GT4 GRAM: A Functionality and Performance Study

Martin Feller[1], Ian Foster[1,2,3], and Stuart Martin[1,2]

Abstract— The Globus Toolkit's pre-Web Services GRAM service ("GRAM2") has been widely deployed on grids around the world for many years. Recent work has produced a new, Web Services-based GRAM service ("GRAM4"). We describe and compare the functionality and performance of the GRAM2 and GRAM4 job execution services included in Globus Toolkit version 4 (GT4). GRAM4 provides significant improvements in functionality and scalability over GRAM2 in many areas. GRAM4 is faster in the case of many concurrent submissions, but slower for sequential submissions and when file staging is involved. (Optimizations to address the latter cases are in progress.) This information should be useful when considering an upgrade from GRAM2 to GRAM4, and when comparing GRAM against other job execution services.

Index Terms— Globus, Grid, Job Execution Service, GRAM, GRAM2, GRAM4, Audit

1 INTRODUCTION

Grid applications frequently require mechanisms for executing remote jobs. While this requirement might appear straightforward, its practical realization can be challenging due to the need to (for example) address security, reliability, and performance concerns; enable both client and server management of resources consumed by remote jobs; propagate error messages and failure notifications; and move data to/from remote computations. Thus, in practice, the creation of secure, reliable, and performant job execution services is difficult.

In Globus Toolkit version 4 (GT4) [1], remote job execution is supported by the Grid Resource Allocation and Management (GRAM) service, which defines mechanisms for submitting requests to execute jobs (defined in a job description language) and for monitoring and controlling the resulting job executions. More precisely, GT4 includes two different GRAM services: the "pre-WS GRAM," or GRAM2 [2], first introduced in GT2, and the newer Web Services-based "WS GRAM," or GRAM4, first included in GT4.

We describe and compare first the functionality (Section 2) and then the performance (Section 3) of GRAM2 and GRAM4: specifically, the code to be distributed as the "pre-WS GRAM" and "WS GRAM" services in the GT4.0.5 release. We see that GRAM4 provides significant improvements in functionality and scalability over GRAM2. GRAM4 performs well for concurrent submissions, but is slower than GRAM2 for sequential submissions and when file staging is involved—areas in which optimizations are planned, along with further functionality improvements, in forthcoming 4.0.x and 4.2.x releases (see Section 4).

Other systems providing functionality similar to GRAM include GridSAM [3], CREAM [4], and WSRF.NET [5]. The functionality and performance data provided here should provide a basis for detailed qualitative and quantitative comparisons with and between those and other systems.

[1] Computation Institute, University of Chicago & Argonne National Laboratory, USA
[2] Math & Computer Science Division, Argonne National Laboratory, Argonne IL, USA
[3] Department of Computer Science, University of Chicago, IL, USA

2 FUNCTIONALITY

We present a point-by-point comparison of GRAM2 and GRAM4 functionality, both in summary form (Table 1) and in more detail in the following. We divide functionality into three parts: security, file staging, and general. In each case, the short description is shaded (in the table) or underlined (in the text) to indicate where GRAM4 offers significantly better functionality than GRAM2. No shading or underline indicates that the two versions offer similar functionality.

We emphasize that this long list of GRAM features does not translate into a complex service for the user. The GRAM2 and GRAM4 client interfaces are simple; the rich functionality described here ensures that remote job execution, monitoring, and management are secure, reliable, and efficient.

2.1 Security Features

Privilege limiting model. 2: *Gatekeeper as root*; 4: *Service with sudo privileges*. In a typical deployment, the GRAM server must be able to start jobs submitted by remote users under different user ids, and thus must be able to execute some code as "root." It is generally viewed as preferable to limit the amount of such "privileged" code. In GRAM2, the entire "gatekeeper" service runs as root. In GRAM4, the GRAM service does not itself require privileges. Instead, it uses "sudo" to invoke operations for which privileges are required.

Authentication. 2: *TLS*; 4: *TLS, Message-level security*. A client can authenticate with a GRAM service using a variety of protocols. In GRAM2, only SSL/TLS is supported; in GRAM4, the standard message-level WS-Security and channel-level WS-SecureConversation are also supported, with the choice of protocol supported by a particular GRAM4 deployment specified in the service configuration.

Credential delegation. 2: *Yes (required)*; 4: *Yes (optional)*. A job submitted to a GRAM service may require a delegated credential [6] if it is to stage files or perform other remote operations for which authentication is required. In GRAM2, a delegated credential is passed with every request. In GRAM4, a separate delegation interface is provided, allowing a client to delegate a credential only when required, and to share a delegated credential among multiple jobs. The GRAM4 approach is more efficient and allows for the use of authentication protocols that don't inherently allow delegation.

Table 1: Functionality comparison of GRAM2 and GRAM4 (see text for details, a shaded box means better)

Feature	GRAM2	GRAM4
1) SECURITY		
Privilege limiting model	Gatekeeper as root	Service with sudo privileges
Authentication options	TLS	TLS, Secure Message, WS-Security
Credential delegation	Yes (required)	Yes (optional)
Credential refresh	Yes	Yes
Share credential delegation among jobs	No	Yes
Authorization callouts	Yes (single PDP callout)	Yes (PDP callout chain)
2) FILE MANAGEMENT		
File staging	Yes	Yes
File staging retry policy	None	RFT supported
Incremental output staging (“streaming”)	Stdout, stderr only	stdout, stderr, & any output files
Standard input access	Yes (from file)	Yes (from file)
Throttle staging work	No	Yes
Load balance staging work	No	Yes
3) GENERAL		
Access protocol	GRAM-specific HTTP	Web Services, SOAP
Job description language	RSL	JDD
Extensible job description language	Yes	Yes
Local resource manager interface	PERL scripts	PERL scripts + SEG
Local resource managers	Fork, Condor, PBS, LSF, ...	Fork, Condor, PBS, LSF, …
Fault tolerance	Yes (client initiated)	Yes (service initiated)
State access: pull	Yes	Yes
State access: push (subscription)	Yes: callbacks	Yes: WS-Notification
Audit logging	Yes (experimental)	Yes (experimental)
At most once job submission	Yes (2 phase commit)	Yes (UUID on create)
Job cancellation	Yes	Yes
Job lifetime management	Yes	Yes
Maximum active jobs	~250	32,000
Parallel job support	Yes	Yes
MPICH-G support	Yes	Yes
Basic Execution Service (BES) interface	No	Prototyped

Credential refresh. 2: *Yes*; 4: *Yes*. Credentials have a lifetime, which may expire before a job has completed execution. Thus, we may want to supply a new credential. Both GRAM2 and GRAM4 provide this capability. In GRAM4, the refresh is performed via the same delegation service used to supply the credential in the first place. A client can both request notification of imminent expiration and refresh the credential.

Share credential delegation among jobs. 2: *No*; 4: *Yes*. See “credential delegation” (above) for explanation.

Authorization callouts. 2: *Yes—single PDP callout*; 4: *Yes—PDP callout chain*. Following authentication, GRAM checks to see whether the request should be authorized. In GRAM2, a single (pluggable) policy decision point (PDP) or authorization function can be called, to check (for example) a “grid-map” file acting as an access control list. In GRAM4, multiple PDPs can be combined together, allowing for richer policies. GRAM4 also allows Policy Information Points to be included in the chain for attribute-based authorization. Thus, for example, a GRAM deployment can be configured to use a PIP that parses VOMs attributes and a PDP that uses those attributes as a part of policy evaluation. Since chain of PDPs can be configured, policies at various levels, such as site-level blacklist policies, can be evaluated in addition to service level policies.

2.2 File Management

File staging. 2: *Yes*; 4: *Yes*. Both GRAM2 and GRAM4 allow job descriptions to specify that files are to be staged prior to job execution and/or during or after job completion.

File staging retry policy. 2: *None*; 4: *RFT supported*. In GRAM2, if a file staging operation fails, the job is aborted. In GRAM4, a failed file staging operation can be retried by the GRAM file staging service—the reliable file transfer (RFT) service [7]. RFT’s retry policy can be set as a service default for all transfers and also be overridden by a client for a specific transfer. For example, the Condor-G client sets the number of retries to five.

Incremental output staging (“streaming”). 2: *Stdout, stderr only*; 4: *Stdout, stderr, and any output files*. It can be useful to obtain access to data produced by a program as it executes. In, GRAM2, a job’s standard output and standard error streams can be accessed in this way. In GRAM4, any output file can also be specified as “streaming.”

Standard input access. 2: *Yes—from a file*; 4: *Yes—from a file*. In both GRAM2 and GRAM4, the contents of a specified file can be passed to a job’s standard input.

Throttle staging work. 2: *No*; 4: *Yes*. A GRAM submission that specifies file staging operations imposes load on the "service node" executing the GRAM service. In GRAM2, this load was not managed, and so many simultaneous submissions could result in a large number of concurrent transfers and thus excessive load on the "service node." GRAM4 can be configured to limit the number of "worker" threads that process GRAM4 work and thus the maximum number of concurrent staging operations. In 4.0.5, the default value for this configuration parameter is 30.

Load balance staging work. 2: *No*; 4: *Yes*. In GRAM2, staging work must be performed on the same "service node" as the GRAM2 service. In GRAM4, staging work can be distributed over several "service nodes." A "GRAM and GridFTP file system mapping" configuration file allows a system administrator to specify one or more GridFTP servers, each associated with a local resource manager (LRM) type and one or more file system mappings. Based on a job's LRM type and file paths in the staging directives, GRAM then chooses the matching GridFTP server(s).

2.3 General

Access protocol. 2: *GRAM-specific HTTP*; 4: *Web Service, SOAP*. GRAM2 uses a custom HTTP-based protocol to transfer requests and replies. GRAM4 uses Web Services, thus providing for a convenient standard representation of protocol messages (WSDL) and enabling the use of standard tooling to develop clients.

Job description language. 2: *RSL*; 4: *JDD*. GRAM2 uses a custom, string-based Resource Specification Language (RSL). GRAM4 supports an XML-based version of RSL, the Job Description Document (JDD). A prototype implementation of the Job Submission Description Language (JSDL) has also been developed for GRAM4: see Section 5.

Extensible job description language. 2: *Yes*; 4: *Yes*. Both RSL and JDD support user-defined extensibility elements.

Local resource manager interface. 2: *Perl scripts*; 4: *Perl scripts + SEG*. A GRAM service that receives a job submission request passes that request (assuming successful authentication and authorization) to a local resource manager (LRM). Both GRAM2 and GRAM4 can interface to many LRMs. GRAM4 monitors the LRM jobs more efficiently by using the scheduler event generator (SEG) instead of polling. Each GRAM4 LRM type requires a SEG implementation

Local resource managers. 2: *Fork, Condor, PBS, LSF*; 4: *Fork, Condor, PBS, LSF*. Both GRAM2 and GRAM4 support a simple "fork" LRM (that simply starts jobs on the same computer as the GRAM server) and a range of other commonly used LRMs, including Portable Batch System (PBS), Load Sharing Facility (LSF), and Condor. Many others are available from third parties such as: LoadLeveler, Sun Grid Engine (SGE), GridWay, etc.

Fault tolerance. 2: *Yes—client initiated*; 4: *Yes—service initiated*. It is important that a GRAM service be fault tolerant, by which we mean that if it fails (e.g., because the computer on which it is running crashes) and is then restarted, it and its clients can reconnect with any running jobs [8]. GRAM2 provides a limited form of fault tolerance, requiring a client to supply a "job contact" that the GRAM service then uses to reconnect with the job. GRAM4 provides a more general solution: it persists the job contact information itself, and thus can monitor and control all jobs that it created, without the involvement of clients.

State access: push (subscription). 2: *Yes—callbacks*; 4: *Yes—WS-Notification*. Both GRAM2 and GRAM4 allow a client to request notifications of changes in job state. In GRAM2, the client registers a call back. In GRAM4, standard WS-Notification operations are applied to the "job status" resource property.

State access: pull. 2: *Yes*; 4: *Yes*. In GRAM4, the service defines a WSRF Resource Property that contains the value of the job state. A client can then use the standard WSRF getResourceProperty operation. In GRAM2, it is a proprietary operation.

Audit logging. 2: *Yes*; 4: *Yes*. This recent enhancement to both GRAM2 and GRAM4 allows an audit record to be inserted into an audit database when a job completes. This mechanisms is used, for example, by TeraGrid to obtain both a unique grid ID for a job and job resource usage data from TeraGrid's accounting. Extensions have already been contributed (for GRAM4 only) by Gerson Galang (APAC Grid) to insert the job's audit record at the beginning of the job and to update the record after submission and again at job end.

At most once job submission. 2: *Yes—two-phase commit*; 4: *Yes—UUID on create*. A simple request-reply job submission protocol has the problem that if the reply message is lost, a client cannot know whether a job has been started. Thus, both GRAM2 and GRAM4 provide protocol features that a client can use to ensure that the same job is not submitted twice. GRAM2 uses a 2-phase commit protocol: the client submits a request, obtains a job contact, and then starts the job. GRAM4 adopts an alternative approach: the client supplies a client-created unique identifier (UUID) and the GRAM service guarantees not to start a job with a duplicate identifier. The GRAM4 approach allows a job submission to proceed with one rather than two roundtrips and is thus more efficient.

Job cancellation. 2: *Yes*; 4: *Yes*. In GRAM4, a client calls the standard WSRF "Destroy" operation to terminate a job. In GRAM2, it is a proprietary operation.

Job lifetime management. 2: *Yes*; 4: *Yes*. Both GRAM2 and GRAM4 provide similar functionality for job state lifetime management, in order for a client to control when a job's state is cleaned up. GRAM2 implements a set of job directives and operations that control this functionality. GRAM4 leverages standard WS-ResourceLifetime operations.

Maximum active jobs. 2: *~250*; 4: *32,000*. GRAM2 creates a "job manager" process for each submitted job, a strategy that both creates excessive load on the "service node" where the GRAM2 service runs and limits the number of jobs that a GRAM2 service can support concurrently. In contrast, GRAM4 runs as a single process that maintains information about each active job in a structure file. In the current implementation, the number of concurrent jobs that can be sup-

ported is limited by the number of files that can be created in a directory; this limit can easily be increased, if desired.

Parallel job support. 2: *Yes*; 4: *Yes*. Both GRAM2 and GRAM4 support jobs of type MPI.

MPICH-G [9] support. 2: *Yes*; 4: *Yes*. GRAM2 supports multi-jobs (i.e., jobs that span multiple computers) via the client-side DUROC library [10], which performs interprocess "bootstrapping" for id, rank, and barrier via a custom jobmanager protocol. GRAM4 uses a multi-job service, which performs "bootstrapping" via a Rendezvous Web Service.

BES interface. 2: *No*; 4: *Prototyped*. The (soon-to-be-standard) Basic Execution Service (BES) specification [11] can easily be implemented in GRAM4 (and has been prototyped), but not in GRAM2 due to the lack of support for Web Services in the latter system.

3 Performance Comparison

We present, in Tables 2 and 3, GRAM2 and GRAM4 performance data for a variety of scenarios. In each row, shading indicates where one version offers significantly better performance than the other; a lack of shading indicates that the two versions offer similar performance.

We conducted all experiments in an environment comprising two computers connected by Gigabit/s Ethernet. Both the client and server are a 4-CPU Dual Core AMD Opteron™ Processor 275 (2.2GHz). The client computer has 3.4 GB memory and runs Scientific Linux CERN Rel 3.0.8; the server has 4 GB memory and runs RHEL 4.x. The server is also connected by Gigabit/s Ethernet to a 240-node Condor pool running Condor v.6.8.1. We used the GRAM2 and GRAM4 code supplied to the Virtual Data Toolkit (VDT) to build version 1.6.0; this code does not correspond to any released GT version, but will soon be available (with additional improvements) as GT4.0.5.

Table 2 presents results for sequential job submissions, in which jobs are submitted in sequence on the client using command line client programs (globusrun for GRAM2 and globusrun-ws for GRAM4), and executed on the server side using the simplest GRAM LRM, "fork," which simply forks a process to execute the application. We do not stage executables, and we execute a simple job that does not involve computation but can be requested to stage in and/or stage out a single file. We also vary the use of delegation. In each run, 100 jobs are submitted, and the time from first submission to last completion is measured at the client, then divided by 100 to get the average per-job time. We see that GRAM4 is somewhat slower than GRAM2 for simple sequential jobs and considerably slower when file operations are involved. The latter slowdown is due to the use of Web Services calls from the GRAM server to a local RFT service. We have plans to improve this situation and expect to match or exceed GRAM2 performance in the near future.

Table 2: Average seconds/job; sequential scenario.

Delegation	StageIn	StageOut	GRAM2	GRAM4
None	None	None	N/A	1.70
Per job	None	None	1.07	1.71
Per job	1x10KB	None	1.78	5.57
Shared	1x10KB	None	N/A	5.41
Per job	1x10KB	1x10KB	2.44	9.08
Shared	1x10KB	1x10KB	N/A	7.91

Table 3 presents results for concurrent jobs. In these experiments, a single Condor-G [12] client submits 1000 jobs to the server, which uses the GRAM Condor LRM interface to execute the jobs on a Condor pool. There is no client-side throttling of jobs; thus, Condor-G submits the jobs as fast as it can. Again, we do not stage the executable, and execute a simple job that does not involve computation but can be instructed to perform simple file operations.

The times given in Table 3 are from first submission to last completion, as measured by the client. We do not divide by 1000 to obtain a per-job time, as there is presumably some "ramp up" and "ramp down" time at the start and end of the experiment, and thus the resulting numbers would perhaps not be accurate "per-job" times. We note that Condor-G automatically shares delegated credentials for GRAM4 jobs, but cannot do so for GRAM2 jobs. We see again that GRAM4 currently performs somewhat less well than GRAM2 when file operations are involved; however, we are encouraged to see that it performs better than GRAM2 in the absence of staging. As noted in the discussion of the sequential results, we plan optimizations to improve staging performance.

Table 3: Average seconds/1000 jobs; concurrent scenario.

Stage In	Stage Out	File Clean Up	Unique Job Dir	GRAM2	GRAM4
None	None	No	No	2552	2100
1x10KB	1x10KB	No	No	2608	3779
1x10KB	1x10KB	Yes	Yes	2698	5695

4 Relevant GT 4.0.x Development Plans

We outline some of the enhancements to GRAM4 functionality and performance that we plan for imminent 4.0.x releases.

Make audit data available during job execution. Currently, GRAM4 audit records are inserted into the audit database at the end of a job. A patch contributed by Gerson Galang of the Australian Partnership for Advanced Computing inserts each audit record at the beginning of the job, update the record after the LRM job submission, and then again at the end of the job.

Improve performance of staging jobs. We have determined that the performance of file operations (e.g., staging and cleanup) can be improved significantly in the case that the GRAM4 and the RFT service that performs those operations are collocated: we simply replace the Web Services calls to the RFT service with local Java object calls.

5 Relevant GT 4.2.x Development Plans

We outline some of the enhancements to GRAM4 functionality and performance that we plan for future 4.2.x releases.

Support JSDL. We plan an alpha-quality version of the Job Submission Description Language (JSDL) [13] for Q1 07. This work will leverage the current GRAM4 internals that are used with the current custom XML job description language.

Connection caching. The Java Web Services Core performs connection caching for communication between clients and services. This optimization should allow a single client submitting many jobs to the same service to realize a performance improvement without making any code modifications. This improvement has been committed to the Globus software repository and will be included in 4.2.x.

Flexible authorization framework. The attribute processing framework now has richer attribute processing, including a normalized attribute representation to combine attributes about entities from disparate sources. The enhanced authorization framework now allows custom combining algorithms, supports distinct access and administrative rights, and provides a default combining algorithm that uses permit override with delegation of rights to ascertain decision. These improvements have been committed and will be included in GT 4.2.x.

6 Conclusions

Grids around the world have used GRAM2 for remote job submission for years. The implementation of a Web Services-based GRAM has taken time due to the concurrent evolution of Web Services standards. However, those developments are now behind us, and the resulting product is superior. We have finally reached a point at which GRAM4 is to be preferred to GRAM2, for the following reasons:

- GRAM4 provides vastly better functionality than GRAM2, in numerous respects.
- GRAM4 provides better scalability than GRAM2, in terms of the number of concurrent jobs that can be supported. It also greatly reduces load on service nodes, and permits management of that load.
- GRAM4 performance is roughly comparable to that of GRAM2. (We still need to improve sequential submission and file staging performance, and we have plans for doing that, and also for other performance optimizations.)

We encourage those deploying applications and developing tools that require remote job submission services to adopt GRAM4, and to provide feedback on their experiences.

7 Acknowledgments

The authors wish to thank those that contributed in various ways to the work reported in this paper, in particular, Rachana Ananthakrishnan, Joe Bester, Lisa Childers, Jarek Gawor, Carl Kesselman, and Peter Lane. This work was supported in part by the National Science Foundation under contract OCI-0534113 and by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357.

8 References

[1] I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems," in *IFIP International Conference on Network and Parallel Computing*, 2005, pp. 2-13.

[2] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems," in *4th Workshop on Job Scheduling Strategies for Parallel Processing*, 1998, pp. 62-82.

[3] W. Lee, A. S. McGough, and J. Darlington, "Performance Evaluation of the GridSAM Job Submission and Monitoring System," in *UK eScience Program All Hands Meeting*, 2005.

[4] P. Andreetto, S. Borgia, A. Dorigo, and others, "CREAM: A Simple, GRID-Accessible, Job Management System for Local Computational Resources;," in *Computing in High Energy and Nuclear Physics* Mumbai, India, 2006.

[5] G. Wasson and M. Humphrey, "Exploiting WSRF and WSRF.NET for Remote Job Execution in Grid Environments," in *International Parallel and Distributed Processing Symposium* Denver CO, 2005.

[6] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A Security Architecture for Computational Grids," in *5th ACM Conference on Computer and Communications Security*, 1998, pp. 83-91.

[7] W. E. Allcock, I. Foster, and R. Madduri, "Reliable Data Transport: A Critical Service for the Grid," in *Building Service Based Grids Workshop, Global Grid Forum 11*, 2004.

[8] D. Thain and M. Livny, "Building Reliable Clients and Services," in *The Grid: Blueprint for a New Computing Infrastructure (2nd Edition)*: Morgan Kaufmann, 2004.

[9] N. Karonis, B. Toonen, and I. Foster, "MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface," *Journal of Parallel and Distributed Computing*, vol. 63, pp. 551-563, 2003 2003.

[10] K. Czajkowski, I. Foster, and C. Kesselman, "Co-allocation Services for Computational Grids," in *8th IEEE International Symposium on High Performance Distributed Computing*, 1999.

[11] A. Grimshaw and others, "Basic Execution Services (BES) Specification," 2007.

[12] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," *Cluster Computing*, vol. 5, pp. 237-246, 2002.

[13] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva, "Job Submission Description Language (JSDL) Specification V1.0," Open Grid Forum, GFD 56 2005.