034115

PHOSPHORUS

Lambda User Controlled Infrastructure for European Research

Integrated Project

Strategic objective:
Research Networking Testbeds

Deliverable reference number D3.9

Final Report on the period M25 – M33

Due date of deliverable: 2009-06-30
Actual submission date: 2009-06-30
Document code: Phosphorus-WP3-D3.9

Start date of project:
October 1, 2006

Duration:
33 Months

Organisation name of lead contractor for this deliverable:
Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.

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

Final Report on the period M25 – M33

Abstract

This report will summarize the results of the final WP3 activities during month 25 – month 33 dedicated to consolidation of middleware implementations, final work on interfaces to G^{2}MPLS and HARMONY, forwarding of authentication and authorization credentials between the middleware layer and network layer, and last experiments made with the final middleware.

Table of Contents

Table of Figures

0 Executive Summary

This report summarizes the results of the final WP3 activities during project month 25 – month 33. The activities in this final phase of the project were dedicated to consolidation of middleware implementations, finalisation of the implementation on interfaces to G^{2}MPLS and HARMONY based on the experience of the PHOSPHORUS demonstrations at the Supercomputing 2008 in Austin, the ICT 2008 in Lyon and the PHOSPHORUS workshop during the TERENA Networking Conference 2009 in Malaga.

Moreover, the implementation of forwarding authentication and authorization credentials between the middleware layer and network layer was completed during this period, and last experiments using with the final middleware have been performed.

For the Intelligent Network Caching Architecture the focus was on algorithms and parameters to optimize the performance of INCA.

.

1 Overview

This final report will summarize the results and achievements of WP3 during month 25 - 33, in particular the middleware integration, the integration of the middleware layer and the network layer, and the demonstrations of the applications using the PHOSPHORUS optical testbed at the Supercomputing 2008 in Austin, the ICT 2008 in Lyon and the PHOSPHORUS workshop during the TERENA Networking Conference 2009 in Malaga.

During the period of the last 9 months considered in this report the main tasks of WP3 were:

- Finishing the middleware development and adaptation (UNICORE, MetaScheduling Service (MSS), Resource Selection Service (RSS), INCA).
- Integration of the Resource Selection Service into the UNICORE Rich Client, final experiments.
- Evaluating the results of the testbed experiments done in the previous phase and the demonstrations at the SC08 and ICT 2008, definition of missing functionality, and implementation of missing functionality. The activity is concluded by final experiments with the enhanced interface.
- Completing the Implementation of the interface between the Grid Resource Management Systems to the G2MPLS system via the G-OUNI protocol as specified jointly with WP2.
- Integration of the Middleware with the AAI for the network layer: Analyzing the solution implemented by WP1 and WP4. Indentify the relevant information for authentication and authorization available on the middleware layer. Define the interface towards the network layer create a seamless chain of trust from the middleware client to the network layer.
- Final work on algorithms and parameters to optimize the performance of INCA.
- Finalize the integration of DDSS with G^{2}MPLS.

- Final demonstration of the application suite selected for the PHOSPHORUS project at the PHOSPHORUS workshop during the TERENA Networking Conference 2009 in Malaga.

The report follows the division of work in tasks, and presents for each task reports on the results of and achievements made. Thus, Section 2 presents the final adaptation and modification activities for the UNICORE system in the testbed, in Section 3 the results of adaption and developments of the MetaScheduling Service are described, Section 0 summarises the integration of the Resource Selection Service into the UNICORE rich client, Section 5 highlights the results of the final developments of the interface between middleware and G^{2}MPLS, and section 6 describes the simulations for the integration of the INCA middleware and G^{2}MPLS. In Section 7 the only application-related activity in this period is presented: the integration of DDSS with G^{2}MPLS. The entire report is summarized in section 0, where we also draw some conclusions.

1.1 Work package objectives and main tasks

The work-package objectives during the last 9 months timeframe encompassed:

- Integration of the Resource Selection Service into the UNICORE Rich Client, final experiments.
- Evaluation of the results of the testbed experiments done in the previous phase and the demonstrations at the SC08 and ICT 2008, definition of missing functionality, and implementation of missing functionality.
- Providing authentication and authorization information obtained in the middleware layer to the network layer supporting AAI mechanisms implemented there.
- Evaluate algorithms and parameters to optimize the performance of INCA the Intelligent Network Caching Architecture.
- Testing the final consolidations and improvements. Experiments will be carried out in the period from month 28 to month 32 in the test-bed.

1.2 Integrated Approach

The demonstrations conducted in the reporting period aim at presenting the successful integration of the four layers the PHOSPHORUS testbeds are built upon:

- Physical layer
- Operating systems layer
- Middleware layer
- Application layer

Details of the achievements demonstrated are described in deliverable D3.8 - Final Report on the achievements and results of WP3 [D3.8] and in the 3rd Annual Activity Report [Y3act].

The developments during the last phase also included finalisation of the interface to make the physical layer additionally accessible via the G^{2}MPLS and the G.O-UNI interface between the middleware and the G^{2}MPLS. More details on G^{2}MPLS can be found in the deliverable “Preliminary Grid-GMPLS Control Plane prototype” [D2.5].

The approach for the integration between the middleware and G^{2}MPLS is described in the “Report on the results of the middleware experiments during the final testbed experiment” [D3.7].

The interface between the application layer and the middleware layer, in other words how the applications may access middleware services and through these middleware services may benefit from all layers of the PHOSPHORUS service stack was described in several deliverables with “Report on the Results of the Application Experiments During the Final Testbed Experiments” [D3.6] being the latest one.

2 UNICORE

For the PHOSPHORUS project we have chosen UNICORE as the middleware to be linked with both the WISDOM and KoDaVis applications. UNICORE provides a plug-in architecture at the client side that allows for application specific plug-ins to be included. The suitability of the plug-in concept has been evaluated for both applications. The work on middleware design started in March 2007. While UNICORE 5 was the system for productive use when PHOSOPHORUS started, WP3 decided to migrate to UNICORE 6 – the web-service based implementation of UNICORE – for the PHOSPHORUS testbed.

In order to fully support all scenarios of the inter-WP demonstrations at SC08 and ICT2008, the UNICORE client plugin for the KoDaVis application was extended to select the desired demonstration scenarios. Additional changes were necessary in the negotiation protocol, to make the reservation of bandwidth as transparent to the application as possible. Data for the KoDaVis application can now be served by servers either selected by the client, the middleware (both unicast from the point of view of the network), or the network (anycast). From the point of view of the application and UNICORE client plugin, they all appear in the same way.

The KoDaVis application has been in a stable state for a long time. As stated above, it has been demonstrated at various occasions. Minor adaptations were required to support the scenarios foreseen by WP2, where either the middleware or network selects the KoDaVis data server. The actual change was necessary to continue to support collaboration among multiple KoDaVis clients, even though their data is served by different data servers.

Demonstrations were given at the SC08 in Austin, Texas, the ICT2008 in Lyon, France, and the TNC2009 in Málaga, Spain. For the demonstrations at TNC2009, a local, transportable testbed has been installed, which contained all PHOSPHORUS components in a few machines.

During the entire period the testbed implementation of UNICORE was following the general advances in UNICORE middleware and regularly updated.

3 MetaScheduling Service

The MetaScheduling Service (MSS) was initially developed in the German project VIOLA. Its main purpose was the co-allocation and reservation of compute and network resources for demanding Grid applications that used more than one compute infrastructure.

Based on the WP1 specification of the network service plane (NSP) in the previous phase the MSS adapter towards the ARGON NRPS (as defined in the VIOLA project) was replaced by an adapter for the NSP. Figure 1 shows the resulting architecture that was used for the demonstrations during the first project review in December 2007 and which then was used as part of the regular PHOSPHORUS testbed infrastructure. As Figure 1 depicts the different NRPS available in the PHOSPHORUS network environment having an interface to the NSP while the request for network resources from the MSS is handled by the NSP and transparently forwarded to the respective NRPS. In the last phase of PHOSPHORUS the HARMONY system became available. Since the interface was slightly different from the one previously used to access the NSP through an adapter it was necessary to rework this interface, which was completed during the last months of PHOSPHORUS.

Figure 1 also shows the interface with the Control Plane and the G^{2}MPLS developed by WP2 in the PHOSPHORUS project. The implementation of these interfaces has been mostly done between month 18 and month 24 with a preliminary version of G^{2}MPLS. Since the final version of G^{2}MPLS became available for the last project phase the implementation of the interface could be adapted and is complete now. More details about the architecture can be found in Section 5 (Interfacing between G²MPLS and Resource Management System via G-OUNI).

The last focus of the MSS extensions was the hand-over of the user credentials via the middleware layer to the network layer. This was achieved for both the Service Plane through the HARMONY system and the Control Plane through G^{2}MPLS. The approach used is based on SAML-Assertions, which contain the user credential information in both cases. Each system that forwards the SAML-Assertion to the next layer adds its own signature to the assertion. Through this signing the downstream systems may verify that the SAML-

Assertion with the user credential information was forwarded by trusted systems and not altered during the transport. The only requirements to perform this verification are the availability of the root certificate(s) of the CA(s) issuing the server/service certificates of the systems forwarding the assertion, and, of course, trust in this CA(s).

With this extension the PHOSPHORUS environment provides a unique Authentication and Authorisation Infrastructure, which seamlessly stretches from the user to the token-based security mechanisms on the network layer.

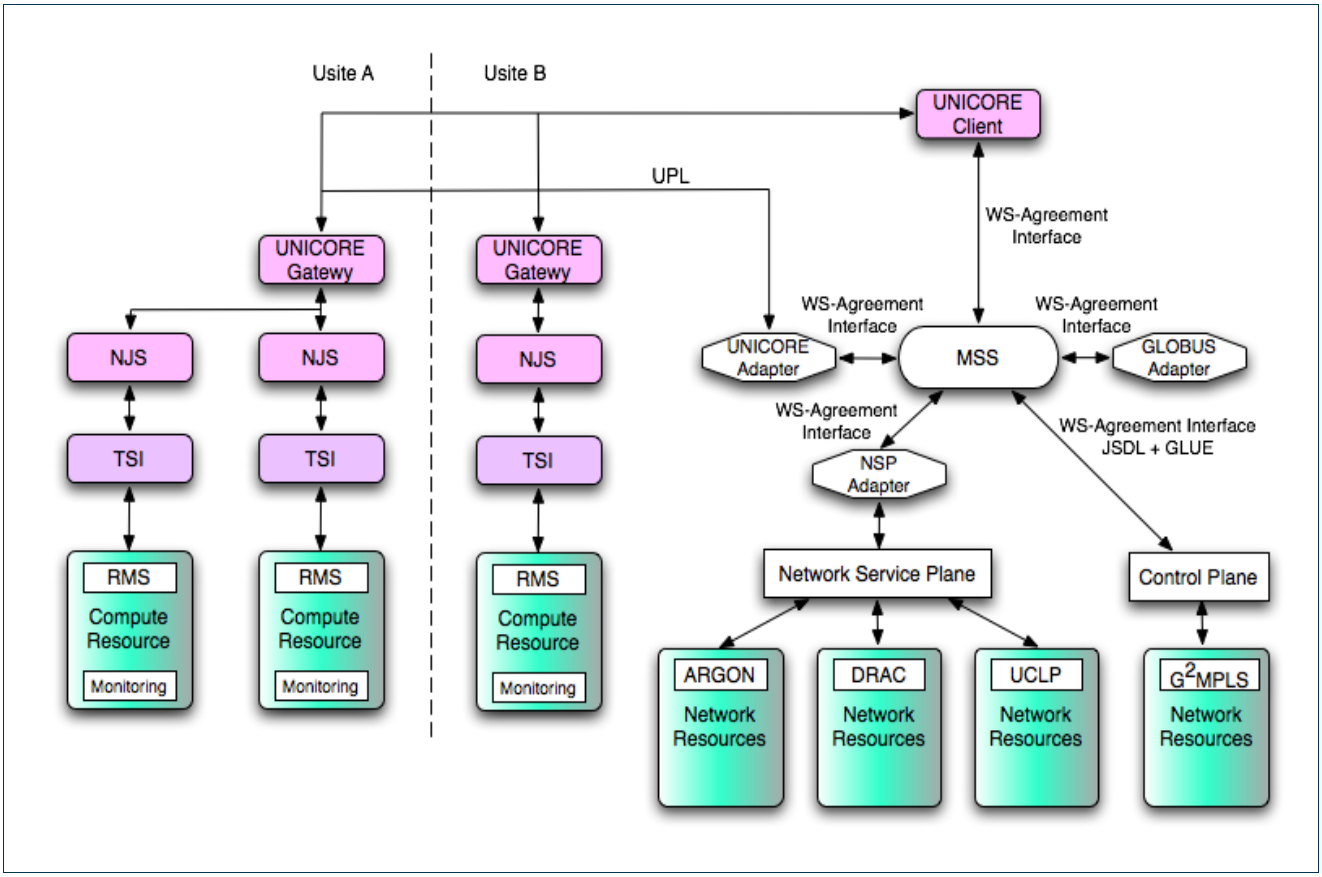

Figure 1: Integration of the MetaScheduling Service and the Network Service Plane and the Control Plane

3.1 Summary of the achievements

Interface between the MetaScheduling Service and the Network Service Plane:

- Now realised through the HARMONY system
- WS-Agreement has been selected as protocol and language for advance reservation of network QoS, e.g. for co-allocation
- Harmony exposes a Web Service Interface
- Forwarding of user credential information as SAML-Assertion to the Service Plane

Interface between the MetaScheduling Service and the G^{2}MPLS:

- WS-Agreement has been selected as protocol and language to request co-allocations
- MSS exposes a co-allocation service for the Control Plane
- Interface to G2MPLS based on BES and GLUE
- Forwarding of user credential information as SAML-Assertion to the Control Plane

Finalisation of integration of MSS components with UNICORE 6

- Support of UNICORE 6 security
    - trust delegation to support complete chain of trust between user, wsag4unicore6 adapter, and UNICORE 6 server components
    - WS-Security for trust delegation verification
- Default implementation of WS-Agreement based SLAs
- File Stage-in / File Stage-out for WISDOM-like applications
    - UNICORE 6 server to server single file transfers
    - UNICORE Basic File Transfer for stage-in / RBYTEIO for stage-out
    - File Stage-in from archive / File Stage-out to archive
    - Transfer of complete directory structures

- Archives are unpacked on target system (stage-in)
- Archives are created on source systems and transferred to target systems (stage-out)
- UNICORE Basic File Transfer for stage-in / RBYTEIO for stage-out

The overall result is WSAG4UNICORE6 – WS-Agreement based UNICORE 6 integration.

4 Resource Selection Service

The Resource Selection Service (RSS) has been designed in order to allow the MetaScheduling Service (MSS) to place a job request automatically on an appropriate computing system taking into account the specific requirements of the job as imposed by the application to be executed.

In the previous phase the work on the functionality of the RSS was completed and tested. However, the full integration into the user client was not yet achieved. This was the focus of the work in the last phase of PHOSPHORUS.

In the course of this work the RSS algorithm has been implemented in the Eclipse Rich Client framework. This allows to contribute the RSS as a plug-in for the UNICORE Rich Client, which is the user client for the UNICORE6 middleware and the MSS. Figure 2 and Figure 3 show the resulting graphical interface for the user.

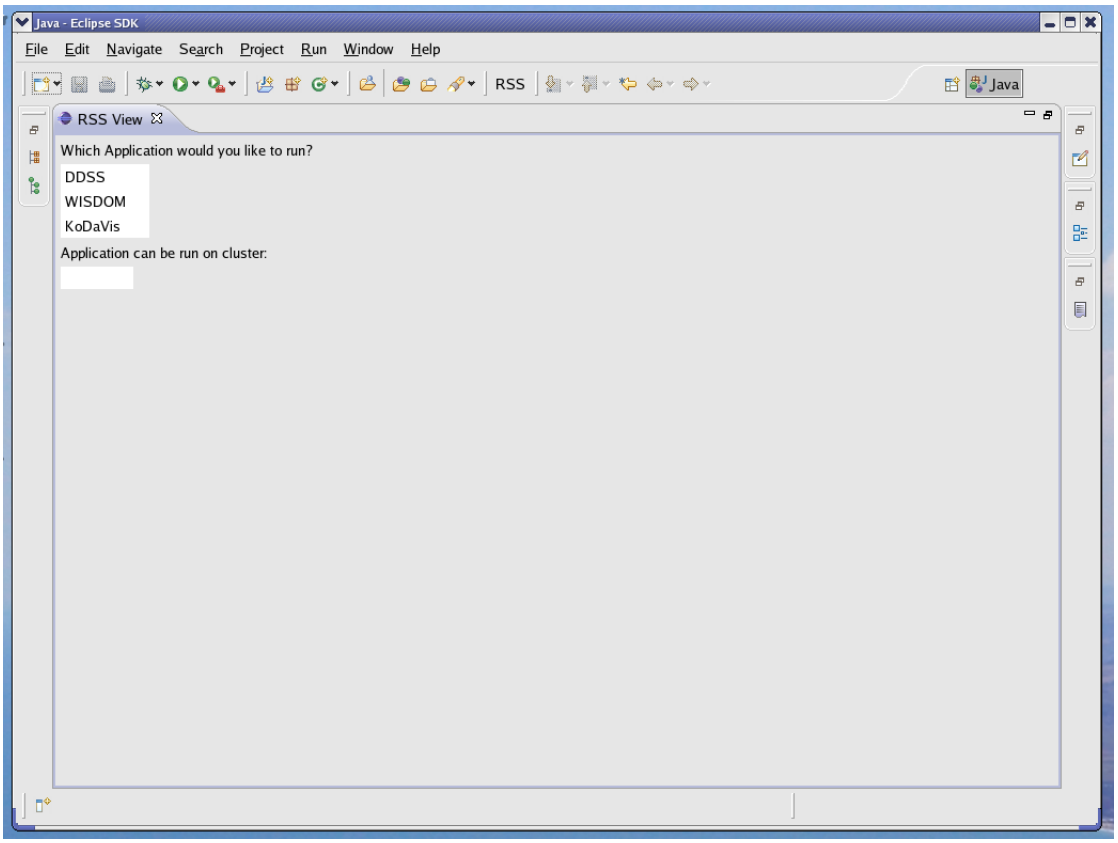

Figure 2: Interface for selecting the application a user wants to execute (note: only applications available on clusters the user has access to are presented)

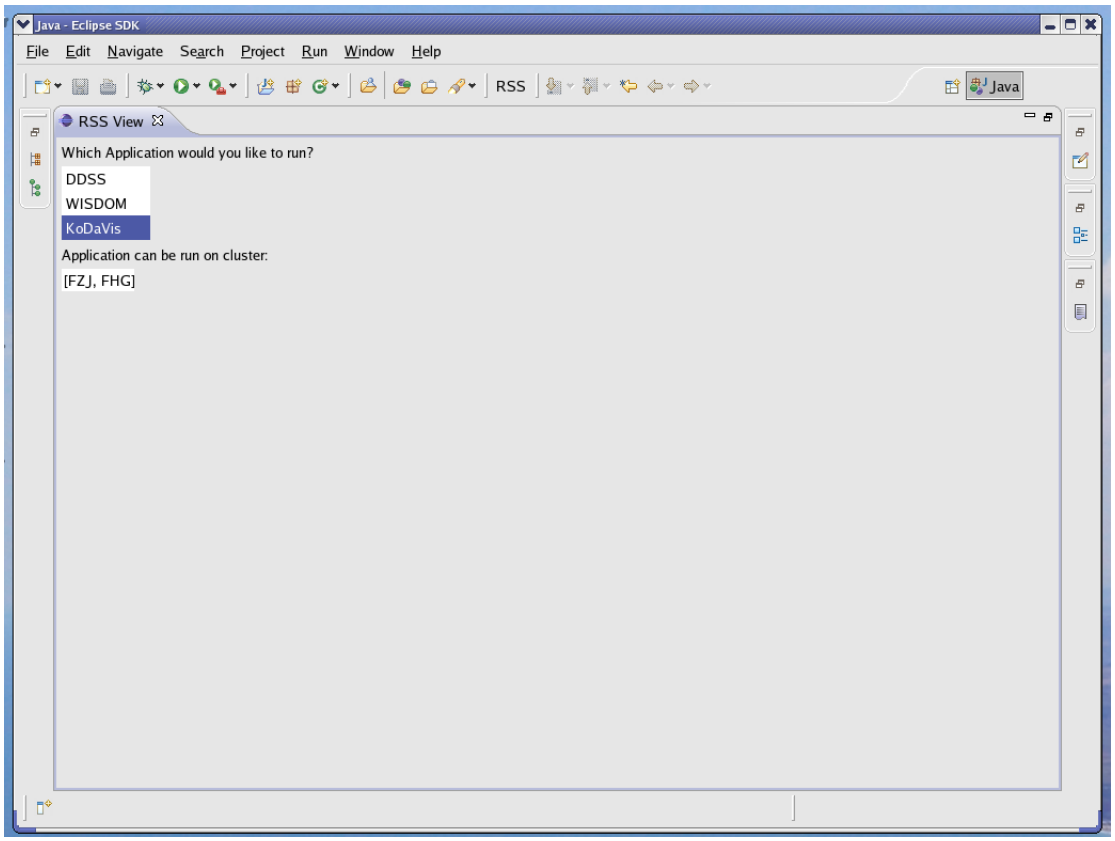

Figure 3: Suitable clusters to execute the application (KoDaVis) selected by the user returned as result of the Resource Selection Service processing.

4.1 Summary of the achievements

The RSS algorithm has been implemented in the Eclipse Rich Client framework, which allows to use the RSS as a plug-in for the UNICORE Rich Client.

5 Interfacing between G²MPLS and Resource Management System via G-OUNI

In the previous phase the implementation of three different interfaces was started to support

- the G²MPLS overlay model
- the G²MPLS integrated model
- the G.OUNI interface (which is actually realised as a G-OUNI proxy)

The functionality of the interfaces is briefly described below. More information about the underlying technologies used, the frameworks and standards the implementation is based upon may be found in D3.7.

5.1 G²MPLS overlay model

In G2MPLS Overlay model, the Grid layer has Grid and network routing knowledge in order to provide Grid resource configuration and monitoring (as in its standard behaviour) plus network resource configuration and monitoring. G2MPLS acts as an information bearer of network and Grid resources and as a configuration “arm” just for the network service part.

This model is intended to be mainly deployed when most of the computational and service intelligence need to be maintained in the Grid layer for specific middleware design and functional behaviours. The Grid scheduler in this case plays the leading role, which is the overall responsible for initiation and coordination of the reservation process through the participating Grid sites and the network in between.

5.2 G²MPLS integrated model

In the G2MPLS Integrated model, most of the functionalities for resource advance reservation and commit are moved to the G2MPLS Network Control Plane. G2MPLS is responsible for scheduling and configuring all the job parts, those related to the Grid sites and those related to the network.

Grid sites are modelled as special network nodes with specific additional Grid resource information. The resulting topology is flat and integrated with respect to the positioning of the Grid layer against the network layer.

5.3 The G-OUNI interface

The Grid Optical User Network Interface (G.OUNI) comprises a number of procedures to facilitate on demand as well as in-advance access to Grid services/resources by interfacing Grid end points and any Grid middleware with any type of network resource provisioning system.

5.4 Communication with the G²MPLS layer

The gSOAP engine is used to implement a web service based communication between G.OUNI gateway and MSS. Since the gSOAP engine does not provide support for stateful web services, currently the OGSA BES (Basic Execution Service) protocol is foreseen as the basic protocol for the MSS-G.OUNI interaction. The OGSA BES interface is implemented as a stateless web service. This has makes the implementation on the G.OUNI gateway site much easier. Furthermore, UNICORE 6 provides an OGSA BES interface since version 6.1. Therefore, this functionality can be accessed directly from the GOUNI gateway. Figure 4 depicts the resulting interoperability between MSS and G2MPLS realised through the components described before.

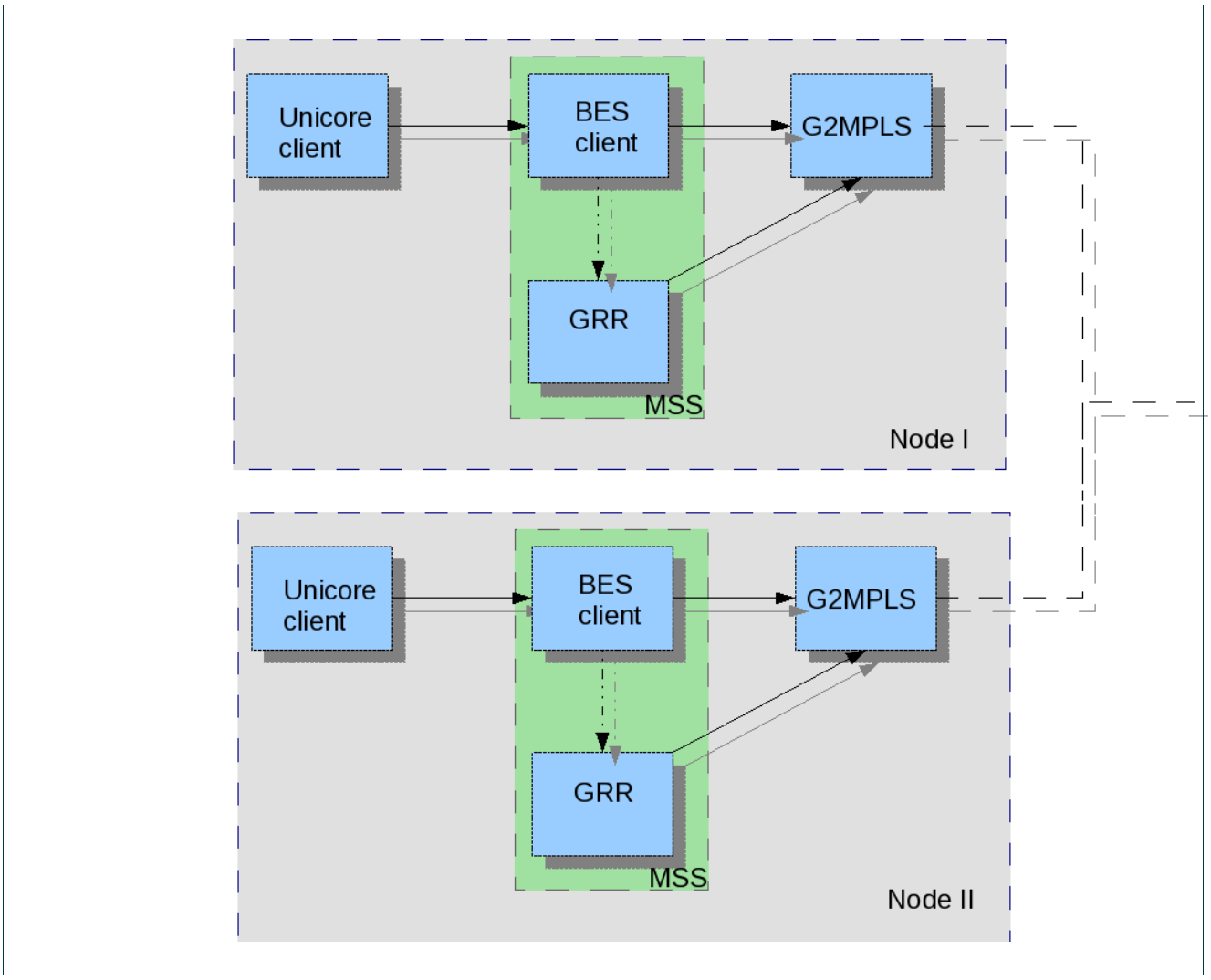

Figure 4: Interoperability and interoperation between MSS and G^{2}MPLS.

5.5 Integration of MSS, G2MPLS, KoDaVis

The interoperability between mentioned parts was a prerequisite of the demonstrations during the PHOSPHORUS workshop at the TERENA Networking Conference 2009. The implementation resulted in a successful demonstration of three previously defined use cases described below.

Figure 4 shows the communication flow of all components involved.

The UNICORE KoDaVis client can trigger MSS for a valid KoDaVis server address. Depending on test scenario, the MSS retrieves a server address either from local registry (GRR) or through G^{2}MPLS. The local registry continuously updates status of free network slots and sends it to G^{2}MPLS. Depending on this information, G^{2}MPLS chooses a KoDaVis data server and MSS sends it to the UNICORE client.

Two or more nodes are required to show the full potential of MSS-G^{2}MPLS interoperability.

Because the UNICORE client and G^{2}MPLS operate with different address conventions (TNA, UNICORE specific IP addresses) the MSS provides a mapping from one to other address system.

Description of the test scenarios:

Three different scenarios have been performed to test the interoperability between the components.

The first and second scenario didn't retrieve data server address from G^{2}MPLS.

In the first scenario the UNICORE client was aware of data server address and only a network reservation was done via G^{2}MPLS (through the CreateActivity() call).

In the second scenario the data server address was retrieved from a local repository (GRR) and sent back to UNICORE client. Also a network reservation was requested via G^{2}MPLS.

In the third scenario, the MSS was able to reserve the network via G^{2}MPLS and also to retrieve a proper data server address. Only the third scenario shows the full reservation and routing capability of the system.

5.6 Summary of the achievements

The interfaces implemented and available in the testbed by the end of the reporting period support both the G^{2}MPLS overlay model and the G^{2}MPLS integrated model.

G²MPLS Overlay Model

- the Network is seen as an additional Grid resource

- it can be used by Grid services as any other resource
- Grid services access G²MPLS via G.OUNI interface
- this supports the more “traditional” Grid approach

G²MPLS Integrated Model

- Grid intelligence is moved to network layer
- G²MPLS is responsible for scheduling and configuring (atomic) jobs
- Grid scheduler are still responsible for workflow execution
- Grid information (e.g. resource description of sites, availability information) is injected into G²MPLS via G.OUNI gateway
- Job submission to G²MPLS via G.OUNI gateway

To realise these interfaces some additional implementation work towards the integration of MSS, UNICORE and G²MPLS have been completed.

In order to enable the MSS to work with UNICORE 6, the new adapter is used to make UNICORE 6 functionality accessible via WS-Agreement protocol. Since UNICORE 6 poses high requirements in terms of security, a number of extensions were made to the MSS environment. These extensions include:

Enabling WS-Security for the MSS in order to digitally sign the SOAP messages send by the MSS client and server components, and to validate the digital signatures of the messages received by the MSS client and server components. Since these concepts are unknown on the network layer the MSS is bridging between these two environments.

Enable the MSS to support UNICORE 6 trust delegations. UNICORE 6 uses SAML 2.0 assertions to delegate trust from the issuer of a job (e.g. a user) to the entity that actually submits a job to UNICORE (e.g. a scheduler). Therefore, additional functionality to generate UNICORE 6 trust delegations (TD) by the MSS client, to validate TD objects via the digital message signature on the server side, and to generate and validate of trust delegation chains is required for the MSS.

Framework properties at month 33

- Glue based interface to inject and receive routing information from/to the G2MPLS
- BES based interface to submit / receive Grid jobs from G²MPLS
- Integration of MSS with G²MPLS via G.OUNI Gateway
- All required components implemented, tested and successfully demonstrated at the TERENA Networking Conference 2009.

6 INCA middleware - Intelligent Network Caching Architecture

6.1 Introduction

INCA is an intelligent storage network that provides data-transfers with high performance. As we depict in the following figure the functionalities of INCA are divided in two layers. The first layer is called Data Management Layer and its purpose is to infuse in the system totally distributed and automated management functionalities. In more detail we test and evaluate three major features that each storage network must have:

The first one is scalability. In order to achieve this goal we have implemented a distributed hash table (DHT) that is used for the distributed metadata maintenance and the distributed location of them.

The second is the balanced use of storage network resources in terms of storage space and network bandwidth. In order to achieve this goal any data chunk is hashed and according to the output of a uniform hash function is stored in the proper storage node.

The third is the fault-tolerance of a storage network. A stable routing algorithm used for the data and metadata location has been developed and has been evaluated.

Through our experiments we have evaluated our architectural decisions and the performance of the algorithms that we have used in order to enhance the storage network with these features. The results show that a DHT is capable to manage well a storage system but special attention has to be paid in order to maintain all the necessary data structures during dynamic conditions.

As for the second layer it is called Storage protocol layer. Its purpose is to provide:

1. Point-to-point buffer transfer in order to store chunks in the chosen (by the upper layer) node;

2. The glue between Data management layer and the storage control layer.

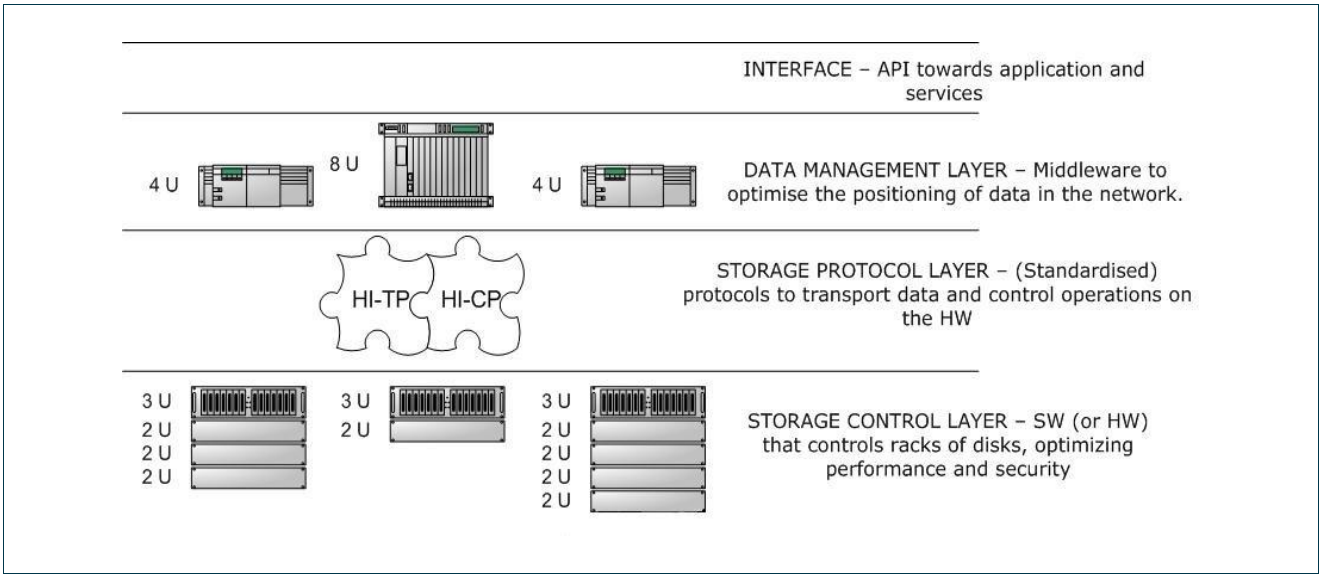

Figure 5: Layers of the intelligent storage network INCA.

In the first stage we have observed that PHOSPHORUS is an environment where a storage network with a totally distributed management will bring out its advantages. In more detail due to the high performance of the underlying network a single point of management that a centralized architecture would have introduced it would act as a bottleneck for the whole system. Additionally the automated and distributed nature of the data placement and retrieval will exploit the resources of such an environment. Due to these reasons we have selected a DHT in order to be used as the basis for the Data management layer. This DHT is called CAN (Content Addressable Network).

CAN is a distributed lookup and routing protocol. All nodes participating in CAN are equal and no hierarchy exists. ID space is organized as a virtual d-torus (a 2-d torus is like the surface a sphere). Each node is responsible for the keys in a zone of the space (in a 2-d torus zone is a rectangular parallel to the axes of the coordinates). Each value (item name) corresponds to a key. Given a key system has the ability to route the message to the node responsible for that key. CAN implements three basic functions item insertion, lookup and delete.

Nodes in CAN are self-organized into an overlay network that represents this virtual coordinate space. Each node has to discover and maintain a routing table with the IP

addresses of the nodes adjoining its own zone. These nodes are called its neighbours. All values are mapped into the space using a uniform hash function. This function creates the coordinates in the space (key) for each data item (value). A query for a key is routed through the overlay network to the node responsible for the key. Fast and reliable routing is the most important service of CAN.

When a node enters the network an existing node's zone is divided in two halves and the new node is responsible for the one of them the old node for the other. Each node has a Virtual Identifier (VID). When a zone is divided to two each child's VID is created by adding a binary digit in its parent VID. To the node with the lower coordinates in the dimension of the division is added a 0 and to the other is added a 1.

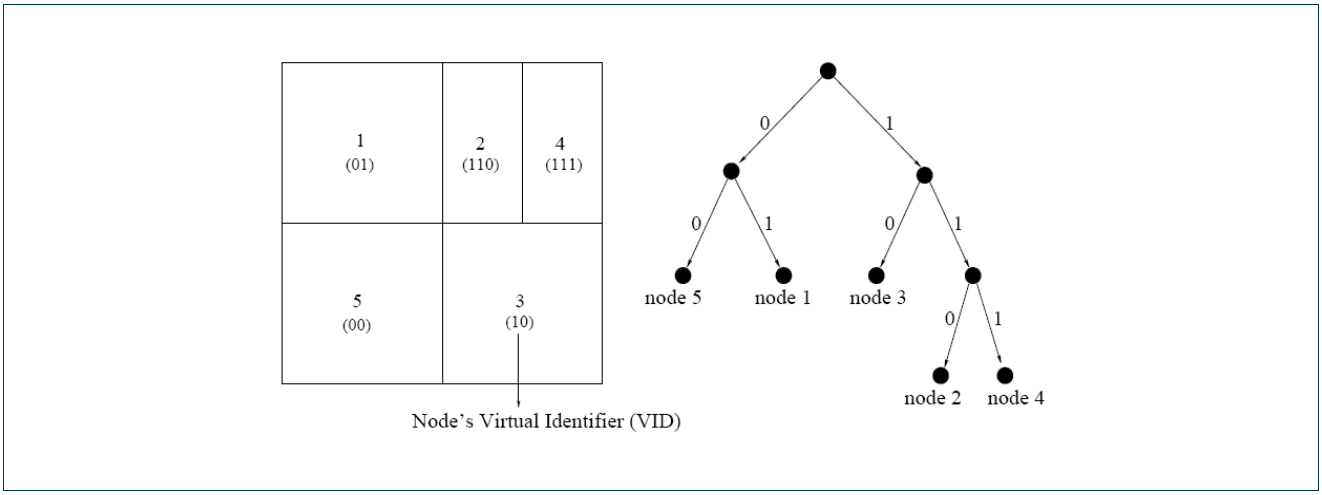

Figure 6: A 2-D CAN with 5 nodes

For a new entrance a node already in the network must be discovered. After its discovery by using CAN's routing mechanisms it could be found an appropriate node to split its zone. Some keys are more popular than others and zones are not uniformly divided. For having a load balanced network an algorithm, which finds an appropriate node for split its zone must be implemented. At last neighbours of the new and old node must be informed for this change so both nodes sending an update message to all of their current neighbours. It is also important to be mentioned that periodic messages are sent to ensure network's stability. A new entrance affects only O(d) nodes regardless of the total number of nodes in the system.

To route between two points in the space a node in CAN forwards a message with the destination coordinates to its neighbour with coordinates closest to the destination. The number of the average hops with this routing mechanism is (d/4) (n1/d) where n is the total number of nodes in the network.

When a node leaves the system its zone and its database (key-value pairs) is taken by a neighbour, which is called takeover node. The two zones are merged into one if is possible or the takeover handles two zones. CAN needs to be tolerant to node failures so database can be replicated in a number of nodes to increase network reliability, to reconstruct the database by asking the participating nodes in the system requires much time and an enormous amount of bandwidth.

We have implemented CAN and with the functionalities that it provides we are able to manage INCA. In more detail when a node of INCA enters the network takes a zone in CAN. Metadata are hashed and assigned to the responsible node. Data are free to move in any node according to the policy of the management function but have to inform the responsible node for their location. For their retrieval the responsible node is asked through routing in CAN and it returns the address of their presence.

As for the storage protocol layer due to the high capabilities of the network TCP has been characterized as inefficient so it was necessary the implementation of a new protocol that will support the architecture above. So we have implemented a new protocol that will be responsible for point-to-point chunk transfers that requested from the layer above. This is parted from two layers:

1. HITP: High-volume INCA Transport Protocol that aims to transport fast huge volumes (Terabytes) of data in an efficient way and dealing with lambda networks.

2. HICP: High-volume INCA Control Protocol, it aims to control HITP and making the glue between middleware and network levels.

6.2 Final developments of the INCA

During these months as we had already clarified the architectural decisions of INCA we focused on algorithms and parameters to optimize the performance of INCA. The technical objectives for the development of this period were:

1. The definition of a routing protocol in CAN as the routing latency through it determines the speed in which we can put and retrieve files though a storage area network.

2. The creation of a mechanism where we can use and balance the distribution of objects through the storage area network according to the bandwidth and storage capabilities of the participating nodes.

3. The development of an algorithm where we fragment objects in a specific block size and we can transfer them efficiently with high bit-rate by exploiting the resources of the network efficiently.

6.3 Results of the final period

A. Routing protocol in CAN. We have experimented during this period with multiple routing protocols. The first was routing according to the proximity of the point in CAN where we want to route agnostic to the underlying network conditions. The second was a routing where each time the message is forwarded to the closest node in the underlying network that is towards the point that we want to route. Finally we have concluded to a routing mechanism where we route each message according to a metric that is correlated with the proximity to the underlying network measured by the RTT (round trip time between nodes) and the distance in the virtual space between the neighbour and the final destination.

B. According to the findings there are two techniques that can work orthogonal towards the balancing mechanism of the storage area network. The first is the creation of an overlay where each node has a portion of the space according to its bottleneck resource and the second is the creation of object Ids dynamically not according to a uniform hash function but according to the remain resources of the participating nodes.

C. At last the fragmentation of objects into small blocks and the development of a flow control protocol is a technique very efficient towards the balancing of the storage resources of the system and the exploitation of underlying network resources.

7 Adaptation, deployment and tests of the PHOSPHORUS DDSS application

7.1 Introduction

In the previous stage of project many tests has been done using DDSS applications to check efficiency of the PHOSPHORUS network. DDSS applications used in that period were modified versions of existing scientific and commercial applications, e.g. GridFTP and Tivoli Storage Manager Backup/Archive application. During period M25-M33 of project many tests with DDSS have been performed, which led to a number of demonstrations to give evidence of the accomplishments achieved.

7.2 Application

For demonstration needs, a special application, which makes backups on remote server has been developed at PSNC. The application integrates *GridFTP* [GridFTP] with *G2MPLS* facility and at once it visualizes all steps of the backup process. This tool has been implemented in *Python* language using *PyGTK* library [PyGTK]. The application has been shown as one of PHOSPHORUS project demos following conferences: SC'08 [www1], ICT 2008 [www2] and TNC'09 [www3].

7.2.1 General description

The main application can be invoked by a special *Bourne shell* script (sh) prepared for this purpose. The script in turn launches a *Python* script. As result we can see a window with a

list of steps, which have already been performed or are actually processed. Below the list is a progress bar, which shows percentage of already finished steps (as shown on Figure 7).

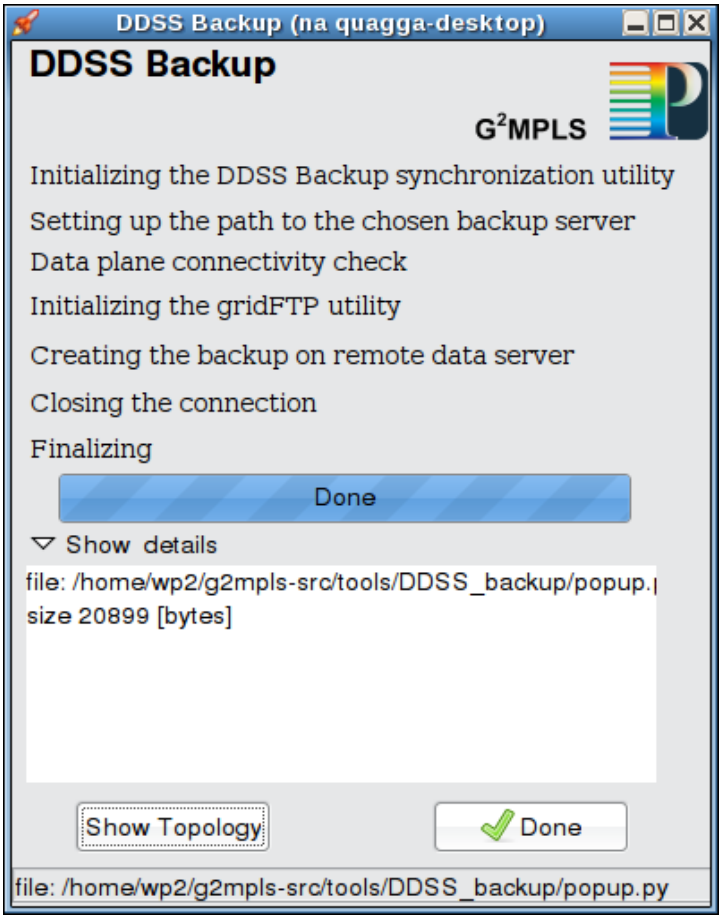

Figure 7: The main window of backup application.

We can also see the details of backups having been run, by using the *Show details* sub-menu. The detailed view shows for which files backup copies have been made and what was the individual size (in bytes).

The application also allows us to trace the topology of the network, which is used to perform backups. This functionality is invoked when pressing *Show Topology* button.

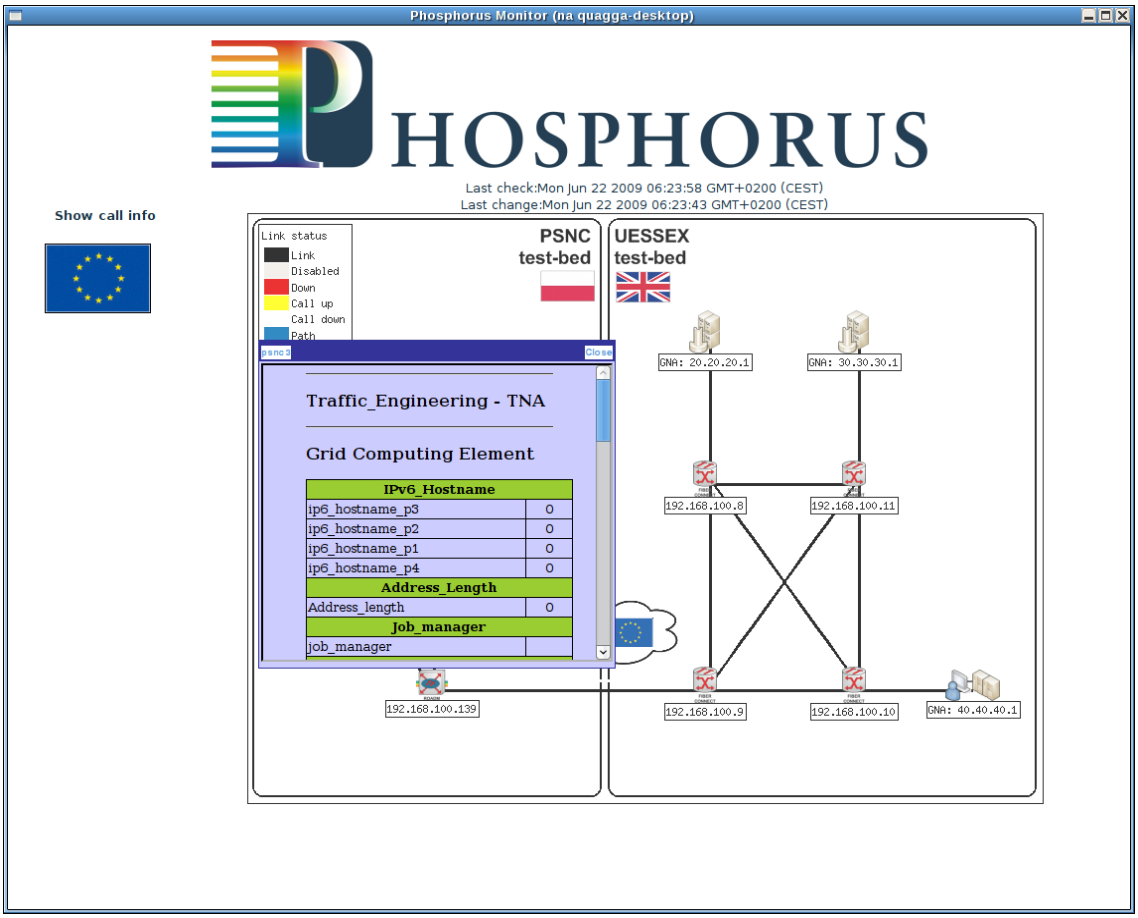

Figure 8: Topology of the PHOSPHORUS network.

The topology view shown in Figure 8 represents two sites: PSNC and ESSEX. This tool is kind of a monitoring facility, which shows the status of devices located at each center. Details of each host in the test-bed and links between the hosts are available. After clicking on a host, detailed information about this host is displayed. Moreover, the topology view is presenting the path reserved for each transfer. This path is marked as a yellow line between a set of hosts in the network.

The application also has been integrated with *Konqueror* – a universal viewing application. In the context menu of *Konqueror* a user can choose the option to backup selected files with the DDSS Backup application. This functionality enables flexible construction of sets of files to backup.

The operation of the application can be described in the following way: the application requests the resources (destination data server and a path), then if any suitable resources can be reached, the tool creates a reservation and provides the path to it. When the transfer is finished, Control Plane vacates the resources.

7.2.2 GridFTP and G^{2}MPLS integration

Generally, the intention of the application implementation was to integrate *GridFTP* and *G^{2}MPLS*. However, for the implementation it was necessary to introduce a common layer, which will be bridging between network and application layer. To provide such functionality two Python modules have been implemented:

- *g2dwrapper*: responsible for calling *G2MPLS* for establishing a path between source and destination hosts,
- *g2dialer*: responsible for tearing down the path.

Both modules are invoked by the main application script. *G2dwrapper* is performed before the initialization of *GridFTP* transfer and *g2dialer*, when the transfer is complete.

7.3 Experiments

This application was developed to visualize the process of link reservation and backup calls. That is why using simple scenarios have been sufficient to verify the functionality. Each scenario comprises the following steps:

- *Initializing the DDSS backup synchronization utility:* in this step data to backup are prepared, all transfer parameters are set and the target backup server is chosen.
- *Setting up a path to the chosen backup server:* send a signal over the possible path to the destination server requesting if it is possible to establish connection along the entire path length. If the response from all components along the path is yes, the g2dialer is launched.
- *Data plane connectivity check:* when the path is reserved, it is checked whether it is possible to send data over the path reserved, i.e. if data plane is ready to transmit data.

- *Initialization of the GridFTP utility*: create a Grid proxy and establish the GridFTP connection using the reserved path.
- *Making a backup on remote storage node:* run the GridFTP file transfer.
- *Closing the connection:* tear down the reserved path.
- *Finalizing:* Wait for the Control Plane properly closing the connection and send the appropriate message.

7.4 Summary

In the last stage of project a special DDSS Backup application has been developed. It integrated one of the applications used in the previous stages of the projects DDSS *GridFTP* with the *G2MPLS* functionality and visualizes the progress of the backup process. The application was successfully shown on conferences and project reviews. Despite of the fact that the application runs only simple scenarios (files transfer – backup), the objectives of the DDSS development have been achieved. This solution enables cooperation of resource reservation tool and backup application. Integration with *Konqueror* allows to perform backups in an easy way and to create own backup scenarios flexibly. Because of the fact that it is implemented in *Python* and *Shell scripts* the solution can easily be ported to new machines.

8 Conclusions

This report highlights the results and achievements of middleware development and consolidation during the last 9 months of the PHOSPHORUS project. The resulting framework of both components on the middleware layer and components on the network layer has been demonstrated during the final PHOSPHORUS workshop at the TERENA Networking Conference 2009 in Malaga, Spain. The PHOSPHORUS application suite, which was used in this demonstration, has been described in the last application-related report D3.6 - Report on the Results of the Application Experiments During the Final Testbed Experiments. The technological basis for the developments during the last 9 months have been described in D3.7 - Report on the results of the middleware experiments during the final testbed experiments.

Individual results achieved during the last 9 months:

The semantic Resource Selection developed and implemented in the previous phase has been integrated in the UNICORE 6 rich client, allowing the user to specify just the name of the application to be executed while the entire source detection and selection based on appropriateness for the specific application is done automatically by the middleware.

With the finalised interface between G^{2}MPLS and the middleware the PHOSPHORUS initial goal has been reached to provide an environment where the resource selection may already start at the network layer based on the information on topology, connectivity and link status available there. The relevant information is then passed from the network layer to the middleware layer and presented to the user for the final decision or to the MetaScheduling Service acting on behalf of the user. The framework allows both operations in the traditional overlay model but also in the advanced integrated model.

The interface between the middleware developed in WP3 and WP1 Service Plane developments has been updated to support the HARMONY system.

Both the Service Plane interface and the Control Plane interface have been extended to allow the transmission of user credential information from the middleware layer to the network layer. With these extensions the PHOSPHORUS testbed provides a seamless chain of trust from the user authentication level to the token-based security mechanisms as implemented by WP1, WP2, and WP4.

Finally, for the Intelligent Network Caching Architecture an evaluation of algorithms and parameters was done to optimize the performance of INCA.

Overall, the last 9 months of the WP3 activities have been quite successful regarding the finalization of the developments of the previous PHOSPHORUS phases. With the achievements of the last 9 months the project provides a full integration of the middleware, the Service Plane and the Control Plane with respect to the holistic management of the computational and network resources. With the last efforts targeting the Authentication and Authorization Infrastructure we achieved a seamless chain of trust from the user to the token based network security, which has not been available in any other project or production environment before.

9 References

[G2MPLS]	"Deployment and Interoperability of the PHOSPHORUS Grid Enabled GMPLS (G2MPLS)" , DOI 10.1109/CCGRID.2008.120
[D2.5]	D2.5 - Preliminary Grid-GMPLS Control Plane prototype, http://www.phosphorus.pl/wiki/images/0/00/PHOSPHORUS-WP2-D2.5v1.0.doc
[D3.6]	D3.6 Report on the Results of the Application Experiments During the Final Testbed Experiments, http://www.phosphorus.pl/wiki/images/3/3e/PHOSPHORUS-D3.6.pdf
[D3.7]	D3.7 Report on the results of the middleware experiments during the final testbed experiment http://www.phosphorus.pl/wiki/images/3/3e/PHOSPHORUS-D3.7.pdf
[GLUE]	GLUE Working Group (GLUE), http://forge.gridforum.org/sf/projects/glue-wg
[OGSA-BES]	OGSA-BES Working Group, http://forge.gridforum.org/sf/go/projects.ogsa-bes-wg/
[D3.8]	D3.8 - Final Report on the achievements and results of WP3
[Y3act]	3rd Annual Activity Report
[GridFTP]	http://www.globus.org/grid_software/data/gridftp.php
[PyGTK]	http://www.pygtk.org
[www1]	Super Computing 2008: http://sc08.supercomputing.org
[www2]	ICT 2008 event: http://ec.europa.eu/information_society/events/ict/2008/index_en.htm
[www3]	TERENA Networking Conference 2009: http://tnc2009.terena.org

10 Acronyms

AAA	Authentication, Authorisation, Accounting
CAN	Content Addressable Network
DDSS	Distributed Data Storage Systems
e2e	end to end
EGEE	Enabling Grids for E-sciencE (European Grid Project)
FC	Fibre Channel
FC-SATA	Fibre Channel to SATA technology (mixed technology used in disk matrices: disk matrix have Fibre Channel ports for hosts connectivity, but contains SATA disk drives)
GEANT2	Pan-European Gigabit Research Network
GEANT+	the point-to-point service in GEANT2
GMPLS	Generalized MPLS (MultiProtocol Label Switching)
G^{2}MPLS	Grid-GMPLS (enhancements to GMPLS for Grid support)
G.O-UNI	Grid Optical User Network Interface (integrating optical networks with grid services)
GT4	Globus Toolkit Version 4 (Web-Service based)
INCA	Intelligent Network Caching Architecture
KoDaVis	Tool for Distributed Collaborative Visualisation
MSS	MetaScheduling Service (a Grid Level Scheduler developed at the Fraunhofer Institute SCAI)
NREN	National Research and Education Network
NRMS	Network Resource Management System
NRPS	Network Resource Provisioning System
PoP	Point of Presence
Protégé	Ontology Editor and Knowledge Acquisition System
QoS	Quality of Service
SAML	Security Assertion Markup Language
SNMP	Simple Network Management Protocol
TOPS	Technology for Optical Pixel-Streaming
TPD	Tiled Panel Display
TUAM	Tool for Universal Annotation and Mediation
UNI	User to Network Interface
UNICORE	European Grid Middleware (UNIiform Access to COmpute REsources)
VLAN	Virtual LAN (as specified in IEEE 802.1p)

VIOLA	A German project funded by the German Federal Ministry of Education and Research (Vertically Integrated Optical Testbed for Large Applications in DFN)
VPN	Virtual Private Network
WISDOM	Wide In Silicio Dockong On Malaria