start
Nack(28) from 192.168.103.2
CR from 192.168.101.2
CR from 192.168.103.2
Nack(45) from 192.168.103.2
Nack(48) from 192.168.103.2
Nack(53) from 192.168.103.2
CR from 192.168.101.2
CR from 192.168.103.2
```

```
fbouhafs@tan1:~/m_ftp$ java m_
Start Receive
1%
3%
NACK (28) from 192.168.102.1
5%
6%
NACK (45) from 192.168.102.1
8%
NACK (48) from 192.168.102.1
NACK (53) from 192.168.102.1
10%
11%
```

```
fbouhafs@cartman:~/m_ftp$ java m_ftp -r 224.10.
Start Receive
1%
3%
packet(28) is lost
5%
7%
packet(45) is lost
8%
packet(48) is lost
packet(53) is lost
10%
11%
```

Bash
CERN
source
Source status
ENS
2
Interception of NACK(28)
Bash
ENS
ENS
1
Packet 28 is lost, sends NACK (28)
recv NACK(28) from TAN and retransmit
3

Figure 13. Augmented screenshot.

6. Conclusions

This paper reported on our experiments in implementing and deploying application-aware services for the grids, with a case study on the reliable multicast feature. 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 which will hopefully help for a larger usage of distributed computing.

The last part of the paper addresses the performance issues by measuring the processing cost of the proposed AAS. The results are very encouraging as most processing costs range from tens of us to hundreds of us 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 would certainly help to disseminate the concept of application-aware components for computational grids by allowing local deployment of AACs, independently of telcommunication carriers and ISP.

References

1. D. Andersen, H. Balakrishnan, M. Kaashoek, R. Morris. Resilient Overlay Networks Proc. 18th ACM SOSP, Banff, Canada, October 2001.
2. R. B. et al. Parsec: A parallel simulation environment for complex systems. *Computer Magazine, 31(10), October 1998, pp77-85.*
3. M. Calderón, M. Sedano, A. Azcorra, C. Alonso. Active Networks Support for Multicast Applications. *IEEE Networks, May/June 1998.*
4. 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.*
5. S. Floyd, V. Jacobson, and Liu C. G. A reliable multicast framework for light weight session and application level framing. In *ACM SIGCOMM'95*, pp342–356.
6. I. Foster, C. Kesselman, J. Nick, S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.*
7. 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.*
8. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl J. Supercomputing Applications, 11(2):115-128, 1997*

9. I. Foster and C. Kesselman, editors. The Grid: Blueprint for a New Computing Infrastructure. *Morgan Kaufmann Publishers, 1999*
10. J.P. Gelas and L. Lefèvre. TAMANOIR : A High Performance Active Network Framework. *Workshop on Active Middleware Services 2000, 9th IEEE International HPDC, Pittsburgh.*
11. Jim Gemmell, Todd Montgomery, Tony Speakman, Nidhi Bhaskar, Jon Crowcroft. The PGM Reliable Multicast Protocol. *IEEE Networks, January 2003.*
12. 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*
13. iGrid 2002 demonstrations. *http://www.igrid2002.org/Applications.pdf.*
14. S. K. Kasera et al. Scalable fair reliable multicast using active services. *IEEE Networks, Special Issue on Multicast*, 2000.
15. Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software Practice and Experience, 32(2):135– 164, February 2002.*
16. M. Litzkow and M. Livny. Experience With The Condor Distributed Batch System. In *IEEE Workshop on Experimental Distributed Systems, October 1990.*
17. M. Maimour and C. Pham. Dynamic Replier Active Reliable Multicast (DyRAM) *Proceedings of the 7th IEEE Symposium on Computers and Communications (ISCC 2002).*
18. 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.*
19. Christos Papadopoulos, Guru M. Parulkar, and George Varghese. An error control scheme for large-scale multicast applications. In *IEEE INFOCOM'98*, pp1188–1996.
20. S. Paul and K. K. Sabnani. Reliable multicast transport protocol (rmtp). *IEEE JSAC, Special Issue on Network Support for Multipoint Comm.*, 15(3):407–421.
21. N. S. V. Rao NetLets: End-To-End QoS Mechanisms for Distributed Computing in Wide-Area Networks Using Two-Paths. Journal of High-Performance Computing Applications, Vol. 16(3), August 2002, pp285-292.
22. 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.
23. D. L. Tennehouse et al. A survey of active network research. *IEEE Comm. Mag.*, pp80–86, January 1997.
24. 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.*
25. L. Wei, H. Lehman, S. J. Garland, and D. L. Tennenhouse. Active reliable multicast. In *IEEE INFOCOM'98.*
26. XTP Forum. *Xpress Transport Protocol Specification*, March 1995.
27. R. Yavatkar, J. Griffioen, and M. Sudan. A reliable dissemination protocol for interactive collaborative applications. In *ACM Multimedia'95.*
28. S. Zabele et al. Improving Distributed Simulation Performance Using Active Networks. In *World Multi Conference, 2000.*

Author's Vitae

Congduc Pham
received his Ph.D. degree in computer science from the University of Paris 6, France in 1997. He is currently an Associate Professor at the University of Lyon 1, France and member of the INRIA RESO project. His research interests include parallel simulation of computer networks, cluster and grid computing, active networking and multicast protocols. He is a member of IEEE.