DESIGNING GENERAL, COMPOSABLE, AND MIDDLEWARE-INDEPENDENT GRID INFRASTRUCTURE TOOLS FOR MULTI-TIERED JOB MANAGEMENT*

Erik Elmroth, Peter Gardfjäll, Arvid Norberg,
Johan Tordsson, and Per-Olov Östberg
Dept. Computing Science and HPC2N, Umeå University, SE-901 87 Umeå, Sweden
{elmroth, peterg, arvid, tordsson, p-o}@cs.umu.se
http://www.gird.se

Abstract We propose a multi-tiered architecture for middleware-independent Grid job management. The architecture consists of a number of services for well-defined tasks in the job management process, offering complete user-level isolation of service capabilities, multiple layers of abstraction, control, and fault tolerance. The middleware abstraction layer comprises components for targeted job submission, job control and resource discovery. The brokered job submission layer offers a Grid view on resources, including functionality for resource brokering and submission of jobs to selected resources. The reliable job submission layer includes components for fault tolerant execution of individual jobs and groups of independent jobs, respectively. The architecture is proposed as a composable set of tools rather than a monolithic solution, allowing users to select the individual components of interest. The prototype presented is implemented using the Globus Toolkit 4, integrated with the Globus Toolkit 4 and NorduGrid/ARC middlewares and based on existing and emerging Grid standards. A performance evaluation reveals that the overhead for resource discovery, brokering, middleware-specific format conversions, job monitoring, fault tolerance, and management of individual and groups of jobs is sufficiently small to motivate the use of the framework.

Keywords: Grid job management infrastructure, standards-based architecture, fault tolerance, middleware-independence, Grid ecosystem.

*Financial support has been received from The Swedish Research Council (VR) under contract number 621-2005-3667. This research was conducted using the resources of the High Performance Computing Center North (HPC2N).

1. Introduction

We investigate designs for a standards-based, multi-tier job management framework that facilitates application development in heterogeneous Grid environments. The work is driven by the need for job management tools that:

- offer multiple levels of functionality abstraction,
- offer multiple levels of job control and fault tolerance,
- are independent of, and easily integrated with, Grid middlewares,
- can be used on a component-wise basis and at the same time offer a complete framework for more advanced functionality,

An overall objective of this work is to provide understanding of how to best develop such tools. Among architectural aspects of interest are, e.g., to what extent job management functionalities should be separated into individual components or combined into larger, more feature-rich components, taking into account both functionality and performance. As an integral part of the project, we also evaluate and contribute to current Grid standardization efforts for, e.g., data formats, interfaces and architectures. The evaluation of our approach will in the long term lead to the establishment of a set of general design recommendations.

Features of our prototype software include user-level isolation of service capabilities, a wide range of job management functionalities, such as basic submission, monitoring, and control of individual jobs; resource brokering; autonomous processing; and atomic management of sets of jobs. All services are designed to be middleware-independent with middleware integration performed by plug-ins in lower-level components. This enables both easy integration with different middlewares and transparent cross-middleware job submission and control.

The design and implementation of the framework rely on emerging Grid and Web service standards [3],[9],[2] and build on our own experiences from developing resource brokers and job submission services [6],[7],[8], Grid scheduling support systems [5], and the SweGrid Accounting System (SGAS) [10]. The framework is based on WSRF and implemented using the Globus Toolkit 4.

2. A Model for Multi-Tiered Job Submission Architectures

In order to provide a highly flexible and customizable architecture, a basic design principle is to develop several small components, each designed to perform a single, well-defined task. Moreover, dependencies between components are kept to a minimum, and are well-defined in order to facilitate the use of alternative components. These principles are adopted with the overall idea that a

specific middleware, or a specific user, should be able to make use of a subset of the components without having to adopt an entire, monolithic system [11].

We propose to organize the various components according to the following layered architecture.

Middleware Abstraction Layer. Similar to the hardware abstraction layer of an operating system, the middleware abstraction layer provides the functionality of a set of middlewares while encapsulating the details of these. This construct allows other layers to access resources running different middlewares without any knowledge of their actual implementation details.

Brokered Job Submission Layer. The brokered job submission layer offers fundamental capabilities such as resource discovery, resource selection and job submission, but without any fault tolerance mechanisms.

Reliable Job Submission Layer. The reliable job submission layer provides a fault tolerant, reliable job submission. In this layer, individual jobs or groups of jobs are automatically processed according to a customizable protocol, which by default includes repeated submission and other failure handling mechanisms.

Advanced Job Submission & Application Layers. Above the three previously mentioned layers, we foresee both an *advanced job submission layer*, comprising, e.g., workflow engines, and an *application layer*, comprising , e.g., Grid applications, portals, problem solving environments and workflow clients.

3. The Grid Job Management Framework (GJMF)

Here follows a brief introduction to the GJMF, where the individual services and their respective roles in the framework are described.

The GJMF offers a set of services which combined constitute a multi-tiered job submission, control and management architecture. A mapping of the GJMF architecture to the proposed layered architecture is provided in Figure 1.

All services in the GJMF offer a user-level isolation of the service capabilities; a separate service component is instantiated for each user and only the owner of a service component is allowed to access the service capabilities. This means that the whole architecture supports a decentralized job management policy, and strives to optimize the performance for the individual user.

The services in the GJMF also utilize a local call structure, using local Java calls whenever possible for service-to-service interaction. This optimization is only possible when the interacting services are hosted in the same container.

The GJMF supports a dynamic one-to-many relationship model, where a higher-level service can switch between lower-level service instances to improve fault tolerance and performance.

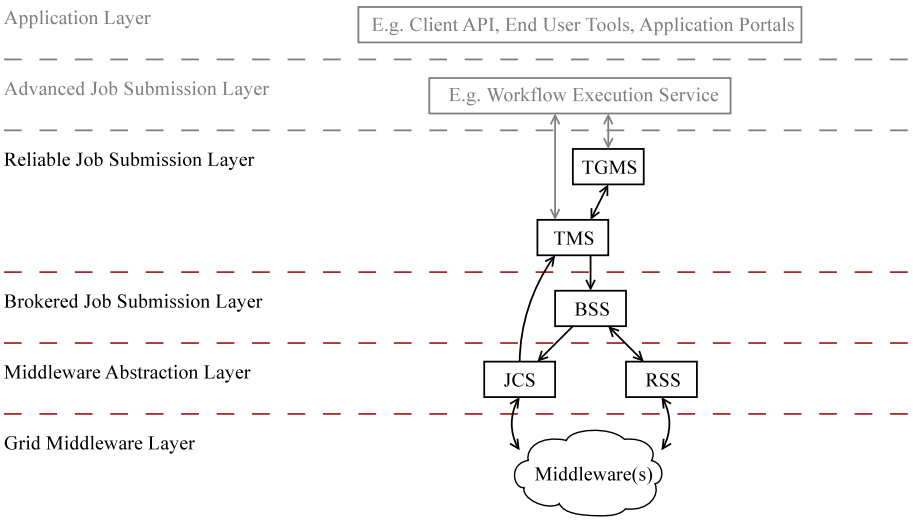

Figure 1. GJMF components mapped to their respective architectural layers.

As a note on terminology, there are two different types of job specifications used in the GJMF: abstract *task* specifications and concrete *job* specifications. Both are specified in JSDL [3], but vary in content. A job specification includes a reference to a computational resource to process the job, and therefore contains all information required to submit the job. A task specification contains all information required except a computational resource reference. The act of brokering, the matching of a job specification to a computational resource, thus transforms a task to a job.

Job Control Service (JCS). The JCS provides a functionality abstraction of the underlying middleware(s) and offers a platform- and middleware-independent job submission and control interface. The JCS operates on jobs and can submit, query, stop and remove jobs. The JCS also contains customization points for adding support for new middlewares and exposes information about jobs it controls through WSRF resource properties, which either can be explicitly queried or monitored for asynchronous notifications. Note that this functionality is offered regardless of underlying middleware, i.e., if a middleware does not support event callbacks the JCS explicitly retrieves the information required to provide the notifications. Currently, the JCS supports the GT4 and the ARC middlewares.

Resource Selection Service (RSS). The RSS is a resource selection service based on the OGSA Execution Management Services (OGSA EMS) [9]. The OGSA EMS specify a resource selection architecture consisting of two services, the Candidate Set Generator (CSG) and the Execution Planning Service (EPS).

The purpose of the CSG is to generate a candidate set, containing machines where the job *can* execute, whereas the EPS determines where the job *should* execute. Upon invocation, the EPS contacts the CSG for a list of candidate machines, reorders the list according to a previously known or explicitly provided set of rules and returns an *execution plan* to the caller.

The current OGSA EMS specification is incomplete, e.g., the interface of the CSG is yet to be determined. Due to this, the CSG and the EPS are in our implementation combined into one service - the RSS. The candidate set generation is implemented by dynamical discovery of available resources using a Grid information service, e.g., GT4 WS-MDS, and filtering of the identified resources against the requirements in the job description. The RSS contains a caching mechanism for Grid information, which alleviates the frequency of information service queries.

Brokering & Submission Service (BSS). The BSS provides a functionality abstraction for brokered task submission. It receives a task (i.e., an abstract job specification) as input and retrieves an execution plan (a prioritized list of jobs) from the RSS. Next, the BSS uses a JCS to submit the job to the most suitable resource found in the execution plan. This process is repeated for each resource in the execution plan until a job submission has succeeded or the resource list has been exhausted. A client submitting a task to the BSS receives an EPR to a job WS-Resource in the JCS as a result. All further interaction with the job, e.g., status queries and job control is thus performed directly against the JCS.

Task Management Service (TMS). The TMS provides a high-level service for automated processing of individual tasks, i.e., a user submits a task to the TMS which repeatedly sends the task to a known BSS until a resulting job is successfully executed or a maximum number of attempts have been made. Internally, the TMS contains a per-user job pool from which jobs are selected for sequential submission. The TMS job pool is of a configurable, limited size and acts as a task submission throttle. It is designed to limit both the memory requirements for the TMS and the flow of job submissions to the JCS. The job submission flow is also regulated via a congestion detection mechanism, where the TMS implements an incremental back-off behavior to limit BSS load in situations where the RSS is unable to locate any appropriate computational resources for the task. The TMS tracks job progress via the JCS and manages a state machine for each job, allowing it to handle failed jobs in an efficient manner. The TMS also contains customization points where the default behaviors for task selection, failure handling and state monitoring can be altered via Java plug-ins.

Task Group Management Service (TGMS). Like the TMS for individual tasks, the TGMS provides an automated, reliable submission solution for groups of tasks. The TGMS relies on the TMS for individual task submission and offers a convenient way to submit groups of independent tasks. Internally, the TGMS contains two levels of queues for each user. All task groups that contain unprocessed tasks are placed in a task group queue. Each task group queue, in turn, contains its own task queue. Tasks are selected for submission in two steps: first an active task group is selected, then a task from this task group is selected for submission. By default, tasks are resubmitted until they have reached a terminal state (i.e., succeeded or failed). A task group reaches a terminal state once all its tasks are processed. A task group can also be suspended, either explicitly by the user or implicitly by the service when it is no longer meaningful to continue to process the task group, e.g., when associated user credentials have expired. A suspended task group must be explicitly resumed to become active. The TGMS contains customization points for changing the default behaviors for task selection, failure handling and state monitoring.

Client API. The Client API is an integral part of the GJMF; it provides utility libraries and interfaces for creating tasks and task groups, translating job descriptions, customizing service behaviors, delegating credentials and contains service-level APIs for accessing all components in the GJMF. The purpose of the GJMF Client API is to provide easy-to-use programmable (Java) access to all parts of the GJMF.

For further information regarding the GJMF, including design documents and technical documentation of the services, see [12].

4. Performance Evaluation

We evaluate the performance of the TGMS and the TMS by investigating the total cost imposed by the GJMF services compared to the total cost of using the native job submission mechanism of a Grid middleware, GT4 WS-GRAM (without performing resource discovery, brokering, fault recovery etc.).

In the reference tests with WS-GRAM, a client sequentially submits a set of jobs using the WS-GRAM Java API, delaying the submission of a job until the previous one has been successfully submitted. All jobs run the trivial /bin/true command and are executed on the Grid resources using the POSIX Fork mechanism. The jobs in a test are distributed evenly among the Grid resources using a round-robin mechanism. The WS-GRAM tests do not include any WS-MDS interaction. No job input or output files are transferred and no credentials are delegated to the submitted jobs. In each test, the total wall clock time is recorded. Tests are performed with selected numbers of jobs, ranging from 1 to 750.

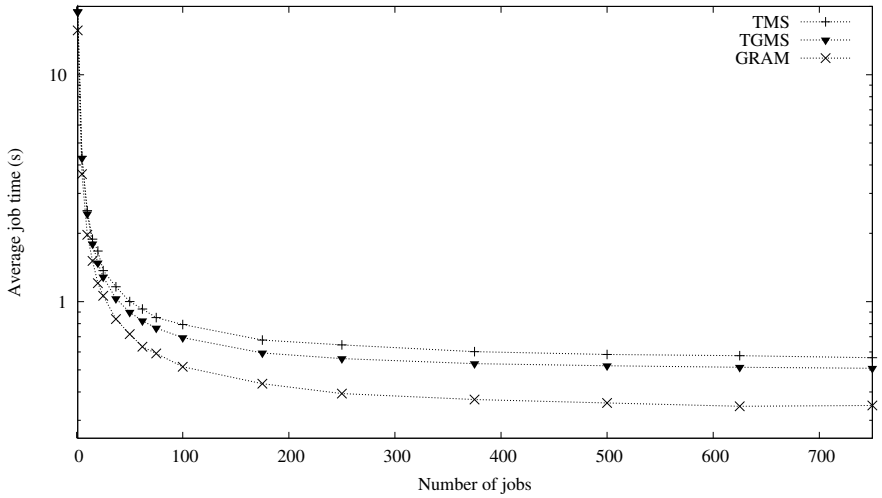

Figure 2. GRAM and GJMF job processing performance.

The configuration of the GJMF tests is the same as for the WS-GRAM tests, with the following additions. For the TGMS tests, user credentials are delegated from the client to the service for each task group (each test). Delegation is also performed only once per test in the TMS case, as all jobs in a TMS test reuse the same delegated credentials. For both the TGMS and the TMS tests, the BSS performs resource discovery using the GT4 WS-MDS Grid information system and caches retrieved information for 60 seconds. In the TMS and TGMS tests, all services are co-located in the same container, to enable the use of local Java calls between the services, instead of (more costly) Web service invocations.

Test Environment. The test environment includes four identical 2 GHz AMD Opteron CPU, 2 GB RAM machines, interconnected with a 100 Mbps Ethernet network, and running Ubuntu Linux 2.6 and Globus Toolkit 4.0.3.

In all tests, one machine runs the GJMF (or the WS-GRAM client) and the other three act as WS-GRAM/GT4 resources. For the GJMF tests, the RSS retrieves WS-MDS information from one of the three resources, which aggregates information about the other two.

Analysis. Figure 2 illustrates the average time required to submit and execute a job for different number of jobs in the test. As seen in the figure, the TGMS offers a more efficient way to submit multiple tasks than the TMS. This is due to the fact that the TMS client performs one Web service invocation per task whereas the TGMS client only makes a single, albeit large, call to the TGMS. The TGMS client requires between 13 (1 task) and 16.6 seconds (750 tasks) to delegate credentials, invoke the Web service and get a reply. For the TMS,

the initial Web service call takes roughly 13 seconds (as it is associated with dynamic class-loading, initialization and delegation of credentials), additional calls average between 0.4 and 0.6 seconds. For the GRAM client, the initial Web service invocation takes roughly 12 seconds. The additional TMS Web service calls quickly become the dominating factor as the number of jobs are increased. When using Web service calls between the TGMS and the TMS this factor is canceled out. Conversely, when co-located with the TMS and using local Java calls, the TGMS only suffers a negligible overhead penalty for using the TMS for task submission. In a test with 750 jobs, the average job time is roughly 0.35 seconds for WS-GRAM, and approximately 0.51 and 0.57 seconds for the TGMS and TMS, respectively.

As the WS-GRAM client and the JCS use the same GT4 client libraries, the difference between the WS-GRAM performance and that of the other services can be used as a direct measure of the GJMF overhead.

In the test cases considered, the time required to submit a job (or a task) can be divided into three parts.

1 The initialization time for GT4 Java clients. This includes time for class loading and run-time environment initialization. This time may vary with the system setup but is considered to be constant for all three test cases.
2 The time required to delegate credentials. This only applies to the GJMF tests, not the test of WS-GRAM. Even though delegated credentials are shared between jobs, the TMS is still slightly slower than the TGMS in terms of credential delegation. The TMS has to retrieve the delegated credential for each task, whereas the TGMS only retrieves the delegated credential once per test.
3 The Web service invocation time. This factor grows with the size of the messages exchanged and affects the TGMS, as a description of each individual task is included in the TGMS input message. The invocation time is constant for the TMS and WS-GRAM tests, as these services exchange fixed size messages.

Summary. When co-hosted in the same container, the GJMF services allots an overhead of roughly 0.2 seconds per task for large task groups (containing 750 tasks or more). The main part of this overhead is associated with Java class loading, delegation of credentials and initial Web service invocation. These factors result in larger average overheads for smaller task groups. For task groups containing 5 tasks, the average overhead per task is less than 1 second, and less than 0.5 seconds for 15 tasks. It should also be noted that, as jobs are submitted sequentially but executed in parallel, the submission time (including the GJMF overhead), is masked by the job execution time. Therefore, when using real world applications with longer job durations than those in the tests, the impact of the GJMF overhead is reduced.

5. Related Work

We have identified a number of contributions that relate to this project in different ways. For example, the Gridbus [16] middleware includes a layered architecture for platform-independent Grid job management; the GridWay Metascheduler [13] offers reliable and autonomous execution of jobs; the GridLab Grid Application Toolkit [1] provides a set of services to simplify Grid application development; GridSAM [15] offers a Web service-based job submission pipeline which provides middleware abstraction and uses JSDL job descriptions; P-GRADE [14] provides reliable, fault-tolerant parallel program execution on the grid; and GEMLCA [4] offers a layered architecture for running legacy applications through grid services. These contributions all include features which partially overlap the functionality available in the GJMF. However, our work distinguishes itself from these contributions by, in the same software, providing i) a composable service-based solution, ii) multiple levels of abstraction, iii) middleware-interoperability while building on emerging Grid service standards.

6. Concluding Remarks

We propose a multi-tiered architecture for building general Grid infrastructure components and demonstrate the feasibility of the concept by implementing a prototype job management framework. The GJMF provides a standards-based, fault-tolerant job management environment where users may use parts of, or the entire framework, depending on their individual requirements. Furthermore, we demonstrate that the overhead incurred by using the framework is sufficiently small (approaching 0.2 seconds per job for larger groups of jobs) to motivate the practical use of such an architecture. Initial tests demonstrate that by proper methods, including reuse of delegated credentials, caching of Grid information and local Java invocations of co-located services, it is possible to implement an efficient service-based multi-tier framework for job management. Considering the extra functionality offered and the small additional overhead imposed, the GJMF framework is an attractive alternative to a pure WS-GRAM client for the submission and management of large numbers of jobs.

Acknowledgments

We are grateful to the anonymous referees for constructive comments that have contributed to the clarity of this paper.

References

[1] G. Allen, K. Davis, K. N. Dolkas, N. D. Doulamis, T. Goodale, T. Kielmann, A. Merzky, J. Nabrzyski, J. Pukacki, T. Radke, M. Russell, E. Seidel, J. Shalf, and I. Taylor. Enabling

applications on the Grid - a GridLab overview. *Int. J. High Perf. Comput. Appl.*, 17(4), 2003.

[2] S. Andreozzi, S. Burke, L. Field, S. Fisher, B. Kónya, M. Mambelli, J. M. Schopf, M. Viljoen, and A. Wilson. Glue schema specification version 1.2 draft 7. http://glueschema.forge.cnaf.infn.it/uploads/Spec/GLUEInfoModel_1_2_final.pdf, March 2007.

[3] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, A. S. McGough, D. Pulsipher, and A. Savva. Job Submission Description Language (JSDL) specification, version 1.0. http://www.ogf.org/documents/GFD.56.pdf, March 2007.

[4] T. Delaittre, T. Kiss, A. Goyeneche, G. Terstyanszky, S.Winter, and P. Kacsuk. GEMLCA: Running legacy code applications as Grid services. *Journal of Grid Computing*, 3(1 – 2):75 – 90, June 2005. ISSN: 1570-7873.

[5] E. Elmroth and P. Gardfjäll. Design and evaluation of a decentralized system for Grid-wide fairshare scheduling. In H. Stockinger, R. Buyya, and R. Perrott, editors, *e-Science 2005, First International Conference on e-Science and Grid Computing*, pages 221–229. IEEE CS Press, 2005.

[6] E. Elmroth and J. Tordsson. An interoperable, standards-based Grid resource broker and job submission service. In H. Stockinger, R. Buyya, and R. Perrott, editors, *e-Science 2005, First International Conference on e-Science and Grid Computing*, pages 212–220. IEEE CS Press, 2005.

[7] E. Elmroth and J. Tordsson. A standards-based Grid resource brokering service supporting advance reservations, coallocation and cross-Grid interoperability. Submitted to *Concurrency and Computation: Practice and Experience*, December 2006.

[8] E. Elmroth and J. Tordsson. Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions. *Future Generation Computer Systems. The International Journal of Grid Computing: Theory, Methods and Applications*, 2007, to appear.

[9] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. Von Reich. The Open Grid Services Architecture, version 1.5. http://www.ogf.org/documents/GFD.80.pdf, March 2007.

[10] P. Gardfjäll, E. Elmroth, L. Johnsson, O. Mulmo, and T. Sandholm. Scalable Grid-wide capacity allocation with the SweGrid Accounting System (SGAS). Submitted to *Concurrency and Computation: Practice and Experience*, October 2006.

[11] Globus. An "Ecosystem" of Grid Components. http://www.globus.org/grid_software/ecology.php. March 2007.

[12] Grid Infrastructure Research & Development (GIRD). http://www.gird.se. March 2007.

[13] E. Huedo, R.S. Montero, and I.M. Llorente. A framework for adaptive execution on Grids. *Software - Practice and Experience*, 34(7):631–651, 2004.

[14] P. Kacsuk, G. Dózsa, J. Kovács, R. Lovas, N. Podhorszki, Z. Balaton, and G. Gombás. P-GRADE: a Grid programming environment. *Journal of Grid Computing*, 1(2):171 – 197, 2003.

[15] W. Lee, A. S. McGough, and J. Darlington. Performance evaluation of the GridSAM job submission and monitoring system. In *UK e-Science All Hands Meeting*, Nottingham, UK, 2005.

[16] S. Venugopal, R. Buyya, and L. Winton. A Grid service broker for scheduling e-Science applications on global data Grids. *Concurrency Computat. Pract. Exper.*, 18(6):685–699, May 2006.