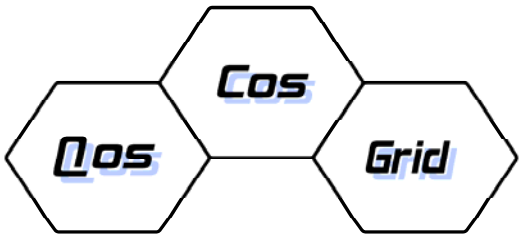

FP6-IST-2005-033883

QosCosGrid

Quasi-Opportunistic Supercomputing for Complex Systems in Grid Environments

Specific Targeted Research or Innovation Project
IST-2005-2.5.4 Advanced Grid Technologies, Systems and Services

D1.4: Second Prototype & Integration of Grid Services Together with QoS-Aware Grid MW Providers (M24)

Due date of deliverable: M24 (31 August 2008)

Actual submission date (of deliverable report): 9 October 2008

Start date of project: 1 September 2006 **Duration**: 30 months

Instytut Chemii Bioorganicznej PAN w Poznaniu (PSNC)

Revision: 1.0

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	-
PP	**Restricted to other programme participants (including the Commission Services)**	**X**
RE	Restricted to a group specified by the Consortium (including the Commission Services)	-
CO	Confidential, only for members of the Consortium (including the Commission Services)	-

Revision history

Document administration information		
Project acronym:	QosCosGrid, **Contract number**: FP6–IST-2005-033883	
Deliverable number:	D1.4	
Full title:	Second Prototype & Integration of Grid Services Together with QoS-Aware Grid MW Providers	
Short name:	Grid Services for Complex Systems	
Document identifier:	QosCosGrid-del-D1.4GridServicesForComplexSystems(M24)-s	
Lead partner short name:	PSNC	
Report version:	1	
Report preparation date:	9/10/2008	
Dissemination level:	PP: Consortium, programme participants, EC services	
Nature:	D: Demonstrator	
Lead author:	Krzysztof Kurowski	
Co-authors:	Mariusz Mamonski, Piotr Grabowski, Yannick Langlois, Guillaume Mecheneau, Thomas Herault, Camille Coti, Mark Ragan	
Status:	-	Draft
	-	Final
	X	**Submitted**

The QosCosGrid Consortium has addressed all comments received, making changes as necessary. Changes to this document are detailed in the change log table below.

Change log		
Date	**Editor**	**Summary of changes made**
01/7/2008	George Kampis	Report template set up
15/8/2008	All	A draft version of deliverable based on input from partners involved in WP1
31/8/2008	All	Final Draft
06/10/2008	Raul Alcantara	Format Revision
09/10/2008	Werner Dubitzky	Finalization for submission to EC.

Executive summary

The main goal of the D1.4 Second Prototype and Integration of Grid Services Together with QoS-Aware Grid Middleware deliverable is to describe the second prototype and integration between Administrative Domain level components, QosCosGrid infrastructure and cross-domain programming and execution environments: QCG-OMPI and ProActive for complex systems developed in WP1 during the last 24 months in a close collaboration with other work packages. Main programming interfaces, their integration and communication protocols among cross-site grid services and applications will be presented as well in this document. To demonstrate advanced features and the usability of QosCosGrid solutions various results of cross-domain application specific benchmarks and testing procedures will be discussed.

This deliverable covers also the following five areas of research and development activities performed in WP1 together with main work package requirements we met:

- Improvements to programming and execution environments, namely QCG OMPI and ProActive, to meet the requirement: “reduce the cost and complexity of complex system (CS) distributed across a wide spectrum of platforms, scheduling systems and grid technologies”.
- Advance reservation extensions to Open DRMAA Service Provider (OpenDSP) and GRMS to meet the requirement: “be able to take advantage of advance reservation capabilities provided by QoS-aware grid middleware”.
- Low level monitoring extensions to Platform LSF and its integration with QCG middleware to meet the requirement: “facilitate integration of distributed CS components, scheduling and data storage systems”.
- Examples and performance results of cross-domain simulations in QCG OMPI and ProActive executed on the QosCosGrid test bed to meet the requirement: “provide CS-oriented interfaces for communication, monitoring and measurement capabilities in order to facilitate development and experiments”.
- Setup Web-based platform called QosCosGrid Gateway to meet the requirement: “allow administrators and end users of CS from various domains to monitor CS components”.

One of the key achievements of WP1 is that after two years we are able to demonstrate a fully functional cross-site test bed connecting computing clusters located in the following administrative domains: PSNC (Poland), ColBud (Hungary), INRIA (France), PFU (Spain), UU (United Kingdom), UvA (Holland) and UQ (Australia). Complex system developers and users not only access and use transparently remote computing resources on a regular basis but also benefit from new application oriented features for inter cluster communications and user friendly interfaces for large scale application control and monitoring.

Figure 1: The geographic map of QosCosGrid test bed at M24.

Table of contents

1. Introduction

The generic QosCosGrid architecture was presented in the previous deliverable D1.3 First Prototype and Deployment of Grid Services for Complex Systems in WP1. In this document we focus mostly on a range of extensions and improvements that have been done in the last reporting period to QosCosGrid software, infrastructure and services classified as Administrative Domain components (see Figure 1). In order to distinguish two reports, note that in the first prototype we concentrated on state-of-the-art analysis of existing systems and services as well as complex systems requirements for cross-domain parallel application programming, execution and advanced control. We also initiated a joint administrative effort of creating a test bed where all adopted and developed software components are deployed and tested. In this way, at that time, we identified a set of missing features and capabilities that have been successfully implemented during the next year to build a consistent second prototype of the whole QosCosGrid system.

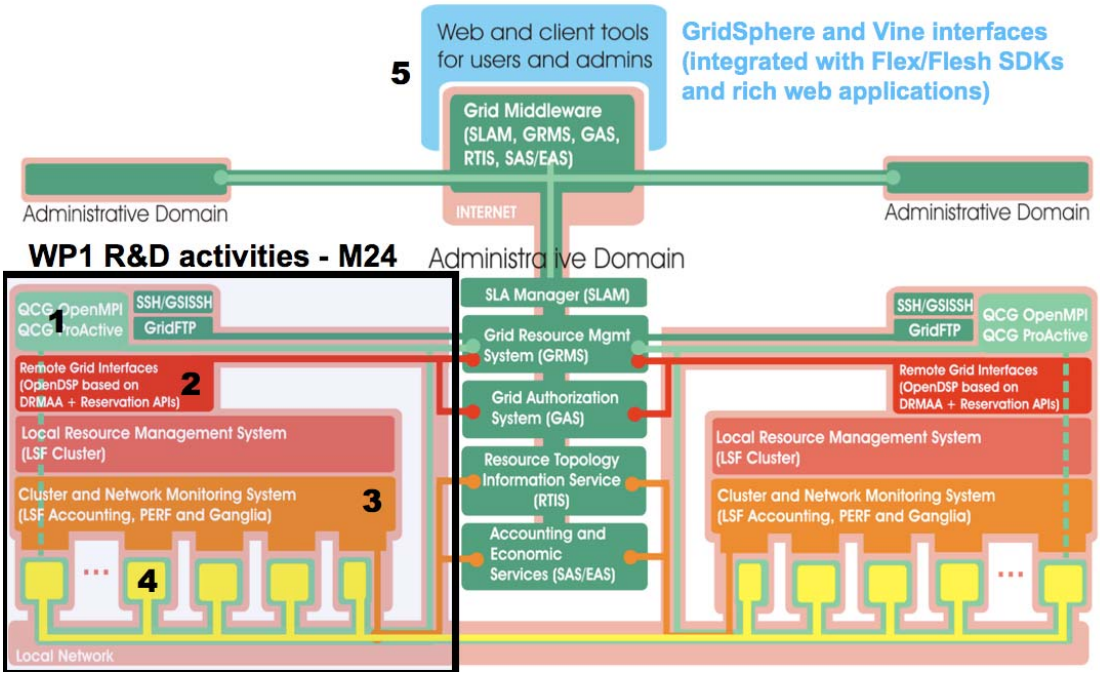

Figure 2: QosCosGrid architecture and main R&D activities in WP1.

To understand the complexity of development in WP1 one should see the QosCosGrid infrastructure as a distributed QosCosGrid system consisting of many independent components, services and interfaces end users can use to program and then use easily for the execution of large scale parallel applications. To show dependencies among software components main WP1 activities and areas of development are highlighted in Figure 2. On the very bottom there is a set of computing resources connected via local or external networks on which parallel complex systems written in C/C++ or Java are executed and controlled by a queuing system (4). Local management operations,

such as advance computing resource reservation as well as job submission and control are performed by a local queuing system (local resource management system), in our case Platform LSF, which had to be extended to offer advanced accounting and monitoring capabilities for QoS/SLA control (3). The local queuing system – Platform LSF, is a key component for WP1 as it manages computing resources within an administrative domain. As our architecture design requirements were to develop generic grid solutions, Platform LSF can be replaced by two other existing systems: SGE and PBSPro as they offer advanced features we need for resource reservation and co-allocation in WP1. Even more critical from the QosCosGrid infrastructure perspective is the remote interface we provide on the top of queuing systems in QosCosGrid – OpenDSP (2). In general, OpenDSP service acting as a main remote gateway in every administrative domain offering on demand access to remote computing resources in the QosCosGrid test bed. An improved version of OpenDSP is used for remote advance resource reservations and job submission via DRMAA-compliant interface in all administrative domains involved. Finally, on the top of OpenDSP two programming and execution environments are located: QCG OMPI and QCG ProActive (1). These tools have been successfully integrated with OpenDSP to provide application-level mechanisms for parallel and distributed cross-cluster computing submission. Additionally, we have developed a set of web-based interfaces for IT administrators (5) to help them monitor and maintain cross-domain computing test bed. All mentioned software components can be seen as a consistent layered stack of software that has to be deployed in on the computing cluster to connect to the QosCosGrid infrastructure.

In the next sections we will describe the current status of mentioned five development areas in WP1 and the way we integrated WP1 software stack to offer required grid interfaces for complex systems.

2. Programming and execution environments: QCG OMPI and QCG ProActive

Two programming and execution environments for parallel cross-domain application development and deployment have been adopted in QosCosGrid, namely: QCG OMPI and QCG ProActive. However, their required various software extensions and functional improvements to be fully integrated with the QosCosGrid infrastructure, in particular Platform LSF, OpenDSP and GRMS. Note, the final integration of both QCG-OMPI and QCG ProActive with GRMS has been scheduled for a final prototype, therefore in this deliverable we explain main integration efforts in WP1 between application level mechanisms and resource management interfaces for cross-domain resource co-allocation, job submission and control at the OpenDSP and LSF level (components denoted as 2 and 3 in Figure 2).

QCG-OMPI is an extended MPI programming environment, providing an adapted run-time environment and MPI library, based on Open MPI 1.3a1 and a set of grid services called Inter-Cluster Communication Services. As a programming and execution environment QCG-OMPI has been used by many QosCosGrid use cases written in C/C++. Alternatively, once we identified legacy applications and

use cases written in Java, we decided in WP1 to provide support for a new Java-based programming environment called ProActive. ProActive uses by default the RMI Java standard library as a portable communication layer, supporting the following communication protocols: RMI, HTTP, Jini, RMI/SSH, and Ibis. With a reduced set of simple primitives, ProActive provides a comprehensive toolkit that simplifies the programming of applications distributed on Local Area Networks (LANs), Clusters, Internet Grids and Peer-to-Peer Intranets for Java-based applications.

Unfortunately, in both cases, there was neither support for multi-user environments nor support for advance reservation and co-allocation required for advanced complex systems application scenarios we have collected in QosCosGrid. Thus, in WP1 we had to develop a new communication grid service called QCG PA Gateway improving the cross-site deployment process of QCG-OMPI and QCG ProActive applications. Moreover, we introduced new inter-cluster communication models for QCG-OMPI and ProActive. The next two subsections present in detail our extensions and modifications to deal with cross-site multi-user parallel application deployment and control.

2.1. New Deployment protocol for OpenDSP and GRMS in ProActive

While developing ProActive extensions in WP1 the following goals were defined to meet QosCosGrid architecture assumptions:

- Preserve standard ProActive library properties (i.e. extend it rather then change it)
- Provide to end users a consistent Job Profile schema as a single document that is used to describe application parameters required for execution as well as resource requirements, in particular network topology and estimated execution time
- Prevent users from necessity of having direct (i.e. over SSH) access to remote clusters machines.

Created extensions (applied to the ProActive 3.9 library) were named QCG-ProActive to distinguish from standard ProActive middleware.

It is worth to mention here that an essential part of every ProActive application lifecycle, especially for end users, is the deployment process. Originally, it is based on creation of remote java virtual machines (JVMs) that are treated as “containers” called Active Objects which later perform real computations. Typically, there are many possible ways how JVMs can be started by the ProActive environment:

- Remote shels: ssh, gsissh, rsh, rlogin.
- Locally via DRMS systems: LSF, PBS, SGE, OAR, PRUN.
- Remotely via grid middleware: globus(GT2, GT3 and GT4), Unicore, gLite.

In practice, to use mentioned protocols to deploy remotely a ProActive based application on a distributed cross-cluster computing infrastructure a lot of

configuration efforts is required. Furthermore, configuration files and the initial setup are subject of frequent changes as they are not generated automatically and must be coded in the ProActive application by a developer well in advance. Thus, ProActive Deployment Descriptor (PAD) as a separated file has been introduced to ProActive in order to fully separate the application code from a deployment protocol. To meet advanced requirements in QosCosGrid we used OpenDSP at the administrative level and GRMS at the grid level as gateways to DRMS systems. Therefore, in the first step we had to design a new deployment protocol for QosCosGrid middleware with two critical components: OpenDSP and GRMS. The figure below illustrates basic steps of a new QosCosGrid deployment model we proposed in WP1.

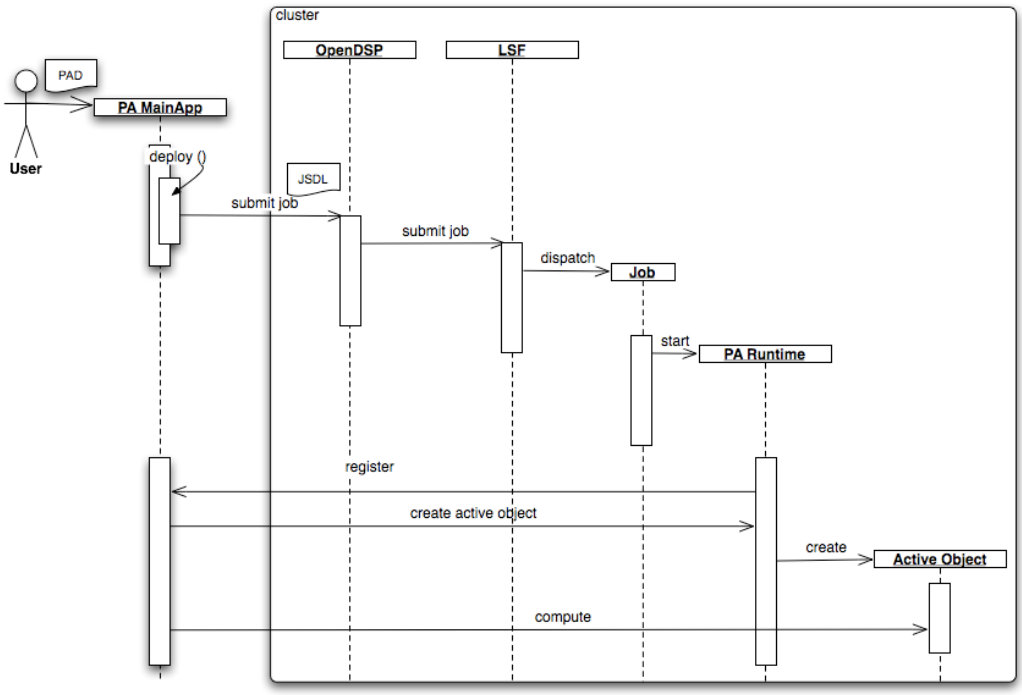

Figure 3: A new deployment for OpenDSP/GRMS in ProActive.

As it is presented on a sequence diagram in Figure 3, in the first step a user has to prepare an appropriate PAD file using a new QCG ProActive deployment model for the QosCosGrid infrastructure. The deployment model is based on the JSDL standard and some XML extensions we defined in close collaboration with WP2 and WP3 called QCG Job Profile to be able to describe advanced topology-aware resource requirements and QoS/SLA parameters of QosCosGrid use cases. The deployment model is provided by a user together with a main ProActive-based application which either contact GRMS or OpenDSP directly to submit a job. Once OpenDSP has been tightly integrated with the underlying queuing system (Platform LSF) via DRMAA routines the job submission request is forwarded asynchronously down to LSF to dispatch a new computing job on a computing cluster. Then, LSF job starts a new ProActive runtime environment. The ProActive environment itself is in charge of the registration callback to the main ProActive application. If the callback is performed successfully the main

application sends back a new call to the ProActive environment to create Active Object(s). Once a new Active Object is created the main application is able to send computing tasks to it and the local calculation process begins under the control of LSF.

2.2. Cross-site QCG-ProActive deployment model

Unfortunately, a typical deployment model in ProActive has many drawbacks, in particular:

- The main ProActive application must stay connected all the time to exchange messages online with remote Active Objects and wait since all of them have finished calculations.
- The network connection between the main ProActive application on the client side and remote Active Objects could be a bottleneck and slow down calculations once many messages might be exchanged.
- To overcome these problems one can run the main ProActive application inside a computing cluster instead of having the main application executed on the client side. However, then we will end up with two phases and in fact two job submissions to OpenDSP and underlying LSF are required:
- First, the execution of the main client application (PA MasterApp) to setup the ProActive environment.
- Second, the execution of the main client application (PA Runtimes) to deploy Active Objects.

The main problem while submitting the main client application (PA MasterApp) and working nodes (ProActive Runtimes) at the same time is that during the second phase some parameters and arguments must be defined and they are generated automatically by ProActive during the first deployment stage. Therefore, we proposed to create an external service in QosCosGrid called ProActive Node Coordinator (PNC) that helps to exchange initial arguments between the PA MasterApp and PA Runtimes. Consequently, a local queuing system (OpenDSP with LSF) does not have to start ProActive Runtimes directly. Instead, we need to provide an appropriate wrapper – QCG PA Wrapper, which connects PNC service and synchronize initial data required to start ProActive Runtimes properly. Additionally, in order to support a cross-cluster ProActive deployment before any job submission request to LSF via OpenDSP there is an appropriate advance resource reservation call. The main difference, from the end user perspective, is that instead of a typical PAD file people in QosCosGrid are using a Job Profile as a language to describe ProActive application requirements and then the Job Profile is automatically converted to corresponding PAD files submitted to OpenDSP or GRMS services. The whole new deployment process in QosCosGrid, including cross-site advance reservation and job submission calls, is presented in detail in Figure 4.

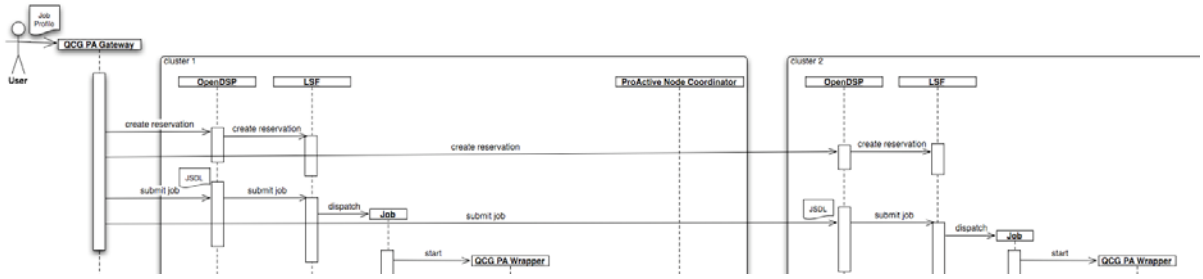

Figure 4: Cross-site resource co-allocation and job submission calls based on a new QosCosGrid deployment model for OpenDSP/GRMS in ProActive.

2.3. New Inter-cluster communication mechanisms in ProActive

The basic ProActive Library transport layer is based on Java RMI. Java RMI, is a Java application programming interface for performing the object equivalent of remote procedure calls. Using RMI a java application can call methods of a remote object on a different Java Virtual Machine. Naturally, the two JVMs could be on the same machine or on different machines connected via a network. The RMI type communication usually consumes one TCP/IP port per one remote object instance and ports are randomly selected. Moreover, it is almost impossible to configure a firewall to forward the RMI traffic which in practice is a big issue we identified in WP1. To deal with this problem the ProActive environment can be configured to use the RMISSH communication protocol, which is in fact a standard RMI protocol tunnelled over the SSH protocol. The example inter-cluster RMISSH communication is shown in the figure below.

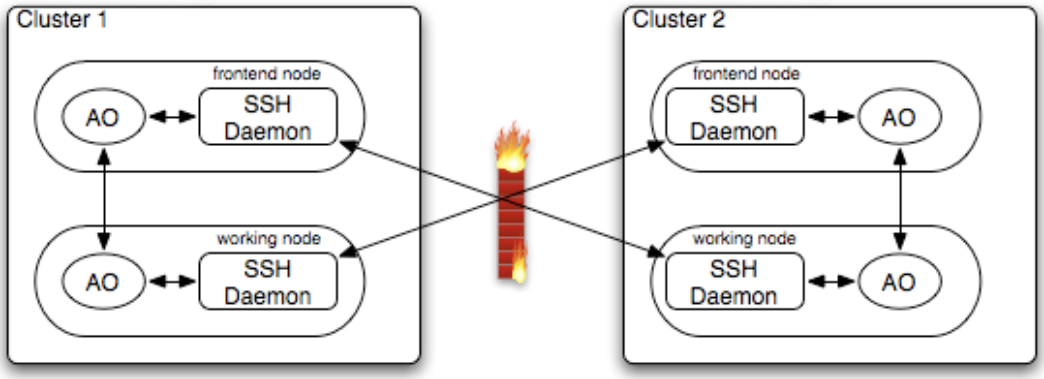

Figure 5: RMISSH communication protocol used by ProActive.

As it is presented above, the communication between Active Objects (AOs) in ProActive within one computing cluster can be established easily whereas external AO calls have to (or often must) go via firewalls due to security requirements and access policies defined in different administrative domains.

When two Active Objects running on hosts belonging to different clusters need to communicate the on-demand SSH tunnel could be established between these

two hosts. Although this solution solves most of the firewall issues, as SSH based communication is typically allowed, the approach has still some drawbacks and additional requirements:

- The public key based authentication must be carefully configured to allow passwordless authentication between each pair of hosts within the whole cross-cluster computing infrastructure (this means that used private keys must be unencrypted which brings some security risks).
- Every host must be accessible externally using SSH protocol, thus clusters behind NAT are not able to communicate using RMISSH protocol.

2.4. QCG-ProActive tunnelling

To give the possibility to use computig clusters behind NAT in cross-cluster deployments a new ProActive communication protocol (named RMIQCG) was developed in WP1. The RMIQCG concept is similar to RMISSH but instead of using the SSH protocol for tunneling TCP/IP connections it takes advantage of the SOCKS protocol. In order to use RMIQCG protocol each computing cluster must have a SOCKS server deployed at the front-end node. The port on which the SOCKS server is listening must be externally available. The example cross-cluster communication using the RMIQCG protocol is illustrated in Figure 6 below.

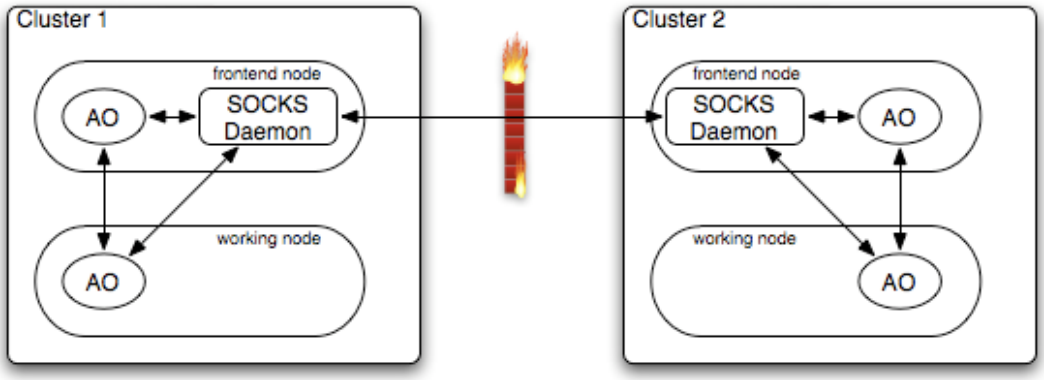

Figure 6: A new ProActive communication protocol for cross-cluster communication based on the SOCKS mechanisms.

As you can see a cross-site communication is established between SOCKS servers deployed on front-end machines in different computing clusters and the internal communication and access to other hosts are protected by the firewall. However, this approach may introduce some scalability problems so currently we perform in WP1 various tests to introduce more advanced tunneling techniques based on QCG-OMPI.

2.5. QCG-OMPI cross-site communication improvements

This section very briefly describes improvements for cross-site communication in QCG-OMPI as well as its integration with OpenDSP and Platform LSF due to the fact that all details are reported in the D1.2: Adapted Version of the OpenMPI Communication Library deliverable.

Currently, in QCG-OMPI Inter-Cluster-Communication Services (ICCS) allow processes located across the grid to communicate with one another in spite of security policies such as network or port address translation (NAT/PAT) and firewalls located between them. Besides communication issues, applications must be able to terminate in spite of the volatility of resources. Regarding the use-case applications and their needs in terms of memory and data dependencies, an application-driven solution is clearly the best approach: the application is in charge of saving its state on regular basis and performing a rollback/recovery on previously stored checkpoints. The QosCosGrid middleware provides application developers with an API to be able to store application checkpoints on reliable and efficiently accessible media. The present state of the prototype allows efficient and transparent inter-cluster communications. The performances of the communication techniques proposed by QCG-OMPI have been evaluated and show a small overhead (Figure 7).

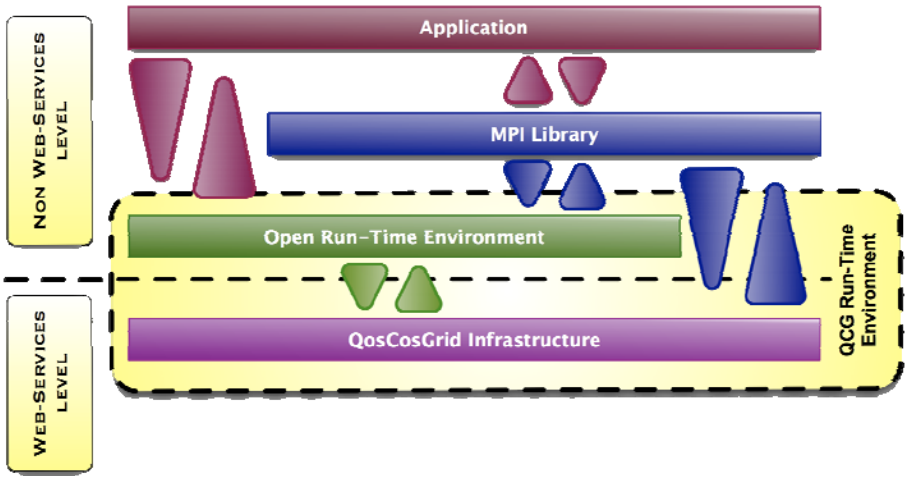

Figure 7: QCG-OMPI generic architecture and its integration with the QosCosGrid infrastructure.

A multi-cluster resource reservation system based on improved OpenDSP and LSF is supported by QCG-OMPI, and the launcher can be used to start the application across clusters. The extended run-time environment can transmit information about the physical underlying topology on which the application is being. This information can be used to adapt the application at run-time and organize communications in order to make better use of the networks' characteristics. A set of efficient hierarchical group communications is provided and has been studied. They make use of the topology information to build adapted hierarchical communicators and use theses communicators to schedule communications. The diagram below (Figure 8) shows some basic steps performed by the QCG-OMPI environment to setup a multi-cluster parallel application execution.

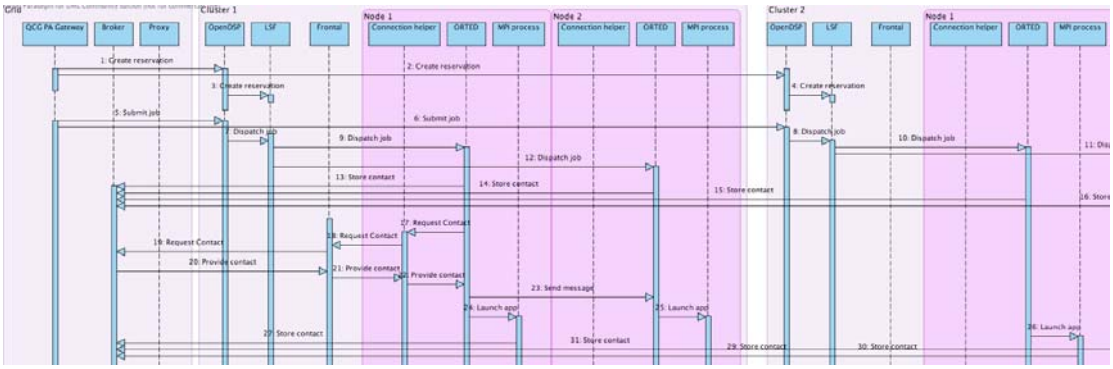

Figure 8: Cross-site resource co-allocation and job submission calls for OpenDSP/GRMS in QCG-OMPI.

A similar to QCG PA Gataway is used in the first step of the QCG-OMPI deployment process to contact OpenDSP services exposing the access to computing clusters managed by queuing systems in different administrative domains. In this steps advance reservation calls are forwarded down to Platform LSF to prepare in advance an appropriate number of computing resources used later for MPI processes. The way QCG-OMPI processes are synchronized is slightly different comparing to QCG ProActive mentioned in previous subsections. In all clusters dispatch job calls in OpenDSP/LSF setup ORTED services together with appropriate initialization parameters to contact a Broker service responsible for management of the whole cross-site MPI application execution. Additionally, a mechanism called Connection helper is invoked on each node to request contact points and "handlers" for intra and internal communication among MPI processes located in different computing clusters.

A significant improvement has been done to QCG-OMPI comparing to the first prototype with respect to the QosCosGrid run-time environment. grid services, such as Broker and Proxy, can be deployed now as persistent services and are not related to a particular MPI job. The set of grid services have to be deployed on the head-nodes by administrators, and will be used by all the jobs running on the cross-cluster environments. Moreover, it is possible to run several MPI processes on a single machine. The previous version of QCG-OMPI could support only one process on a given machine, this limitation has been relieved.

3. Remote Grid Interfaces – OpenDSP and advance reservation capabilities

Various analysis of system level APIs to Platform LSF for advance reservation encouraged us to design a set of common advance reservation APIs - called AR API for different queuing systems, in particular SGE and PBS. Therefore, based on DRMAA (Distributed Resource Management API Application) routines we have added and integrated with internal mechanisms in OpenDSP a new generic library for advance reservation management that can be also easily integrated with other queuing systems to provide a unified interface for application developers. Then, OpenDSP exposing new advanced features has been integrated with external clients, in particular QCG PA Gateway (presented in in

the previous section), GRMS and QosCosGrid Gateway for job submission and management of cross-domain experiments. New APIs and extensions to DRMAA 1.0 will be provided as a feedback for DRMAA Working Group at Open Grid Forum.

Main AR API routines have been described below. As mentioned above, in principle the AR library design has been based upon DRMAA and Platform LSF API. This explains similar naming convention and proposed routines.

3.1. AR library initalization

```
1 int ar_init(const char *contact,
2     char *error_diagnosis, size_t error_diag_len);
```

The ar_init call initializes the AR library. Required before using any AR API calls which need to communicate with the underlying queuing system. This routine can be called only once before corresponding ar_exit call. The optional and implementation dependant contact string may denote how to connect to queuing system. The ar_exit call below simply cleans up and closes the library.

```
1 int ar_exit(char *error_diagnosis, size_t error_diag_len);
```

3.2. Adding advance reservations

```
1 int ar_add_reservation(
2     char *ar_id, size_t ar_id_len,
3     const ar_reservation_template_t *art,
4     char *error_diagnosis, size_t error_diag_len
5     );
```

The ar_add call adds advance reservation with specification defined in the art structure. The argument ar_id stores the advance reservation id received from the underlying queuing system. Currently several requirements may be defined in ar_reservation_template_t.

```
/* non-vector reservation attributes */
#define AR_RESERVATION_TYPE "ar_reservation_type"
#define AR_RESERVATION_NAME "ar_reservation_name"
#define AR_RESERVATION_START_TIME "ar_reservation_start_time"
#define AR_RESERVATION_DURATION "ar_reservation_duration"
#define AR_ARCHITECTURE "ar_architecture"
#define AR_SOFTWARE "ar_software"
#define AR_NICENESS "ar_niceness"
#define AR_N_NODES "ar_n_nodes"
#define AR_N_CPUS "ar_n_cpus"
#define AR_CPU_TIME "ar_cpu_time"
#define AR_MEMORY "ar_memory"
#define AR_LOCKED_MEMORY "ar_locked_memory"
#define AR_ADDRESS_SPACE "ar_address_space"
#define AR_FILESYSTEM_SPACE "ar_filesystem_space"
#define AR_HOME_SPACE "ar_home_space"
#define AR_TMP_SPACE "ar_tmp_space"
#define AR_SWAP_SPACE "ar_swap_space"

/* vector attributes */
#define AR_V_NODES "ar_v_nodes"
```

3.3. Checking and removing advance reservations

```
int ar_remove_reservation(const char *ar_id,
    char *error_diagnosis, size_t error_diag_len);
```

Then, the ar_check_reservation call checks the reservation with a given ar_id and retrieves detailed information about it from the underlying queueing system. This is still in a conceptual phase since different queuing systems describe advance reservation using different categories. For example SGE supports the idea of "slots", while Platform LSF uses simply the number of "processors". These terms are not always fully equivalent.

```
int ar_check_reservation(
    const char *ar_id,
    ar_information_template_t **ari,
    char *error_diagnosis, size_t error_diag_len
    );
```

3.4. Job submission using advance reservation

Note that job submission is not covered by AR API. One should use directly queuing system API for job submission and control, or take advantage of DRMAA routines as we did in OpenDSP. During the submission process, there is a possibility to pass in a job template defined in DRMAA a queuing system dependant attribute (this is called drmaa_native_specification). This can be used to pass a reservation id into which job is supposed to be submitted.

3.5. Other improvements to OpenDSP

We managed to release a new version of OpenDSP supporting OGSA-BES and HPC-profile OGF specifications. A final version of the computing service is

scheduled for the next reporting period but a new release of OpenDSP is under testing procedures in QosCosGrid test bed. To meet administrators needs we have re-implemented internal OpenDSP mechanisms to complete privilege separation with the service. In a nutshell, privilege separation uses two processes: the privileged parent OpenDSP process that monitors the progress of the unprivileged child processes responsible for job submission and monitoring of different end users. Moreover, a well defined interface between privileged parent and unprivileged child allows the child to delegate operations that require privileges.

4. Low level extensions to Platform LSF

4.1. Global advance reservation in Platform LSF

In order for the meta scheduler, in our case GRMS, to propose a scheduling plan for future jobs it needs to analyse current status of the resources i.e. which ones are available and when.

The grid is composed of several clusters which are themselves potentially used outside of QosCosGrid. Thus it is important, while the meta scheduler does its work, to block new job submission requests. Otherwise as the scheduling plan can take several minutes to complete, the picture of free/used resources known by the meta-scheduler could be different from the reality. So two steps are important there:

1. Prevent new jobs submissions to the local schedulers to guarantee that information of resources availability is not altered while the meta scheduler is analysing them.
2. Give a list of available resources with details such as: hostnames, number of slots/processors and time periods.

4.2. Current solution

Presentation

The goal is to offer a new command/C function to reserve the maximum number of available slots in a cluster using the LSF Advance Reservation feature. It allows to book resources for a given period and for a specific user or group of users that no one else can use. Only a Administrator-level user can sets reservations.

Using this technique the “grid user” can book resources that will be reported back to the meta scheduler. All those resouces will not be available to any other users and so acts like a lock. Once the meta scheduler completes the scheduling plan using those information it removes the global reservation and reserves resources for the different users according to the plan.

General algorithm

The algorithm consists of three main steps:

1. Discovery of available slots and time periods (host by host)
2. Book resources according to these information.
3. Store booked resources information in a data structure to report back to the caller.

Technical details

Here are some implementations details.

```
typedef struct one_rsv {
  int procs;
  char *time_window;
  struct one_rsv *next;
} one_rsv;

typedef struct {
  char *hostname;
  one_rsv *rsv_list;
  one_rsv *last_elt;
} rsvs_per_hosts;
```

rsvs_per_hosts is a linked list structure where each element references one host and the list of its availability (struct one_rsv) described by the number of available processors/slots and the corresponding period.

```
int qcg_lock(char *begin_time, char *end_time, char *global_rsv_id, char
*username, rsvs_per_hosts *list);
          begin_time: IN
end_time:   IN
global_rsv_id: IN reservation name
username: IN
list: OUT list of every booked resources

int qcg_del_lock(char *rsv_global_id);
global_rsv_id: IN reservation name
```

The implementation does not have an efficient nor optimal discovery/booking algorithm. Currently it only books totally free resources i.e. those which do not have any reservations places on them. A difficulty comes from the fact a host is composed of several slots/processors and then it is not trivial to represent the picture of availability.

4.3. LSF Network Monitoring extensions

Usually in a cluster scheduling does not involve network metrics. It is rather based on hosts metrics like memory or CPU usage. When dealing with more complex applications using topologies with different needs of latency and bandwidth then it becomes important.

Moreover with QosCosGrid the goal is to run cross-domains applications on different clusters from different ADs using each different interconnects (with different). And all those clusters/ADs can be far away from each other which implies potential big differences of latency and bandwidth depending on which clusters the applications are running on.

Latency and bandwidth are two important metrics and can be used independently or altogether to fulfill an application resources requirements. Some applications rely on latency but do not need to share huge amounts of data across processes. Other applications need huge bandwidth and so scheduling in a cluster instead of cross-domains could be important to ensure fast interconnect bandwidth will be available instead of relying on internet connections which are less predictable and powerful (both for bandwidth and latency purposes).

We focused on cross-domains measures, as it is a main goal of QosCosGrid to support cross-domains applications. Currently there are no intra cluster measures but the solution should follow the same mechanisms so we just have to adapt for intra and inter clusters architectures.

The solution is based on NetPipe that provides a simple way to measure latency and bandwidth between 2 hosts. The initial choice for NetPipe is due to its simplicity and its multiple algorithms and implementations for different kind of protocols and interconnects such as:

- Tcp
- MPI
- Shared memory
- Myninet
- etc.

To perform the measure, NetPipe is started on the first node and waiting for data to send back to the sender. NetPipe is started on the second node with the address of the first one to exchange data and to compute the latency and bandwidth between those two nodes.

```
First_host> ./NPtcp
Second_host> ./NPtcp –h first_host
…
```

For cross-domains monitoring the TCP version is enough for now but if we decide to extend or to have more precision for intra clusters measures then we could use some of the other NetPipe plugins to prove some typical interconnects. Gateways of every cluster are known so the idea is each cluster periodically probes the remote gateways and stores the results locally. To follow the monitoring architecture wirh PERF, a PERF collector has been added to handle the network monitoring (probing and results storage).

Current status and limitations

The PERF module is currently deployed on INRIA and PSNC and is easy to deploy everywhere else as it comes with an installer script. Only remote gateways are measured and the list of gateways is actually read from a local static file instead of being asked to RTIS, which is the central service of information. So the file needs to be updated each time a new cluster/gateway is added to the grid.

The TCP version of NetPipe has the big inconvenient to use static ports. This leads to 3 problems:

- Open ports have to be available on each gateway (what if there is no ports or the cluster policy is not to leave open ports)
- There can be some security issues with open ports even this can be tracked on the firewall level or to use some authentication protocols.
- If nodeA and nodeB probes nodeC at the same time the communications have to be on different ports on nodeC and as they are static we have to define them in the local gateways list file, which are different on each cluster.

Optimizations and next steps

The biggest issue now is how to surpass the ports limitation, which makes NetPipe not convenient to use and raises the problem of security. The solution we will study is to use the MPI version of NetPipe with QCG-OMPI:

1. There is no need to define specific ports as QCG-OMPI infrastructure handles everything so the gateways list does not have to define ports where to connect and this list can be the same on every clusters and then removed once we query RTIS to get that information.
2. For the same reasons it can also be used for intra clusters monitoring without any ports conflicts.
3. We reuse another QosCosGrid component to achieve our goal.

Next important step is intra-cluster monitoring and compliance monitoring. The intra cluster monitoring can follow the same architecture than the one described previously except there is no call to RTIS as everything can be done querying the LSF cluster. Regarding compliance monitoring, the goal is to make sure once the application is over that it got all resources the used asked for. This topic needs to be studied but a good idea would be to query the application from time to time to ask for its bandwidth consumption for example. This can also be done through a PERF plugin that will update the database.

4.4. Platform LSF and OpenMPI integration

The previous deliverable relating to the first prototype of the integration described the work done at that time and the goal (i.e. the final prototype) we wanted to achieve. The main point is the independence of every clusters that are part of a grid which is managed (with other services) by GRMS. Now, when dealing with cross-domains applications, it means requesting resources from

different LSF clusters. Basically the LSF clusters report their own resources availability and GRMS according to this information will start jobs on those clusters. This is the important point: GRMS starts the jobs. Referring to the following picture (LSF/QCG-OMPI integration 1st prototype), the problem here is that the main command start LSF jobs on remote clusters and even if it does work it is not something that GRMS will notice easily. It is cleaner to say that only GRMS is able to submit jobs.

Current prototype

The goal was to develop the final prototype scenario. We encountered some issues at the QCG-OMPI level, which prevent the ORTEDs to correctly join the master ORTED, so the picture as stated before is not possible yet as it needs some code changes to QCG-OMPI and this is not the objective to modify the way QCG-OMPI works but rather adapting the resources provider (aka LSF) to the MPI framework behaviour and needs. So a new scenario was born, which is a mix of the 1st prototype and the idea of the final one. The following picture describes it (Figure 9).

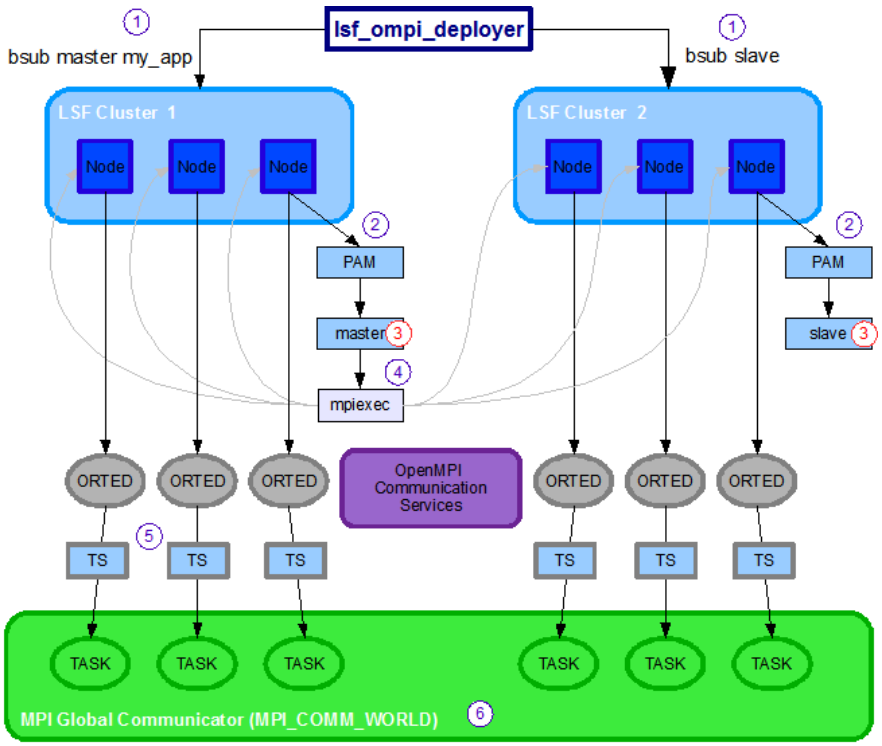

Figure 9: The second prototype of QCG-OMPI and Platform LSF integration.

Actually we keep the PAM/TS framework but LSF jobs are started independently. Then the mpiexec (or mpirun or master ORTED) command starts the ORTEDs and processes in its original way (i.e. no ORTEDs join the master one by itself). To achieve that the mpiexec command needs to know every hosts that will be involved in the job aka machinefile. So on top of the previous picture there is another mechanism (that takes place at step 3 on the figure), which gathers information needed to create the machinefile.

This is a simple way to exchange files between clusters and especially designed to gather text files in an LSF environment. On each gateway node we define two directories:

- TO_REMOTE: /tmp/lsfompi_remote
- TO_LOCAL: /tmp/lsfompi_local

A daemon is also running on those gateways in background (manage as a service by EGO) and periodically checks new files in those two directories. A file put in the TO_REMOTE directory means it should exported to a remote cluster. A file in the TO_LOCAL directory means it should be dispatched to local nodes.

As it is directly designed for LSF and QCG-OMPI there is a basic protocol so files read the right destination with right information in it. Now the files are exchanged using the scp command (SSH protocol).

The slave script creates its local machinefile based on the hosts assigned to the LSF job and named following the job name. Three lines are added at the top of the file:

- The cluster gateway where is located the mpiexec command (master ORTED) for this job. This information was passed as an argument to the slave script at the LSF submission time.
- The user name of the job owner.
- A keyword all or remote:
    - All: the file is sent to every hosts involved in the LSF job
    - Master: the file is sent to the first node of the LSF job

Then the slave script sends this file to the TO_REMOTE directory of its own cluster's gateway where is located the daemon.

If a file is in TO_REMOTE:

- The first line of the file is the remote node where the file must be sent in the TO_LOCAL directory (the first line is removed before sending).
- If a file is in TO_LOCAL:
- The file name is composed of the grid jobname (the same LSF jobname for all jobs across clusters part of the same grid job) so it can be used to ask the LSF master which machines are used for this job.
- The first line of the file is the user name and is used to know in which directory (usually "/tmp/username-jobname") put the file.
- If the last line is the keyword "master" then the file is copied (using scp) to the first host of the local LSF job machines list. If it's "all", all nodes will received the file. For the integration of LSF and QCG-OMPI only the "master" keyword is used for the moment.

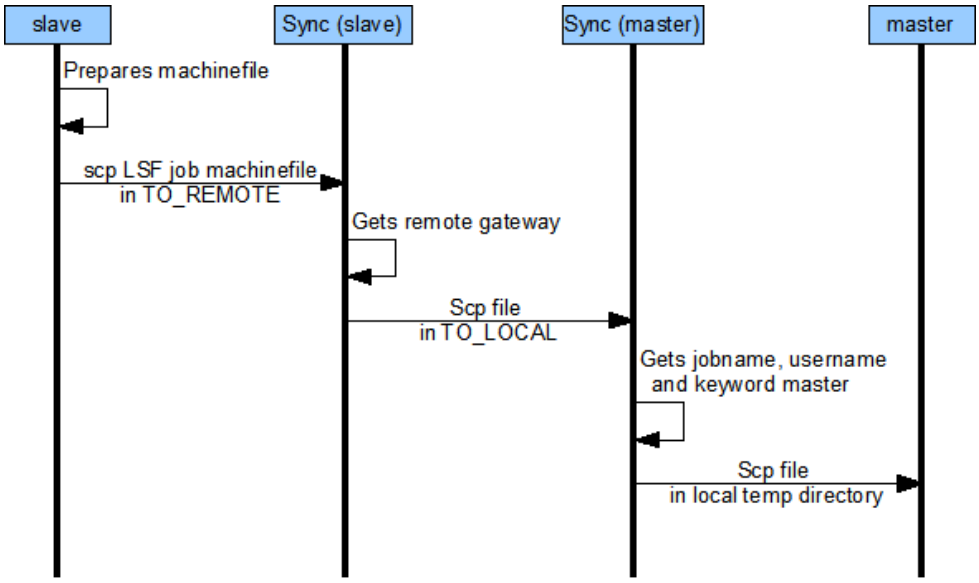

Figure 10: Platform LSF and QCG-OMPI cross-cluster synchronization.

Optimizations

There are some improvements to do on the jobs termination. The first job is the mpiexec command so when the MPI application ends the job ends too. But the other jobs on the remote cluster are waiting for an end and the way PAM/TS are used here is not the usual way but it does not prevent to correctly complete MPI simulations. Some improvements could also be done on the files exchange protocol so it is more robust and could be more general than exchanging text files only. Future work will also concern how to integrate QCG-OMPI Fault Tolerance capabilities with LSF.

5. QCG-OMPI and QCG ProActive basic performance tests

In order to demonstrate the usability of second prototype of the QosCosGrid system in WP1 we have performed a set of benchmarks and tests to measure communication time, latency and bandwidth for parallel applications written in QCG-OMPI in our test bed. A simple parallel 2 process MPI application was recompiled with QCG-OMPI and executed within a cluster, as a cross-cluster experiment between two computing clusters located in Europe (INRIA and PSNC) and as a long distance cross-cluster application executed simultaneously in Europe and Australia (PSNC and UQ). As expected we have obtained very interesting results showing a huge difference among these three different runs (see Figure 11). The testing application was performing basic calculations together with asynchronous bi-directional communication between two processes. The first parameter we measured was an average communication time between two MPI processes while increasing a number of data transferred. For communication intensive MPI application with a relatively small data packages transferred the internal MPI communication was around one hundred times faster than the cross-cluster QCG-OMPI execution in Europe and more than thousand times faster than the cross-cluster execution between clusters located in Europe and Australia. However, for a bigger size of messages

exchanged between MPI processes the communication time was much closer to results obtained in cross-clusters QCG-OMPI executions. Note, for cross-cluster connections we used not optimized Internet based TCP/IP connection channels protected by the SSL/TLS protocol. It is also interesting to notice how the random Internet traffic was changing during our tests and consequently influenced the communication time in cross-cluster executions whereas the internal cluster communication was dependent only on the size of messages. Therefore, it is clear that nowadays communication intensive type parallel applications would not gain a lot from cross-site executions. However, as demonstrated by many use cases in WP3, the possibility to exchange data among MPI processes allocated on a large number of computing resources in different clusters as well as the access to a bigger distributed memory would help users to deal with bigger instances of problems or reduce a total execution time.

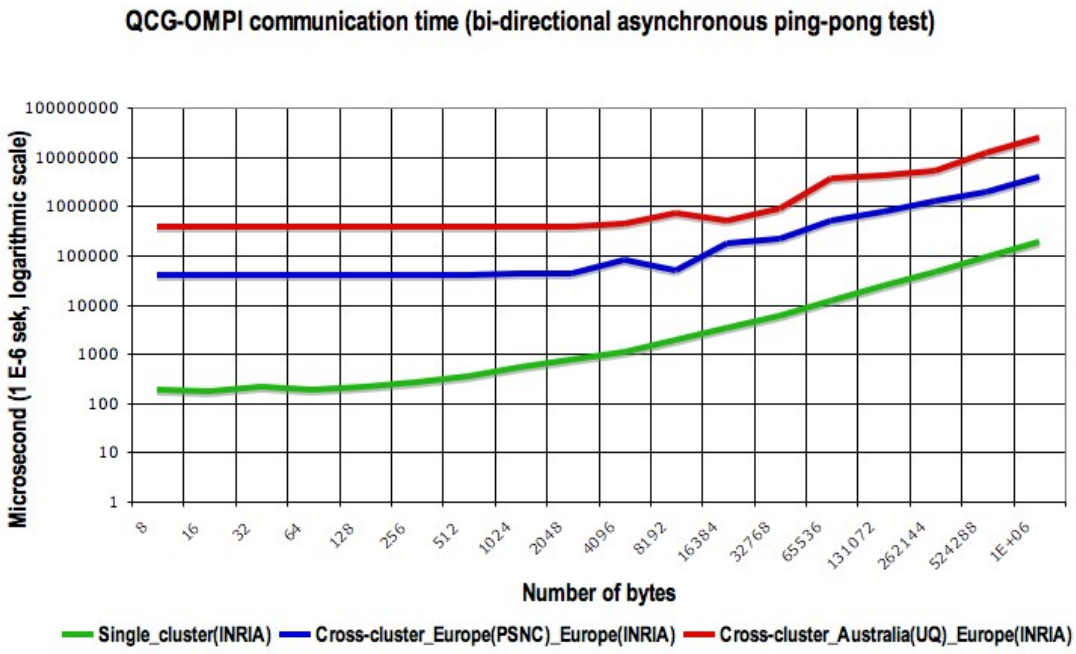

Figure 11: QCG-OMPI inter- and intra-cluster tests.

The next two diagrams (Figure 12 and Figure 13) show two other important network parameters: bandwidth and latency we measured during the bi-directional asynchronous ping-pong QCG-OMPI tests. There is no surprise that the bandwidth within the cluster is much better than cross-site connection using a regular Internet connection in the QosCosGrid test bed. It is important to mention here that using dedicated Internet channels among computing clusters we may improve significantly the bandwidth and consequently reduce the communication time for cross-cluster parallel application executions.

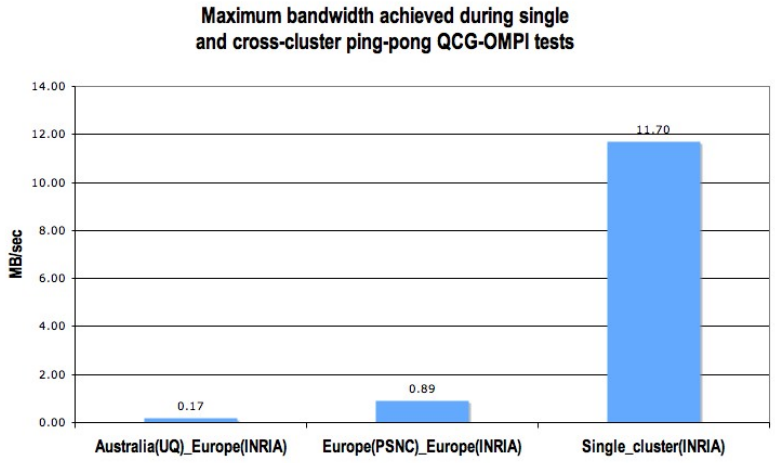

Figure 12: Maximum bandwidth achieved during the QCG-OMPI tests.

Even more interesting was to measure the minimum latency during QCG-OMPI internal and cross-cluster executions. It is clear that for cross-site parallel application executions geographical locations matter. The minimum latency we achieved between Europe and Australia was around 0.1684s which was about two thousands more than the minimum internal cluster latency. However, the latency was only three times slower than the theoretical latency measured for the connection between Europe and Australia with a speed of light!

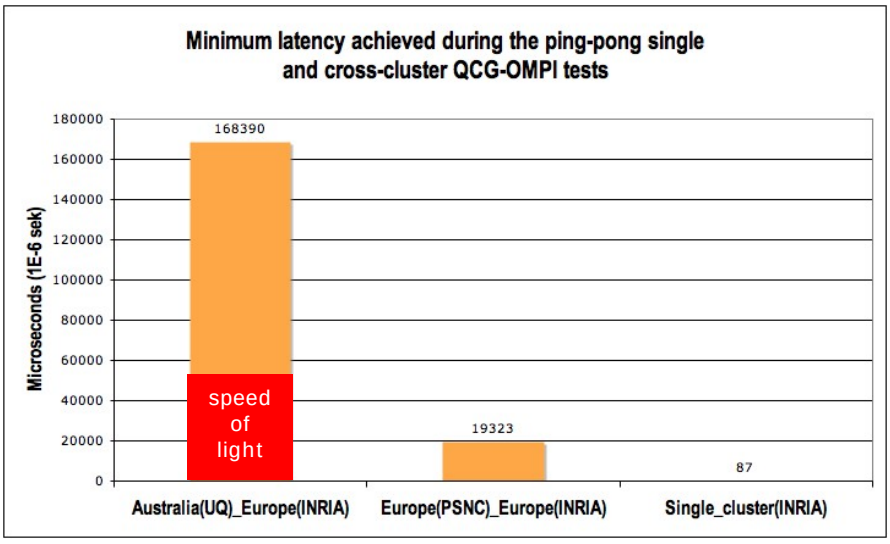

Figure 13: Maximum bandwidth achieved during the QCG-OMPI tests.

6. Web based QosCosGrid Gateway and test procedures

6.1. QosCosGrid Gateway and test procedures

A service-oriented architecture proposed for QosCosGrid consists of various services that developers created and setup at administrative domain and grid layers. The services that they develop have published interfaces and can be also easily tested by developers writing unit tests for different components.

Developers can use test suits to run both periodic and on-demand tests using our web-based QosCosGrid Gateway (see Figure 13). All results are collected in a database so we can perform some statistical analysis of the reliability of different services as well as find out more details about past failures.

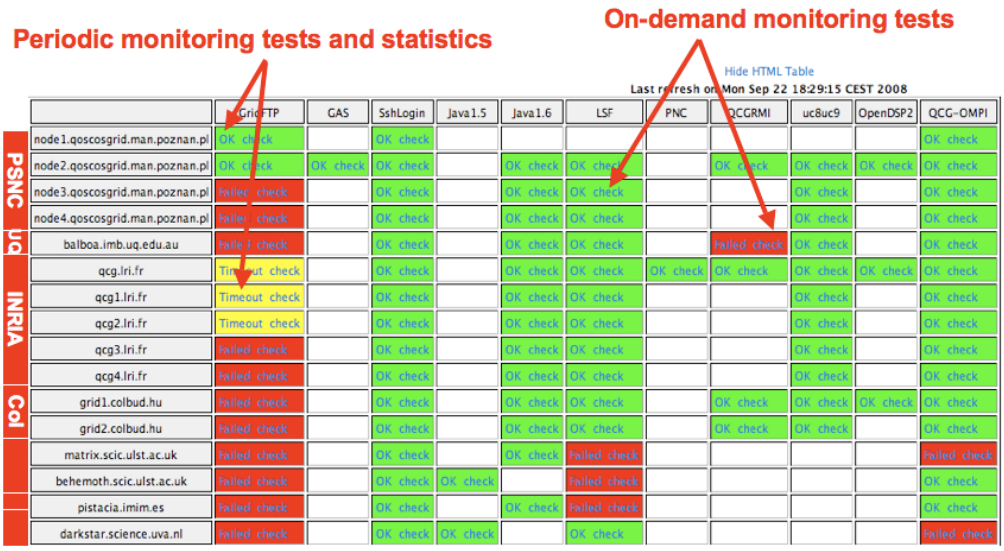

Figure 14: Web based QosCoGrid test suite created to perform periodic and on-demand tests of different components and services deployed in the QosCosGrid test bed.

Clearly, the test suite and tests it includes can be run to validate the service independently from any application that uses the service. It is also a good practice to run the unit tests during an automated build process. There is no reason for a QA tester to test an application if the unit tests did not complete successfully. More and better testing usually means fewer defects and a higher overall level of quality.

Additionally, we have recently added a new test bed at QosCosGrid Gateway showing monitoring and computing resource characteristics to provide a feedback for both end users, developers and administrators in different administrative domains. Various monitoring tests are invoked periodically and their results are presented as graphical maps and diagrams on the web (see Figure 14). The first two maps show results of bi-directional tests of cross-domain QCG OMPI and QCG ProActive applications measuring bandwidth and latency among front-end machines of computing clusters in different administrative domains connected via Internet. We have added also useful Gantt charts showing local and cross-domain job executions as well as advance reservation and co-allocation of computing resources in time.

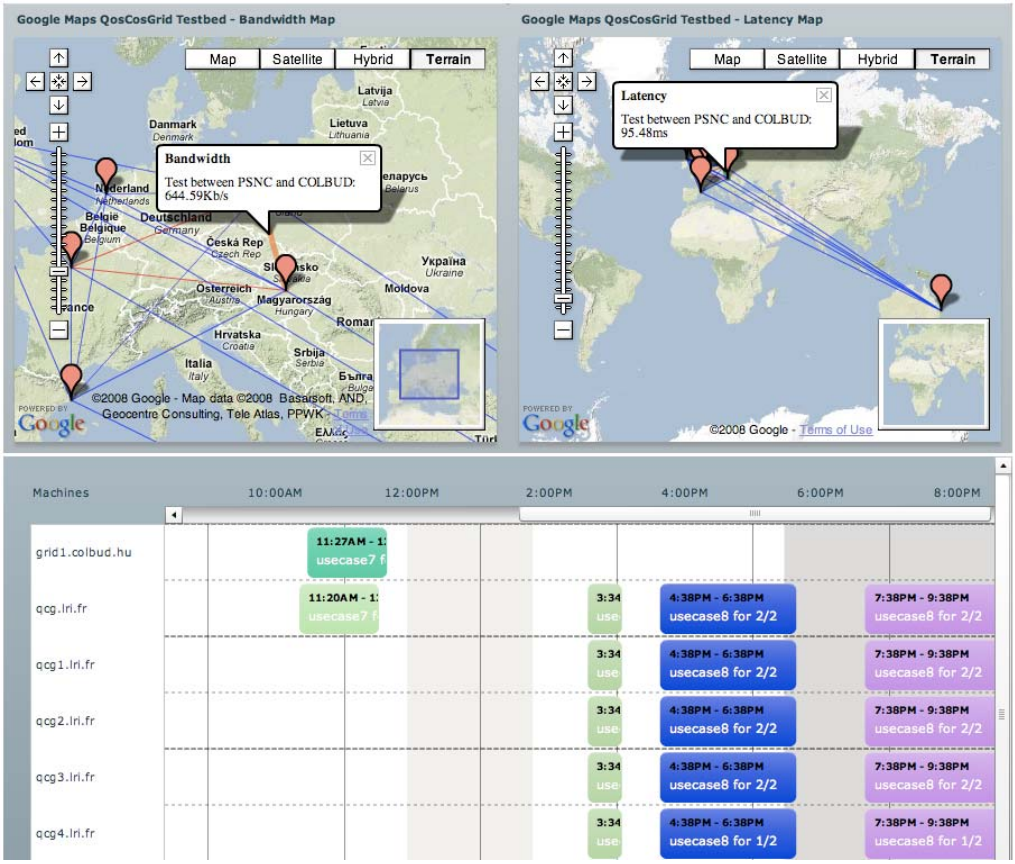

Figure 15: Web based test suite to perform periodic and on-demand tests of different components and services deployed in the QosCosGrid test bed.

6.2. Security issues in WP1

From the QosCosGrid security perspective administrative domain (AD) and grid components must enable secure sharing and control of all resources (computing, applications, storage, network, etc.) across geographically dispersed domains. Therefore, we have compared and analyzed security, privacy and data integrity issues in the QosCosGrid test bed where EU and non-EU partners are involved. In general, security tools are concerned with establishing the identity of members and resources (authentication), protecting network communications, and determining who is allowed to perform what actions (authorization), supporting functions such as managing user credentials and maintaining AD group membership information as well as accounting statistics (accounting) that must be have been collected in ADs for SLA, billing or charging of users.

List of figures

List of tables

