Flexible and Dynamic Control of Network Quality of Service in Grid environments: the QoSINUS approach*

Pascale Vicat-Blanc Primet[2] Johan Montagnat[1]
Fabien Chanussot[2]

[1] CNRS UMR5515, CREATIS, INSA de Lyon, 69621 Villeurbanne, France
[2] INRIA-ENS LIP, RESO, 46, allée d'Italie, 69 007 Lyon, France

1 Abstract

Grids rely on a complex interconnection of IP domains that may exhibit changing performance characteristics and may offer different quality of service (QoS) facilities. We examine the case of a biomedical application distributed over a grid and show how it may suffer from uncontrolled communication performance. Then we present the QoSINUS service that dynamically allocates the network resources to Grid flows in order to match their specific QoS requirements under different load conditions. The aim of this approach is to optimize the end to end performances the heterogeneous mix of grid flows gets from the network to enhance the individual application's performance as the overall grid infrastructure performance and utilization level. The QoSINUS service is based on the *programmable network* approach that offers flexibility, evolutivity and enables dynamic adaptation to network load variations. Finally results of QoSINUS experiments conducted in the context of the `eToile` french grid testbed based on the high speed and DiffServ capable research network infrastructure, `VTHD`, are presented.

2 Introduction

The purpose of Computational Grids is to aggregate a large collection of shared resources (computing, communication, storage, information) to build an efficient and very high performance computing environment for data-intensive or computing-intensive applications [6].

*This work was funded by the RNTL eToile project of the French ministry of research, by the European Commission program IST-2000-25182 through the EU DataGrid project and IST-2001 DataTAG project, the INRIA project RESO, and the french ministry for research ACI-GRID project.

There are many possible utilizations of a grid, not only intensive computing but also interactive usage and remote access to expensive instruments or huge scientific data bases exploration for instance. Many data transfer patterns will occur in such an environment. Bulk data transfer from one location to an other and time constrained message exchanges of fine-grained parallel applications may have to coexist. Consequently, in Grids, the quality of service (QoS) the heterogeneous mix of flows receives may affect the overall grid performance, as well as each individual application performance.

But the underlying communication infrastructure of these large scale distributed environments is a complex interconnection of multi IP domains, with changing performance characteristics and different quality of service facilities. *The Grid Network cloud* may thus exhibit extreme heterogeneity in performance and reliability that can considerably affect the global application performances. The total lack of resource control in IP networks may be responsible for performance problems that are not acceptable in grids environment.

This paper is organized as follows: we examine the case of a biomedical application distributed over a grid and show how it may suffer from uncontrolled commmunication performance in section 3. Then, section 4 presents the QoSINUS service that has been designed to dynamically allocate controlled network resources to Grid flows and to flexibly manage the network quality of service at the edge of the grid network cloud. The results of experiments conducted in the context of the French grid testbed eToile are detailed in section 5.

3 QoS in Grids

3.1 The medical imaging example

Several analysis [2, 11] have shown that the QoS requirements spectrum of Grid flows is broad comprising needs like the guaranteed delivery of a complete huge data file, the predictability of the TCP throughput, a stability-level for data-delivery. To understand the specificity of QoS requirements of grid applications, we introduce the example of a *content-based query of medical image databases* application.

Medical imaging is data intensive due to the size of medical images. For instance, a CT-scanner usually produces 3D images made of 50 to 100 slices. Each slice is a 512^2 image. Each voxel intensity is encoded on 16 bits. This results in a 25 to 50 Mb image. A typical 3D Magnetic Resonance Image is 256^2 per slice, 128 slices, 2 bytes per voxel resulting in a 16 Mb image. Content-based query of images on a grid requires the distribution of large data set over the network.

Digital medical images are stored in databases. The simplest data structures are flat files: one file for each (2D or 3D) image. Meta-data on image files (e.g. patient name, acquisition type, etc) are usually stored in relational tables and used for querying the available data. Beyond simple queries on meta-data, physicians want to search for images close to a sample they are interested in, so that they can study cases similar to the image they are looking at. This involves an analysis of the image contents to determine images from a database that are similar to a sample image.

Several similarity measures may be used to measure the differences between images [8, 10, 9]. Although these computations are not very intensive, the comparison of a sample image against a complete database may become intractable due to the size of medical databases. However, the computation can be easily distributed on a computation grid since the comparisons of the sample image with each image of the database are independent.

Figure 1 illustrates a content-based query on a computation grid. A sample image is available on the physician's computer. A complete database is available from a storage site. The images are distributed over the computation grid to perform similarity measurements. The result returned to the user interface is a score for each image in the database. The physician can then select the highest score images for visualization.

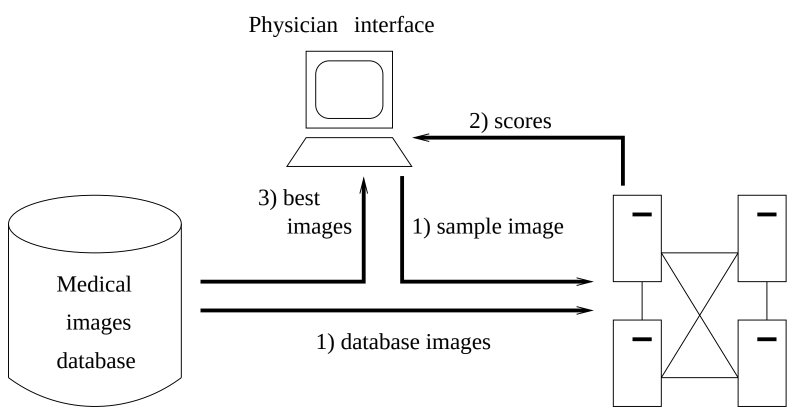

Figure 1: Content-based query on medical images

3.2 Performance analysis and Network requirements

The application does not exhibit real-time constraints. However, computations should be fast enough for a reasonable use in clinical practice (fast means minutes in this context, 30 minutes at most). The computation time of the similarity measures is rather low (from few seconds to 15 minutes on a 1GHz Pentium processor, depending on the measures estimated and the input image size). Therefore, the network transmission of the images to the computing sites may become a bottleneck.

Let S the total search time, n the image number, T the transfer time of an image I between two nodes and C the computing time of image I with another image J. Assuming, all computations are done on one processor and the database is located in the same site, the total search time S_1 will be: $S_1 = nC$.

If, given site N_a the location of the end user, site N_b the location of the DataBase, site N_c the location of a computing node, we assume that the effective rate r (TCP throughput) on links $N_a - N_b$, $N_a - N_c$ and $N_b - N_c$ are equivalent and stable, then the transfer time T of an image is constant and depends on the size l of the image I: $T = \frac{l}{r}$. If the operations are dispatched on k processors and the database cut in k equal partitions, S_k the total search time will be:

$$S_k = \frac{nC}{k} + \frac{n}{k}T + kT$$

with $\frac{n}{k}T$ being the time to transfer the k partitions to the k processors and kT the time to transfer the image I to the k processors.

We denote SpeedUp the speedup obtained with a search deployed over the grid.

$$\text{SpeedUp} = \frac{S_1}{S_k} = \frac{nkC}{(nC) + (n + k^2)T} \quad (1)$$

Consequently we have:

1. If $\frac{C}{T}$ is low, the speedup goes to 0. In this condition, it is better to adopt a centralised approach.
2. If $\frac{C}{T}$ is high, the speedup goes to k. It is then very interesting to split the database in as much partitions as possible and distribute the computing load to the maximum available computing nodes.
3. If $\frac{C}{T} = 1$, the speedup goes to $\frac{nk}{2n+k^2}$, depending on the ratio $\frac{n}{k}$.

Let's consider now that T is not constant, going from lower than C to higher, what may be the usual case on a classical TCP/IP best effort congested link, performances will be difficult to predict. The appropriate choice of k becomes very complex and the global application performance impossible to determine. We conclude that, on a grid, it is very important to have an accurate evaluation of the data transmission and a stable throughput because it can impact the distribution strategy and the overall performance. To optimize the application performance, a grid scheduler should interact with the network performance measurement system and the network QoS control system according to the following steps:

1. initialisation: estimate of T, T_{estim}, using a grid network performance measurement tool. Compute $\frac{C}{T_{estim}}$;

2. scheduling: determine k based on C, T_{estim} and n;

3. execution: ensure that T remains as close as possible to T_{estim} using a network QoS control system.

3.3 How to control communication performances in Grids?

The Grid network cloud is a complex aggregation of heterogeneous domains offering various performances guarantees and QoS strategies that Grid designer cannot control. Distributed applications, like the medical imaging application, generally rely on TCP protocol running over IP for wide area communications. IP is a connectionless protocol and therefore it has no inherent mechanisms to deliver guarantees according to traffic contracts. When congestion occur, end to end flows experience varying latency (queues on interfaces) or traffic loss that can damage dramatically the end to end throughput.

The problem of network performance control can be studied from different view points in grids:

- Monitoring and forecasting systems (like the Network Weather Service [14], [7], or network performance measurement architecture designed for the DataGrid project [4]), measure and predict end to end performances and help to characterize the links and the network behavior. Such services can be used to perform the first step of our previous example.

- Enhanced network services that increase the network performance control are required to assure the communication performance. Such approaches have to rely on transport protocols and QoS services deployed in IP networks. Such services may help to achieve the goal of the last step identified above.

Generally, two solutions have been adopted to improve and control the communication performances at network level:

- the first one consists in masking the variability of the network performances, by integrating compensation mechanisms directly in the transport protocols, like in TCP or FEC.

- an other approach consists in adding control in the network in order to avoid performance variation and to offer some guarantees in transfer. This approach is known as IP Quality of service.

A large effort to provide QoS architecture at network level has been done during the last ten years. Three types of QoS approaches have been studied for service differentiation at packet level:

- services that provide strict guarantees (absolutes) with resource reservation like in the standardized IntServ architecture [3] and RSVP [?],
- services that provide strict guarantees without resource reservation and state management in core routers like in CoreStateless model [?] based on dynamic packet state concept,
- services that offer statistical guarantees (differentiated services) to aggregates through packet prioritisation in approaches like DiffServ [1]

The IntServ architecture presents a scalability problem because this model requires that control and forwarding state for all flows are maintained in routers. The corestateless approach requires the help of routers to compute flow virtual clocks with mechanisms not available in actual equipments. This approach is not deployed in existing infrastructures. The DiffServ architecture is a more scalable and manageable standardised architecture. Two services are frequently deployed: EF, that provides minimized delay, and AF that assures a minimal bandwidth. However DiffServ [1] is a *pure in network* solution that cares on packets (level 3 building block) and aggregates while end to end users pay attention to the performances of individual flows (level 4 abstract unit). DiffServ applies dedicated per-hop behaviors to aggregates of packets. Consequently, end to end flows are not assured to receive bounded guarantees. Moreover, users applications have no way to specify in advance and control the service they expect. Finally, as the QoS resources are scare and/or costly, they have to be finely managed.

The next section describes the QoSINUS architecture that has been designed to dynamically control the QoS performance of grid flows.

4 The QoSINUS approach

4.1 Goals and usage scenario

The approach we propose aims to enable Grid applications, that may strongly benefit from QoS guarantees, to simply access the various QoS capabilities offered by IP domains. A Grid oriented QoS API and a programmable QoS service have been designed to introduce flexibility and dynamic in the management, control and realization of end to end, flow level QoS in Grid context. The aims of the QoSINUS service are to:

1. provide heterogeneous Grid flows with a mean to specify their QoS objectives;
2. dynamically map these objectives with the IP QoS services provided in the network, according to the state of the link, the QoS mechanisms configured and the experienced performances.

This approach combines application aware and infrastructure aware components activated within the network, at the interface of the Grid computing domain and the Grid long distance networks, as shown in figure **??**. Such an approach increases slightly the complexity at the frontier points, but maintains the core network and the grid applications simplicity.

>From the user point of view the QoSINUS service is invoked in two phases, as presented in figure 2. During a first programming phase, a QoS request is transmitted to the QoSINUS service. Then the sender transmits data packets: the QoSINUS decision component chooses dynamically the appropriated class of service to cross the domain, trying to simultaneously optimize the QoS resource usage and the flow performance.

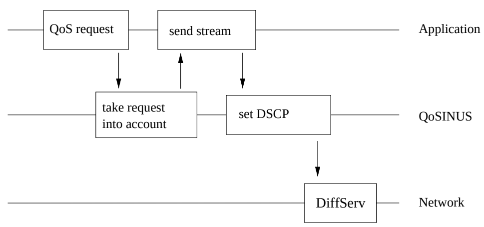

Figure 2: QoSINUS Usage scenario.

4.2 Design model

4.2.1 API

The first issue to solve is to allow grid applications to specify and control their QoS. This issue is addressed by a dedicated API with 4 functions:

- *QoS_Set* lets an application specify its QoS needs in terms of delay, rate and loss, or transmission schedule and rate.
- *QoS_Invoke* and *QoS_Release* that are called by the client application to let QoSINUS know that the data stream for which QoS was requested is about to start or has ended.
- *QoS_Request* lets the client application specify its QoS needs in a more detailed way than QoS_Set. The QoS request is expressed in an Xml format. The corresponding schema is given in listing **??**.

4.2.2 Service architecture

The dynamic mapping of the flows QoS specification to the existing IP QoS facilities is addressed by a service architecture that combines flow aware and infrastructure aware components.

We have identified four types of component for flexible QoS programming and control:

- programming component,
- performance measurement component,
- adaptive control component,
- enforcement component.

The QoS programming component is invoked for initiating, propagating and storing the flow QoS goals in the programmable nodes of the path.

The QoS performance measurement is responsible for the characterisation of the specific paths. In a DiffServ context, this component can measure the performances of each DS class on each grid path with active out of band probes or filter and gather flow performances in band.

The adaptive control components is responsible for class mapping and for adapting the packet marking influencing the forwarding performances regarding the QoS goals and the current state of network classes. It monitors and updates the allocated bandwidth of each class and acts as a decision component that decides which class has to be attributed to the packets when the performance measurement component indicates some change.

The QoS enforcement service, intercepts and adapts the data flows when it is necessary. It realizes the packet marking and conditioning functions. Other components, that are not designed in this architecture, may be added for status report exchange between programmable nodes for example.

Various tables and soft states are associated to this service: Class allocation table that stores the status of the allocated class bandwidth, *QoS goal* structure that maintains the flows objectives and *QOS flow status* that monitors the flow.

The adaptive control component is flexible, extensible, replaceable and it is simultaneously dependant on the infrastructure QoS properties and on the flow properties. It represents the kernel of the QoSINUS service. It decides the adaptation according to the performance experienced by the packets of each particular flow. With DiffServ, this adaptation means dynamic rate shaping or packet remarking.

The QoSINUS programmable service class diagram is presented in figure 3. In that figure, QaSEngine and QaSXMLIf correspond to the programming component, QaSMonitoringIf corresponds to the performance measurement component interface, QaSMapper is the adaptive control component that can be a static mapper, an adaptive mapper or an Ack-React mapper, QaSKernelIf represents the interface to the enforcement component.

Different mappings and adaptive algorithms have been designed. The simplest one consist in a static mapping. It performs the allocation decision on static thresholds. For example, if a delay constraint is expressed and premium capacity is still available, an EF bandwidth corresponding to the requested one will be allocated. All packets belonging to this flows will be

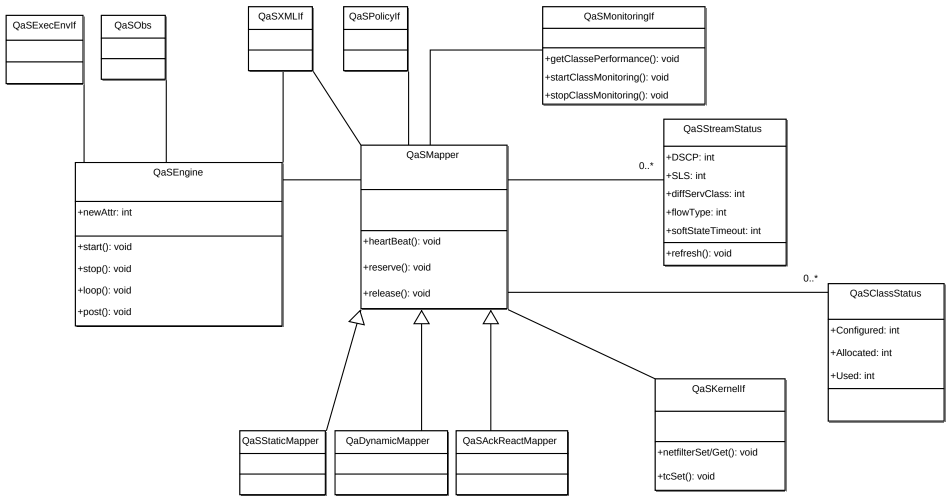

Figure 3: The QoSINUS class diagram.

marked EF. Adaptive algorithms attempt to fulfil the requirements of the flows while optimising the classes utilization. The classes are ordered by level of performance they offer. Adaptive mappers are thus responsible for monitoring the performances of the flows and dynamically allocating class resources. An example of such an algorithm is described in the next section.

4.2.3 Ack-based earliest deadline first TCP stream ordering

The Ack-based mapping algorithm aims at ensuring that TCP data transfers are completed within the specified schedule, and at allocating the DiffServ classes optimally.

This algorithm assumes that the application is able to specify the schedule it plans to follow for it's data transfer. The schedule is specified in terms of start date, stop date and amount of data to transfer.

The streams are ordered according to their deadlines: the stream with the earliest deadline is taken care of first.

The algorithm tracks the achieved performance of each TCP stream it controls, and tries to compensate if it gets lower than the requested performance.

The following actions are taken by the programmable QoS service, that correspond to the performance measurement, adaptive control and enforcement components mentioned in the previous section.

Ack-based transmission tracking The system evaluates the amount of data transmitted at a given moment in time using the TCP acknowledgment messages flowing from the flow's receiver back to the sender.

The acknowledgment number included in the TCP ACK packets is not exactly equal to the actual amount of data successfully received. But it is a good estimate of the amount of transmitted data.

Earliest deadline first ordering The system uses an earliest deadline first ordering scheme, to order the streams it manages. Comparing the initial requested schedule with the actual amount of data transmitted for each stream, the system is able to determine which streams are behind schedule. The stream behind schedule with the earliest deadline is given the maximum priority, i.e. is marked in the best performing DiffServ class.

The assumption here is that a stream with a later deadline will have the opportunity to catch up later on. This allows for example, to lower a massive bulk transfer throughput a little bit, when smaller urgent communications have to take place. This approach should thus lead to a more efficient use of the network resource.

5 Experiments

5.1 The e-Toile VTHD testbed

The e-Toile project [12] is an experimental wide area grid testbed [1]. One of the original aim of the project is to focus on the High Performance Grid Networking dimension and to evaluate the benefit that grid middlewares and applications can get from enhanced networking technologies.

The VTHD (vraiment très haut débit) network infrastructure [13] interconnects the e-Toile grid nodes with access link of 1 to 2 Gbps. The e-Toile middleware relies on existing middleware (Globus [5], Grid Engine...) and integrates new building blocks. The e-Toile middleware aims to fully exploit the power of the networking infrastructure.

5.2 VTHD DiffServ implementation

The Gigabit VTHD [?] Backbone provides four DiffServ classes (EF, TCP AF, UDP AF and BE). The principle of the DiffServ configuration in VTHD is that a bandwidth share is statically provisioned in the edge routers and allocated to the four DS classes. Each access point has to control and to shape the traffic injected in each class. Table 1 gives the absolute rate allocated to each DiffServ class.

The first experimentation consisted in evaluating the raw performances of the DiffServ classes obtained from one edge to another. Best Effort and EF TCP performances have been measured in five network loads conditions: no background traffic, 267Mbits/s, 535Mbits/s, 840Mbits/s and 1000Mbits/s

[1] e-toile is a RNTL project (réseau national de recherche en logiciel) funded by French Ministry of Research

DS Class	Share
Expedited Forwarding	10%
Assured Forwarding for TCP	30%
Assured Forwarding for UDP	30%
Best Effort	30%

Table 1: The statical bandwidth share of VTHD DiffServ classes.

UDP background traffic. Figure 4 shows the performance (TCP Throuput and RTT) of Best Effort and EF respectively.

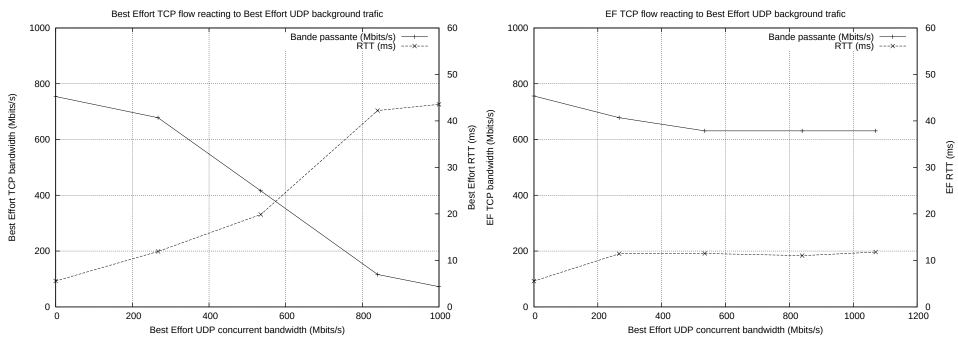

Figure 4: Best Effort and EF reacting to background traffic.

This simple experiment shows that the EF traffic is correctly protected while Best effort traffic is strongly affected by increasing load. The Best Effort TCP throughput is divided by 10. This results corresponds to the expected TCP/IP backoff behaviour.

The next section demonstrates the impact that this kind of network behaviour may have on the medical imaging application performance.

5.3 Expected effect of the Network QoS on the application performance

Let's use the database described in table 2 as an use case of the medical imaging application presented in section 3.1.

Number of images	238
Image dimensions	181×217×181
Image size	13.6 MB
Database size	3.1 GB

Table 2: Studied image database characteristics.

Each images grey level range may be undersampled prior to processing. This loss of precision brings an increased computation time. The following experiments are therefore using undersampled version of the original images to 8 and 12 bits when possible.

5.3.1 Application's computation time

The medical application involves similarity measure computations over image volumes. The complexity for all similarity measures depends on the size and the dynamic range of medical images. Furthermore, the complexity of computations varies from on measure to another (see [8] for details). Table 3 summarizes the measured computation time (in seconds) for the similarity measures (excluding image I/O and undersampling) on pairs of the above mentionned database images, the 6 similarity measures proposed, and every possible undersampling to 8, 12, and 16 bits. The times where measured on a 800MHz Intel pentium III processor.

Undersampling	SD	SSD	CC	RC	Woods	MI
8 bits	0.795	0.798	0.810	0.808	0.810	0.813
12 bits	13.79	14.46	16.17	16.05	15.87	15.92
16 bits	697.3	697.4	700.3	702.6	802.7	804.3

Table 3: Computation times for similarity measures computation.

5.3.2 Network transfer time

Table 4 shows network file transfer time measured in the five network load conditions studied in section 5.2 for Best Effort and EF. This correspond to the measurement of $\frac{n}{k}T$ for the database mentioned above (807MB with $k = 4$, 323MB with $k = 10$, and 32MB with $k = 100$).

		transfer (s)		
Class used	Concurrent traffic	807MB	323MB	32MB
Best Effort	0 Mbits/s	9.0	3.7	0.4
	267 Mbits/s	10.0	4.0	0.4
	535 Mbits/s	16.3	6.5	0.6
	840 Mbits/s	58.4	23.2	2.2
	1070 Mbits/s	109.3	65.4	3.8
EF	0 Mbits/s	9.1	3.7	0.4
	267 Mbits/s	10.0	4.0	0.4
	535 Mbits/s	10.7	4.3	0.4
	840 Mbits/s	10.7	4.3	0.4
	1070 Mbits/s	10.7	4.3	0.4

Table 4: Network transfer time in different traffic conditions, corresponding to Database transfers.

		k		
T_{estim}(s)	C(s)	4	10	100
0.17	0.8	3.26	7.68	9.86
	15.37	3.95	9.85	67.76
	734.1	4	10	99.01

Table 5: Application speedup using $k \in \{4, 10, 100\}$ processors on a congested EF class network.

		k		
T_{estim}(s)	C(s)	4	10	100
0.93	0.8	1.79	3.77	1.96
	15.37	3.76	9.21	27.76
	734.1	3.99	9.98	94.83

Table 6: Application speedup using $k \in \{4, 10, 100\}$ processors on a congested BestEffort class network.

5.3.3 Performance estimation

According to table 3, the mean application execution time (C) is 0.8s (for 8 bits undersampled images), 15.37s (for 12 bits undersampled images), or 734.1s (for 16 bits images). Given that $n = 238$, it is possible to estimate the speedup using equation 1 for various network conditions and values of k.

According to table 4, estimation T_{estim} of the time needed to transfer an image using the EF class when there is a 840 Mbits/s background traffic is 0.17s. It is 0.93s when using Best Effort in the same traffic conditions.

Table 5 shows the speedup obtained with EF and 840 Mbits/s of background traffic. Table 6 shows the speedup when using Best Effort class in the same traffic conditions.

These tables demonstrate that the EF DiffServ class significantly improves the speedup for shorter computation times where the network overhead is most damaging the computation time.

They also demonstrate that it is possible to estimate the number of processors to use depending on the application execution time and network performance.

It is interesting to note that the estimation of the best k to use might vary a lot for different network performances (case where T_{estim}=0.93 versus T_{estim}=0.17 with C=0.8).

This emphasizes the need of keeping the $\frac{C}{T}$ stable enough during the application's execution.

That's where QoSINUS proves to be usefull, guaranteeing the stability of a given TCP throughput and making an efficient use of the DiffServ class resources.

The next section illustrates how that service can help to keep the network transmission time as close as T_{estim} as possible.

5.4 Ack-React experimentation

The goal of this experiment is to demonstrate the use of the TCP Ack packets when evaluating the throughput of a transmission and choosing the DiffServ class to map a stream into.

The simple algorithm below is used to choose the DiffServ class to use. Given Cl in BE,EF, the DiffServ class of the current packet flow, Q_{trans} the amount of transmitted data and Q_{req} the requested amount of data to transmit at time t. The DiffServ allocation algorithm as follows:

```
Cl = BE
for each received ack
        estimate Qtrans

for each control time period
        if Qtrans < Qreq
                Cl = EF class
        else
                Cl = BE class
```

Three sites are involved, in Lyon (I), Paris (II) and Grenoble (III). Those sites are linked by the VTHD network, as shown in figure 5

A TCP stream is flowing from Lyon to Grenoble. Background traffic is emitted from Lyon to Paris, that produces a congestion on Lyon's router, and affects the TCP stream's performance.

The QoSINUS programmable service is deployed at the VTHD border in Lyon, and controles the TCP streams emitted from Lyon to the network.

This test's procedure is as follows:

- The application, that needs to transmit 486 MBytes of data, sends its QoS request. The stream is scheduled between t + 10 seconds and t + 70 seconds. The requested rate is 64 Mbps.
- About 11 seconds later, the TCP stream is started.
- At t + 20 seconds, the background traffic is triggered, at 1 Gbps.
- At t + 57 seconds, the background traffic is stopped.
- The application stops sending data whenever its done with the transmission of its 486 MBytes of data.

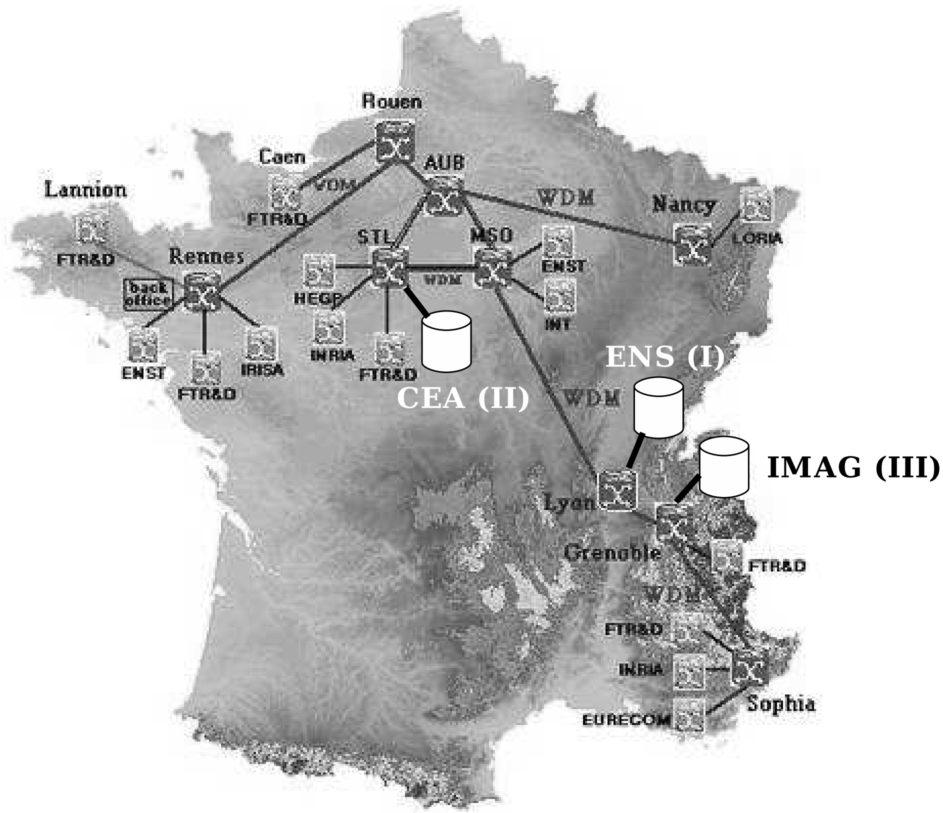

Figure 5: Network topology

5.4.1 Impact of the background traffic

The first step consists in observing the impact the background traffic has on the TCP stream, if nothing is done to protect it.

Figure 6 shows how the TCP stream behaves in this case.

As long as the concurrent traffic is not emitted, the TCP stream achieves some 120 Mbps, and gets ahead of schedule.

But it gets much slower (arround 15 Mbps), once the concurrent traffic is started. The stream is thus behind schedule when the transmission completes.

5.4.2 Reacting to the Ack packets

The same procedure is repeated. But the DiffServ class mapping algorithm is turned on, to check that it's able to protect the TCP stream's performance.

Figure 7 shows what happens in this case.

The TCP stream is affected the same way as before by the background traffic. But here, the system is able to detect that the achieved transmission

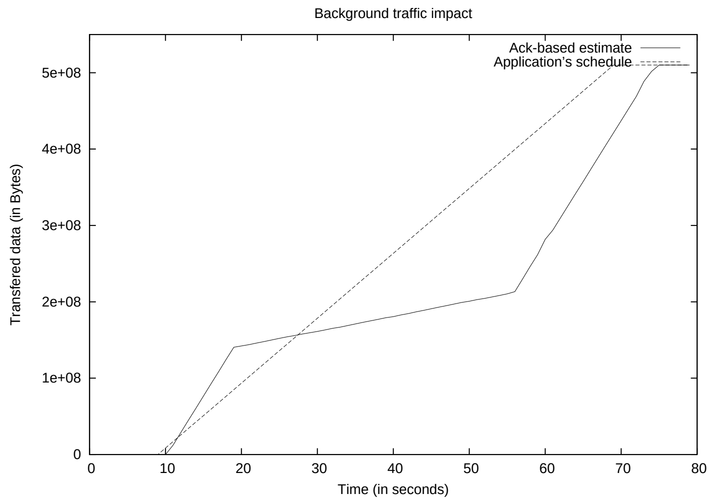

Figure 6: ACK based adaptation turned off

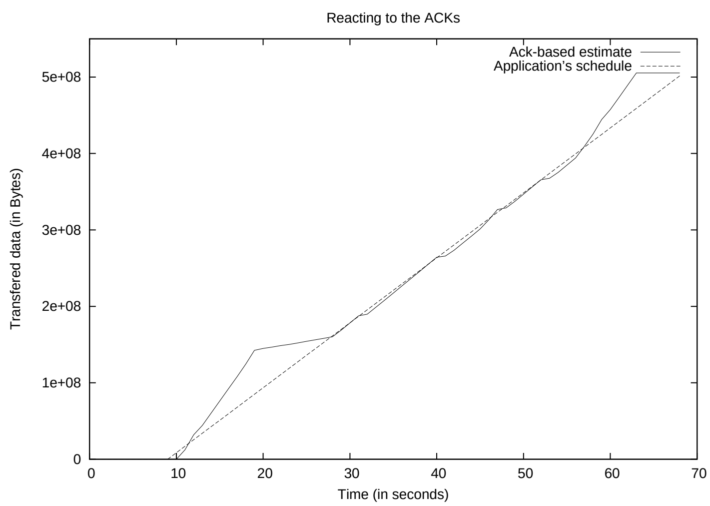

Figure 7: ACK based adaptation turned on

performance gets lower than the requested one (arround t + 28 seconds).

The system reacts accordingly, and alternatively marks the stream's packets in EF, to boost the transmission rate, and back to Best Effort, when it catches up. In this case, the achieved performance curve sticks to the requested performance curve and the transmission is able to end within schedule.

6 Conclusion

This paper presented the problem of mapping the QoS requirements of grid flows to the QoS services offered by IP network services. The results obtained on an experimental grid testbed show the benefit the Grid application can expect from IP QoS services available in current academicals network infrastructures. A medical image analysis application corresponding to a real usecase has been exposed and its theoretical performances analysed. This application has been selected as it is strongly dependent on the network capacity and representative of grid application flow performance requirements. A flexible service named QoSINUS that has been designed to dynamically manage the QoS classes of an IP domain and serve the needs of grid application is proposed. The use of this service by grid applications and middleware offers a very flexible, transparent and efficient QoS control. An adaptive marking algorithm has been proposed and is very promising. In the case of pricing, it can be very important from an economical poiunt of view, to save EF bandwidth. More experimentation now needs to take place, focusing on the earliest deadline first ordering of several concurrent streams. An extension of QoSINUS in the context of heterogenous multidomain network is also under study. The programmable network approach and the modular architecture of QoSINUS also permit to easily deploy new algorithms and to study other QoS approaches like bandwidth on demand allocation.

References

[1] Steven Blake, David Black, Mark Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss. An architecture for differentiated services. Internet Request For Comments RFC 2475, Internet Engineering Task Force, December 1998.

[2] F. Bonnassieux, P. Primet, and P. Clarke. Network provisioning for the datagrid testbed1. Technical report, EU DATAGRID report Deliverable D7.1 - Approved by the EC January 2001, 2001.

[3] Robert Braden, David Clark, and Scott Shenker. Integrated services in the internet architecture: an overview. Internet Request For Comments RFC 1633, Internet Engineering Task Force, June 1994.

[4] European datagrid ist project. `http://www.edg.org/`.

[5] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, July 1997.

[6] Ian Foster and Carl Kesselman. The grid : Blueprint for a new computing infrastructure. *Morgan Kaufmann Publishers Inc.*, 1998.

[7] Benjamin Gaidioz, Richard Wolski, and B. Tourancheau. Synchronizing network probes to avoid measurement intrusiveness with the network weather service. In *Proc. 9th IEEE Symp. on High Performance Distributed Computing*, 2000.

[8] J. Montagnat, H. Duque, J.-M. Pierson, V. Breton, L. Brunie, and Magnin I.E. Medical Image Content-Based Queries using the Grid. In *Healthgrid'03*, pages 138–147, Lyon, France, January 2003.

[9] G.P. Penney, J. Weese, J.A. Little, P. Desmedt, D.LG. Hill, and D.J. Hawkes. A Comparison of Similarity Measures for Use in 2D-3D Medical Image Registration. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI'98)*, volume 1496 of *LNCS*, pages 1153–1161, Cambridge, USA, October 1998. Springer.

[10] A. Roche, G. Malandain, A. Ayache, and S. Prima. Towards a Better Comprehension of Similarity Measures Used in Medical Image Registration. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI'99)*, volume 1679 of *LNCS*, pages 555–564, Cambridge, UK, September 1999. Springer.

[11] P. Vicat-Blanc/Primet. High performance grid networking in the datagrid project. *special issue Future Generation Computer Systems*, January 2003.

[12] P. Vicat-Blanc/Primet, G. Romier, and M. Soberman. Le projet e-toile: développement et mise en oeuvre d'une grille haute performance. In *Proceedings of "Journées Nationales du RNTL 2002"*, 2002.

[13] Vthd. `http://www.vthd.org/`.

[14] R. Wolski. Dynamically forecasting network performance using the Network Weather Service. *Cluster Computing*, 1(1):119–132, 1998.