Report No.2 on Globus Toolkit 4 evaluation by joint JINR (Dubna), KIAM and SINP MSU team

***Phase 2*:** Evaluation 1 (1/06 – 20/07)

GT4 General characteristic

Applicability. GT4 may be used:
- as independent middleware for a grid;
- being based on widely accepted protocols of Web services (WS) stack, GT4 is highly interoperable and could be used along with other middleware packages or separate services with the same basis.

Architecture. GT4 implements WS architecture, enriching it with additional techniques to access the data that should persist between different invocations of services, in a consistent and interoperable manner. It is hopeful, that novel WS-Resource approach will facilitate not only interoperability, but will also simplify the development of stateful applications that function in a multi-user and multi-session mode.

Structure.
- WS Core – program environment that supports services functioning;
- a number of ready-to-use services;
- development tools:
  - tools for client programs and services creation (API, WSDL generation, deployment);
  - API for accessing the services of GT4;
  - user-level utilities for grid operation.

Functionality. GT4 services cover four areas: Security, Data Management, Execution Management, Information Services. Table 1 shows the list of GT4 services in comparison with those of gLite.

Some examples of GT4 applications:
- DSTC and Monash University have announced the release of Nimrod/G v3.0, that provides support for Globus version 4.0, using the native GT 4.0 web services.
- In Condor-G the problem of full support for GT4 GRAM is decided (GGF-14).
- Gridcast (Belfast e-Science Center) migrates from GT3 to GT4 [1].

GT4 and gLite

In each main area gLite has services, which are absent in GT4, but are substantial:
- Security. Virtual organization support (VOMS gLite);
- Execution Management.
  - Resource virtualization (Workload Management System).
  - Support of computational resource internal organization: CE, reliability (Condor-C), logging and bookkeeping (L&B), job monitoring.
- Data Management. Support of storage resource internal organization: SE, SRM; file catalogue (FireMan).

Table 1.

Globus Toolkit 4	gLite
User Interface	1. User Interface
Security - CAS - Delegation Service - WS A&A: o Authorization Framework o Message/Transport-level Security - Credential Management: o SimpleCA o MyProxy o GSI-OpenSSH	2. Security a. VOMS (DataTAG, EDG) b. GSI (Globus) c. Authentication and authorization for C and Java based (web) services (EDG)
Data Management - RFT - GridFTP - RLS - OGSA-DAI - Data Replication Service (tech preview)	3. Storage Element a. File Transfer/Placement (EGEE) b. glite-I/O (AliEn) c. GridFTP (Globus) d. SRM: Castor (CERN), dCache (FNAL, DESY), other SRMs 4. Catalog a. File and Replica Catalog (EGEE) b. Metadata Catalog (EGEE)
Execution Management - WS GRAM - Pre-WS GRAM - Globus Teleoperations Control Protocol (GTCP) - Workspace Management Service (WMS)	5. Computing Element a. Gatekeeper, WSS (Globus) b. Condor-C (Condor) c. CE Monitor (EGEE) d. Local batch system (PBS, LSF, Condor) 6. Workload Management a. WMS (EDG) b. Logging and bookkeeping (EDG) c. Condor-C (Condor)
Information Services WS MDS (MDS4) - Aggregator Framework - Index Service - Trigger Service - WebMDS	7. Information and Monitoring a. R-GMA (EDG)

Common Runtime Components - Java WS Core - C WS Core - Python WS Core - XIO - C Common Libraries	

On the other hand gLite incorporates some components from GT, though from previous version (GT2).

- Grid Security Infrastructure
- GridFTP
- Gatekeeper.

Risks assessment.

GT4 is issued relatively recently and some of it's components are still under development.

GT4 includes 3 WS Core implementation (C, Java, Python), but presently all GT4 services are developed for the Java WS Core only, which has worse performance characteristics than C container. The usage of Java as an instrumental language may complicate the development and reengineering of services.

GT4 does not implement the whole stack of WS specifications (Transfer, Messaging, Quality of Service, Business Logic), and apparently WS Core will evolve. Besides, WS stack is not matured enough to be wholly stable, for the alternatives see [2]. A risk assessment of several contemporary WS protocols is presented in paper [3].

There exist alternative to WSRF approaches to managing state, WS-GAF [4], for example.

GT4 WS Core

Service Hosting Environment (or container) is the domain-independent logic used to receive a SOAP message and identify and invoke the appropriate code to handle the message, and potentially also to provide related administration functions.

GT4 container:
- implements SOAP over HTTP as a message transport protocol, and both transport-level and WS-Security message-level security for all communications;
- implements WS-Addressing, WSRF, and WS-Notification functionality;
- supports logging via Log4j, which implements the Jakarta Commons Logging API;
- defines WSRF WS-Resources with properties providing access to information about services deployed in the container and container properties such as version and start time.

GT4 has three Web services hosting environments (containers) with different performance properties, supported Web services implementation languages, security support, and so forth: Java WS Core, C WS Core, Python WS Core. It is important, that all GT4 ready-to-use services are implemented only for Java WS Core.

WS Core architecture divides common logic of request processing from the implementation code of services. Common part consists presently of a suite of handlers (security, dispatch) and some other mechanisms (Lifetime Management, Notification Producer/Consumer), and is extendible. This architecture possibly could be used for the implementation of various grid-specific functions, such as logging, accounting, job monitoring in a way that will not affect a service code.

WS specifications in GT4. GT4 are underlied by:
- Web services architecture (XML, SOAP, WSDL);
- WS-Security and other specifications relating to security;
- WS-Addressing, WSRF, and WS-Notification specifications used to define, name, and interact with stateful resources.

Several other specifications are planned to be used in a future GT implementations:
- Web services distributed management (WSDM) for managing GT components;
- WS-CIM – a means of representing physical and virtual resources.

The situation with Web services specifications is complicated due to competing proposed specifications. In particular, Microsoft and others [2] recently proposed WS-Transfer, WS-Eventing, and WS-Management, which define similar functionality to WSRF, WS-Notification, and WSDM, respectively, but using different syntax.

GT4 Java container's administrating and management

Management of Java WS container includes two actions: start container (globus-start-container utility) and stop container (globus-stop-container utility).

globus-start-container

There is a possibility to configure container profile. This means that the same installation of Java WS Core can be used to run different containers each with different configuration. We in KIAM tested this feature and it works fine.
Stored information about resources may be used for recovery purposes. The resources will be available again after container's restart.

Service creation

There are three methods to create Web service:

- creating service with java2wsdl and wsdl2java tools, having java interface of service
- GT4's approach for services creating (The Globus Toolkit 4 Programmer's Tutorial by Borja Sotomayor)
- Matthew Smith's and Bernd Freisleben's approach for GT4 service creation [5]

First method is universal and most popular, but it doesn't support WS-Resource. That's why programmer has to write WSDL file, in which WS-Resource is described. But this approach is preferable for designing services which do not use WSRF (common Web services).
Second and third methods support WS-Resources. Borja Sotomayor's method requires knowing SOAP, WSDL, WSDD, etc. and more complex for understanding and production. Nevertheless, it is general purpose approach for creating all types of services (WSRF and non-WSRF).
Matthew Smith and Bernd Freisleben have created special library and other additional files which help to make service creation easier. Unlike developing a GT4 Service the traditional way, you will not have to mess around with WSDL or JNDI configurations to create your service.

Conclusion
We propose that service creation have to be originated from service interface, but not from WSDL description. WSDL is a machine-oriented language and very difficult for human. Therefore, the third method is more suitable for this purpose. But for the moment this approach supports only services and clients written on Java.

WSRF and GT4 WS Core

General information
The WS Resource Framework (WSRF) specifications define a generic and open framework for modeling and accessing stateful resources using Web services. This framework comprises mechanisms to describe views on the state (WS-ResourceProperties), to support management of the state through properties associated with the Web service (WS-ResourceLifetime), to describe how these mechanisms are extensible to groups of Web services (WS-ServiceGroup), and to deal with faults (WS-BaseFaults).

The WS-Notification family of specifications defines a pattern-based approach to allowing Web services to disseminate information to one another. This framework comprises mechanisms for basic notification (WS-Notification), topic-based notification (WS-Topics), and brokered notification (WS-BrokeredNotification).

Experiences with WSRF application:

1. Evidently the most valuable experience of WSRF/WS-N application presents GT4 itself. As we know, WS-Gram, WS-MDS, Workspace Management Service actively make use of the stateful resources for modeling jobs and information about sites.
2. Several systems have been developed with the aid of WSRF::Lite

- WEDS: a WSRF-based Environment for Distributed Simulation [6]
- Grid data management middleware for managing dynamic Grid file system sessions [7].

WSRF implementations comparison

This part is based on the paper [8]. By now five different WS-Resource Framework and WS-Notification (WSRF/WSN) implementations exist and their comparison is of interest. There are five WSRF implementations:

- GT4-Java, the Java WS Core of GT4;
- GT4-C, the C WS Core of GT4;
- pyGridWare, a Python WSRF implementation [9], which is distributed with GT4 (Python WS Core);
- the Perl-based WSRF::Lite [10];
- WSRF.NET, an implementation of WSRF/WSN on the .NET Framework [11].

Functional comparison:

- Significant commonalities are seen with regard to dispatching and SOAP processing techniques, though in each implementation they are different and only GT4-C implements the original techniques.
- **Security.** GT4-Java, GT4-C, pyGridWare and WSRF.NET support three security protocols: 1) TLS/SSL transport-level security protocol; 2) Secure Conversation and 3) SecureMessage. WSRF::Lite supports only transport-level security.
- **Persistence.** WS-Resource's persistence can be achieved by holding the resource in memory, writing it to disk, storing it in a database or by custom method (for the last mode each system provides an interface). GT4-Java, GT4-C, pyGridWare and WSRF::Lite persist WS-Resources in memory by default and come with modules that allow resources to be saved to disk, providing the ability to survive server failure at the cost of some performance. WSRF.NET uses a database by default.
- **Lifetime Management** includes resource creation and resource destruction. Several systems handle creation similarly, providing a create() method, but GT4-Java does not define a specific create() operation. Different implementations remove expired resources via different mechanisms: periodic resource deletion, separate service, garbage collection, event handling.
- **WS-Notification.** WSRF.NET implements all three WSN specifications. GT4-Java and pyGridWare do not implement WS-Brokered Notification and have some other limitations. GT4-C does not implement producer-side notification. WSRF::Lite does not support any Notification specifications.
- **Authorization.** GT4-Java, pyGridWare, and WSRF.NET define an authorization callout that allows the service developer to provide custom authorization. GT4-C implements three built-in mechanisms. WSRF::Lite client passes security information to the WS-Resource through environment variables, leaving the WS-Resource implementation to implement its own authorization. GT4-Java provides a flexible infrastructure level framework for making authorization de-

cisions.GT4-C supports basic host, identity, and self authorization. The integration of an authorization callout interface is planned.

As can be seen, the implementations have significant differences in functionality, as well as in programming models, and performance.

Interoperability. It is regarded as possibility to use a client of one implementation against a service of another. Being consistent with the WS-Interoperability Basic Profile, the five implementations achieve a base level of interoperability with regard to XML, HTTP, SOAP, and WSDL.
But the implementations, nevertheless, are not fully interoperable. Namespace incompatibilities are the key concern, as well as several technical details (HTTP headers, application-specific portions of messages) which lies outside the WSRF/WSN specifications). In current situation interoperability issues are due to both the specifications themselves and idiosyncrasies of the projects' toolkits.

Future
All implementations plan further development.
The bulk of Java WS Core's functionality was submitted to the Apache Software Foundation's new Apollo and Hermes projects. Future Java WS Core development will occur within these projects.

GT4 WS Core performance evaluation

WS testing plan
Typical response time and client CPU time measurements.
Estimations of overhead expenses of GT4 Java container and common WSRF operations with dummy service (Java implementation) and client (Java/C implementation).
Consecutive dummy service method invocations.
Scenarios: 1) no security container (http); 2) security container (https, X.509 signing).
Scalability and robustness at concurrent and successive client runnings.
How quantity of created ws-resources affects on processing request time?

In all tests we used two identical machines - Pentium IV, 2.8 GHz, 512 Mb RAM, RedHat Linux 9 – one for GT4 client, other for GT4 server – both in local network (Ethernet 100 Mbps)

Preliminary results

Estimation of the overhead expenses of GT4 Java container

Aim. The test has been carried out to estimate the overhead expenses of GT4 Java container (i.e. first part which is common for invocation of all services) and client application.
Test's scheme. Dummy service/client Java implementations are used. The dummy service has a single method that returns an input value as output. The client consists of N consecutive invocations of the method.
Time measurements have been carried out by "time" command built into the client utilities to call dummy service.
Server response time (server wall time + network delay) was estimated as (client wall time) – (client CPU time), where client wall time = real time, client CPU time = user

time + system time (see tables 1,2). We have assumed that network delay = 0 because of network ping between our two machines is less than 1 ms.
We ran this test for N=1, 100, 10 000, 100 000.

Results

The results for no-security (http) GT4 Java container

Number of invocations	Results by "time" command	Average time of method invocation on client, ms	Average time of method invocation on server, ms
1	real: 0m2.573 user: 0m2.130 sys: 0m0.090	2573	353
100	real: 0m3.787 user: 0m2.990 sys: 0m0.100	37.87	6.97
10 000	real: 1m23.734 user: 0m50.460 sys: 0m3.400	8.37	2.98
100 000	**real: 12m40.708 user: 7m45.190 sys: 0m34.940**	**7.60**	**2.60**

The following is observed, that with increasing number of invocations N, the average time of method invocation (client and server) decreases. The point is that the very first (single) invocation of service, measured on the client side, gives time about 2 seconds and about 300 ms on server. Then the time is reduced down to 7 ms on client and to 2 ms on server (after a large number of invocations, > 1000). This effect is likely concerned with the first invocation penalty (JVM doing just-in-time compilations, or other "startup" expenses) but slow decline remains unexplained.

The results for security (https, X.509 signing) GT4 Java container

Number of invocations	Results by "time" command	Average time of method invocation on client, ms	Average time of method invocation on server, ms
1	real: 0m3.532 user: 0m2.850 sys: 0m0.080	3532	602
100	real: 0m17.683 user: 0m4.690 sys: 0m0.280	176.83	127.13
10 000	real: 14m21.553 user: 4m34.090 sys: 0m19.070	86.15	56.83
100 000	**real: 141m37.058 user: 45m34.780 sys: 2m44.690**	**84.97**	**55.97**

Estimation of average times for common WSRF operations

Aim. The purposes of this test are estimation of average times for common WSRF operations (in GT4 Java container) on server side and client side and investigation the influence of the number of created resources on the request time.

Test's scheme. We'll use "counter service" and create Java client implementation.

There are four scenarios of a client in which resource creation, setting resource property, getting resource property and resource destruction are examined accordingly. In each scenario client consists of N consecutive invocations of corresponding base WSRF operations such as CreateResource (take part in all scenarios), SetResourceProperty, GetResourceProperty and DestroyResource.
Time measurements carry out by "time" command built into the client utilities to call "counter service".
Server response time (server wall time + network delay) were estimated as (client wall time – client CPU time), where "client wall time" is real time and "client CPU time" is user time + system time. We have assumed that network delay = 0 because of network ping between our two machines is less than 1 ms.

Estimation of GT4 Java container scalability and robustness
Aim. The test has been carried out to estimate GT4 Java container scalability (i.e. to determine the maximum possible number of parallel method invocations before server failure).
Test's scheme. Dummy service/client Java implementations are used. The dummy service has a single method returns an input value as output. The client consists of M threads running concurrently with N consecutive invocations of the method each. Server node CPU consumption has been estimated by "vmstat" and "top" commands built into the client utilities.
We ran this test for (M=100, N=10), (M=200, N=10), (M=100, N=100).

Results for no-security (http) GT4 Java container

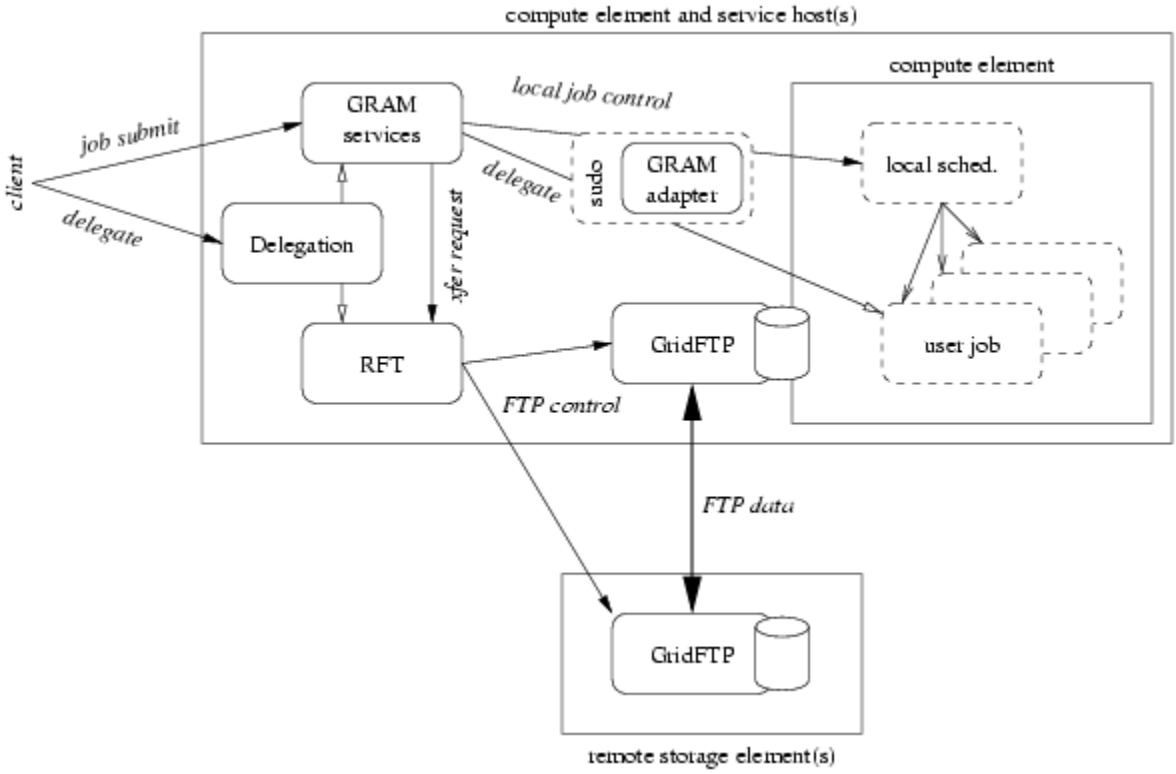
Number of threads (M)	Number of invocations (N)	Average server node consumption, %
100	10	47
200	10	53
100	100	70

Because of server wall time is much less than the client CPU time (see test 1), it was not possible to create sufficient load on the server from single client host. Already at M > 300 the error from the client side have occurred (java.lang.OutOfMemoryError), i.e. the client is loaded on 100 %. So, we plan to use client C implementation for carrying out this test.

GT4 WS-GRAM

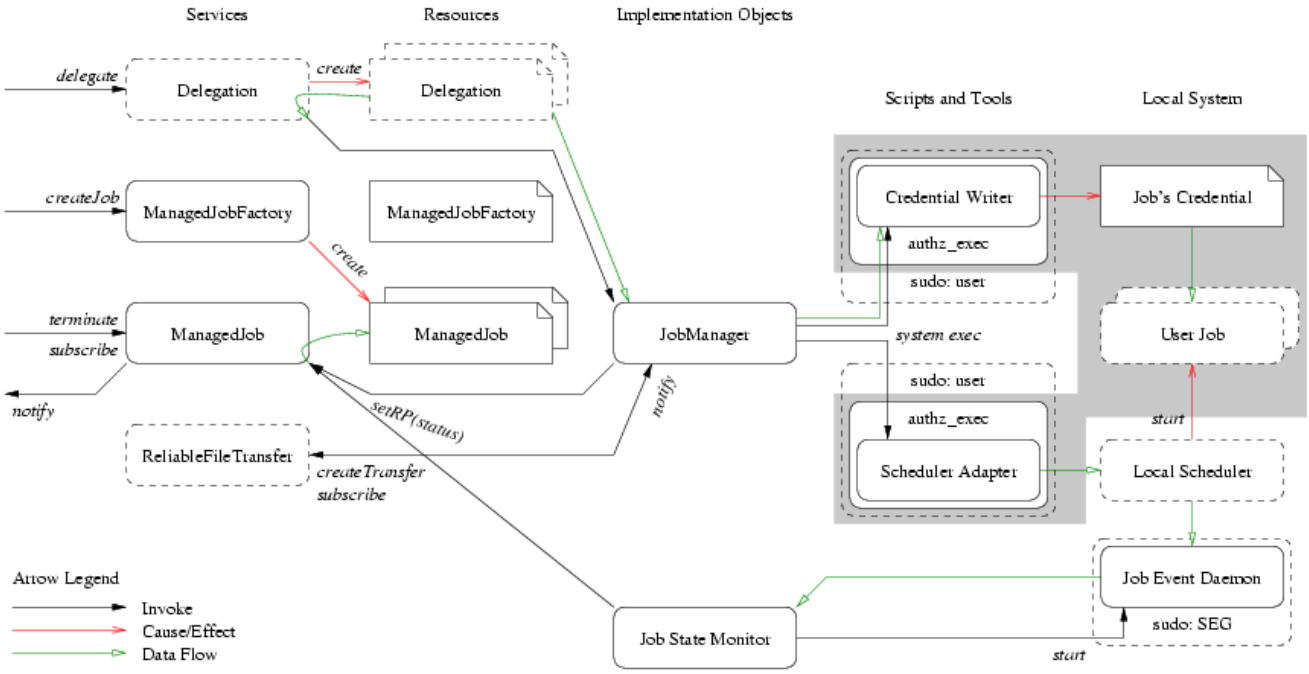
Grid Resource Allocation and Management Service (GRAM) is intended for secure submission, monitoring and control of job on remote computing resources with coordinated file staging. These resources may be controlled by local resource manager such as PBS, LSF etc.

Architecture and implementation of WS-GRAM



WS-GRAM is implemented as a suite of web-services under GT4 Java WS-container, providing Web services interfaces consistent with WSRF model.

In GT4 developers refused high secure but non scalable and low managed personal Java container (user host environment - UHE) for each user as in GT3. Instead of service instance in UHE each submitted job is presented as ws-resource. It is more scalable solution although less secure. As result GT4 performance is significantly more that GT3 performance.



Structure. GRAM services suite consists of two web-services:

- ManagedJobFactory exposed distinct ws-resource for each local resource manager. ManagedJobFactory provides an interface to create ManagedJob resource in order to perform a job in local scheduler. This ws-resource has properties containing site and local resource manager descriptions and GLUE data in GLUE schema format. The properties are published in default Index Service (WS-MDS).
- ManagedJob exposed distinct ws-resource for each submitted job. Managed-Job provides an interface to monitor the status of the job or to terminate the resource (to cancel the job). Client can subscribe (in accordance with WS-Notification) for notification of status this resource.

Used services. WS-GRAM suite uses other GT4 components – Delegation Service and Reliable File Transfer (RFT) Service.

Delegation Service replaces the implicit delegation of previous GRAM solutions with explicit service operations, by which client delegates credentials for use by RFT service or transferring to submitted job. Delegated credentials are presented as ws-resource. End point reference on this resource is transferred to ManagedJobFactory. Single Delegation credentials may be shared between multiple short-lived jobs. Delegation service retains ability to refresh credentials for long-lived jobs.

Reliable File Transfer Service is invoked by the WS-GRAM services to stage files before and after job computations. Files are transferred to (from) user workspace in GRAM site and then may be accessed for submitted job by means of NFS or local scheduler. Integration with RFT provides a much more robust file staging manager than the ad hoc solution from previous GRAM versions. To transfer staging files RFT web-service acts as a so-called third-party client to the gridFTP servers. The integration with GridFTP replaces the legacy GASS data transfer protocol.

Adapter Perl scripts. A control of each local scheduler is provided in the form of adapter Perl scripts, following GRAM adapter plugin API. These adapters implement scheduler-specific submission, job exit detection, job cancellation and job exit status

determination processes. This adapter may take into account "extension" job description elements and interpret this information in accordance with concrete scheduler.

Scheduler Event Generator (SEG) is a program which uses scheduler-specific monitoring modules to generate job state change events. These events are propagated to the Job State monitor and then transferred to ws-resource, presented running job.

Workspace Management Service may be used with WS-GRAM to submit job in Unix user account managed by account pooling (LCMAPS).

Authentication and authorization

WS-GRAM utilize secure Web-service invocation as provided by WSRF core of GT4 for all job-management and file-management messages.

Authentication. Secure communication includes mutual authentication. To authenticate user must have valid X509 proxy credentials. Usually WS-GRAM shows host-credentials.

Authorization. User jobs are executed within Unix user accounts. User mapping is extracted from grid-mapfile, which is assumed to contain a set of one-line entries, each specifying the DN of authorized user and the local Unix user. GRAM can also access other sources for authorization and mapping information by configuring service security descriptor to define custom authorization mechanisms (Policy Decision Points class implemented interface org.globus.wsrf.security.PDP).

Sudo. To run scheduler adapter and write delegated credentials to user account context WS-GRAM use sudo – defacto standard Unix utility. This mechanism replaces the root-privilege gatekeeper from pre-WS GRAM in order to avoid running an entire Java container as root. Sudo configuration file sudoers must correspond to grid-mapfile.

Workspace Management Service (WMS) allows a Grid client to dynamically create and manage a workspace (Unix account) on a remote site. This service is currently in a tech preview release (pre-alpha). Workspace creation and management can be implemented in different ways according to site policies and preferences. Currently, WMS implementation supports account pooling implementations, based on the LCMAPS implementation. To integrate WMS with WS-GRAM site administrator may define user mapping to all accounts from the pool in grid-mapfile. Then a client (potentially a broker or a CE) would create Unix account (and then manage it) through the WMS interface and request execution through WS-GRAM in this account (by means of "localUserId" element of job description file).

User interface and client API

globusrun-ws (written on C) is an user utility to submit job into WS-GRAM and manage submitted job. Client may submit job in an interactive or batch mode. In interactive mode the utility returns only when job terminated; status changes are communicated back to user. In a batch mode the utility returns immediately. User may request EPR to ManagedJob resource created for this job. EPR is used to get current job status, monitor (attach to a running job) and terminate the job.
The utility returns a unique submission identifier that may be used for resubmission if a

job submission request times out. GRAM then ensures that the job is not submitted more then once.

Job description file. It is an XML-file containing job description elements. These elements defines job features such as executable file, work directory, command line arguments, environment variables, memory, time, files for stdin, stdout, stderr and many other. XML job description inherits RSL job description from previous versions GT.
Executable, stdin, stdout, stderr are defined as local files in user workspace on GRAM site. To stage these files to (from) the workspace user must define "fileStageIn" ("fileStageOut") elements.
"fileStageIn" element define (remote URL, local file in user workspace) pair that should be staged to the GRAM site (user workspace) before job execution.
"fileStageOut" element define (local file in user workspace, remote URL) pair that should be staged from the GRAM site (user workspace) after job execution.
"localUserId" element define Unix account that the client would like the job to run as. This requires that there is a mapping in the grid-mapfile between the callers proxy subject and this account.
"extensions" element used to define client-specific data that may be interpret by local scheduler adapter.

Client API. To write Java client application submitting and managing job into WS-GRAM user must use Java WS Core API, Delegation API and WS-GRAM API. For C there is not Delegation API, but there are C WS Core API and C WS-GRAM API only. It is essential disadvantage if user wants to create C client application because he must write C implementation of the delegation by means of C WS Core API.

Metrics: performance, reliability, robustness

WS-GRAM testing plan

Typical response time and client CPU time measurements.
Measurements of max throughput (jobs/min) with globusrun-ws (simple job, job with delegation and file staging).
Concurrent (all jobs together and with uniform load) runnings of globusrun-ws.
Throughput in dependence of concurrent jobs number.
Variants: 1) fork; 2) PBS with disabled running.
Robustness at concurrent and successive runnings.
How quantity of created ws-resources affects on throughput?

In all tests we use two identical machines - Pentium IV, 2.8 GHz, 512 Mb RAM, RedHat Linux 9 – one for GT4 client, other for GT4 server – both in local net (Ethernet 100 Mbit)

Preliminary results

Successive runnings of globusrun-ws with simple job - /bin/date
Times were measured by **time** command that gives (wall time, user time, system time)
Response time = wall time – (user time + system time)
Client CPU time = user time + system time
2 series of 400 runnings – typical response time – 1,75 ± 0,1sec
– typical client CPU time – 0,25 ± 0,02sec
Sometime there were response time's fluctuations up to10 sec.

Client did not return any mistakes, but one time there was error in server container log:

```
2005-07-04 15:28:55,501 ERROR container.GSIServiceThread [Thread-
249,process:120] Error processing request
……
```

Concurrent (all jobs together) runnings of globusrun-ws with simple job - /bin/date

Given number of jobs concurrently started in background (with &). Then **wait** command was executed.
In every series wall time, user time, system time was measured (by **time** command on client machine).
Client CPU time was the same as in successive runnings.
Throughput = (jobs number)/ total wall time

Jobs number	50	75	100	125	150	200	250	300	400
Throughput jobs/min	63	65	68	66	65	64	65	Failed	Failed

When the number of jobs becomes sizeable (more than 50), we in first get errors in container log file:

(1)
```
2005-07-04 15:28:55,501 ERROR container.GSIServiceThread [Thread-
249,process:120] Error processing request
…..
```
or

(2)
```
2005-07-04 15:33:58,899 ERROR handler.AddressingHandler
[Thread-556,invoke:120] Exception in AddressingHandler
AxisFault
…..
```
or

(3)
```
2005-07-04 15:52:22,607 ERROR
utils.JobStateMonitorSubscriptionManager
[Thread-23,subscribe:178] unable to monitor job for state changes
org.globus.exec.monitoring.AlreadyRegisteredException
…..
```

Starting from the number of processes 300 the errors appear in client reply:

```
Submitting job...Failed.
globusrun-ws: Error submitting job
globus_soap_message_module: Failed sending request
ManagedJobFactoryPortType_createManagedJob.
globus_xio: Operation was canceled
globus_xio: Operation timed out
```

or

```
Destroying job...Failed.
globusrun-ws: Unable to destroy job: Error destroying job
globus_soap_message_module: Failed sending request
ManagedJobPortType_Destroy.
globus_xio: Operation was canceled
globus_xio: Operation timed out
```

It is mean that server has not time to process all requests.

After the mistake (3) (with number of processes = 300) client shell script hang-up at **wait** command

We requested the Globus team about these mistakes and received (from Peter Lane) following answer:

"We have had some success in preventing some of these problems, but those changes are only in our CVS trunk and won't be available until 4.2."

Concurrent (with uniform load) runnings of globusrun-ws with simple job - /bin/date

Uniform load is provided by concurrent starters. Each starter runs globusrun-ws successively in cycle. Test script runs given number of starters in background. Then test script waits 1 min and counts number of finished jobs during given time interval. Throughput = (number of finished jobs) / (time interval)

Time interval (min)	5	5	5	5	5	5
Starters number	20	50	100	200	250	300
Throughput jobs/min	65	71	79	70	52	Failed

Errors' picture is nearly equal to one in above ("all together") test with the exception of mistake (3).

Data management

Data management testing:
GridFTP

- Check of the FridFTP installation test
- Study and check of work of GridFTP server
- Preparation of GridFTP configuration files for two modes:
  - Standard mode of starting of the GridFTP server (root, port 2811)
  - Start on behalf of the dedicated user
- Study of transfer of files in a command mode in a local network by two machines
- Preparation of the tests for hard mode work on an exchange of files in a local network (duration of stable work, big volumes of the transmitted data),
- Preparation of the measuring tests
- Approbation of GridFTP with external GT4 installation

RFT

- Installation of the PostgreSQL
- Check of the RFT installation test
- Preparation of the tests for study of exchanges with RFT in a local network
- Check of work of RFT services under the protocols HTTP and HTTPS,
- Check of work of RFT with SOAP servers such as **Container** and **Tomcat**
- Check RFT on stability of long work and with the large volumes of the data
- Preparation of the measuring tests
- Approbation of RFT with external GT4 installation .

Standard mode for GT4 running on two machines at JINR:

- **GridFTP** server (root, port 2811)
- **Tomcat,** serving 2 protocols: HTTP (port 8080) and HTTPS (port 8443)
- **PostgresSQL**, database RFTdatabase for RFT

Hardware:
2 machines – P4 3Ghz, 1.5 GB RAM
OS: FC3 (Fedora Core 3), **SLC** (Scientific Linux Cern Release 3.0.4, 3.05)
JINR local network: 100Mb/s

Main evaluation results:

Special software:

- Installed and configured **PostgreSQL** and database RFTdatabase for RFT component
- Configured Tomcat for simultaneously serving of the two protocols: HTTP (port 8080) and HTTPS (port 8443) in files $TOMCAT_HOME/webapps/wsrf/WEB-INF/web.xml and $TOMCAT_HOME/conf/server.xml

Note: Tomcat uses some GT4 modules and for correct running **must be** started in $GLOBUS_LOCATION directory

GridFTP
Installation GridFTP test:

[omii03] //opt/globus/gt4.0/test/globus_ftp_client_test > ./TESTS.pl

```
Started server at port 60135
Running sanity check
Server appears sane, running tests
globus-ftp-client-caching-get-test.....ok
globus-ftp-client-caching-transfer-test.....ok
globus-ftp-client-create-destroy-test.....ok
globus-ftp-client-exist-test.....ok
globus-ftp-client-extended-get-test.....ok
globus-ftp-client-extended-put-test.....ok
        1/100 unexpectedly succeeded
globus-ftp-client-extended-transfer-test.....ok
globus-ftp-client-get-test.....ok
globus-ftp-client-lingering-get-test.....ok
globus-ftp-client-multiple-block-get-test....ok
globus-ftp-client-partial-get-test.....ok
globus-ftp-client-partial-put-test.....ok
globus-ftp-client-partial-transfer-test.....ok
globus-ftp-client-plugin-test.....ok
globus-ftp-client-put-test.....ok
        1/124 unexpectedly succeeded
globus-ftp-client-size-test.....ok
globus-ftp-client-transfer-test.....ok
globus-ftp-client-user-auth-test.....ok
All tests successful (2 subtests UNEXPECTEDLY SUCCEEDED).
Files=18, Tests=2567, 680 wallclock secs (183.91 cusr + 41.15 csys = 225.06 CPU)
```

The test confirms a correctness of GT4 installation and basic serviceability of GridFTP

Examples of GridFTP transfers:

Example1. GridFTP server run behalf of the root (port 2811). All users DN mapped in /etc/grid-security/grid-mapfile. Transfer request :

.$GLOBUS_LOCATION/bin/globus-url-copy –dbg –vb gsiftp://omii03.jinr.ru:2811/tmp/file45m gsiftp://omii02.jinr.ru:2811/bin/file45m.tmp

Example 2. GridFTP server run behalf of the dedicated user "globus"

- All users DN mapped in /home/globus/.globus.gridmap.
- **Start server**

  grid-proxy-init

  $GLOBUS_LOCATION/sbin/globus-gridftp-server –p 2811 –S
- **Transfer request:**

$GLOBUS_LOCATION/bin/globus-url-copy –dbg –vb -ss "/O=Grid/OU=GlobusTest/OU=simpleCA-omii02.jinr.ru/OU=jinr.ru/CN=Globus 0MII03 admin" -ds " O=Grid/OU=GlobusTest/OU=simpleCA-omii02.jinr.ru/OU=jinr.ru/CN=Globus administrator" gsiftp://omii03.jinr.ru:2811/tmp/file45m gsiftp://omii02.jinr.ru:2811/tmp/file45m.tmp

Test FTPTEST

The test for two machines FTPTEST, carrying out in the given cycle of operation, is prepared. One cycle of this test includes two operations:

- **PUT:** Transfer of a file to other machine
- **GET:** Reading of the transferred file back

For running of the test it is required started GridFTP servers on both machines (root, port 2811) and presence of the valid proxy certificate of the client.

This test is a part of the complex test for data transfers in GT4 (Look results of complex testing)

Transfer of the large files.

The test for two machines **terra-ftp-test.sh**, carrying out in the given cycle of operation is prepared. At each cycle, this test sends a file by the size 1GB to other machine at JINR local network. For check of transfer of the data 1TB in the cycle 1000 is given.

Test result:

Cycles	**Summary time**	**Av. trans. speed**	FAILURES
1000	32 h 35 m	8.5 MB/s	None

GridFTP conclusion:

General impression about GridFTP: the transfer of the data with this GT4 application is reliable enough and is simple in circulation

RFT, Reliable File Transfer

PostgreSQL 7.3.10 and database **RFTdatabase** are established according to the documentation on RFT

Installation RFT test

The basic task of the installation test - to check up correctness of installation components and their basic serviceability

Notes:

- In Binary Installer for FC3 there are no files **runtests.xml** and **test.properties**
- For start of the test it is necessary to make complex (difficult) configuration adjustments
- Being started in systems FC3, SLC this test did not manage fully The questions of testing were sent to **gt4-friends**. One of the answers to our inquiry: "*The file not found error is just a race condition between one test finishing and deleting the files and other test needing it. you can ignore that*" (Ravi Madduri)

Command-line client operations

The work of two command-line **rft** and **rft-delete** in various modes is checked up:

- standard running of GridFTP server (root, port 2811)
- Container as SOAP server with HTTP (port 8080) and HTTPS (port 8443)
- Tomcat as SOAP server with HTTP (port 8080) and HTTPS (port 8443)

Some examples:

```
rft -h omii03.jinr.ru -m msg  -f transfer2.xfr
rft-delete -h omii03.jinr.ru -m msg  -f delete.xfr
rft-delete -h omii03.jinr.ru -r 8443 -m trans  -f delete.xfr
```

The tests RFT for complex testing of transfers of the data

For realization of a complex cycle of testing of transfers of the data with application RFT two tests are prepared: **RFTTEST** and **SRFTTEST**, carrying out in a given cycle operations:

- Transfer of a file on the local machine, operation **rft**

- Removal of this file, operation **rft-delete**
- Transfer of a file on other machine, operation **rft**
- Removal of this file, operation **rft-delete**

The test SRFTTEST requires access to services with use of the protocol **https** and port **8443.** One Tomcat sample is enough forrunning of both tests.

2.4. GridFTP and RFT. The complex test for check of stability of work with the data exchange

Work on duration with the maximal charge

Conditions of realization of the test:

- Two similar machines in a local network JINR (Dubna) with standard GT4 installation were used;
- By each machine both servers GridFTP (root, port 2811) and Tomcat on 2 ports (8080, 8443) by the protocols http and https are started;
- - By each machine the databases RFTdatabase in PostgreSQL for support RFT are established;
- As additional loading on GT4, operational systems and SOAP servers (in this case Tomcat) the test **WSTEST** was used. The test WSTEST (ws-test.sh), carrying out the reference by the given cycle to custom Grid services of two machines is prepared. Used services: the standard test counter-client and examples (first, singleton, factory, RP, RL, params) from the Sotomayor's Tutorial;
- By each machine 4 tests (WSTEST, FTPTEST, RFTTEST and SRFTTEST) were simultaneously started on behalf of 4 clients; as a whole 8 simultaneously working tests
- For transfer the files in 50MB were used
- For each test the counter of recurrences (at the rate of work of the test within **two days)** was set, on this time the proxy certificates were created.

The results of testing are given below. It is possible to consider these results encouraging in an estimation of stability GT4 in work with the data management. Used CPU time of machines reached from time to time 30 %.

Test	Cycle	Duration of work (hours)	Failures
FTPTEST	10 000	40	None
RFTTEST	2000	40, 45	None
SRFTTEST	2000	44	None
WSTEST	1000	42	None

Transfer of the big data arrays (in 1 Tb) in a local network JINR (technical speed is 100 Mb)

The LHC tasks require steady transfer of the large files (terra bytes). Therefore it was important to check up GT4 ability to process steadily such flows of the data

Conditions of realization of the test:

- Running GridFTP server on each machine (root, port 2811)
- Running Tomcat on each machine (http, port 8080)
- The file in 1 GB was continuously consistently (1000 times) transferred to other machine in a local network of JINR.

Testing results:

	Command-line	Summary time	Av. speed	Failures
RFT	Rft	32h 35m	8.5 Mbyte/s	None
GRIDFTP	Globus-url-copy	25h 14m	11.00 Mbyte/s	None

GT4 gridFTP server and RFT service tests for file transfers with file size 10Mb, 50 Mb, 100Mb, 500Mb and 1000 Mb (1 Mb=1048576 bytes).

Client-side values of parameters (tables 1 and 3) were gotten by execution "time" command together with correspondent client command. Server side parameters were received from "vmstat" command output (shows virtual memory statistics) during test execution with 1 second interval.

Measured values of parameters during file transfer tests from client to remote gridFTP server are represented in Table 1 and 2. Comparison of execution times of scp and GridFTP file transfers are shown on the figure 1.

Table 1. Client gridFTP parameters.

Parameter	***File size, Mb***				
	10	***50***	***100***	***500***	***1000***
real, sec	1.205	4.862	9.337	45.217	107.08
user, sec	0.08	0.211	0.339	1.645	3.181
sys, sec	0.033	0.181	0.342	1.693	5.437
transfer speed, Mb/sec	8.3	10.28	10.71	11.06	9.34

real - elapsed real time;
user - total number of CPU-seconds that the process spent in user mode;
sys - total number of CPU-seconds that the process spent in kernel mode;
transfer speed – value which is given by division file size by real time;

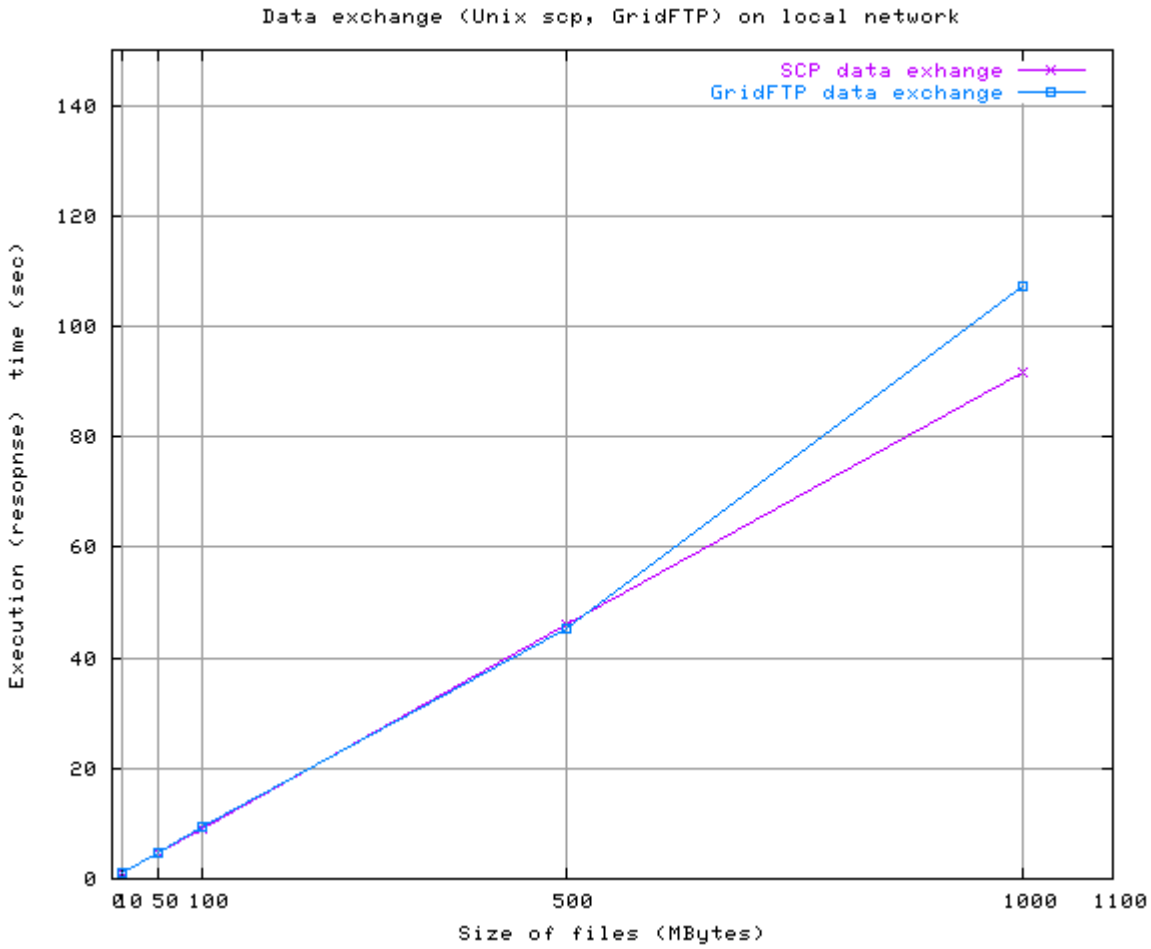


Figure 1. Comparison of execution times of SCP and GridFTP file transfers.

Table 2. Server gridFTP parameters.

Parameter	***File size, Mb***				
	10	***50***	***100***	***500***	***1000***
Summary user and system time, sec	1.4	3.58	6.67	32.44	66.93
Average CPU usage, % (user+system)	7.78 (4.00+3.78)	6.07 (4.37+1.69)	4.94 (3.9+1.04)	7.34 (6.43+0.91)	4.86 (4.15+0.72)

Parameter	***File size, Mb***				
	10	***50***	***100***	***500***	***1000***
Time points	18	59	135	442	1376

Measured values of parameters during file transfer test using RFT service are represented in Table 3 and 4. Comparison of execution times of SCP and RFT file transfers are shown on the figure 2.

Table 3. Client RFT parameters.

Parameter	***File size, Mb***				
	10	***50***	***100***	***500***	***1000***
real, sec	25.069	20.043	24.913	66.531	117.259
user, sec	4.992	5.049	5.594	5.572	5.681
sys, sec	0.195	0.217	0.23	0.239	0.278
transfer speed, Mb/sec	0.4	2.49	4.57	7.52	8.53

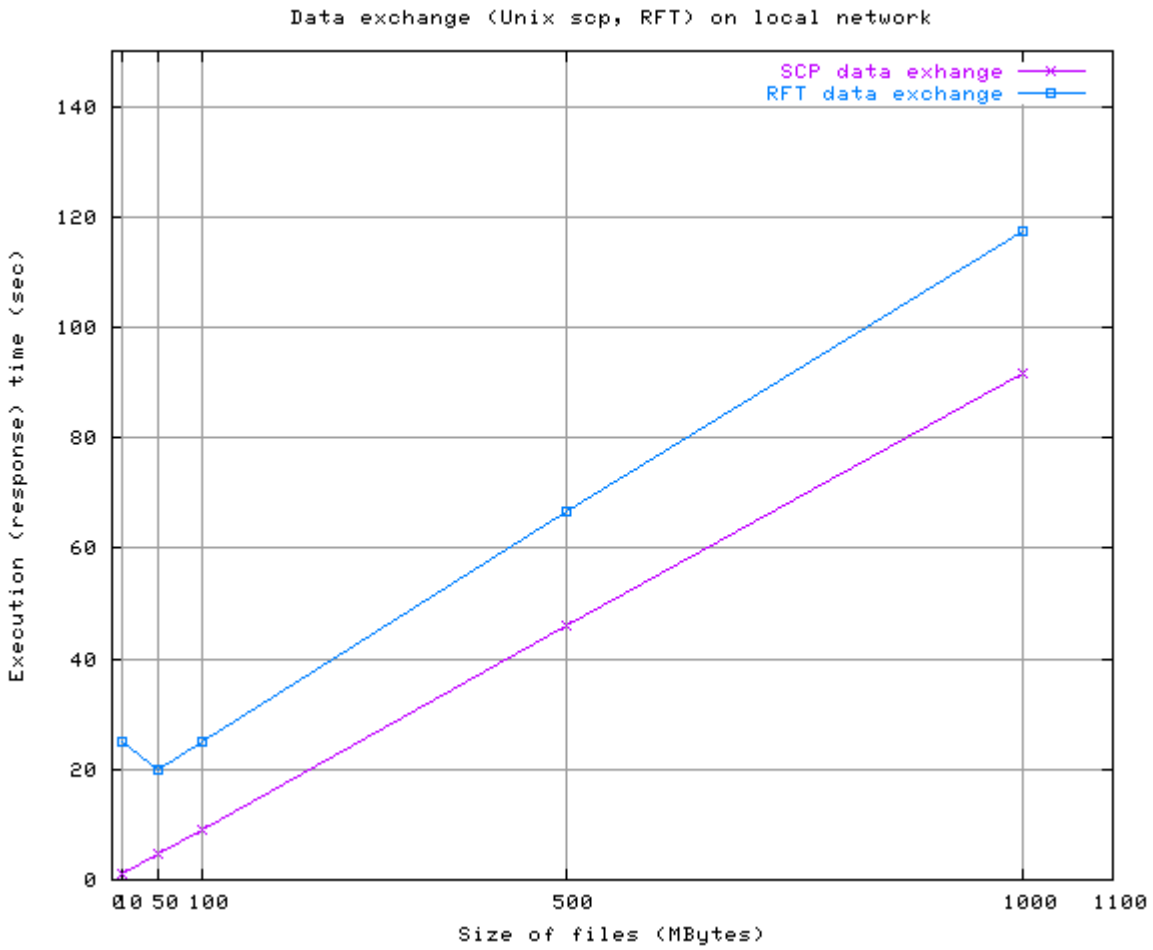


Figure 2. Comparison of execution times of SCP and RFT file transfers.

Table 4. Server RFT parameters.

	File size, Mb				
Parameter	***10***	***50***	***100***	***500***	***1000***
Summary user and system time, sec	1.48	4.81	8.83	34.8	72.61
Average CPU usage, % (user+system)	0.55 (0.30+0.25)	2.31 (1.81+0.5)	3.49 (2.84+0.65)	3.62 (3.08+0.55)	6.52 (5.63+0.89)
Time points	269	208	253	960	1114

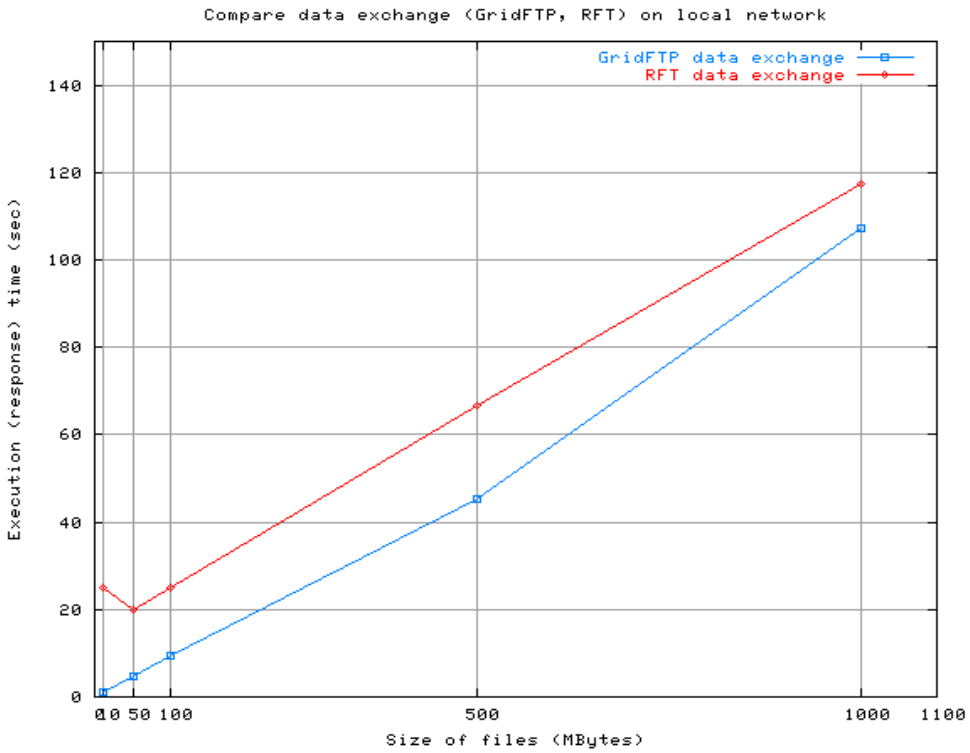


Figure 3. Comparison of execution times of GridFTP and RFT file transfers

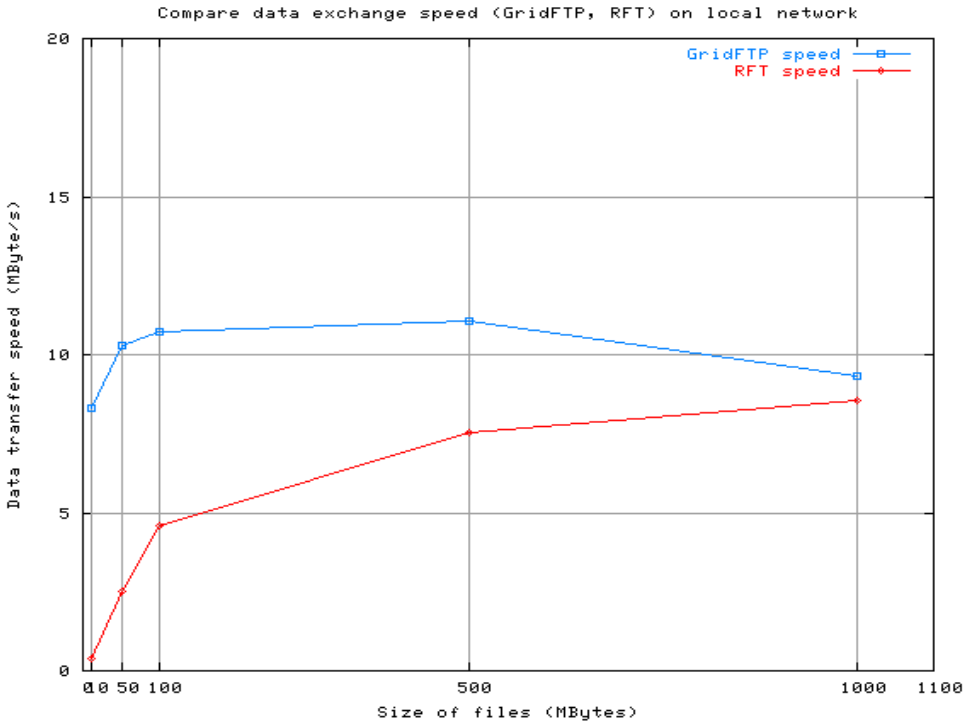


Figure 4. Speed comparison of GridFTP and RFT file transfers.

General conclusion on GT4 Data Management (components GridFTP and RFT) : the transfer of the data with this GT4 application is reliable enough and is simple in circulation

GT4 GSI

Authentication

The security layer in globus, known as GSI, uses SSL/TLS mechanism for the mutual authentication. After the communication is established the encryption is turned off by default, but it can be specified to keep the shared key encryption over the communication channel. Single sign-on services are provided using the standard X.509 proxy certificates (RFC 3820). GSI supports both message-level security and transport-level security. Transport-level security is presented with TLS encryption of the communication channel. Anonymous mode (encrypted channel without credentials) is also supported. Message-level security is implemented according to WS-Security [12] and WS-SecureConversation [13] specification which is encrypting the payload or part of the payload of the SOAP messages. The implementation supports integrity protection, encryption and replay prevention (verifying that the same message is not received several times).
End entities are identified using X.509 certificates. Delegation service is implemented allowing the clients to delegate and renew its proxy certificates to the services, however the implementation mat be not compliant to the WS-Trust [14] specification since the latter is not yet well-defined. Username/password authentication may be used for the anonymous TLS mode, but all advanced security features like delegation, etc. are not supported in this mode.

Authorization

GT4 GSI supports grid-mapfile authorization and SAML[15]. SAML defines formats for a number of types of security assertions and a protocol for retrieving those assertions. GSI uses SAML AuthorizationDecision assertion for communicating with CAS and third-party services like PERMIS for authorizing access to the GT4-based services. It was not clear from the documentation if the VOMS service is supported, but it should work since the VOMS is .

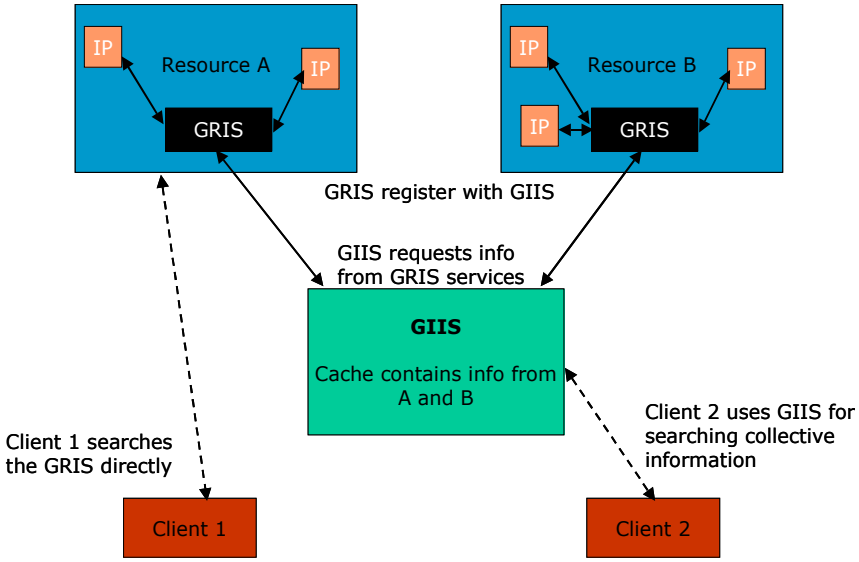
MDS4 vs R-GMA (and MDS2)

In this notes we collected facts for comparison of MDS4 (GT4) and R-GMA (accepted as Information and Monitoring Service in gLite (EGEE middleware)). We go through the following points:

- General architecture
- Components
- Information Sources/Providers
- Information model
- Specification, protocols
- Query language
- Data flow
- APIs
- Clients and user interface
- Summary

General architecture

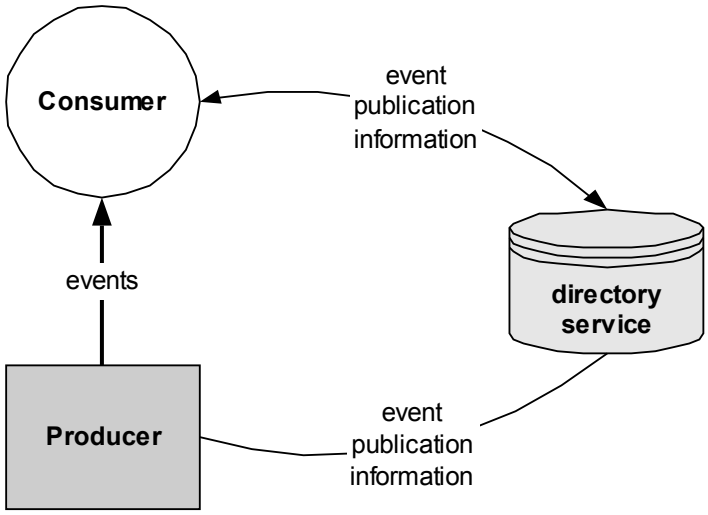
MDS2



MDS2 Architecture

- Standard mechanism for publishing and discovery
- Decentralized, hierarchical structure
- Soft-state protocols
- Caching
- Grid Security Infrastructure (GSI) credentials

R-GMA



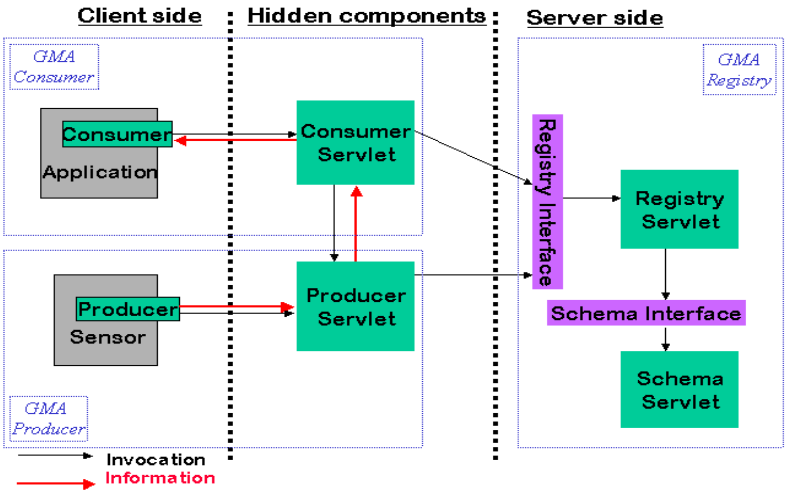
GGF Grid Monitoring Architecture (GMA)

GMA as defined currently:

- Defines only the basic concepts
- No APIs, protocols, schema (data model) defined

R-GMA (see [18]):

- a *relational* implementation of the GMA
  - developed within the European DataGrid (EDG);
  - brings the power and flexibility of the relational model;
- R-GMA creates the impression that you have one RDBMS per VO - is a way of using the relational model in a *Grid* environment (not a general distributed RDBMS);
- all the producers of information are
  - quite independent;
  - relational in the sense that Producers announce what they have to publish via an SQL CREATE TABLE statement and publish with an SQL INSERT and that Consumers use an SQL SELECT to collect the information they need (more formal description of R-GMA see [19])
- Conforms to the Web Services Architecture:
  - all operations (defined by WSDL) are requested by applications through an exchange of messages with the service;
  - the sequence and format of messages required for each operation, including any parameters, is specified in the WSDL;
- Used Java Servlet technologies: each component has the bulk of its implementation in a Servlet;
- Focus on notification of events
- User can subscribe to a flow of data with specific properties directly from a data source.



R-GMA Architecture

MDS4

Architecture is based on the key WSRF concept = Resource Properties:

- Every service advertises Resource Properties
  - Monitoring data is "baked right in"
  - WSRF has common mechanism to expose a state data to requestors for query, update and change notification
  - Service-level concept, not host-level concept
- Thus:"native" information source

Components

MDS2

- Producers:
  - Grid Resource Information Service (GRIS)
- Consumers/Index:
  - Grid Information Index Service (GIIS)

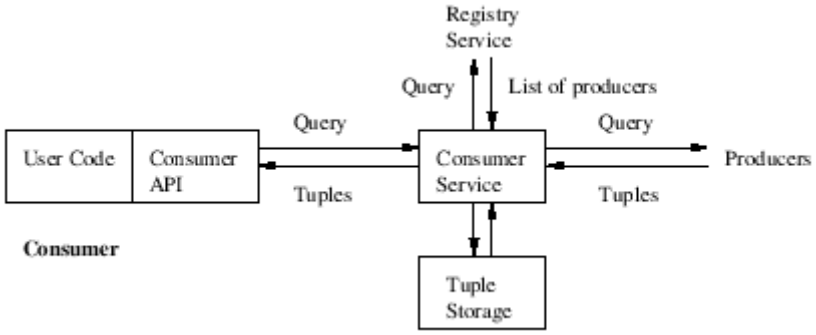
R-GMA

- Four principal components running on one or more servers (each service is defined by a WSDL document):
  - Registry
  - Schema
  - Consumers
  - Producers

- The ***registry*** stores information about all producers currently available.
  - Registry records details about the Producer, which include the description and view of the data published, but not the data itself.
  - To avoid a bottleneck and single point of failure a code has been written to allow multiple copies of the registry to be maintained.
  - Each one acts as master of the information which was originally stored in that Registry instance and has copies of the information from other Registry instances.
  - Synchronization is carried out frequently.
  - The system makes use of soft state registration to make it robust. Producers and Consumers both commit to communicate with their servlet within a certain time. A time stamp is stored in the Registry, and if nothing is heard by that time, the

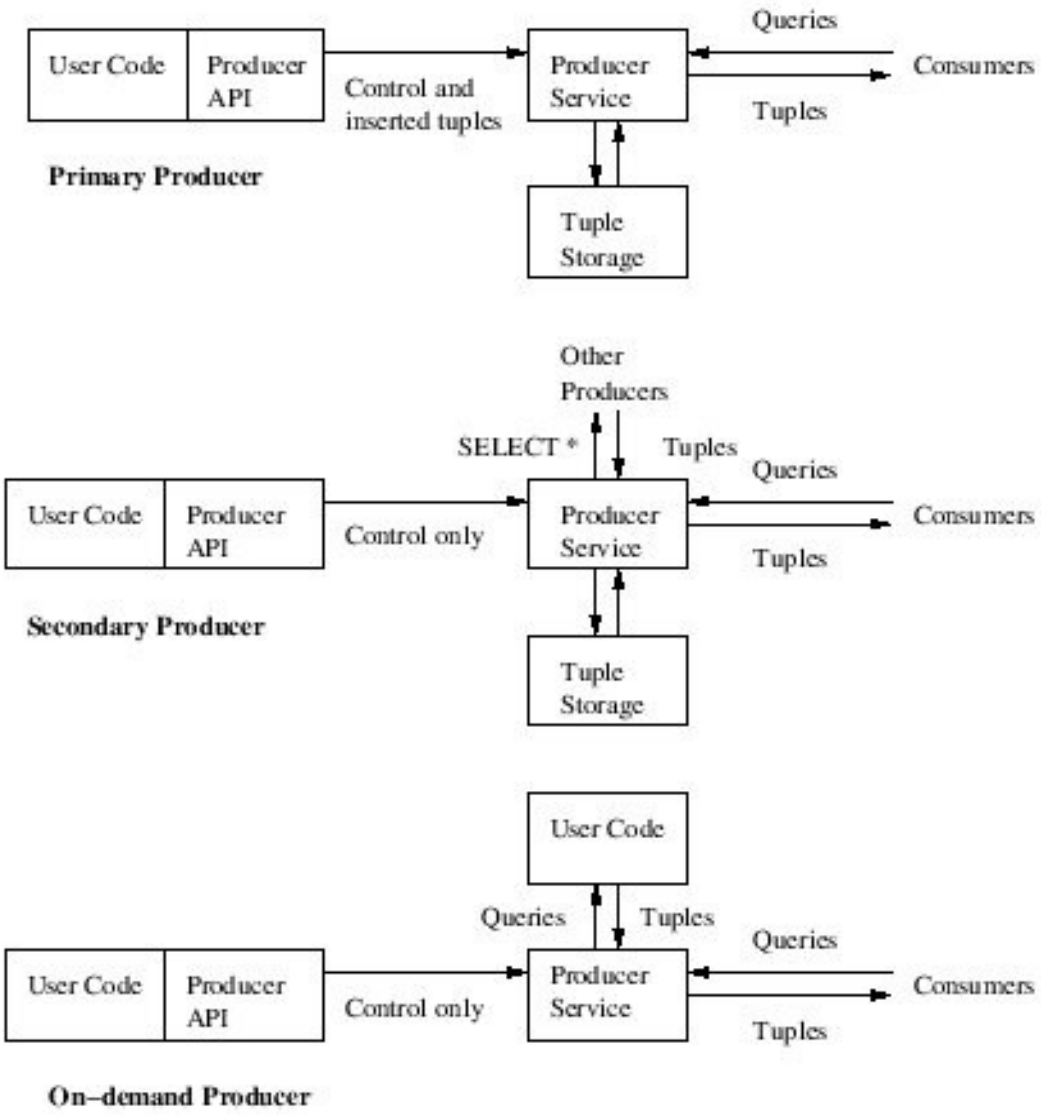
Producer or Consumer is unregistered.

- ***Schema***
  - The description of the data is actually stored as a reference to a table in the Schema. In practice the Schema is colocated with the Registry.
  - For service location and monitoring one defines a pair of tables:
    - Service - publishes its existence and how to contact, each Service tuple includes the type of the service and a URI for the service where the hostname within the URI is where the serice is located. (Eventually these will all be URLs to contact the service)
    - ServiceStatus - each service provider specifies a command (invoked locally on each machine running a service) which can be run to obtain service status; the information is then collected by an Archiver to a LatestProducer.
    - Thus:
      - the Service table says what should exist
      - the ServiceStatus gives the current state Grid wide.
- ***Consumers:***



R-GMA consumer

- There are *three classes* and *five types* of producers:
  - classes:
    - Primary Producer,
    - Secondary Producer,
    - On-demand Producer,



Classes of R-GMA producers

- Five different types of Producers:
  - DataBaseProducer: supports history queries - writes each record to an RDBMS. This is slow (compared to a StreamProducer) but it can handle joins;
  - StreamProducer: supports continuous queries and writes information to a memory structure where it can be picked up by a Consumer;
  - ResilientProducer: similar to the StreamProducer but information is backed up to disk so that no information is lost in the event of a system crash.
  - Latest-Producer: supports latest queries by holding only the latest records in an RDBMS;
  - CanonicalProducer: somewhat different as there is no user interface to publish data via an SQL INSERT statement, instead it triggers user code to answer an SQL query (the other Producers are all Insertable - this means that they all have an interface accepting an SQL INSERT statement).
- Other valuable components:
  - ***Archiver*** which is a combined Consumer-Producer:
    - a user has to tell an Archiver what to collect and it does so on his behalf.
  - The ***mediator*** (which is hidden behind the Consumer interface) is the component which makes R-GMA easy to use. Producers are associated with views on a

virtual data base. Currently views have the form

SELECT * FROM <table> WHERE <predicate>

- This view definition is stored in the Registry. When queries are posed, the Mediator uses the Registry to find the right Producers and then combines information from them.

- Clients (see **Clients, User interface)**

MDS4

Components:

Information sources

Native

Add-on

Higher level services

Index

- Index Service collects information from (potentially) many sources and publishes it in one place.
- Index Service is both registry ***and cache***
- Subscribes to information providers
  - Data, datatype, data provider information
- Caches last value of all data
- In memory default approach
- Each GT4 container has a default Index Service that keeps track of resources that have been created within the container.

- MDS has a soft consistency model
  - Published information is recent, but not guaranteed to be the absolute latest
  - Each registration into an Index Service is subject to soft-state lifetime management
  - Reg's have expiry times and must be periodically renewed
  - Index is self-cleaning, since outdated entries disappearing automatically
  - It is recommended that the Index Servers be run in one of two modes:
    - a *public index*, in which all Index data is collected through anonymous queries and access is granted to everyone;
    - a *personal index*, in which all index data is collected using credentials delegated by an individual and access is restricted to that same individual.

Trigger

- Compound consumer-producer service
- Subscribe to a set of resource properties
- Evaluate that data against a set of pre-configured conditions (triggers; defined in a configuration file).
- When a condition matches, email is sent to pre-defined address
- Uses service groups as part of their administrative interface, to keep track of what information they are to collect.
- NB: This (trigger) functionality, inspired by a similar capacity in Hawkeye

Archiver (not available in GT4.0 far but is planned for future releases)

- Compound consumer-producer service
- Subscribe to a set of resource properties
- Data put into database (Xindice)
- Other consumers can contact database archive interface

- Other valuable components:
  - MDS4 Aggregate Framework:
    - General framework for building services that collect and aggregate data
      - Index and Trigger service both use this, in fact they are both specializations

of a more general *aggregator framework*
Clients (see **Clients, User interface)**

Information Sources/Providers

MDS2
User code + GRIS

R-GMA
User code + R-GMA Producers

MDS4
- XML Based - not LDAP
- "Native" information sources
  - All GT4 services:
    - In the case of a Query or Subscription source, the information provider is a WSRF-compliant service
      - Query => WS-ResourceProperty;
      - Subscription => WS-Notification mechanisms;
- "Add-on" information sources
  - in the case of an Execution source, the information provider is an executable program that obtains data via some domain-specific mechanism.
    - Code that generates resource property information (were called service data providers in GT3)
  - GRAM - GT4 Job Submission Interface
    - Queue information, number of CPUs available and free, job count information, some memory statistics and host info for head node of cluster
  - Several GT4 services
    - Reliable File Transfer Service (RFT)/GridFTP
      - Service status data, number of active transfers, transfer status, information about the resource running the service
    - Community Authorization Service (CAS)
      - Identifies the VO served by the service instance
    - Replica Location Service (RLS) - not a WS
      - Location of replicas on physical storage systems (based on user registrations) for later queries
        are not WSRF services => there exist plans (see [22]) to build a service to talk to it to advertise needed resource properties;
  - Interfaces to both Hawkeye and Ganglia
    - Not WS so these are Execution Sources
    - Basic host data (name, ID), processor information, memory size, OS name and version, file system data, processor load data
    - Some condor/cluster specific data
  - Other data can also be gathered this way *but*
    - Interfaces to other probes, archives are needed

Summary on the MDS4 providers

Name	Info source	Source Type	Information Provided

Hawkeye	Condor pool	Execution	Basic host data (name, ID), processor information, memory size, OS name and version, file system data, processor load data, and other basic Condor host data.
Ganglia	Cluster	Execution	Basic host data (name, ID), memory size, OS name and version, file system data, processor load data, and other basic cluster data.
GRAM	GT4 grid resource allocation and management service	Query, Subscription	Processor information, memory size, queue information, number of CPUs available and free, job count information, and some memory statistics
RFT	GT4 reliable file transfer service	Query, Subscription	RFT service status data, number of active transfers, transfer status, information about the resource running the service
CAS	GT4 community authorization service	Query, Subscription	Identifies the VO served by the service instance
RLS	GT4 replica location service	Execution	Location of replicas on physical storage systems (based on user registrations) for later queries.
Basic	Every GT4 Web service	Query, Subscription	ServiceMetaDataInfo element includes start time, version, and service type name

Information model

ALL:

information sources *can* publish information according to the ***GLUE (***=Grid Laboratory Uniform Environment) ***schema***

- Schema developed by DataTAG for EU/USA interoperability.
- Modelled in UML

MDS2

LDAP version (implementations) of ***GLUE***

R-GMA

SQL version (implementations) of ***GLUE***

MDS4

XML *mapping* of the ***GLUE*** (for Hawkeye, Ganglia and GRAM)

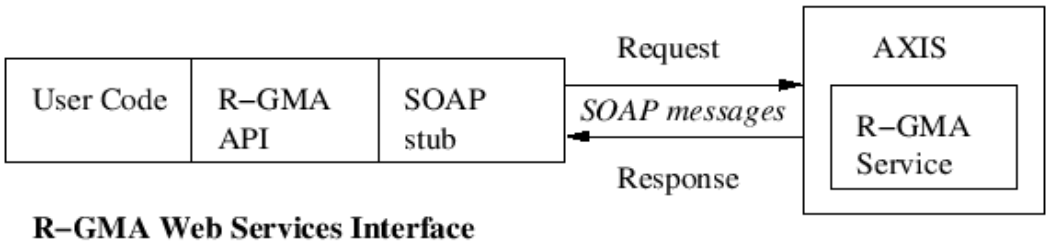
Specification, protocols

MDS2

LDAP

R-GMA

- uses "SOAP messaging over http/s", in a request/response pattern, for all user-to-service and service-to-service communications apart, from streaming
  - Apache AXIS software implements the SOAP interface on the server.



R–GMA Web Services Interface

Resource Framework (WSRF) in R-GMA:

- For each of the principal services, R-GMA has a corresponding factory service to create resources; this allows R-GMA to support services with different attributes (quality of service): the user specifies the attributes required (e.g. reliable delivery of tuples) and the factory service creates a new resource with the URL of a suitable service. The user only needs to know the URL of their local factory service.
- Each instance of a producer or consumer exists as a resource on a server;
- A resource contains the private data or threads associated with that particular instance (such as the tuple-storage and tuple-streamer in a Primary Producer);
- It is created by an R-GMA service, when a user sends a "create" request, resides on the server with the service, and is given an identifier which is passed back to the client.
- The client then includes the resource identifier with all subsequent requests relating to that instance (the API takes care of this);
- the concept of soft-state registration is used:
  - A resource is normally destroyed at the explicit request of the user, but in order to protect itself from an accumulation of redundant resources, an R-GMA service requires the user to specify a termination interval (in a restricted range) when it creates the resource. If the service doesn't hear from the user for any period exceeding the termination interval, the resource is destroyed.
  - The registry protects itself in the same way, against producers and consumers which register then disappear, so a periodic keep-registered message has also to be sent to the registry, by the consumer and producer services.
  - R-GMA has a Resource Framework (borrowing from the WSRF) to manage the life-cycle of resources running on a server. This includes sending keep-registered messages to the registry on behalf of producer and consumer resources. It is implemented as part of the R-GMA services, and is hidden from the user.
- Registry and Schema instances also exist as resources created by a Registry or Schema service.
- This allows a single service to host registries or schemas for multiple virtual organisations. However they are not managed by the Resource Framework, the VO is the resource identifier, and they do not time out.

MDS4

Web Services Standards Used By MDS4:

- WS-ResourceProperties
  - defines a mechanism by which Web Services can describe and publish *resource properties*, or sets of information about a resource. Resource property types are defined in the service's WSDL, and the resource properties themselves can be retrieved, in the form of XML documents, using WS-ResourceProperties query operations.
- WS-BaseNotification

- defines a subscription/notification interface for accessing resource property information.
- WS-ServiceGroup
  - defines a mechanism for grouping related resources and/or services together as *service groups*.

Query language

MDS2

LDAP-based queries (*ldapsearch*)

R-GMA

There are four types of query:

- continuous: provides the client with all results matching the query as they are published; a continuous query is therefore acting as a filter on a published stream of data;
- latest: used to find the current value of something;
- history: might be seen as the more traditional one, where you want to make a query over some time period including "all time";
- static: supported by *On-demand* producers - database-like queries and do not contain R-GMA time-stamps; the primary purpose of an On-demand producer is to allow large databases to be accessed through the R-GMA infrastructure, without the overhead of copying tuples into a producer service.

The set of queries that a particular producer supports is recorded in the registry. All query types except static can take an optional time interval parameter.

MDS4

- Resource properties may be queried by name or via XPath [XPATH] queries
  - Because RPs are stored in XML, we may use the XPath query language to search inside the document; for instance,
    - count(//*[local-name()='Entry'])

    returns a count of elements named Entry

Data flow

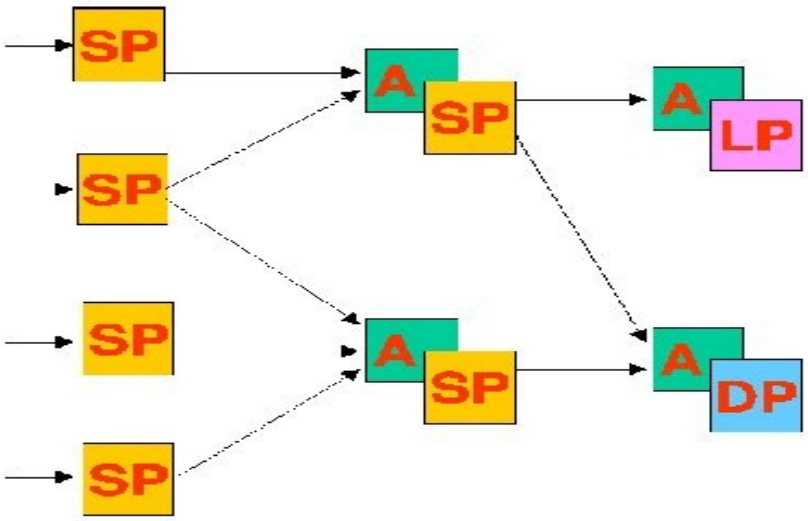
MDS2

- Pull only data models

R-GMA

- Push and pull data models

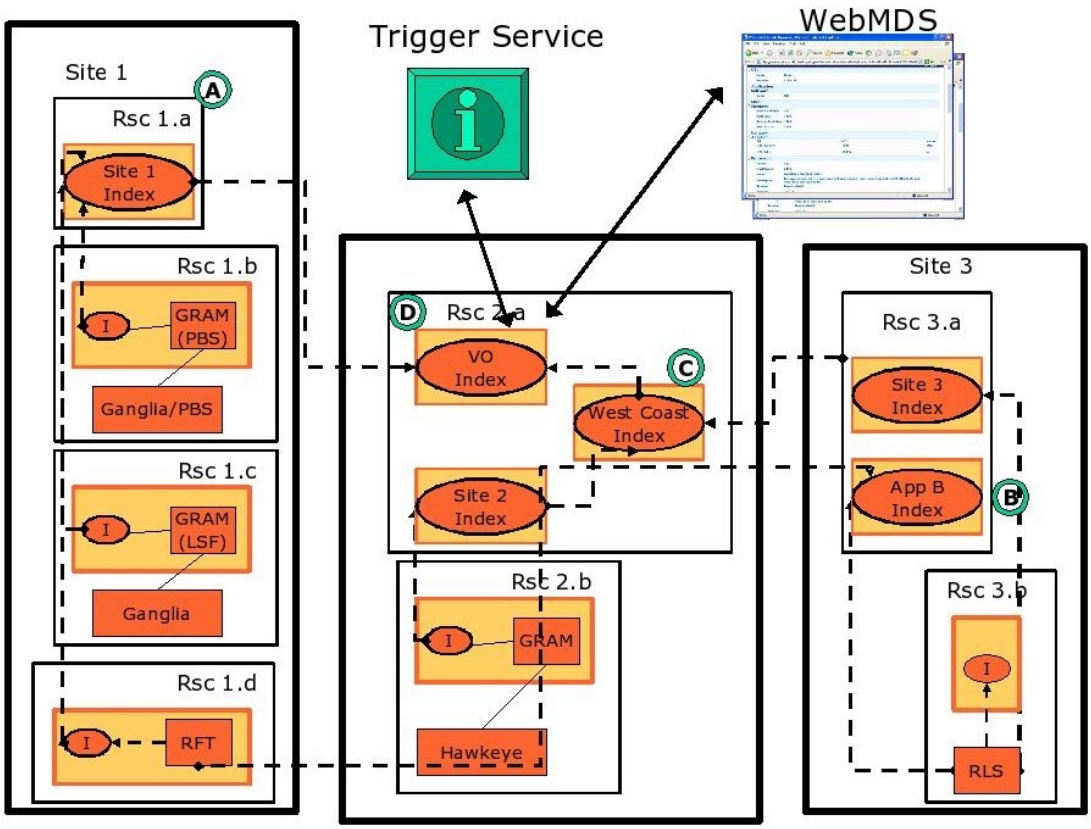
A possible topology of R-GMA components (SP=StreamProducer, A= Archiver, LP=Latest-Producer, DP=DataBaseProducer):



MDS4

- Push and pull data models
- No single global Index provides information about every resource on the Grid
  - Hierarchies or special purpose index's are common
  - Each virtual organization will have different policies on who can access its resources
  - Registering a container index into a VO index
- Users can get information in two ways
  - Query from resources directly:
    - Information more recent,
    - Higher load on resources and on requestor
    - Need to know which resources exist in order to query them
  - Query from index:
    - Do not need to know existence of each resource
    - Information is older

Use case (MDS4 sample deployment):



APIs

MDS2

- C API

R-GMA

- Multiple APIs in Java, C++, C, Python and Perl are available to communicate with the servlets;

MDS4

- Java and C APIs

Clients & user interface

MDS2

LDAP command line tools + third party tools (like GridIce etc.)

R-GMA

Tools/clients:

- a command line tool (in Python),
- a Java graphical display tool,
- the R-GMA Browser (accessible from a Web browser without any R-GMA installation.

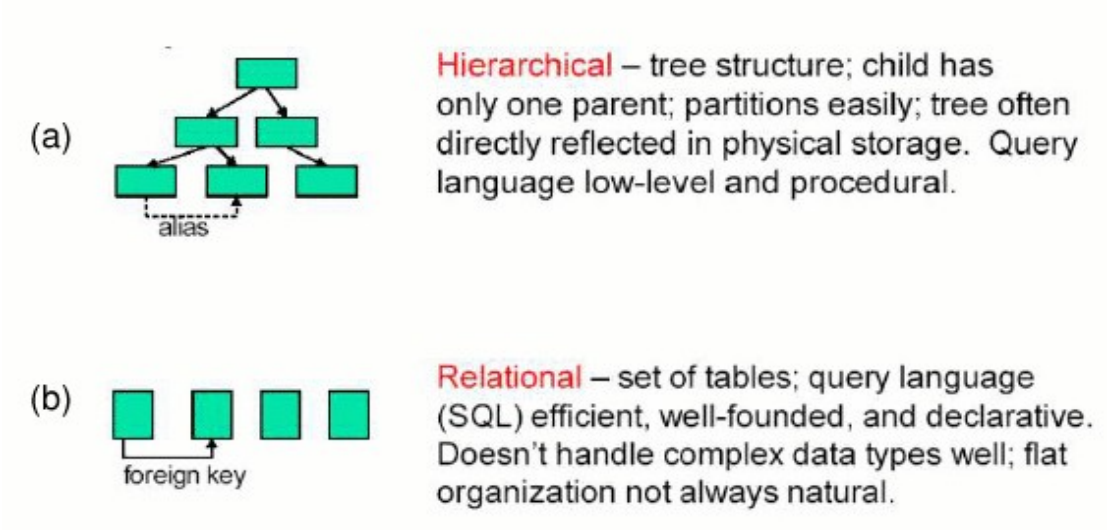
MDS4

- Command line, Java and C APIs

- MDSWeb Viz service
  - Web-based interface to WSRF resource property information
  - User-friendly front-end to the Index Service
  - Uses standard resource property requests to query resource property data
  - XSLT transforms to format and display them
  - Customized pages are simply done by using HTML form options and creating your own XSLT transform

Summary

Two types of data structure in Monitoring and Information systems:
(a) hierarchical = MDS4
(b) relational = R-GMA



Hierarchical

- scale-up to global extend
- multi-organizational modeling quite naturally (regions, institutions, departments)
- complex queries are efficient but need to be handcrafted in code
- Uses XML-based query language (Xpath in the case of MDS4) - rapidly becoming the *de facto* standard for retrieving and exchanging data

Relational

- support complex queries, joins, aggregate operations
- richer data model, complex relationships
- designed also for frequent updates, support of transactions

- Uses SQL – powerful and more common query language

But: The query languages for both cases may converge on the basis of an XML query language, XQuery (see [SQL_XML]

Some performance comparison (taken from [21]):

Comparison of Index service performance for 5 monitoring systems (values in italics are curve-fitting approximations):

Monitoring System	1 client		10 Clients		50 Clients		100 Clients	
	Single client queries /sec	Response Time (msec)	Single client queries /sec	Response Time (msec)	Single client queries /sec	Response Time (msec)	Single client queries /sec	Response Time (msec)
MDS2 w/cache	0.88	129	0.45	147	0.92	153	0.93	182
MDS2 w/o cache	0.45	1219	0.15	5534	0.77	29,175	0.91	40,410
R-GMA	0.92	61	0.03	277	0.24	3230	0.89	9734
Hawkeye	0.93	79	0.02	106	0.12	113	0.68	463
MDS-4	24	40	*16.8*	n/a	*3.29*	n/a	0.85	1243

Resources

References

[1]. http://www.globus.org/alliance/events/ggf14/GGF14-TH.pdf

[2]. Alternative Software Stacks for OGSA-based Grids. Marty Humphrey, Glenn Wasson, Yuliyan Kiryakov, Sang-Min Park, David Del Vecchio, Norm Beekwilder, Jim Gray. http://www.cs.virginia.edu/~humphrey/papers/AlternativeSoftwareStacksOGSA.pdf

[3]. Assessing the Risk and Value of Adopting Emerging and Unstable Web Services Specifications. Savas Parastatidis, Jim Webber. http://www.cs.ncl.ac.uk/research/pubs/trs/papers/851.pdf

[4]. WS-GAF: A Framework for Building Grid Applications using Web Services. Savas Parastatidis, Jim Webber, Paul Watson, Thomas Rischbeck. http://www.cc-pe.net/CCPEwebresource/c816webber/c816wsgaf.pdf

[5]. http://ds.informatik.uni-marburg.de/MAGE/sg/tutorial/index.html

[6]. Introducing WEDS: a WSRF-based Environment for Distributed Simulation. Technical Report Number UKeS-2004-07. Peter Coveney, Jamie Vicaryy, Jonathan Chinz, Matt Harve. http://www.nesc.ac.uk/technical_papers/UKeS-2004-07.pdf

[7]. Supporting Application-Tailored Grid File System Sessions with WSRF-Based Services. Ming Zhao, Vineet Chadha Renato, J. Figueiredo. http://www.acis.ufl.edu/~ming/research/hpdc-14.pdf

[8]. State and Events for Web Services: A Comparison of Five WS-Resource Framework and WS-Notification Implementations.

[9]. http://dsd.lbl.gov/gtg/projects/pyGridWare/

[10]. http://www.sve.man.ac.uk/Research/AtoZ/ILCT

[11]. M. Humphrey, G. Wasson, M. Morgan, and N. Beekwilder. An Early Evaluation of WSRF and WSNotification via WSRF.NET. 2004 Grid Computing Workshop (associated with Supercomputing 2004). Nov 8 2004, Pittsburgh, PA.

[12]. IBM, Microsoft, and VeriSign, Web Services Security Language (WS-Security).

[13]. IBM, et al., Web Services Secure Conversation Language (WSSecureConversation) Version 1.0, December 18.

[14]. IBM, et al., Web Services Trust Language (WS-Trust).
[15]. Security Association Markup Language (SAML) Specification v.1.0, http://www.oasis-open.org/committees/security/
[16]. J. Schopf, and B. Clifford, "Monitoring Clusters and Grid", http://www.grids-center.org/news/clusterworld/0804GridFinal.pdf
[17]. J. Hofer, "VO Grid Technologies, Grid Information Services and Monitoring", http://dps.uibk.ac.at/~radu/grid/lecture4_1.pdf
[18]. A.Cooke et al. "Relational Grid Monitoring Architecture (R-GMA)", http://www.-gridpp.ac.uk/papers/ah03_148.pdf
[19]. A.Cooke et al. "R-GMA: An Information Integration System for Grid Monitoring", http://www.macs.hw.ac.uk/dis4g/publications/coopis.pdf
[20]. Information and Monitoring Service (R-GMA), System Specification, https://edms.cern.ch/document/490223
[21]. Jennifer M. Schopf, Mike D'Arcy, Neill Miller, Laura Pearlman, Ian Foster, Carl Kesselman, Monitoring and Discovery in a Web Services Framework: Functionality and Performance of the Globus Toolkit's MDS4, SC05 http://www-unix.mcs.anl.gov/~schopf/Pubs/mds-sc05.doc
[22]. Jennifer M. Schopf, Distributed Monitoring and Information Services for the Grid www.nesc.ac.uk/events/local/abstracts/100120051600.html
[23]. Xuehai Zhang , Jeffrey Freschel (??), Scalability comparison of MDS2, Hawkeye, R-GMA, www.mcs.anl.gov/~jms/Pubs/xuehaijeff-hpdc2003.pdf
[24]. J. E. Funderburk, S. Malaika and B. Reinwald "XML programming with SQL/XML and XQuery"