GT 4.0 Pre WS GRAM

GT 4.0 Pre WS GRAM

Table of Contents

List of Tables

Special note about Pre-WS GRAM documentation

The GT4 release includes the Pre-WS version of GRAM (formerly referred to as GRAM2), for legacy purposes only - it will be deprecated at some future time as experience is gained with the WS implementation. Because a lot of documentation exists for GRAM2 many links point into this old documentation. New features had been added to GRAM2 and information about these can only be found in the latest Toolkit documentation.

Chapter 1. Execution Management: Key Concepts

1. Overview

The Globus Toolkit provides both a suite of web services and a "pre-web services" Unix server suite to submit, monitor, and cancel jobs on Grid computing resources. Both systems are known under the moniker "GRAM", while "WS GRAM" refers only to the web service implementation. Jobs are computational tasks that may perform input/output operations while running which affect the state of the computational resource and its associated file systems. In practice, such jobs may require coordinated staging of data into the resource prior to job execution and out of the resource following execution. Some users, particularly interactive ones, benefit from accessing output data files as the job is running. Monitoring consists of querying and subscribing for status information such as job state changes.

Grid computing resources are typically operated under the control of a scheduler which implements allocation and prioritization policies while optimizing the execution of all submitted jobs for efficiency and performance. GRAM is not a resource scheduler, but rather a protocol engine for communicating with a range of different local resource schedulers using a standard message format.

For more detailed information about the concepts behind the software implementation, see WS GRAM Approach[1].

2. Conceptual details

A number of concepts underly the purpose and motivation for GRAM. These concepts are divided into broad categories below.

2.1. Targeted job types

GRAM is meant to address a range of jobs where arbitrary programs, reliable operation, stateful monitoring, credential management, and file staging are important. GRAM is not meant to serve as a "remote procedure call" (RPC) interface for applications not requiring many of these features, and furthermore its interface model and implementation may be too costly for such uses. The WS GRAM service will become cheaper over time as the underlying web service technologies improve, but as with the older pre-web service GRAM, its protocols will always involve multiple round-trips to support these advanced features that are not required for simple RPC applications.

The underlying WSRF core environment used for WS GRAM should also be considered as a direct application-hosting solution for lightweight, shared applications. In other words, if an application requires only modest input and output data transport in a stateless manner (all that matters is the result data or fault signal) and will be invoked many times, it may be a good candidate for exposure as an application-specific Web service.

2.2. Component architecture

Rather than consisting of a monolithic solution, GRAM is based on a component architecture at both the protocol and software implementation levels. This component approach serves as an ideal which shapes the implementation as well as the abstract design and features.

[1] WS_GRAM_Approach.html

GRAM service suite	Job management with GRAM makes use of multiple types of service: • Job management services represent, monitor, and control the overall job life cycle. These services are the job-management specific software provided by the GRAM solution. • File transfer services support staging of files into and out of compute resources. GRAM makes use of these existing services rather than providing redundant solutions; WS GRAM has further refactored some file transfer mechanisms present in pre-web service GRAM. • Credential management services are used to control the delegation of rights among distributed elements of the GRAM architecture based on users' application requirements. Again, GRAM makes use of more general infrastructure rather than providing a redundant solution, and WS GRAM has continued this refactoring to better separate credential management at the protocol level.
Service model	For WS GRAM, the Globus Toolkit software development environment, and particularly WSRF core, is used to implement distributed communications and service state. For pre-web service GRAM, the "gatekeeper" daemon and GSI library are used for communications and service dispatch.
Scheduler adapters	A scripted plug-in architecture is provided by GRAM to enable extension with adapters to control a variety of local schedulers.

2.3. Security

Secure operation	WS GRAM utilizes WSRF functionality to provide for authentication of job management requests as well as to protect job requests from malicious interference, while pre-web service GRAM uses GSI and secure sockets directly. The use of GRAM does not reduce the ability for system administrators to control access to their computing resources. The use of GRAM also does not reduce the safety of jobs users run on a given computing resource.
Local system protection domains	To protect users from each other, jobs are executed in appropriate local security contexts, e.g. under specific Unix user IDs based on details of the job request and authorization policies. Additionally, GRAM mechanisms used to interact with the local resource are design to minimize the privileges required and to minimize the risks of service malfunction or compromise.
Credential delegation and management	A client may delegate some of its rights to GRAM services in order to facilitate the above functions, e.g. rights for GRAM to access data on a remote storage element as part of the job execution. Additionally, the client may delegate rights for use by the job process itself. With pre-web service GRAM, these two uses of rights are inseparable, while WS GRAM provides separate control for each purpose (while still allowing rights to be shared if that is desired).
Audit	To assist with normal accounting functions as well as to further mitigate risks from abuse or malfunction, GRAM uses a range of audit and logging techniques to record a history of job submissions and critical system operations.

2.4. Job Management

Reliable job submission	WS GRAM provides an "at most once" job submission semantics. A client is able to check for and possibly resubmit jobs, in order to account for transient communication errors without risk of running more than one copy of the job. Similarly, pre-web service GRAM provides a two-phase submission mechanism to submit and then commit a job to be run.
Job cancellation	While many jobs are allowed to run to their natural completion, GRAM provides a mechanism for clients to cancel (abort) their jobs at any point in the job life cycle.

2.5. Data Management

Reliable data staging	WS GRAM provides for reliable, high-performance transfers of files between the compute resource and external (gridftp) data storage elements before and after the job execution. Pre-web service GRAM can also stage with gridftp systems but with less flexible reliable-transfer logic driving its requests.
Output monitoring	GRAM supports a mechanism for incrementally transferring output file contents from the computation resource while the job is running. WS GRAM uses a new mechanism to allow arbitrary numbers of files to be transferred in this fashion, while pre-web service GRAM only supports incremental transfer of the job's standard output and error streams.

2.6. Task coordination

Parallel jobs	Some jobs are parallel, meaning that they consist of more than one simultaneous tasks. These tasks are often hosted on parallel computing hardware to provide increased computational throughput.
Task rendezvous	Some parallel programming environments, such as MPI, provide mechanisms for all tasks in a parallel computation to find out about each other: to know the number of peer tasks as well as possibly to exchange some information between tasks. Native parallel programming models often support this within the local job start mechanism. GRAM provides a mechanism for task rendezvous which job applications may use if they do not have another more appropriate solution. This mechanism may be used directly by application code or by intervening middleware libraries, e.g. libraries that are designed to present a simplified programming model to applications run via GRAM. GRAM does not require tasks to be coordinated, but addresses this requirement in order to facilitate a wider range of applications. The protocols and APIs used to support task rendezvous are different for WS GRAM and pre-web service GRAM.

3. Related documents

The following links include internal or external documents that expand on some of these key concepts:

- Pre WS GRAM Approach[2]
- WS GRAM Approach[3]

[2] ../prewsgram/Pre_WS_GRAM_Approach.html
[3] ../wsgram/WS_GRAM_Approach.html

Chapter 2. GT 4.0 Pre WS GRAM Approach

1. Introduction

The Globus Resource Allocation Manager (GRAM) is the lowest level of Globus resource management architecture. GRAM allows you to run jobs remotely, providing an API for submitting, monitoring, and terminating your job.

When a job is submitted, the request is sent to the gatekeeper of the remote computer. The gatekeeper handles the request and creates a job manager for the job. The job manager starts and monitors the remote program, communicating state changes back to the user on the local machine. When the remote application terminates, normally or by failing, the job manager terminates as well.

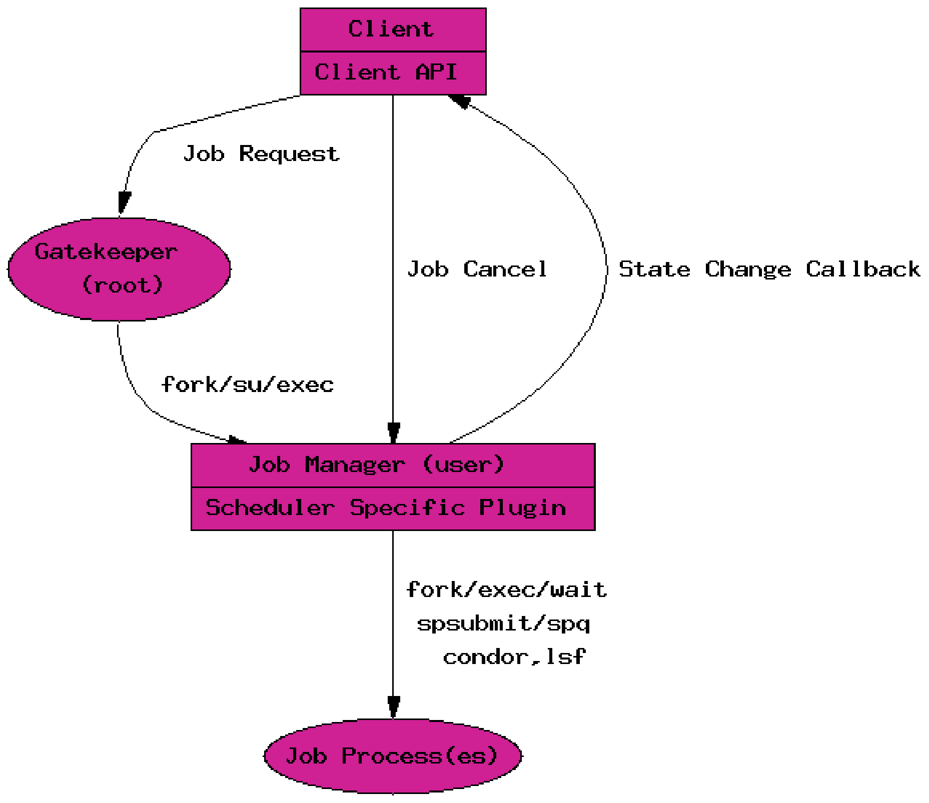
GRAM is responsible for

- Parsing and processing the **Resource Specification Language** (RSL) specifications that outline job requests. The request specifies resource selection, job process creation, and job control. This is accomplished by either denying the request or creating one or more i processes (jobs) to satisfy the request.
- Enabling remote monitoring and managing of jobs already created.

The Resource Specification Language (RSL) is a structured language by which resource requirements and parameters can be outlined by a user.

To run a job remotely, a GRAM gatekeeper (server) must be running on a remote computer, listening at a port; and the application needs to be compiled on that remote machine. The execution begins when a GRAM user application runs on the local machine, sending a job request to the remote computer. The executable, stdin and stdout, as well as the name and port of the remote computer, are specified as part of the job request. The job request is handled by the gatekeeper, which creates a job manager for the new job. The job manager handles the execution of the job, as well as any communication with the user.

The architecture of GRAM is diagrammed below:



Resource

An entity capable of running one or more processes on behalf of a user.

Client

The process that is using the resource allocation client-side API.

Job

A process or set of processes resulting from a job request. Jobs are grouped, so any error in one job results in the mutual termination of all others in the group. If the job is killed by the client, all processes are terminated, and the job itself is finally terminated as well.

Job Request

A request to gatekeeper to create one or more job processes, expressed in the supplied Resource Specification Language. This request guides

- resource selection (when and where to create the job processes)
- job process creation (what job processes to create)
- job control (how the processes should execute

2. Components

2.1. Gatekeeper

A process, running as root, which begins the process of handling allocation requests. It exists on the remote computer before any request is submitted. When the gatekeeper receives an allocation request from a client, it

- resource selection (when and where to create the job processes)
- mutually authenticates with the client,
- maps the requestor to a local user,
- starts a job manager on the local host as the local user, and
- passes the allocation arguments to the newly created job manager.

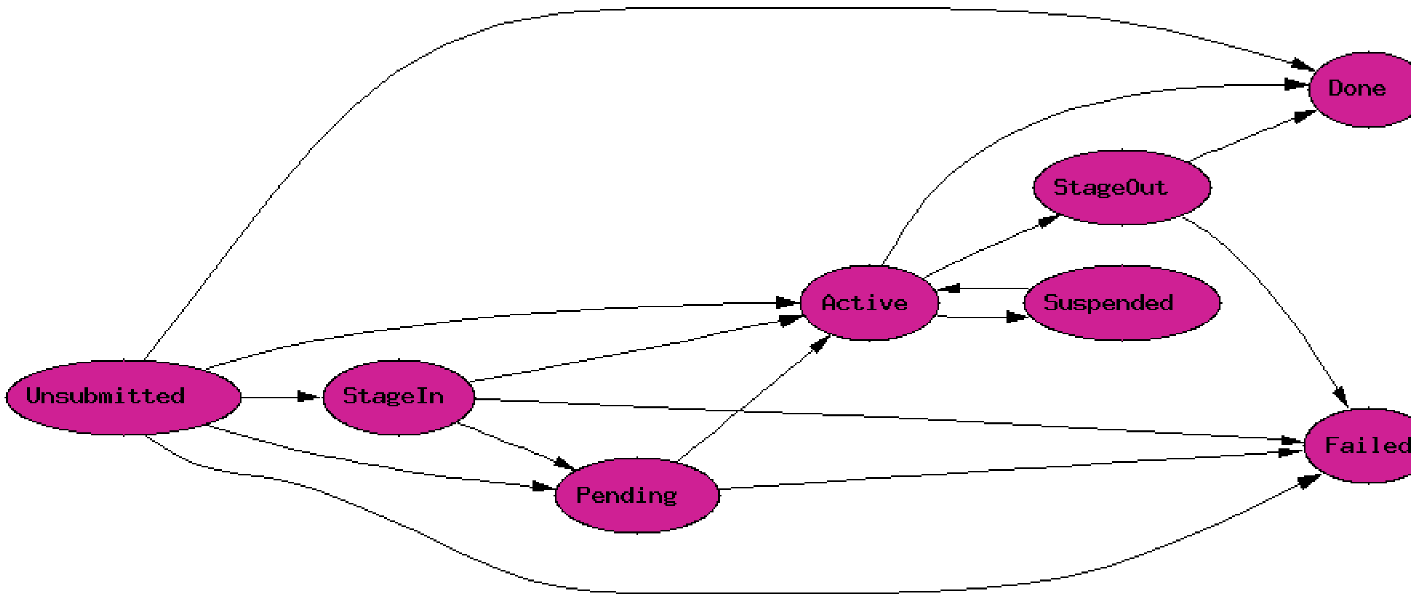
2.2. Job Manager

One job manager is created by the gatekeeper to fulfill every request submitted to the gatekeeper. It starts the job on the local system, and handles all further communication with the client. It is made up of two components:

- Common Component - translates messages received from the gatekeeper and client into an internal API that is implemented by the machine specific component. It also translates callback requests from the machine specific components through the internal API into messages to the application manager.
- Machine-Specific Component - implements the internal API in the local environment. This includes calls to the local system, messages to the resource monitor, and inquiries to the MDS.

3. Job States

The GRAM supports the following scheduling model. A user or resource broker submits a job request, which initially registers as a pending job. The job then undergoes state changes according to this state diagram:



Unsubmitted

The job has not yet been submitted to the scheduler. A job state callback for this state is never sent; rather it was introduced for the case when the job manager is stopped and restarted before the job is submitted. This state was introduced in GRAM 1.5 (Globus 2.0).

StageIn

The job manager is staging executable, input, or data files to the job. Jobs which do not involve any staging will not enter this state. This state was introduced in GRAM 1.6.

Pending

The job has been submitted to the scheduler, but resources have not yet been allocated for the job.

Active

The job has received all of it's resources, and the application is executing.

Suspended

The job has been stopped temporarily by the scheduler. Only some schedulers will cause a job to enter the Suspended state. This state was introduced in GRAM 1.5 (Globus 2.0).

StageOut

The job manager is staging output files from the job manager host to remote storage. Jobs which do not involve any staging will not enter this state. This state was introduced in GRAM 1.6.

Done

The job completed successfully.

Failed

The job terminated before completion, as a result of an error, or a user or system cancel.

4. Audit

Table 2.1. Audit Logging Support

GRAM job auditing direct to DB	GRAM can be configured to write a job audit record to a file that is ready for uploading into a Database. This can be useful for exposing and integrating GRAM job information with a Grid's existing accounting infrastructure. A case study for TeraGrid can be read here[1]
Local scheduler logging	For systems using a local batch scheduler, all of the accounting and logging facilities of that scheduler remain available for the administrator to track jobs whether submitted through GRAM or directly to the scheduler by local users.

[1] http://www.teragridforum.org/mediawiki/index.php?title=GRAM4_Audit

Chapter 3. GT 4.0 Pre WS GRAM: User Guide

1. Introduction

GRAM services provide secure job submission to many types of job schedulers for users who have the right to access a job hosting resource in a Grid environment. The existence of a valid proxy is in fact required for job submission. All GRAM job submission options are supported transparently through the embedded request document input. In fact, the job startup is done by submitting a client-side provided job description to the GRAM services. This submission can be made by end-users with the GRAM command-line tools.

2. Commandline Tools

Gram Clients[1]

3. Resource Specification Language (RSL)

Resource Specification Language (RSL)[2]

4. RSL Attributes

RSL Attributes[3]

5. Job Execution Environment

Job Execution Environment[4]

[1] http://www.globus.org/toolkit/docs/2.4/admin/guide-user.html#gram
[2] http://www.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html
[3] http://www.globus.org/api/c-globus-4.0.3/globus_gram_job_manager/html/globus_job_manager_rsl.html
[4] http://www.globus.org/api/c-globus-4.0.3/globus_gram_job_manager/html/globus_gram_job_manager_job_execution_environment.html

6. GRAM Error Codes

Table 3.1. Gram Error Codes

#	Name	Description
0		Success
1	GLOBUS_GRAM_PROTOCOL_ERROR_PARAMETER_NOT_SUPPORTED	one of the RSL parameters is not supported
2	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_REQUEST	the RSL length is greater than the maximum allowed
3	GLOBUS_GRAM_PROTOCOL_ERROR_NO_RESOURCES	an I/O operation failed
4	GLOBUS_GRAM_PROTOCOL_ERROR_BAD_DIRECTORY	jobmanager unable to set default to the directory requested
5	GLOBUS_GRAM_PROTOCOL_ERROR_EXECUTABLE_NOT_FOUND	the executable does not exist
6	GLOBUS_GRAM_PROTOCOL_ERROR_INSUFFICIENT_FUNDS	of an unused INSUFFICIENT_FUNDS
7	GLOBUS_GRAM_PROTOCOL_ERROR_AUTHORIZATION	authentication with the remote server failed
8	GLOBUS_GRAM_PROTOCOL_ERROR_USER_CANCELLED	the user cancelled the job
9	GLOBUS_GRAM_PROTOCOL_ERROR_SYSTEM_CANCELLED	the system cancelled the job
10	GLOBUS_GRAM_PROTOCOL_ERROR_PROTOCOL_FAILED	data transfer to the server failed
11	GLOBUS_GRAM_PROTOCOL_ERROR_STDIN_NOT_FOUND	the stdin file does not exist
12	GLOBUS_GRAM_PROTOCOL_ERROR_CONNECTION_FAILED	the connection to the server failed (check host and port)
13	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_MAXTIME	the provided RSL 'maxtime' value is not an integer
14	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_COUNT	the provided RSL 'count' value is not an integer
15	GLOBUS_GRAM_PROTOCOL_ERROR_NULL_SPECIFICATION_TREE	the job manager received an invalid RSL
16	GLOBUS_GRAM_PROTOCOL_ERROR_JM_FAILED_ALLOW_ATTACH	the job manager failed in allowing others to make contact
17	GLOBUS_GRAM_PROTOCOL_ERROR_JOB_EXECUTION_FAILED	the job failed when the job manager attempted to run it
18	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_PARADYN	an invalid paradyn was specified
19	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_JOBTYPE	the provided RSL 'jobtype' value is invalid
20	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_GRAM_MYJOB	the provided RSL 'myjob' value is invalid
21	GLOBUS_GRAM_PROTOCOL_ERROR_BAD_SCRIPT_ARG_FILE	the job manager failed to locate an internal script argument file

#	Name	Description
22	GLOBUS_GRAM_PROTOCOL_ERROR_ARG_FILE_CREATION_FAILED	the job manager failed to create an internal script argument file
23	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_JOBSTATE	the job manager detected an invalid job state
24	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_SCRIPT_REPLY	the job manager detected an invalid script response
25	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_SCRIPT_STATUS	the job manager detected an invalid script status
26	GLOBUS_GRAM_PROTOCOL_ERROR_JOBTYPE_NOT_SUPPORTED	the provided RSL 'jobtype' value is not supported by this job manager
27	GLOBUS_GRAM_PROTOCOL_ERROR_UNIMPLEMENTED	unused ERROR_UNIMPLEMENTED
28	GLOBUS_GRAM_PROTOCOL_ERROR_TEMP_SCRIPT_FILE_FAILED	the job manager failed to create an internal script submission file
29	GLOBUS_GRAM_PROTOCOL_ERROR_USER_PROXY_NOT_FOUND	the job manager cannot find the user proxy
30	GLOBUS_GRAM_PROTOCOL_ERROR_OPENING_USER_PROXY	the job manager failed to open the user proxy
31	GLOBUS_GRAM_PROTOCOL_ERROR_JOB_CANCEL_FAILED	the job manager failed to cancel the job as requested
32	GLOBUS_GRAM_PROTOCOL_ERROR_MALLOC_FAILED	system memory allocation failed
33	GLOBUS_GRAM_PROTOCOL_ERROR_DUCT_INIT_FAILED	the interprocess job communication initialization failed
34	GLOBUS_GRAM_PROTOCOL_ERROR_DUCT_LSP_FAILED	the interprocess job communication setup failed
35	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_HOST_COUNT	the provided RSL 'host count' value is invalid
36	GLOBUS_GRAM_PROTOCOL_ERROR_UNSUPPORTED_PARAMETER	one of the provided RSL parameters is unsupported
37	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_QUEUE	the provided RSL 'queue' parameter is invalid
38	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_PROJECT	the provided RSL 'project' parameter is invalid
39	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_EVALUATION_FAILED	the provided RSL string includes variables that could not be identified
40	GLOBUS_GRAM_PROTOCOL_ERROR_BAD_RSL_ENVIRONMENT	the provided RSL 'environment' parameter is invalid
41	GLOBUS_GRAM_PROTOCOL_ERROR_DRYRUN	the provided RSL 'dryrun' parameter is invalid
42	GLOBUS_GRAM_PROTOCOL_ERROR_ZERO_LENGTH_RSL	the provided RSL is invalid (an empty string)
43	GLOBUS_GRAM_PROTOCOL_ERROR_STAGING_EXECUTABLE	the job manager failed to stage the executable

#	Name	Description
44	GLOBUS_GRAM_PROTOCOL_ERROR_STAGING_STDIN	the job manager failed to stage the stdin file
45	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_JOB_MANAGER_TYPE	the requested job manager type is invalid
46	GLOBUS_GRAM_PROTOCOL_ERROR_BAD_ARGUMENTS	the provided RSL 'arguments' parameter is invalid
47	GLOBUS_GRAM_PROTOCOL_ERROR_GATEKEEPER_MISCONFIGURED	the gatekeeper failed to run the job manager
48	GLOBUS_GRAM_PROTOCOL_ERROR_BAD_RSL	the provided RSL could not be properly parsed
49	GLOBUS_GRAM_PROTOCOL_ERROR_VERSION_MISMATCH	there is a version mismatch between GRAM components
50	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_ARGUMENTS	the provided RSL 'arguments' parameter is invalid
51	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_COUNT	the provided RSL 'count' parameter is invalid
52	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_DIRECTORY	the provided RSL 'directory' parameter is invalid
53	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_DRYRUN	the provided RSL 'dryrun' parameter is invalid
54	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_ENVIRONMENT	the provided RSL 'environment' parameter is invalid
55	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_EXECUTABLE	the provided RSL 'executable' parameter is invalid
56	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_HOST_COUNT	the provided RSL 'host_count' parameter is invalid
57	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_JOBTYPE	the provided RSL 'jobtype' parameter is invalid
58	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_MAXTIME	the provided RSL 'maxtime' parameter is invalid
59	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_MYJOB	the provided RSL 'myjob' parameter is invalid
60	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_PARADYN	the provided RSL 'paradyn' parameter is invalid
61	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_PROJECT	the provided RSL 'project' parameter is invalid
62	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_QUEUE	the provided RSL 'queue' parameter is invalid
63	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_STDERR	the provided RSL 'stderr' parameter is invalid
64	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_STDIN	the provided RSL 'stdin' parameter is invalid
65	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_STDOUT	the provided RSL 'stdout' parameter is invalid

#	Name	Description
66	GLOBUS_GRAM_PROTOCOL_ERROR_OPENING_JOBMANAGER_SCRIPT	the job manager failed to locate an internal script
67	GLOBUS_GRAM_PROTOCOL_ERROR_CREATING_PIPE	the job manager failed on the system call pipe()
68	GLOBUS_GRAM_PROTOCOL_ERROR_FCNTL_FAILED	the job manager failed on the system call fcntl()
69	GLOBUS_GRAM_PROTOCOL_ERROR_STDOUT_FILENAME_FAILED	the job manager failed to create the temporary stdout filename
70	GLOBUS_GRAM_PROTOCOL_ERROR_STDERR_FILENAME_FAILED	the job manager failed to create the temporary stderr filename
71	GLOBUS_GRAM_PROTOCOL_ERROR_FORKING_EXECUTABLE	the job manager failed on the system call fork()
72	GLOBUS_GRAM_PROTOCOL_ERROR_EXECUTABLE_PERMISSIONS	the executable file permissions do not allow execution
73	GLOBUS_GRAM_PROTOCOL_ERROR_OPENING_STDOUT	the job manager failed to open stdout
74	GLOBUS_GRAM_PROTOCOL_ERROR_OPENING_STDERR	the job manager failed to open stderr
75	GLOBUS_GRAM_PROTOCOL_ERROR_OPENING_CACHE_USER_PROXY	the cache file could not be opened in order to relocate the user proxy
76	GLOBUS_GRAM_PROTOCOL_ERROR_OPENING_CACHE	cannot access cache files in ~/.globus/.gass_cache, check permissions, quota, and disk space
77	GLOBUS_GRAM_PROTOCOL_ERROR_INSERTING_CLIENT_CONTACT	the job manager failed to insert the contact in the client contact list
78	GLOBUS_GRAM_PROTOCOL_ERROR_CLIENT_CONTACT_NOT_FOUND	the contact was not found in the job manager's client contact list
79	GLOBUS_GRAM_PROTOCOL_ERROR_CONTACTING_JOB_MANAGER	connecting to the job manager failed. Possible reasons: job terminated, invalid job contact, network problems, ...
80	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_JOB_CONTACT	the syntax of the job contact is invalid
81	GLOBUS_GRAM_PROTOCOL_ERROR_UNDEFINED_EXE	the executable parameter in the RSL is undefined
82	GLOBUS_GRAM_PROTOCOL_ERROR_CONDOR_ARCH	the job manager service is misconfigured. condor arch undefined
83	GLOBUS_GRAM_PROTOCOL_ERROR_CONDOR_OS	the job manager service is misconfigured. condor os undefined
84	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_MIN_MEMORY	the provided RSL 'min_memory' parameter is invalid
85	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_MAX_MEMORY	the provided RSL 'max_memory' parameter is invalid
86	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_MIN_MEMORY	the RSL 'min_memory' value is not zero or greater

#	Name	Description
87	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_MAX_MEMORY	the RSL 'max_memory' value is not zero or greater
88	GLOBUS_GRAM_PROTOCOL_ERROR_HTTP_FRAME_FAILED	the creation of a HTTP message failed
89	GLOBUS_GRAM_PROTOCOL_ERROR_HTTP_UNFRAME_FAILED	parsing incoming HTTP message failed
90	GLOBUS_GRAM_PROTOCOL_ERROR_HTTP_PACK_FAILED	the packing of information into a HTTP message failed
91	GLOBUS_GRAM_PROTOCOL_ERROR_HTTP_UNPACK_FAILED	an incoming HTTP message did not contain the expected information
92	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_JOB_QUERY	the job manager does not support the service that the client requested
93	GLOBUS_GRAM_PROTOCOL_ERROR_SERVICE_NOT_FOUND	the gatekeeper failed to find the requested service
94	GLOBUS_GRAM_PROTOCOL_ERROR_JOB_QUERY_DENIAL	the jobmanager does not accept any new requests (shutting down)
95	GLOBUS_GRAM_PROTOCOL_ERROR_CALLBACK_NOT_FOUND	the client failed to close the listener associated with the callback URL
96	GLOBUS_GRAM_PROTOCOL_ERROR_BAD_GATEKEEPER_CONTACT	the gatekeeper contact cannot be parsed
97	GLOBUS_GRAM_PROTOCOL_ERROR_POE_NOT_FOUND	the job manager could not find the 'poe' command
98	GLOBUS_GRAM_PROTOCOL_ERROR_MPIRUN_NOT_FOUND	the job manager could not find the 'mpirun' command
99	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_START_TIME	the provided RSL 'start_time' parameter is invalid
100	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_RESERVATION_HANDLE	the provided RSL 'reservation_handle' parameter is invalid
101	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_MAX_WALL_TIME	the provided RSL 'max_wall_time' parameter is invalid
102	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_MAX_WALL_TIME	the RSL 'max_wall_time' value is not zero or greater
103	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_MAX_CPU_TIME	the provided RSL 'max_cpu_time' parameter is invalid
104	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_MAX_CPU_TIME	the RSL 'max_cpu_time' value is not zero or greater
105	GLOBUS_GRAM_PROTOCOL_ERROR_JM_SCRIPT_NOT_FOUND	the job manager is misconfigured, a scheduler script is missing
106	GLOBUS_GRAM_PROTOCOL_ERROR_JM_SCRIPT_PERMISSIONS	the job manager is misconfigured, a scheduler script has invalid permissions
107	GLOBUS_GRAM_PROTOCOL_ERROR_SIGNALING_JOB	the job manager failed to signal the job
108	GLOBUS_GRAM_PROTOCOL_ERROR_UNKNOWN_SIGNAL_TYPE	the job manager did not recognize/support the signal type

#	Name	Description
109	GLOBUS_GRAM_PROTOCOL_ERROR_GETTING_JOBID	the job manager failed to get the job id from the local scheduler
110	GLOBUS_GRAM_PROTOCOL_ERROR_WAITING_FOR_COMMIT	the job manager is waiting for a commit signal
111	GLOBUS_GRAM_PROTOCOL_ERROR_COMMIT_TIMED_OUT	the job manager timed out while waiting for a commit signal
112	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_SAVE_STATE	the provided RSL 'save_state' parameter is invalid
113	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_RESTART	the provided RSL 'restart' parameter is invalid
114	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_TWO_PHASE_COMMIT	the provided RSL 'two_phase' parameter is invalid
115	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_TWO_PHASE_COMMIT	the RSL 'two_phase' value is not zero or greater
116	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_STDOUT_POSITION	the provided RSL 'stdout_position' parameter is invalid
117	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_STDOUT_POSITION	the RSL 'stdout_position' value is not zero or greater
118	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_STDERR_POSITION	the provided RSL 'stderr_position' parameter is invalid
119	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_STDERR_POSITION	the RSL 'stderr_position' value is not zero or greater
120	GLOBUS_GRAM_PROTOCOL_ERROR_RESTART_FAILED	the job manager restart attempt failed
121	GLOBUS_GRAM_PROTOCOL_ERROR_NO_STATE_FILE	the job state file doesn't exist
122	GLOBUS_GRAM_PROTOCOL_ERROR_READING_STATE_FILE	could not read the job state file
123	GLOBUS_GRAM_PROTOCOL_ERROR_WRITING_STATE_FILE	could not write the job state file
124	GLOBUS_GRAM_PROTOCOL_ERROR_OLD_JM_ALIVE	old job manager is still alive
125	GLOBUS_GRAM_PROTOCOL_ERROR_TTL_EXPIRED	job manager state file TTL expired
126	GLOBUS_GRAM_PROTOCOL_ERROR_SUBMIT_UNKNOWN	it is unknown if the job was submitted
127	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_REMOTE_IO_URL	the provided RSL 'remote_io_url' parameter is invalid
128	GLOBUS_GRAM_PROTOCOL_ERROR_WRITING_REMOTE_IO_URL	could not write the remote io url file
129	GLOBUS_GRAM_PROTOCOL_ERROR_STDIO_SIZE	the standard output/error size is different
130	GLOBUS_GRAM_PROTOCOL_ERROR_JM_STOPPED	the job manager was sent a stop signal (job is still running)

#	Name	Description
131	GLOBUS_GRAM_PROTOCOL_ERROR_USER_PROXY_EXPIRED	the user proxy expired (job is still running)
132	GLOBUS_GRAM_PROTOCOL_ERROR_JOB_UNSUBMITTED	the job was not submitted by original jobmanager
133	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_COMMIT	the job manager is not waiting for that commit signal
134	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_SCHEDULER_SPECIFIC	the provided RSL scheduler specific parameter is invalid
135	GLOBUS_GRAM_PROTOCOL_ERROR_STAGE_IN_FAILED	the job manager could not stage in a file
136	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_SCRATCH	the scratch directory could not be created
137	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_CACHE	the provided 'gass_cache' parameter is invalid
138	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_SUBMIT_ATTRIBUTE	the RSL contains attributes which are not valid for job submission
139	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_STDIO_UPDATE_ATTRIBUTE	the RSL contains attributes which are not valid for stdio update
140	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_RESTART_ATTRIBUTE	the RSL contains attributes which are not valid for job restart
141	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_FILE_STAGE_IN	the provided RSL 'file_stage_in' parameter is invalid
142	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_FILE_STAGE_IN_SHARED	the provided RSL 'file_stage_in_shared' parameter is invalid
143	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_FILE_STAGE_OUT	the provided RSL 'file_stage_out' parameter is invalid
144	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_GASS_CACHE	the provided RSL 'gass_cache' parameter is invalid
145	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_FILE_CLEANUP	the provided RSL 'file_cleanup' parameter is invalid
146	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_SCRATCH	the provided RSL 'scratch_dir' parameter is invalid
147	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_SCHEDULER_SPECIFIC	the provided scheduler-specific RSL parameter is invalid
148	GLOBUS_GRAM_PROTOCOL_ERROR_UNDEFINED_ATTRIBUTE	a required RSL attribute was not defined in the RSL spec
149	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_CACHE	the gass_cache attribute points to an invalid cache directory
150	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_SAVE_STATE	the provided RSL 'save_state' parameter has an invalid value
151	GLOBUS_GRAM_PROTOCOL_ERROR_OPENING_VALIDATION_FILE	the job manager could not open the RSL attribute validation file
152	GLOBUS_GRAM_PROTOCOL_ERROR_READING_VALIDATION_FILE	the job manager could not read the RSL attribute validation file

#	Name	Description
153	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_PROXY_TIMEOUT	the provided RSL 'proxy_timeout' is invalid
154	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_PROXY_TIMEOUT	the RSL 'proxy_timeout' value is not greater than zero
155	GLOBUS_GRAM_PROTOCOL_ERROR_STAGE_OUT_FAILED	the job manager could not stage out a file
156	GLOBUS_GRAM_PROTOCOL_ERROR_JOB_CONTACT_NOT_FOUND	the job contact string does not match any which the job manager is handling
157	GLOBUS_GRAM_PROTOCOL_ERROR_DELEGATION_FAILED	proxy delegation failed
158	GLOBUS_GRAM_PROTOCOL_ERROR_LOCKING_STATE_LOCK_FILE	the job manager could not lock the state lock file
159	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_ATTR	an invalid globus_io_clientattr_t was used.
160	GLOBUS_GRAM_PROTOCOL_ERROR_NULL_PARAMETER	an null parameter was passed to the gram library
161	GLOBUS_GRAM_PROTOCOL_ERROR_STILL_STREAMING	the job manager is still streaming output
162	GLOBUS_GRAM_PROTOCOL_ERROR_AUTHORIZATION_DENIED	the authorization system denied the request
163	GLOBUS_GRAM_PROTOCOL_ERROR_AUTHORIZATION_SYSTEM_FAILURE	the authorization system reported a failure
164	GLOBUS_GRAM_PROTOCOL_ERROR_AUTHORIZATION_DENIED_JOB_ID	the authorization system denied the request - invalid job id
165	GLOBUS_GRAM_PROTOCOL_ERROR_AUTHORIZATION_DENIED_EXECUTABLE	the authorization system denied the request - not authorized to run the specified executable
166	GLOBUS_GRAM_PROTOCOL_ERROR_RSL_USER_NAME	the provided RSL 'user_name' parameter is invalid.
167	GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_USER_NAME	the job is not running in the account named by the 'user_name' parameter

7. Usage Scenarios

There is no content available at this time.

8. Troubleshooting

Globus Toolkit 2.x Error FAQ[5]

[5] http://www.globus.org/toolkit/docs/2.4/faq_errors.html

Chapter 4. GT 4.0 Pre WS GRAM: Admin Guide

1. Building and Installing GRAM

Gram will be installed during the normal installation of the GT4 and needs no extra steps.

2. Configuring GRAM

2.1. Configuration Files

GRAM uses the following configuration files and directories:

1. globus-gatekeeper.conf[1]
2. globus-job-manager.conf[2]
3. grid-services/[3]
4. /etc/grid-security/grid-mapfile[4]

1. globus-gatekeeper.conf

Here is the default globus-gatekeeper.conf:

```
-x509_cert_dir /etc/grid-security/certificates
-x509_user_cert /etc/grid-security/hostcert.pem
-x509_user_key /etc/grid-security/hostkey.pem
-gridmap /etc/grid-security/grid-mapfile
-home /usr/local/globus
-e libexec
-logfile var/globus-gatekeeper.log
-port 2119
-grid_services etc/grid-services
-inetd
```

- *-x509_cert_dir* specifies where to find the trusted CA certificates.
- *-x509_user_cert* specifies where to find the gatekeeper cert.
- *-x509_user_key* specifies where to find the gatekeeper key.
- *-gridmap* specifies where to find the grid-mapfile.
- *-home* specifies where the -e and -logfile variables are relative to. By default, this is your $GLOBUS_LOCATION.

[1] #s-gram-admin-configfile-gatekeeper
[2] #s-gram-admin-configfile-jobmanager
[3] #s-gram-admin-configfile-gridservices
[4] #s-gram-admin-configfile-gridmapfile

- *-e* specifies where to find scripts.
- *-logfile* specifies where the gatekeeper should put its log.
- *-port* specifies what port the gatekeeper will run on.
- *-grid_service* specifies where the directory which contains the configured jobmanagers is.
- *-inetd* specifies that the gatekeeper should exit after dealing with one request. That is because inetd will launch a copy of the gatekeeper for every request that comes in to the port in -port. If you are running a gatekeeper by hand, don't use this flag.

2. globus-job-manager.conf

Here is an example globus-job-manager.conf:

```
-home "/home/bacon/pkgs/globus-2.4"
-globus-gatekeeper-host bacon.mcs.anl.gov
-globus-gatekeeper-port 2119
-globus-gatekeeper-subject "/O=Grid/O=Globus/CN=bacon.mcs.anl.gov"
-globus-host-cputype i686
-globus-host-manufacturer pc
-globus-host-osname Linux
-globus-host-osversion 2.2.19-4.7mdk
-save-logfile on_error
-state-file-dir /home/bacon/pkgs/globus-2.4/tmp
-machine-type unknown
```

See Job Manager Configuration[5] for details. Note that the entries in this file are combined with the entries in $GLOBUS_LOCATION/etc/grid-services for any specific jobmanager.

3. grid-services/

$GLOBUS_LOCATION/etc/grid-services contains one file per configured jobmanager. The default jobmanager is contained in a file named "jobmanager". Actually this is a symbolic link to one of the jobmanager files located in the same directory that will be used as the default jobmanager. Here are the contents of an example file for a fork jobmanager:

```
stderr_log,local_cred - /home/bacon/pkgs/globus-2.4/libexec/globus-job-manager globus-job-
```

To install additional jobmanagers, you need to download the scheduler-specific jobmanager package from the download page[6].

4. /etc/grid-security/grid-mapfile

The grid-mapfile specifies the list of authorized users of this resource. Each entry is a pairing of a subject name and a local user account. The location of this file is specified in globus-gatekeeper.conf

2.2. Configure Inetd and Xinetd

While running globus-personal-gatekeeper as a user is a good test, you will want to configure your machine to run globus-gatekeeper as root, so that other people will be able to use your gatekeeper. If you just run the personal gatekeeper, you won't have authority to su to other user accounts. To setup a full gatekeeper, you will need to make the following modifications as root:

[5] #s-gram-admin-jobmanager-config
[6] http://www.globus.org/toolkit/downloads/development/

In /etc/services, add the service name "gsigatekeeper" to port 2119.

```
gsigatekeeper      2119/tcp                   # Globus Gatekeeper
```

Depending on whether your host is running inetd or xinetd, you will need to modify its configuration. If the directory /etc/xinetd.d/ exists, then your host is likely running xinetd. If the directory doesn't exist, your host is likely running inetd. Follow the appropriate instructions below according to what your host is running.

Inetd

For inetd, add the following entry, all on one line, to /etc/inetd.conf. Be sure to replace GLOBUS_LOCATION below with the actual value of $GLOBUS_LOCATION in your environment.

```
gsigatekeeper stream tcp nowait root
    /usr/bin/env env LD_LIBRARY_PATH=GLOBUS_LOCATION/lib
    GLOBUS_LOCATION/sbin/globus-gatekeeper
    -conf GLOBUS_LOCATION/etc/globus-gatekeeper.conf
```

This entry has changed from the entry provided for the gatekeeper in the Globus Toolkit 2.0 Administrator's Guide. The reason is that if you followed the instructions from the install section, you do not have a static gatekeeper. This requires you to set the LD_LIBRARY_PATH so that the gatekeeper can dynamically link against the libraries in $GLOBUS_LOCATION/lib. To accomplish the setting of the environment variable in inetd, we use /usr/bin/env (the location may vary on your system) to first set LD_LIBRARY_PATH, and then to call the gatekeeper itself.

The advantage of this setup is that when you apply a security update to your installation, the gatekeeper will pick it up dynamically without your having to rebuild it.

Xinetd

For xinetd, add a file called "globus-gatekeeper" to the /etc/xinetd.d/ directory that has the following contents. Be sure to replace GLOBUS_LOCATION below with the actual value of $GLOBUS_LOCATION in your environment.

```
service gsigatekeeper
{
   socket_type  = stream
   protocol     = tcp
   wait         = no
   user         = root
   env          = LD_LIBRARY_PATH=GLOBUS_LOCATION/lib
   server       = GLOBUS_LOCATION/sbin/globus-gatekeeper
   server_args  = -conf GLOBUS_LOCATION/etc/globus-gatekeeper.conf
   disable      = no
}
```

This entry has changed from the entry provided for the gatekeeper in the Globus Toolkit 2.0 Administrator's Guide. The reason is that if you followed the instructions from the install section, you do not have a static gatekeeper. This requires you to set the LD_LIBRARY_PATH so that the gatekeeper can dynamically link against the libraries in $GLOBUS_LOCATION/lib. To accomplish the setting of the environment variable in xinetd, we use the "env =" option to set LD_LIBRARY_PATH in the gatekeeper's environment.

The advantage of this setup is that when you apply a security update to your installation, the gatekeeper will pick it up dynamically without your having to rebuild it.

After you have added the globus-gatekeeper service to either inetd or xinetd, you will need to notify inetd (or xinetd) that its configuration file has changed. To do this, follow the instructions for the server you are running below.

Inetd

On most Linux systems, you can simply run `killall -HUP inetd` On other systems, the following has the same effect: ps aux | grep inetd | awk '{print $2;}' | xargs kill -HUP

Xinetd

On most linux systems, you can simply run `/etc/rc.d/init.d/xinetd restart`. Your system may also support the "reload" option. On other systems (or if that doesn't work), see man xinetd.

At this point, your gatekeeper will start up when a connection comes in to port 2119, and will keep a log of its activity in `$GLOBUS_LOCATION/var/globus-gatekeeper.log.` However, it does not yet have any authorization mapping between certificate subjects and usernames. You will need to create a file named `/etc/grid-security/grid-mapfile` which consists of single line entries listing a certificate subject and a username, like this:

```
"/O=Grid/O=Globus/OU=your.domain/CN=Your Name"    youruserid
```

You can check your subject name using `grid-cert-info -subject`. There are utility commands in $GLOBUS_LOCATION/sbin/grid-mapfile* for adding entries, removing entries, and checking consistency.

2.3. Advanced Configuration

Advanced configuration of GRAM consists of the following tasks:

1. Adding jobmanagers[7]

2. Adding trust to a new CA/removing trust from an old CA[8]

3. Starting your own CA[9]

1. Adding jobmanagers

For information about how to add a job manager for Condor, PBS, or LSF please look here[10]

2. Adding trust to a new CA/removing trust from an old CA

The set of trusted Certificate Authorities is contained in the /etc/grid-security/certificates directory. By default, that directory contains two entries. One, called 42864e48.0 is the public certificate of the Globus CA. The other, called 42864e48.signing_policy is the signing policy for the Globus CA certificate.

The name "42864e8" comes from the openssl -hash option. If you create your own Certificate Authority, you can use the command openssl x509 -in yourcert.pem -noout -hash to determine its hash value. You will need to place a copy of that public certificate, under the name hash.0 (where "hash" corresponds to the output of the openssl command) in the /etc/grid-security/certificates of every Toolkit installation which you want to trust certificates which your CA has signed. Additionally, you will have to create a hash.signing_policy file which contains the DN of your CA, as well as the namespace for which your CA signs.

Namespaces for CAs are designed to be unique. If you do establish your own CA, do not use the "/O=Grid/O=Globus" namespace. That is reserved for the Globus CA.

[7] #add-jobmanagers
[8] #add-ca
[9] #start-ca
[10] #s-gram-adding-jobmanager

Removing trust for a particular CA is as easy as deleting the two files which correspond to the CA. First, look for the .signing_policy which corresponds to the CA you want to remove. Then remove both the .signing_policy and .0 file that correspond to that hash.

3. Starting your own CA

There is a Globus package named Simple CA[11] which is designed to help you establish a CA for your test Grid.

3. Job Manager

The GRAM Job Manager program starts and monitors jobs on behalf of a GRAM client application. The job manager is typically started by the Gatekeeper program. It interfaces with a local scheduler to start jobs based on a job request RSL string.

3.1. Job Manager Setup

Job managers for Fork, PBS, LSF and Condor are included in the toolkit. But only the fork job manager is installed by default during a normal installation of the toolkit. The others must be installed separately if they are needed.

To install them from a source distribution, follow these steps:

1. go to the installer directory (e.g. gt4.0.3-all-source-installer)
2. `make gt4-gram-[pbs|lsf|condor]`
3. `make install`

Using PBS as the example, make sure the scheduler commands are in your path (qsub, qstat, pbsnodes). For PBS, another setup step is required to configure the remote shell for rsh access:

```
% cd $GLOBUS_LOCATION/setup/globus
% ./setup-globus-job-manager-pbs --remote-shell=rsh
```

The following links give extra information what parameters can be added to the setup scripts of the different scheduler adapters:

- Condor Job Manager Setup[12]
- PBS Job Manager Setup[13]
- LSF Job Manager Setup[14]

3.2. Job Manager Configuration

Job Manager Configuration[15]

[11] http://www.globus.org/toolkit/docs/4.0/security/simpleca/
[12] http://www.globus.org/api/c-globus-4.0.3/globus_gram_job_manager_setup_condor/html/main.html
[13] http://www.globus.org/api/c-globus-4.0.3/globus_gram_job_manager_setup_pbs/html/main.html
[14] http://www.globus.org/api/c-globus-4.0.3/globus_gram_job_manager_setup_lsf/html/main.html
[15] http://www.globus.org/api/c-globus-4.0.3/globus_gram_job_manager/html/globus_gram_job_manager_configuration.html

3.3. RSL Validation File Format

RSL Validation File Format[16]

3.4. Job Execution Environment

Job Execution Environment[17]

3.5. RSL attributes

RSL Attributes[18]

3.6. Adding job managers

The fork job manager scheduler will be installed during a normal installation of the toolkit and will be installed as the default job manager service (e.g. $GLOBUS_LOCATION/grid-services/jobmanager). Additional job manager scheduler packages installed will be installed using the convention "jobmanager-<scheduler-name>" (e.g. $GLOBUS_LOCATION/grid-services/jobmanager-pbs).

Information on how to install an additional job manager for Condor, PBS or LSF can be found here[19].

All job manager scheduler setup packages have the argument "-service-name <name>" in order to install a non-fork scheduler as the default job manager service. For example, this command will set the pbs scheduler as the default job manager service:

```
% setup-globus-job-manager-pbs -service-name jobmanager
```

If you need to alter the behavior of the job manager scheduler interface, or you want to create a new job manager scheduler interface for a scheduler that is not available, see this tutorial web page. The details of how to make a client submit to a non-default gatekeeper is covered in the user's guide section.

Note: If you wish to have your job manager report into your MDS, you need to install the appropriate GRAM Reporter setup package for your scheduler. The GRAM Reporter setup packages for each scheduler can be found on the download page[20].

The details of how to make a client submit to a non-default gatekeeper is covered in the user's guide section.

Note: If you wish to have your job manager report into your MDS, you need to install the appropriate GRAM Reporter setup package for your scheduler. The GRAM Reporter setup packages for each scheduler can be found on the download page[21].

4. Scheduler Event Generator / Job Manager Integration

Note: This feature is only available beginning from version 4.0.5 of the GT

[16] http://www.globus.org/api/c-globus-4.0.3/globus_gram_job_manager/html/globus_gram_job_manager_rsl_validation_file.html
[17] http://www.globus.org/api/c-globus-4.0.3/globus_gram_job_manager/html/globus_gram_job_manager_job_execution_environment.html
[18] http://www.globus.org/api/c-globus-4.0.3/globus_gram_job_manager/html/globus_job_manager_rsl.html
[19] #s-gram-admin-jobmanager-setup
[20] http://www.globus.org/toolkit/downloads/development/
[21] http://www.globus.org/toolkit/downloads/development/

4.1. Introduction

A new option available in 4.0.5 is a new method for the pre-ws GRAM Job Manager to monitor the jobs it submits to the local scheduler. After installing, you can configure a job manager to use the new event based method for monitoring jobs, instead of the script-based polling implementation.

This change consists of a few parts

- A new script `globus-job-manager-event-generator` which translates scheduler-specific log information to a general form which the job manager can parse. This script may need to be run as a privileged account in order to parse the log files, depending on the log permissions. This script MUST be running in order for Job Manager processes to receive job state change notifications from the scheduler.
- A new SEG module `globus_scheduler_event_generator_job_manager` which parses a log file to determine which job state changes occur for jobs being managed by a pre-WS GRAM Job Manager.
- Changes to the globus-gram-job-manager program to use the Scheduler Event Generator API to look for job state change events in a log file instead using scripts to query the scheduler state.

4.2. globus-job-manager-event-generator

The globus-job-manager-event-generator script creates a log of all scheduler events related to a particular scheduler instance. This script was created for two purposes

- To avoid requiring that all GRAM user's have the privileges to read the scheduler's log file. Users may not be allowed read access to the scheduler's log files on all sites. The Job Manager processes is run under the user's local account (as mapped in the gridmap file), it is this processes that will be updated for job status via the SEG log file instead of directly from the scheduler's log file.
- To provide a simple format for the scheduler event generator logs so that the job manager will be able to quickly recover state information if the job manager is terminated and restarted. Some scheduler logs are difficult to parse, or inefficient for seeking to a particular timestamp (as is necessary for recovering job state change information). The data written by this script is easily locatably by date, and it is simple to remove old job information without compromising current job manager execution.

One instance of the globus-job-manager-event-generator must be running for each scheduler type to be implemented using the Scheduler Event Generator interface to receive job state changes. This program is located in the sbin subdirectory of the GLOBUS_LOCATION. The typical command line for this program is `$GLOBUS_LOCATION/sbin/globus-job-manager-event-generator -s SCHEDULER_TYPE`, where SCHEDULER_TYPE is the scheduler name of the Scheduler Event Generator module which should be used to generate events (lsf, condor, pbs).

For example, to start the event generator program to monitor an LSF batch system:

```
$GLOBUS_LOCATION/sbin/globus-job-manager-event-generator -s lsf
```

NOTE: if the globus-job-manager-event-generator is not running, no job state changes will be sent from any job manager program which is configured to use the Scheduler Event Generator.

4.3. Job Manager Configuration

By default, the job manager is configured to use the pre-WS GRAM script-based polling method. A new command line option (`-seg`) was added to the globus-job-manager program to enable using the Scheduler Event Generator-driven job state change notifications.

There are two ways to configure the job manager to use the scheduler event generator: globally, in the $GLOBUS_LOCATION/etc/globus-job-manager.conf file, or on a per-service basis in the service entry file in the `$GLOBUS_LOCATION/etc/grid-services` directory.

4.3.1. Global Job Manager Configuration

To enable using the Scheduler Event Generator interface for all Job Managers started from a particular GLOBUS_LOCATION, add a line containing the string

```
-seg
```

to the file $GLOBUS_LOCATION/etc/globus-job-manager.conf.

EXAMPLE $GLOBUS_LOCATION/etc/globus-job-manager.conf:

```
-home "/opt/globus"
-globus-gatekeeper-host globus.yourdomain.org
-globus-gatekeeper-port 2119
-globus-gatekeeper-subject "/O=Grid/OU=Your Organization/CN=host/globus.yourdomain.org"
-globus-host-cputype i686
-globus-host-manufacturer pc
-globus-host-osname Linux
-globus-host-osversion 2.6.10
-save-logfile on_error
-state-file-dir /opt/globus/tmp/gram_job_state
-machine-type unknown
-seg
```

4.3.2. Scheduler-specific Job Manager Configuration

To enable using the Scheduler Event Generator interface for a particular Job Manager, add the string -seg to the end of the line in the service's file in the `$GLOBUS_LOCATION/etc/grid-services` directory.

EXAMPLE $GLOBUS_LOCATION/etc/grid-services/jobmanager-lsf:

```
stderr_log,local_cred - /opt/globus/libexec/globus-job-manager globus-job-manager -conf /o
```

No SEG with Job Manager fork

The Job Manager fork does not support using the Scheduler Event Generator. If the -seg option is passed to a fork Job Manager, it will be ignored.

4.4. globus-job-manager-event-generator Configuration

The globus-job-manager-event-generator program requires that the globus_job_manager_event_generator setup package be installed and run. This setup package creates the `$GLOBUS_LOCATION/etc/globus-job-manager-seg.conf` file and initializes a directory to use for the scheduler logs.

By default, this setup script will create a configuration entry and directory for each scheduler installed on the system. For each scheduler to be handled by the globus-job-manager-event-generator program, there must be an entry in the file in the pattern:

```
<SCHEDULER_TYPE>_log_path=<PATH>
```

The two variable substitutions for this pattern are

SCHEDULER_TYPE

Must match the name of the scheduler-event-generator module for the scheduler (supported with GT 4.0 are lsf, condor, and pbs).

PATH

A path to a directory which must be writable by the account which will run the `globus-job-manager-event-generator` program for the SCHEDULER_TYPE, and world-readable (or readable for a group which contains all users which will run jobs via GRAM on that system). Each directory specified in the configuration file must be unique, or behavior is undefined.

EXAMPLE $GLOBUS_LOCATION/etc/globus-job-manger-seg.conf:

```
lsf_log_path=/opt/globus/var/globus-job-manager-seg-lsf
pbs_log_path=/opt/globus/var/globus-job-manager-seg-pbs
```

In this example, pbs and lsf schedulers are configured to use distinct subdirectories of the `/opt/globus/var/` directory.

NOTE: For best performance, the log paths should be persistent across system reboots and mounted locally (non-networked).

NOTE: If a scheduler is added after the configuration step is done, administrator must rerun the setup package's script (`$GLOBUS_LOCATION/setup/globus/setup-seg-job-manager.pl`) or modify the configuration file and create the required directory with appropriate permissions.

4.5. Running the globus-job-manager-event-generator

The globus-job-manager-event-generator must be running when jobs are submitted to the Job Manager if job state changes are to be detected. One instance of the `globus-job-manager-event-generator` program must be running for each scheduler type which is handled by a Job Manager and configured to use the Scheduler Event Generator interface.

The command line for the globus-job-manager-event-generator program is `globus-job-manager-event-generator -s SCHEDULER_TYPE`. The SCHEDULER_TYPE should match the pattern of a log_path entry in the `$GLOBUS_LOCATION/etc/globus-job-manager-seg.conf` as described above.

NOTE: Remember, if your scheduler logs have restrictive permissions, then this script must be run by an account which has privileges to read those files.

NOTE: Old log files created by the globus-job-manager-event-generator script may be deleted if the administrator is certain that there are no jobs which will restart and require the old information. The names of the log files correspond to the dates when the events occurred. If there is at least one log file in the directory, then when the globus-job-manager-event-generator is restarted, it will resume logging from the timestamp of the newest event in that log file.

4.6. Troubleshooting the globus-job-manager-event-generator

PROBLEM: The globus-job-manager-event-generator program terminates immediately with the output:

```
Error: SCHEDULER not configured
```

SOLUTION 1: Make sure that you specified the correct name for the SCHEDULER module on the command line to the `globus-job-manager-event-generator` program

SOLUTION 2: There is no entry for lsf in the `$GLOBUS_LOCATION/etc/globus-job-manager-seg.conf` file. See the section on globus-job-manager-event-generator Configuration.

PROBLEM: The globus-job-manager-event-generator program terminates immediately with the output:

```
Fault: globus_xio: Operation was canceled
```

SOLUTION: The scheduler module selected on the command line could not be loaded by the Globus Scheduler Event Generator. Check that the name is correct, the module is installed, and the setup script for that module has been run.

PROBLEM: The Job Manager never receives any events from the scheduler.

SOLUTION 1: Verify that the directory specified in the `$GLOBUS_LOCATION/etc/globus-job-manager-seg.conf` for the scheduler exists, is writable by the account running the `globus-job-manager-event-generator` and is readable by the user account running the job manager.

SOLUTION 2: Verify that the globus-job-manager-event-generator program is running.

SOLUTION 3: Verify that the globus-job-manager-event-generator program has permissions to read the scheduler logs. To help diagnose this, run (as the account you wish to run the globus-job-manager-event-generator as) the command

```
$GLOBUS_LOCATION/libexec/globus-scheduler-event-generator -s <SCHEDULER_TYPE> -t 1
```

You should see events printed to the stdout of that process if it is working correctly.

5. Audit Logging

Note: This feature is only available beginning from version 4.0.5 of the GT

5.1. Overview

GRAM provides mechanisms to provide access to audit and accounting information associated with jobs that are submitted to local resource manager (LRM) like PBS, LSF, Condor by GRAM. GRAM is not a local resource manager but rather a protocol engine for communicating with a range of different local resource managers using a standard message format. In some scenarios it is desirable to get an overview over the usage of the underlying LRM like

- What kind of jobs had been submitted via GRAM?
- How long did the processing of a job take?
- How many jobs had been submitted by user X?

The following three usecases give a better overview about the meaning and purpose of auditing and accounting:

1. **Group Access**. A grid resource provider allows a remoteservice (e.g., a gateway or portal) to submit jobs on behalf of multiple users. The grid resource provider only obtains information about the identity of the remote submitting service and thus does not know the identity of the users for which the grid jobs are submitted. This group access is allowed under the condition that the remote service store audit information so that, if and when needed, the grid resource provider can request and obtain information to track a specific job back to an individual user.
2. **Query Job Accounting**. A client that submits a job needs to be able to obtain, after the job has completed, information about the resources consumed by that job. In portal and gateway environments where many users submit

many jobs against a single allocation, this per-job accounting information is needed soon after the job completes so that client-side accounting can be updated. Accounting information is sensitive and thus should only be released to authorized parties.

3. **Auditing**. In a distributed multi-site environment, it can be necessary to investigate various forms of suspected intrusion and abuse. In such cases, we may need to access an audit trail of the actions performed by a service. When accessing this audit trail, it will frequently be important to be able to relate specific actions to the user.

The audit record of each job is stored in a DBMS and contains

- *job_grid_id*: String representation of the resource EPR
- *local_job_id*: Job/process id generated by the scheduler
- *subject_name*: Distinguished name (DN) of the user
- *username*: Local username
- *idempotence_id*: Job id generated on the client-side
- *creation_time*: Date when the job resource is created
- *queued_time*: Date when the job is submitted to the scheduler
- *stage_in_grid_id*: String representation of the stageIn-EPR (RFT)
- *stage_out_grid_id*: String representation of the stageOut-EPR (RFT)
- *clean_up_grid_id*: String representation of the cleanUp-EPR (RFT)
- *globus_toolkit_version*: Version of the server-side GT
- *resource_manager_type*: Type of the resource manager (Fork, Condor, ...)
- *job_description*: Complete job description document
- *success_flag*: Flag that shows whether the job failed or finished successfully
- *finished_flag*: Flag that shows whether the job is already fully processed or still in progress

While audit and accounting records may be generated and stored by different entities in different contexts, we assume here that audit records are generated by the GRAM service itself and accounting records by the LRM to which the GRAM service submits jobs. Accounting records could contain all information about the duration and the resource-usage of a job. Audit records are stored in a database indexed by a Grid job identifier (GJID), while accounting records are maintained by the LRM indexed by a local job identifier (JID).

GRAM Service GJID creation

The GRAM2 service returns a "job contact" string that is used to control the job. The job contact is by default an acceptable GJID format, so it is the GJID. The GRAM2 client and service does not need to convert it in any way.

To connect the two sets of records, both audit and accounting records, we require that GRAM records theJID in each audit record that it generates. It is then straightforward for an audit service to respond to requests like 'give me the charge of the job with JID x' by first selecting matching record(s) from the audit table and then using the local JID(s) to join to the accounting table of the LRM to access relevant accounting record(s).

We propose a Web Service interface for accessing audit and accounting information. OGSA-DAI is a WSRF service that can create a single virtual database from two or more remote databases. In the future, other per-job information like job performance data could be stored using the GJID or local JID as an index, and then made available in the same virtual database. The rest of this chapter focuses on how to configure GRAM to enable Audit-Logging. A case study for TeraGrid can be read here[22]

OGSA-DAI is available here: http://www.globus.org/toolkit/docs/4.0/techpreview/ogsadai/[23]

Supported DBMSs: Currently schemas for MySQL and PostgreSQL are available to store the audit records that contain the information mentioned above. It shouldn't be too difficult to create schemas for other relational DBMSs.

The audit record of a job is stored at the end of the processing cycle of a job, either when it's completely processed or failed.

5.2. Functionality

Audit logging in Pre WS GRAM is realized in the following way:

- The job manager writes a file to disk for each job. This file contains the audit record. The format of an audit record file that is logged to the database is a comma-separated list of double-quoted strings.
- The audit records are not inserted into the GRAM audit database directly. The Job Manager will, at final job termination (FAILED or DONE state) write a record to a unique file in a directory specified by a configuration file. These audit files must be uploaded by a program which can be called manually or be run periodically as a cron job. The program is a perl script and is located in `${GLOBUS_LOCATION}/libexec/globus-gram-audit` and creates audit records in the configured database from the user audit files. Once the record is uploaded, the program will remove the audit file.

  Here's an example on how a crontab entry must look like in order to run the script every 15 minutes:

  ```
  0,15,30,45 * * * * ${GLOBUS_LOCATION}/libexec/globus-gram-audit
  ```

  The script gets the necessary parameters to connect to the database from the configuration file `${GLOBUS_LOCATION}/etc/globus-job-manager-audit.conf`, which is described below.

  The reason for Pre-WS GRAM to use a different method than WS GRAM, writing audit files instead of direct DB inserts, is to have a more trusted entity writing to the database, instead of requiring all Pre-WS GRAM users with the ability to do so.

5.3. Configuration

Audit logging is turned off by default. To enable audit logging, the following must be done:

1. Add the following line to `${GLOBUS_LOCATION}/etc/globus-job-manager.conf`:

   ```
   -audit-directory <audit record directory>
   ```

2. Create the audit record directory specified in the above configuration file that has the permissions `rws-wsrwx`
3. Add the following lines to the Log4j configuration in `$GLOBUS_LOCATION/etc/log4j.properties` to enable audit logging:

[22] http://www.teragridforum.org/mediawiki/index.php?title=GRAM4_Audit
[23] http://www.globus.org/toolkit/docs/4.0/techpreview/ogsadai/

```
# GRAM AUDIT
log4j.category.org.globus.exec.service.exec.StateMachine.audit=DEBUG, AUDIT
log4j.appender.AUDIT=org.globus.exec.utils.audit.AuditDatabaseAppender
log4j.appender.AUDIT.layout=org.apache.log4j.PatternLayout
log4j.additivity.org.globus.exec.service.exec.StateMachine.audit=false
```

4. Edit `${GLOBUS_LOCATION}/etc/globus-job-manager-audit.conf` to have the correct database connection parameters. Example:

   ```
   DRIVERCLASS:com.mysql.jdbc.Driver
   USERNAME:john
   PASSWORD:foo
   URL:jdbc:mysql://myhost/auditDatabase
   ```

5. Follow the instructions on the WS-GRAM pages[24] to create the database.

5.4. Dependencies

Database

Currently database schemas for PostgreSQL and MySQL are provided to create the audit database table.

6. Testing GRAM

First launch a gatekeeper by running the following (as yourself, not root):

```
% grid-proxy-init -debug -verify
% globus-personal-gatekeeper -start
```

This command will output a contact string like `hostname:4589:/O=Grid/O=Globus/CN=Your Name`. Substitute that contact string for <contact> in the following command:

```
% globus-job-run <contact> /bin/date
```

You should see the current date and time. At this point you can stop the personal gatekeeper and destroy your proxy with:

```
% globus-personal-gatekeeper -killall
% grid-proxy-destroy
```

Please note that the above instructions are just for testing, and do not install a fully functioning gatekeeper on your machine for everyone to use. Installing a system-level gatekeeper for everyone to use will be covered in the configuration section [25]of this guide.

[24] ../wsgram/admin-index.html#s-wsgram-Interface_Config_Frag-audit-logging-config
[25] #s-gram-admin-configuring

7. Usage statistics collection by the Globus Alliance

No usage statistic package is sent after the completion of a job like it's done in WS-GRAM (see here[26]).

[26] ../wsgram/admin-index.html#s-wsgram-admin-usage

Chapter 5. GT 4.0 Pre WS GRAM: Developer Guide

1. Introduction

There is no content available at this time.

2. Public interface

2.1. Scheduler Event Generator

- General information[1]
- SEG protocol[2]
- SEG API[3]
- SEG Tests[4]

2.2. Client

Client API[5]

2.3. GRAM Proctocol

- General information[6]
- GRAM Protocol Functions[7]
- Job States[8]
- Signals[9]
- GRAM Errors[10]
- GRAM Protocol Message Format[11]

[1] http://www.globus.org/api/c-globus-4.0.3/globus_scheduler_event_generator/html/main.html
[2] http://www.globus.org/api/c-globus-4.0.3/globus_scheduler_event_generator/html/seg_protocol.html
[3] http://www.globus.org/api/c-globus-4.0.3/globus_scheduler_event_generator/html/group__seg__api.html
[4] http://www.globus.org/api/c-globus-4.0.3/globus_scheduler_event_generator_test/html/main.html
[5] http://www.globus.org/api/c-globus-4.0.3/globus_gram_client/html/main.html
[6] http://www.globus.org/api/c-globus-4.0.3/globus_gram_protocol/html/main.html
[7] http://www.globus.org/api/c-globus-4.0.3/globus_gram_protocol/html/group__globus__gram__protocol__functions.html
[8] http://www.globus.org/api/c-globus-4.0.3/globus_gram_protocol/html/group__globus__gram__protocol__job__state.html
[9] http://www.globus.org/api/c-globus-4.0.3/globus_gram_protocol/html/group__globus__gram__protocol__job__signal.html
[10] http://www.globus.org/api/c-globus-4.0.3/globus_gram_protocol/html/group__globus__gram__protocol__error.html
[11] http://www.globus.org/api/c-globus-4.0.3/globus_gram_protocol/html/globus_gram_protocol.html

3. Tutorials

3.1. GRAM Job Manager Scheduler Tutorial

This tutorial describes the steps needed to build a GRAM Job Manager Scheduler interface package. The audience for this tutorial is a person interested in adding support for a new scheduler interface to GRAM. This tutorial will assume some familiarty with GTP, autoconf, automake, and Perl. As a reference point, this tutorial will refer to the code in the LSF Job Manager package.

3.1.1. Writing a Scheduler Interface

This section deals with writing the perl module which implements the interface between the GRAM job manager and the local scheduler. Consult the Job Manager Scheduler Interface section[12] of this manual for a more detailed reference on the Perl modules which are used here.

The scheduler interface is implemented as a Perl module which is a subclass of the Globus::GRAM::JobManager module. Its name must match the scheduler type string used when the service is installed. For the LSF scheduler, the name is *lsf*, so the module name is *Globus::GRAM::JobManager::lsf* and it is stored in the file `lsf.pm`. Though there are several methods in the JobManager interface, they only ones which absolutely need to be implemented in a scheduler module are submit, poll, cancel.

We'll begin by looking at the start of the lsf source module, lsf.in (the transformation to lsf.pm happens when the setup script is run. To begin the script, we import the GRAM support modules into the scheduler module's namespace, declare the module's namespace, and declare this module as a subclass of the Globus::GRAM::JobManager module. All scheduler packages will need to do this, substituting the name of the scheduler type being implemented where we see *lsf* below.

```
use Globus::GRAM::Error;
use Globus::GRAM::JobState;
use Globus::GRAM::JobManager;
use Globus::Core::Paths;

...

package Globus::GRAM::JobManager::lsf;

@ISA = qw(Globus::GRAM::JobManager);
```

Next, we declare any system-specifc values which will be substituted when the setup package scripts are run. In the LSF case, we need the know the paths to a few programs which interact with the scheduler:

```
BEGIN
{
    $mpirun = '@MPIRUN@';
    $bsub   = '@BSUB@';
    $bjobs  = '@BJOBS@';
    $bkill  = '@BKILL@';
```

[12] #

```
}
```

The values surrounded by the at-sign (such as `@MPIRUN`) will be replaced by with the path to the named programs by the find-lsf-tools script described below.

3.1.1.1. Writing a constructor

For scheduler interfaces which need to setup some data before calling their other methods, they can overload the `new` method which acts as a constructor. Scheduler scripts which don't need any per-instance initialization will not need to provide a constructor, the Globus::GRAM::JobManager constructor will do the job.

If you do need to overloaded this method, be sure to call the JobManager module's constructor to allow it to do its initialization, as in this example:

```
sub new
{
    my $proto = shift;
    my $class = ref($proto) || $proto;
    my $self = $class->SUPER::new(@_);

## Insert scheduler-specific startup code here
return $self;
}
```

The job interface methods are called with only one argument, the scheduler object itself. That object contains the a Globus::GRAM::JobDescription object (`$self->{JobDescription}`) which includes the values from the RSL string associated with the request, as well as a few extra values:

job_id

The string returned as the value of JOB_ID in the return hash from submit. This won't be present for methods called before the job is submitted.

uniq_id

A string associated with this job request by the job manager program. It will be unique for all jobs on a host for all time.

cache_tag

The GASS cache tag related to this job submission. Files in the cache with this tag will be cleaned by the cleanup_cache() method.

Now, let's look at the methods which will interface to the scheduler.

3.1.1.2. Submitting Jobs

All scheduler modules must implement the submit method. This method is called when the job manager wishes to submit the job to the scheduler. The information in the original job request RSL string is available to the scheduler interface through the `JobDescription` data member of it's hash.

For most schedulers, this is the longest method to be implemented, as it must decide what to do with the job description, and convert them to something which the scheduler can understand.

We'll look at some of the steps in the LSF manager code to see how the scheduler interface is implemented.

In the beginning of the submit method, we'll get our parameters and look up the job description in the manager-specific object:

```
sub submit
{
    my $self = shift;
    my $description = $self->{JobDescription};
```

Then we will check for values of the job parameters that we will be handling. For example, this is how we check for a valid job type in the LSF scheduler interface:

```
if(defined($description->jobtype())
{
    if($description->jobtype !~ /^(mpi|single|multiple)$/)
    {
        return Globus::GRAM::Error::JOBTYPE_NOT_SUPPORTED;
    }
    elsif($description->jobtype() eq 'mpi' && $mpirun eq "no")
    {
        return Globus::GRAM::Error::JOBTYPE_NOT_SUPPORTED;
    }
}
```

The lsf module supports most of the core RSL attributes, so it does more processing to determine what to do with the values in the job description.

Once we've inspected the JobDescription we'll know what we need to tell the scheduler about so that it'll start the job properly. For LSF, we will construct a job description script and pass that to the `bsub` command. This script is a bourne shell script with some special comments which LSF uses to decide what constraints to use when scheduling the job.

First, we'll open the new file, and write the file header:

```
$lsf_job_script = new IO::File($lsf_job_script_name, '>');

$lsf_job_script->print<<EOF;
#! /bin/sh
#
# LSF batch job script built by Globus Job Manager
#
EOF
```

Then, we'll add some special comments to pass job constraints to LSF:

```
if(defined($queue))
{
    $lsf_job_script->print("#BSUB -q $queue\n");
}
if(defined($description->project()))
{
    $lsf_job_script->print("#BSUB -P " . $description->project() . "\n");
}
```

Before we start the executable in the LSF job description script, we will quote and escape the job's arguments so that they will be passed to the application as they were in the job submission RSL string:

At the end of the job description script, we actually run the executable named in the JobDescription. For LSF, we support a few different job types which require different startup commands. Here, we will quote and escape the strings in the argument list so that the values of the arguments will be identical to those in the initial job request string. For this Bourne-shell syntax script, we will double-quote each argument, and escaping the backslash (\), dollar-sign ($), double-quote ("), and single-quote (') characters. We will use this new string later in the script.

```
@arguments = $description->arguments();

foreach(@arguments)
{
    if(ref($_))
    {
         return Globus::GRAM::Error::RSL_ARGUMENTS;
    }
}
if($arguments[0])
{
    foreach(@arguments)
    {
         $_ =~ s/\\/\\\\/g;
         $_ =~ s/\$/\\\$/g;
         $_ =~ s"/\\\"/g;
         $_ =~ s/`/\\\`/g;

         $args .= '"' . $_ . '" ';
    }
}
else
{
    $args = "";
}
```

To end the LSF job description script, we will write the command line of the executable to the script. Depending on the job type of this submission, we will need to start either one or more instances of the executable, or the mpirun program which will start the job with the executable count in the `JobDescription`:

```
if($description->jobtype() eq "mpi")
{
    $lsf_job_script->print("$mpirun -np " . $description->count() . " ");

    $lsf_job_script->print($description->executable()
                            . " $args \n");
}
elsif($description->jobtype() eq 'multiple')
{
    for(my $i = 0; $i < $description->count(); $i++)
    {
        $lsf_job_script->print($description->executable() . " $args &\n");
    }
    $lsf_job_script->print("wait\n");
}
else
{
    $lsf_job_script->print($description->executable() . " $args\n");
}
```

Next, we submit the job to the scheduler. Be sure to close the script file before trying to redirect it into the submit command, or some of the script file may be buffered and things will fail in strange ways!

When the submission command returns, we check its output for the scheduler-specific job identifier. We will use this value to be able to poll or cancel the job.

The return value of the script should be either a GRAM error object or a reference to a hash of values. The Globus::GRAM::JobManager documentation lists the valid keys to that hash. For the submit method, we'll return the job identifier as the value of JOB_ID in the hash. If the scheduler returned a job status result, we could return that as well. LSF does not, so we'll just check for the job ID and return it, or if the job fails, we'll return an error object:

```
    $lsf_job_script->close();

    $job_id = (grep(/is submitted/,
                   split(/\n/, `$bsub < $lsf_job_script_name`)))[0];
    if($? == 0)
    {
        $job_id =~ m/<([^>]*)>/;
        $job_id = $1;

        return { JOB_ID => $job_id };
    }

    return Globus::GRAM::Error::INVALID_SCRIPT_REPLY;
}
```

That finishes the submit method. Most of the functionality for the scheduler interface is now written. We just have a few more (much shorter) methods to implement.

3.1.1.3. Polling Jobs

All scheduler modules must also implement the poll method. The purpose of this method is to check for updates of a job's status, for example, to see if a job has finished.

When this method is called, we'll get the job ID (which we returned from the submit method above) as well as the original job request information in the object's JobDescription. In the LSF script, we'll pass the job ID to the `bjobs` program, and that will return the job's status information. We'll compare the status field from the `bjobs` output to see what job state we should return.

If the job fails, and there is a way to determine that from the scheduler, then the script should return in its hash both

```
JOB_STATE => Globus::GRAM::JobState::FAILED
```

and

```
ERROR => Globus::GRAM::Error::<ERROR_TYPE>->value
```

Here's an excerpt from the LSF scheduler module implementation:

```
sub poll
{
    my $self = shift;
    my $description = $self->{JobDescription};
    my $job_id = $description->jobid();
    my $state;
    my $status_line;

    $self->log("polling job $job_id");

    # Get first line matching job id
    $_ = (grep(/$job_id/, `$bjobs $job_id 2>/dev/null`))[0];

    # Get 3th field (status)
    $_ = (split(/\s+/))[2];

    if(/PEND/)
    {
        $state = Globus::GRAM::JobState::PENDING;
    }
    elsif(/USUSP|SSUSP|PSUSP/)
    {
        $state = Globus::GRAM::JobState::SUSPENDED
    }
    ...
    return {JOB_STATE => $state};
}
```

3.1.1.4. Cancelling Jobs

All scheduler modules must also implement the cancel method. The purpose of this method is to cancel a running job.

As with the `poll` method described above, this method will be given the job ID as part of the JobDescription object held by the manager object. If the scheduler interface provides feedback that the job was cancelled successfully, then we can return a JOB_STATE change to the FAILED state. Otherwise we can return an empty hash reference, and let the poll method return the state change next time it is called.

To process a cancel in the LSF case, we will run the bkill command with the job ID.

```
sub cancel
{
    my $self = shift;
    my $description = $self->{JobDescription};
    my $job_id = $description->jobid();

    $self->log("cancel job $job_id");

    system("$bkill $job_id >/dev/null 2>/dev/null");

    if($? == 0)
    {
        return { JOB_STATE => Globus::GRAM::JobState::FAILED }
    }
    return Globus::GRAM::Error::JOB_CANCEL_FAILED;

}
```

3.1.1.5. End of the script

It is required that all perl modules return a non-zero value when they are parsed. To do this, make sure the last line of your module consists of:

```
1;
```

3.1.2. Setting up a Scheduler

Once we've written the job manager script, we need to get it installed so that the gatekeeper will be able to run our new service. We do this by writing a setup script. For LSF, we will write the script `setup-globus-job-manager-lsf.pl`, which we will list in the LSF package as the **Post_Install_Program**.

To set up the Gatekeeper service, our LSF setup script does the following:

1. Perform system-specific configuration.
2. Install the GRAM scheduler Perl module and register as a gatekeeper service.
3. **(Optional)** Install an RSL validation file defining extra scheduler-specific RSL attributes which the scheduler interface will support.
4. Update the GPT metadata to indicate that the job manager service has been set up.

3.1.2.1. System-Specific Configuration

First, our scheduler setup script probes for any system-specific information needed to interface with the local scheduler. For example, the LSF scheduler uses the `mpirun`, `bsub`, `bqueues`, `bjobs`, and `bkill` commands to submit, poll, and cancel jobs. We'll assume that the administrator who is installing the package has these commands in their path. We'll use an autoconf script to locate the executable paths for these commands and substitute them into our scheduler

Perl module. In the LSF package, we have the `find-lsf-tools` script, which is generated during bootstrap by autoconf from the `find-lsf-tools.in` file:

```
## Required Prolog

AC_REVISION($Revision: 1.1 $)
AC_INIT(lsf.in)

# checking for the GLOBUS_LOCATION

if test "x$GLOBUS_LOCATION" = "x"; then
    echo "ERROR Please specify GLOBUS_LOCATION" >&2
    exit 1
fi

...

## Check for optional tools, warn if not found

AC_PATH_PROG(MPIRUN, mpirun, no)
if test "$MPIRUN" = "no" ; then
    AC_MSG_WARN([Cannot locate mpirun])
fi

...

## Check for required tools, error if not found

AC_PATH_PROG(BSUB, bsub, no)
if test "$BSUB" = "no" ; then
    AC_MSG_ERROR([Cannot locate bsub])
fi

...

## Required epilog - update scheduler specific module

prefix='$(GLOBUS_LOCATION)'
exec_prefix='$(GLOBUS_LOCATION)'
libexecdir=${prefix}/libexec

AC_OUTPUT(
    lsf.pm:lsf.in
)
```

If this script exits with a non-zero error code, then the setup script propagates the error to the caller and exits without installing the service.

3.1.2.2. Registering as a Gatekeeper Service

Next, the setup script installs it's perl module into the perl library directory and registers an entry in the Globus Gatekeeper's service directory. The program globus-job-manager-service[13] (distributed in the job manager program setup package) performs both of these tasks. When run, it expects the scheduler perl module to be located in the `$GLOBUS_LOCATION/setup/globus` directory.

```
$libexecdir/globus-job-manager-service -add -m lsf -s jobmanager-lsf;
```

3.1.2.3. Installing an RSL Validation File

If the scheduler script implements RSL attributes which are not part of the core set supported by the job manager, it must publish them in the job manager's data directory. If the scheduler script wants to set some default values of RSL attributes, it may also set those as the default values in the validation file.

The format of the validation file is described in th RSL Validation File Format[14] section of the documentation. The validation file must be named *scheduler-type*.rvf and installed in the `$GLOBUS_LOCATION/share/globus_gram_job_manager` directory.

In the LSF setup script, we check the list of queues supported by the local LSF installation, and add a section of acceptable values for the *queue* RSL attribute:

```
open(VALIDATION_FILE,
     ">$ENV{GLOBUS_LOCATION}/share/globus_gram_job_manager/lsf.rvf");

# Customize validation file with queue info
open(BQUEUES, "bqueues -w |");

# discard header
$_ = <BQUEUES>;
my @queues = ();

while(<BQUEUES>)
{
    chomp;

    $_ =~ m/^(\S+)/;

    push(@queues, $1);
}
close(BQUEUES);

if(@queues)
{
    print VALIDATION_FILE "Attribute: queue\n";
    print VALIDATION_FILE join(" ", "Values:", @queues);

}
```

[13] http://www-unix.globus.org/api/c-globus-2.4/perl/globus-job-manager-service.html
[14] http://www-unix.globus.org/api/c-globus-2.4/globus_gram_job_manager/html/globus_gram_job_manager_rsl_validation_file.html#globus_gram_job_manager_rsl_validation_file

```
close VALIDATION_FILE;
```

3.1.2.4. Updating GPT Metadata

Finally, the setup package should create and finalize a `Grid::GPT::Setup`. The value of $package must be the same value as the gpt_package_metadata *Name* attribute in the package's metadata file. If either the `new()` or `finish()` methods fail, then it is considered good practice to clean up any files created by the setup script. From `setup-globus-job-manager-lsf.pl`:

```
my $metadata =
new Grid::GPT::Setup(
        package_name => "globus_gram_job_manager_setup_lsf");

...

$metadata->finish();
```

3.1.3. Packaging

Now that we've written a job manager scheduler interface, we'll package it using GPT to make it easy for our users to build and install. We'll start by gathering the different files we've written above into a single directory `lsf`.

- lsf.in
- find-lsf-tools.in
- setup-globus-job-manager.pl

3.1.3.1. Package Documentation

If there are any scheduler-specific options defined for this scheduler module, or if there any any optional setup items, then it is good to provide a documentation page which describes these. For LSF, we describe the changes since the last version of this package in the file `globus_gram_job_manager_lsf.dox`. This file consists of a doxygen mainpage. See www.doxygen.org for information on how to write documentation with that tool.

3.1.3.2. configure.in

Now, we'll write our configure.in script. This file is converted to the configure shell script by the bootstrap script below. Since we don't do any probes for compile-time tools or system characteristics, we just call the various initialization macros used by GPT, declare that we may provide doxygen documentation, and then output the files we need substitions done on.

```
AC_REVISION($Revision: 1.1 $)
AC_INIT(Makefile.am)

GLOBUS_INIT
AM_PROG_LIBTOOL

dnl Initialize the automake rules the last argument
AM_INIT_AUTOMAKE($GPT_NAME, $GPT_VERSION)
```

```
LAC_DOXYGEN("../", "*.dox")

GLOBUS_FINALIZE

AC_OUTPUT(
        Makefile
        pkgdata/Makefile
        pkgdata/pkg_data_src.gpt
        doxygen/Doxyfile
        doxygen/Doxyfile-internal
        doxygen/Makefile
)
```

3.1.3.3. Package Metadata

Now we'll write our metadata file, and put it in the pkgdata subdirectory of our package. The important things to note in this file are the package name and version, the post_install_program, and the setup sections. These define how the package distribution will be named, what command will be run by `gpt-postinstall` when this package is installed, and what the setup dependencies will be written when the Grid::GPT::Setup object is finalized[15].

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE gpt_package_metadata SYSTEM "package.dtd">

<gpt_package_metadata Format_Version="0.02" Name="globus_gram_job_manager_setup_lsf" >

  <Aging_Version Age="0" Major="1" Minor="0" />
  <Description >LSF Job Manager Setup</Description>
  <Functional_Group >ResourceManagement</Functional_Group>
  <Version_Stability Release="Beta" />
  <src_pkg >

    <With_Flavors build="no" />
    <Source_Setup_Dependency PkgType="pgm" >
      <Setup_Dependency Name="globus_gram_job_manager_setup" >
        <Version >
          <Simple_Version Major="3" />
        </Version>
      </Setup_Dependency>
      <Setup_Dependency Name="globus_common_setup" >
        <Version >
          <Simple_Version Major="2" />
        </Version>
      </Setup_Dependency>
    </Source_Setup_Dependency>

    <Build_Environment >
      <cflags >@GPT_CFLAGS@</cflags>
      <external_includes >@GPT_EXTERNAL_INCLUDES@</external_includes>
      <pkg_libs > </pkg_libs>
```

[15] http://www-unix.globus.org/api/c-globus-2.4/globus_gram_job_manager/html/globus_gram_job_manager_interface_tutorial.html#updating_gpt_metadata

```
      <external_libs >@GPT_EXTERNAL_LIBS@</external_libs>
    </Build_Environment>

    <Post_Install_Message >
      Run the setup-globus-job-manager-lsf setup script to configure an
      lsf job manager.
    </Post_Install_Message>

    <Post_Install_Program >
      setup-globus-job-manager-lsf
    </Post_Install_Program>

    <Setup Name="globus_gram_job_manager_service_setup" >
      <Aging_Version Age="0" Major="1" Minor="0" />
    </Setup>

  </src_pkg>

</gpt_package_metadata>
```

3.1.3.4. Automake Makefile.am

The automake Makefile.am for this package is short because there isn't any compilation needed for this package. We just need to define what needs to be installed into which directory, and what source files need to be put inot our source distribution. For the LSF package, we need to list the `lsf.in`, `find-lsf-tools`, and code>setup-globus-job-manager-lsf.pl scripts as files to be installed into the setup directory. We need to add those files plus our documentation source file to the EXTRA_LIST variable so that they will be included in source distributions. The rest of the lines in the file are needed for proper interaction with GPT.

```
include $(top_srcdir)/globus_automake_pre
include $(top_srcdir)/globus_automake_pre_top

SUBDIRS = pkgdata doxygen

setup_SCRIPTS = \
    lsf.in \
    find-lsf-tools \
    setup-globus-job-manager-lsf.pl

EXTRA_DIST = $(setup_SCRIPTS) globus_gram_job_manager_lsf.dox

include $(top_srcdir)/globus_automake_post
include $(top_srcdir)/globus_automake_post_top
```

3.1.3.5. Bootstrap

The final piece we need to write for our package is the `bootstrap` script. This script is the standard bootstrap script for a globus package, with an extra line to generate the `find-lsf-tools` script using autoconf.

```
#!/bin/sh
```

```
# checking for the GLOBUS_LOCATION

if test "x$GLOBUS_LOCATION" = "x"; then
    echo "ERROR Please specify GLOBUS_LOCATION" >&2
    exit 1
fi

if [ ! -f ${GLOBUS_LOCATION}/libexec/globus-bootstrap.sh ]; then
    echo "ERROR: Unable to locate \${GLOBUS_LOCATION}/libexec/globus-bootstrap.sh"
    echo "       Please ensure that you have installed the globus-core package and"
    echo "       that GLOBUS_LOCATION is set to the proper directory"
    exit
fi

. ${GLOBUS_LOCATION}/libexec/globus-bootstrap.sh

autoconf find-lsf-tools.in > find-lsf-tools
chmod 755 find-lsf-tools

exit 0
```

3.1.4. Building, Testing, and Debugging

With this all done, we can now try to build our now package. To do so, we'll need to run

```
% ./bootstrap
% ./gpt-build
```

If all of the files are written correctly, this should result in our package being installed into `$GLOBUS_LOCATION`. Once that is done, we should be able to run gpt-postinstall to configure our new job manager.

Now, we should be able to run the command

```
% globus-personal-gatekeeper -start -jmtype lsf
```

to start a gatekeeper configured to run a job manager using our new scripts. Running this will output a contact string (referred to as <contact-string> below), which we can use to connect to this new service. To do so, we'll run globus-job-run to submit a test job:

```
% globus-job-run <contact-string> /bin/echo Hello, LSF
Hello, LSF
```

3.1.4.1. When Things Go Wrong

If the test above fails, or more complicated job failures are occurring, then you'lll have to debug your scheduler interface. Here are a few tips to help you out.

Make sure that your script is valid Perl. If you run

```
perl -I$GLOBUS_LOCATION/lib/perl \
    $GLOBUS_LOCATION/lib/perl/Globus/GRAM/JobManager/lsf.pm
```

You should get no output. If there are any diagnostics, correct them (in the lsf.in file), reinstall your package, and rerun the setup script.

Look at the Globus Toolkit Error FAQ[16] and see if the failure is perhaps not related to your scheduler script at all.

Enable logging for the job manager. By default, the job manager is configured to log only when it notices a job failure. However, if your problem is that your script is not returning a failure code when you expect, you might want to enable logging always. To do this, modify the job manager configuration file to contain "-save-logfile always" in place of "-save-log on_error".

Adding logging messages to your script: the JobManager object implements a `log` method, which allows you to write messages to the job manager log file. Do this as your methods are called to pinpoint where the error occurs.

Save the job description file when your script is run. This will allow you to run the `globus-job-manager-script.pl` interactively (or in the Perl debugger). To save the job description file, you can do

```
$self->{JobDescription}->save("/tmp/job_description.$$");
```

in any of the methods you've implemented.

4. Usage scenarios

There is no content available at this time.

5. Debugging

There is no content available at this time.

6. Troubleshooting

There is no content available at this time.

7. Related Documentation

Information about other C-APIs of the GT can be found here[17]

[16] http://www.globus.org/toolkit/docs/2.4/faq_errors.html
[17] http://www.globus.org/api/c-globus-4.0.3/

Chapter 6. GT 4.0 Release Notes: Pre-WS GRAM

1. Component Overview

The Grid Resource Allocation and Management (GRAM) service provides a single interface for requesting and using remote system resources for the execution of "jobs". The most common use of GRAM is remote application execution and control. It is designed to provide a uniform, flexible interface to job scheduling systems.

2. Feature Summary

Features new in release 4.0

- The ability to select the account under which the remote job will be run. If a user's grid credential is mapped to multiple accounts, then the user can specify in the RSL under which account the job should be run.

Other Supported Features

- Remote job execution and management
- Uniform and flexible interface to batch scheduling systems
- File staging before and after job execution
- Data streaming of stdout/err during jobs execution

Deprecated Features

- None

3. Bug Fixes

- Bugzilla url to bugs fixed since previous stable version
- ...
- Bugzilla url to bugs fixed since previous stable version

4. Known Problems

- Bugzilla url here
- ...
- Bugzilla url here

5. Technology Dependencies

Pre-WS GRAM depends on the following GT components:

- C Common Libraries
- Pre-WS Authentication and Authorization
- XIO
- GridFTP

Pre-WS GRAM depends on the following 3rd party software. The dependency exists only for the batch schedulers configured, thus making job submissions possible to the batch scheduling service:

- PBS
- Condor
- LSF
- other batch schedulers... (where the GRAM scheduler interface has been implemented)

6. Tested Platforms

Tested Platforms for Pre WS GRAM

- Linux

7. Backward Compatibility Summary

Protocol changes since GT version 3.2

- GRAM job requests may append the desired local username after the `@` symbol in the service path (`jobmanager-fork@username`). This is done automatically when the username RSL attribute is present.

API changes since GT version 3.2

- Implementation change: the GRAM client library now parses the RSL string and checks for the `username` attribute. Previously, the client library did no parsing of the RSL string.

Exception/error changes since GT version 3.2

- `GLOBUS_GRAM_PROTOCOL_ERROR_RSL_USER_NAME` - `username` attribute value is not a literal string.
- `GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_USER_NAME` - `username` attribute present but not running as the specified user (most likely using an old client.)

Schema changes since GT version 3.2

- There is a new RSL attribute `username`. If present, the job will be submitted to run as the specified user. If the job manager is not running as that user, it will not run the job.

8. For More Information

Click here[1] for more information about this component.

[1] http://www.globus.org/toolkit/docs/2.4/

Chapter 7. GT 4.0.1 Incremental Release Notes: Pre-WS GRAM

1. Introduction

These release notes are for the incremental release 4.0.1. It includes a summary of changes since 4.0.0, bug fixes since 4.0.0 and any known problems that still exist at the time of the 4.0.1 release. This page is in addition to the top-level 4.0.1 release notes at http://www.globus.org/toolkit/releasenotes/4.0.1.

For release notes about 4.0 (including feature summary, technology dependencies, etc) go to the Pre-WS GRAM 4.0 Release Notes[1].

2. Changes Summary

Other than a bug fix, no change has occurred for Pre-WS GRAM.

3. Bug Fixes

The following bug was fixed for Pre-WS GRAM:

- Bug 3589:[2] Pre-WS GT4 globusrun -a mode cores on a USATLAS site

4. Known Problems

No problems are known to exist for Pre-WS GRAM at the time of the 4.0.1 release.

5. For More Information

Click here[3] for more information about this component.

[1] http://www.globus.org/toolkit/docs/4.0/execution/prewsgram/Pre_WS_GRAM_Release_Notes.html
[2] http://bugzilla.globus.org/globus/show_bug.cgi?id=3589
[3] http://www.globus.org/toolkit/docs/2.4/

Chapter 8. GT 4.0.2 Incremental Release Notes: Pre-WS GRAM

1. Introduction

These release notes are for the incremental release 4.0.2. It includes a summary of changes since 4.0.1, bug fixes since 4.0.1 and any known problems that still exist at the time of the 4.0.2 release. This page is in addition to the top-level 4.0.2 release notes at http://www.globus.org/toolkit/releasenotes/4.0.2.

For release notes about 4.0 (including feature summary, technology dependencies, etc) go to the Pre-WS GRAM 4.0 Release Notes[1].

2. Changes Summary

Other than a bug fix, no change has occurred for Pre-WS GRAM.

3. Bug Fixes

The following bug was fixed for Pre-WS GRAM:

- Bug 3355:[2] $JAVA_HOME/bin/java vs [[PATH]] java
- Bug 3411:[3] Condor-G and pre-WS GT4 problems
- Bug 4112:[4] Add Solaris 5.9 to find-condor-tools utility
- Bug 3798:[5] Massive memory leaks in GRAM client library
- Bug 3942:[6] globus_gram_client_job_request() is MT-unsafe

4. Known Problems

The following problems are known to exist for Pre-WS GRAM at the time of the 4.0.2 release:

- Bug #1037:[7] globus-gram-reporter does not publish additional LSF job ...
- Bug #1551:[8] Race condition in job manager
- Bug #1550:[9] Fixes for race condition in job manager

[1] http://www.globus.org/toolkit/docs/4.0/execution/prewsgram/Pre_WS_GRAM_Release_Notes.html
[2] http://bugzilla.globus.org/globus/show_bug.cgi?id=3355
[3] http://bugzilla.globus.org/globus/show_bug.cgi?id=3411
[4] http://bugzilla.globus.org/globus/show_bug.cgi?id=4112
[5] http://bugzilla.globus.org/globus/show_bug.cgi?id=3798
[6] http://bugzilla.globus.org/globus/show_bug.cgi?id=3942
[7] http://bugzilla.globus.org/globus/show_bug.cgi?id=1037
[8] http://bugzilla.globus.org/globus/show_bug.cgi?id=1551
[9] http://bugzilla.globus.org/globus/show_bug.cgi?id=1550

- Bug #1868:[10] globus_gram_reporter is missing from GT 3.2.1 PreWS distr...
- Bug #1934:[11] Gatekeeper's syslog output cannot be controlled
- Bug #2235:[12] GT3.2.1 globus_gram_client_tools-4.3 patch broken
- Bug #3373:[13] globus removes the temporary job directory before pbs wri...
- Bug #3425:[14] personal gatekeeper does not work on AIX
- Bug #3428:[15] globus-personal-gatekeeper -list failing
- Bug #2739:[16] Gatekeeper AuthZ/Gridmap Callout result logging
- Bug #2741:[17] catching SIGSEGV if dynamic loading of authorization modu...
- Bug #4150:[18] do not use break in Perl, use last
- Bug #4235:[19] globus-job-manager doesn't exit if the job fails.
- Bug #4335:[20] Incorrectly terminated message in globus-gram-job-manager...

5. For More Information

Click here[21] for more information about this component.

[10] http://bugzilla.globus.org/globus/show_bug.cgi?id=1868
[11] http://bugzilla.globus.org/globus/show_bug.cgi?id=1934
[12] http://bugzilla.globus.org/globus/show_bug.cgi?id=2235
[13] http://bugzilla.globus.org/globus/show_bug.cgi?id=3373
[14] http://bugzilla.globus.org/globus/show_bug.cgi?id=3425
[15] http://bugzilla.globus.org/globus/show_bug.cgi?id=3428
[16] http://bugzilla.globus.org/globus/show_bug.cgi?id=2739
[17] http://bugzilla.globus.org/globus/show_bug.cgi?id=2741
[18] http://bugzilla.globus.org/globus/show_bug.cgi?id=4150
[19] http://bugzilla.globus.org/globus/show_bug.cgi?id=4235
[20] http://bugzilla.globus.org/globus/show_bug.cgi?id=4335
[21] http://www.globus.org/toolkit/docs/2.4/

Chapter 9. GT 4.0.3 Incremental Release Notes: Pre-WS GRAM

1. Introduction

These release notes are for the incremental release 4.0.3. It includes a summary of changes since 4.0.2, bug fixes since 4.0.2 and any known problems that still exist at the time of the 4.0.3 release. This page is in addition to the top-level 4.0.3 release notes at http://www.globus.org/toolkit/releasenotes/4.0.3.

For release notes about 4.0 (including feature summary, technology dependencies, etc) go to the Pre-WS GRAM 4.0 Release Notes[1].

2. Changes Summary

Pre-WS GRAM has performed a security fix and a bug fix since GT 4.0.2.

3. Bug Fixes

The following bug has been fixed for Pre-WS GRAM since GT 4.0.2:

- Bug 4605:[2] leak in rsl copy recursive

4. Known Problems

The following problems are known to exist for Pre-WS GRAM at the time of the 4.0.3 release:

- Bug 1037:[3] globus-gram-reporter does not publish additional LSF job ...
- Bug 1551:[4] Race condition in job manager
- Bug 1550:[5] Fixes for race condition in job manager
- Bug 1868:[6] globus_gram_reporter is missing from GT 3.2.1 PreWS distr...
- Bug 1934:[7] Gatekeeper's syslog output cannot be controlled
- Bug 2235:[8] GT3.2.1 globus_gram_client_tools-4.3 patch broken
- Bug 3373:[9] globus removes the temporary job directory before pbs wri...

[1] http://www.globus.org/toolkit/docs/4.0/execution/prewsgram/Pre_WS_GRAM_Release_Notes.html
[2] http://bugzilla.globus.org/globus/show_bug.cgi?id=4605
[3] http://bugzilla.globus.org/globus/show_bug.cgi?id=1037
[4] http://bugzilla.globus.org/globus/show_bug.cgi?id=1551
[5] http://bugzilla.globus.org/globus/show_bug.cgi?id=1550
[6] http://bugzilla.globus.org/globus/show_bug.cgi?id=1868
[7] http://bugzilla.globus.org/globus/show_bug.cgi?id=1934
[8] http://bugzilla.globus.org/globus/show_bug.cgi?id=2235
[9] http://bugzilla.globus.org/globus/show_bug.cgi?id=3373

- Bug 3425:[10] personal gatekeeper does not work on AIX
- Bug 3428:[11] globus-personal-gatekeeper -list failing
- Bug2739:[12] Gatekeeper AuthZ/Gridmap Callout result logging
- Bug 2741:[13] catching SIGSEGV if dynamic loading of authorization modu...
- Bug 4150:[14] do not use break in Perl, use last
- Bug 4235:[15] globus-job-manager doesn't exit if the job fails.
- Bug 4335:[16] Incorrectly terminated message in globus-gram-job-manager...

5. For More Information

Click here[17] for more information about this component.

[10] http://bugzilla.globus.org/globus/show_bug.cgi?id=3425
[11] http://bugzilla.globus.org/globus/show_bug.cgi?id=3428
[12] http://bugzilla.globus.org/globus/show_bug.cgi?id=2739
[13] http://bugzilla.globus.org/globus/show_bug.cgi?id=2741
[14] http://bugzilla.globus.org/globus/show_bug.cgi?id=4150
[15] http://bugzilla.globus.org/globus/show_bug.cgi?id=4235
[16] http://bugzilla.globus.org/globus/show_bug.cgi?id=4335
[17] http://www.globus.org/toolkit/docs/2.4/

Chapter 10. GT 4.0.4 Incremental Release Notes: Pre-WS GRAM

1. Introduction

These release notes are for the incremental release 4.0.4. It includes a summary of changes since 4.0.3, bug fixes since 4.0.3 and any known problems that still exist at the time of the 4.0.4 release. This page is in addition to the top-level 4.0.4 release notes at http://www.globus.org/toolkit/releasenotes/4.0.4.

For release notes about 4.0 (including feature summary, technology dependencies, etc) go to the Pre-WS GRAM 4.0 Release Notes[1].

2. Changes Summary

Pre-WS GRAM had 4 bug fixes since GT 4.0.3.

3. Bug Fixes

- Bug 4860:[2] LSF jobmanager doesn't preserve environment variables with spaces
- Bug 4969:[3] Broken contact string with globus-personal-gatekeeper and simpleCA
- Bug 4620:[4] SEGV on updating credential for GRAM
- Bug 4976:[5] wrong values for host and port in configfile of globus-personal-gatekeeper

4. Known Problems

- Bug 1550:[6] Fixes for race condition in job manager
- Bug 1934:[7] Gatekeeper's syslog output cannot be controlled
- Bug 3373:[8] globus removes the temporary job directory before pbs wri...
- Bug 3428:[9] globus-personal-gatekeeper -list failing
- Bug 2739:[10] Gatekeeper AuthZ/Gridmap Callout result logging
- Bug 2741:[11] catching SIGSEGV if dynamic loading of authorization modu...

[1] http://www.globus.org/toolkit/docs/4.0/execution/prewsgram/Pre_WS_GRAM_Release_Notes.html
[2] http://bugzilla.globus.org/globus/show_bug.cgi?id=4860
[3] http://bugzilla.globus.org/globus/show_bug.cgi?id=4969
[4] http://bugzilla.globus.org/globus/show_bug.cgi?id=4620
[5] http://bugzilla.globus.org/globus/show_bug.cgi?id=4976
[6] http://bugzilla.globus.org/globus/show_bug.cgi?id=1550
[7] http://bugzilla.globus.org/globus/show_bug.cgi?id=1934
[8] http://bugzilla.globus.org/globus/show_bug.cgi?id=3373
[9] http://bugzilla.globus.org/globus/show_bug.cgi?id=3428
[10] http://bugzilla.globus.org/globus/show_bug.cgi?id=2739
[11] http://bugzilla.globus.org/globus/show_bug.cgi?id=2741

- Bug 4235:[12] globus-job-manager doesn't exit if the job fails.
- Bug 4360:[13] globus-job-get-output bug prevents output delivery, PBS jobmanager affected. See also globus-job-clean, globus-job-cancel
- Bug 4450:[14] GT 4.0.1 globus-gatekeeper hang on UC/ANL TeraGrid
- Bug 4730:[15] MPI Jobs using Globus LSF in HP XC Cluster....
- Bug 4747:[16] Need evaluation of patch to JobManager.pm
- Bug 4771:[17] date bug in job manager log file
- Bug 4905:[18] setup-globus-gram-job-manager.pl fails silently when globus-version is missing

5. For More Information

Click here[19] for more information about this component.

[12] http://bugzilla.globus.org/globus/show_bug.cgi?id=4235
[13] http://bugzilla.globus.org/globus/show_bug.cgi?id=4360
[14] http://bugzilla.globus.org/globus/show_bug.cgi?id=4450
[15] http://bugzilla.globus.org/globus/show_bug.cgi?id=4730
[16] http://bugzilla.globus.org/globus/show_bug.cgi?id=4747
[17] http://bugzilla.globus.org/globus/show_bug.cgi?id=4771
[18] http://bugzilla.globus.org/globus/show_bug.cgi?id=4905
[19] http://www.globus.org/toolkit/docs/2.4/

Chapter 11. GT 4.0 Component Fact Sheet: Pre-Web Service Grid Resource Allocation and Management (Pre-WS GRAM)

1. Brief component overview

The Grid Resource Allocation and Management (GRAM) service provides a single interface for requesting and using remote system resources for the execution of "jobs". The most common use of GRAM is remote application execution and control. It is designed to provide a uniform, flexible interface to job scheduling systems.

2. Summary of features

Features new in release 4.0

- The ability to select the account under which the remote job will be run. If a user's grid credential is mapped to multiple accounts, then the user can specify in the RSL under which account the job should be run.

Other Supported Features

- Remote job execution and management
- Uniform and flexible interface to batch scheduling systems
- File staging before and after job execution
- Data streaming of stdout/err during jobs execution

Deprecated Features

- None

3. Usability summary

Usability improvements for Pre-WS GRAM:

- improvement #1
- ...
- improvement #n

4. Backward compatibility summary

Protocol changes since GT version 3.2

- GRAM job requests may append the desired local username after the `@` symbol in the service path (`jobmanager-fork@username`). This is done automatically when the username RSL attribute is present.

API changes since GT version 3.2

- Implementation change: the GRAM client library now parses the RSL string and checks for the `username` attribute. Previously, the client library did no parsing of the RSL string.

Exception/error changes since GT version 3.2

- `GLOBUS_GRAM_PROTOCOL_ERROR_RSL_USER_NAME` - `username` attribute value is not a literal string.
- `GLOBUS_GRAM_PROTOCOL_ERROR_INVALID_USER_NAME` - `username` attribute present but not running as the specified user (most likely using an old client.)

Schema changes since GT version 3.2

- There is a new RSL attribute `username`. If present, the job will be submitted to run as the specified user. If the job manager is not running as that user, it will not run the job.

5. Technology dependencies

Pre-WS GRAM depends on the following GT components:

- C Common Libraries
- Pre-WS Authentication and Authorization
- XIO
- GridFTP

Pre-WS GRAM depends on the following 3rd party software. The dependency exists only for the batch schedulers configured, thus making job submissions possible to the batch scheduling service:

- PBS
- Condor
- LSF
- other batch schedulers... (where the GRAM scheduler interface has been implemented)

6. Tested platforms

Tested Platforms for Pre WS GRAM

- Linux

7. For More Information

Click here[1] for more information about this component.

[1] index.html