High Performance Distributed Computing (HPDC-11), Edinburgh, Scotland, July 24-26, 2002

InfoGram: A Grid Service that Supports Both Information Queries and Job Execution

Gregor von Laszewski[1], Ian Foster[1], Jarek Gawor[1], Andreas Schreiber[1,2], Carlos J. Peña[1,3]
[1] Argonne National Laboratory, 9700 S. Cass Ave, Argonne IL 60439, U.S.A.
[2] German Aerospace Center (DLR), Linder Höhe, 51147 Cologne, Germany
[3] State University of New York at Stony Brook, Stony Brook, NY 11794, U.S.A.
gregor@mcs.anl.gov

Contents

InfoGram: A Grid Service that Supports Both Information Queries and Job Execution

Gregor von Laszewski[1], Jarek Gawor[1], Carlos J. Peña[1,2] and Ian Foster[1]
[1] Argonne National Laboratory, 9700 S. Cass Ave, Argonne IL 60439, U.S.A.
[2] State University of New York at Stony Brook, Stony Brook, NY 11794, U.S.A.
gregor@mcs.anl.gov

Abstract

The research described in this paper is performed as part of the Globus Project. It introduces a new Grid service called InfoGram that combines the ability of serving as information service and as a job execution service. Previously, both services were architected and implemented within the Globus Toolkit as two different services with different wire protocols. Our service demonstrates a significant simplification of the architecture while treating job submissions and information queries alike. The advantage of our service is that it provides backwards compatibility to existing Grid services, while at the same time providing forwards compatibility to the emerging Web services world. Part of the work conducted within this effort is already reused by the current Open Grid Services Architecture prototype implementation.

1. Introduction

The Grid approach is an important development in the discipline of computer science and engineering[30]. It is making rapid progress on several levels, including the definition of terminology, the design of an architecture and framework [13], the application in scientific problems [5, 4], and the creation of physical instantiations of Grids on a production level [10, 3, 2]. Grids provide an infrastructure that allows for flexible, secure, coordinated resource sharing among dynamic collections of individuals, resources, and organizations.

Over the past few years, the Globus Project has developed the Globus Toolkit [18] that provides a basic Grid middleware toolkit, which includes elementary services to address Grid management issues related to resource management, security, information, and data management [30]. Two of the most important Grid services that are provided by the Globus Toolkit are the information service and the job execution service.

The *information service* returns information about the capabilities and the state of the Grid infrastructure. The Globus Toolkit provides such an information service called Monitoring and Directory Service (MDS) [31, 7], formerly known as Metacomputing Directory Service.

The *job execution service* controls the submission and execution of jobs on remote machines. The Globus Toolkit provides such a service under the name Grid Resource Allocation and Management (GRAM) service [9]. GRAM processes requests for execution, performs resource allocation, monitors, and controls job execution. Furthermore, a limited amount of information related to the capabilities and availability regarding the job execution service for a Grid resource can be exposed through an information provider to the MDS. This information includes, for example, the name of the queue, details about the mode of operation, and other important features that may guide the process of job submission by the user. Authentication to MDS and GRAM are handled through the Grid Security Infrastructure (GSI) [6].

The information and job execution service have so far existed as separate services within the Globus Toolkit. Considerable software engineering effort is necessary to implement, maintain, and deploy these services while at the same time support interoperability. We argue that this complexity can be reduced significantly by alternative approaches to both protocol design and implementation. To test this hypothesis, we developed a prototype that promises a significant simplification in all aspects previously mentioned. We have termed our prototype InfoGram in order to acknowledge its dual purpose.

Our research has the following objectives and goals:

- Design of a simplified Grid service architecture to provide a unified service for information, monitoring, and job submission.
- Develop this service while providing backwards compatibility by adhering to standard Grid protocols.

- Support multiple information return request formats such as LDIF and XML.
- Improve the reliability of the job execution and in a second phase while replacing the protocol used to perform the Job submission with SOAP.
- Provide an open framework that can be easily adapted to interact with local schedulers and extract information through custom designed information providers.
- Provide a framework that is based on GSI and its application within the Globus toolkit to map Global Grid User Identifiers to local account names.
- Develop this service while providing forwards compatibility to Web services.
- Build a groundwork for a Web services based implementation of Globus services.

The rest of this paper is structured as follows: First, we discuss the Globus Toolkit services GRAM and MDS in more detail. We outline how the operational integration of these services is achieved in production Grids. Next, we present the enhancements to the GRAM service that allow constructing our InfoGram service. We demonstrate that our service provides a significant architectural simplification but at the same time provides enhancements currently not available in the Globus Toolkit. Additionally, we show that this new service can still be integrated into the existing MDS concept. Finally, we outline how such a service can be used as part of Grid applications.

2. Execution Service

To contrast our differences to the Globus GRAM it is necessary to revisit the architecture of the Globus GRAM service. The basic structure of a GRAM service (version 1.1.x) and its interaction with clients relevant for our discussion is depicted in Figure 1. A GRAM service provides the basic functionality for secure and uniform access to remote computational resources. The functionality of GRAM can be explained as part of a typical three tier architecture. Before we include our enhancements to this architecture (Section 6), we explain the functionality of each tier in more detail.

Client Tier. A client can submit a job to a remote resource and can check on its status either through polling the status of the job or through event notification to the client through the GRAM Service. To allow identification of the job, a job handle (often referred to GlobusID) is returned on job startup so that it can be used for later connection, including from other remote clients with appropriate authorization. For example, this job handle can be used to contact the job and issue a cancellation.

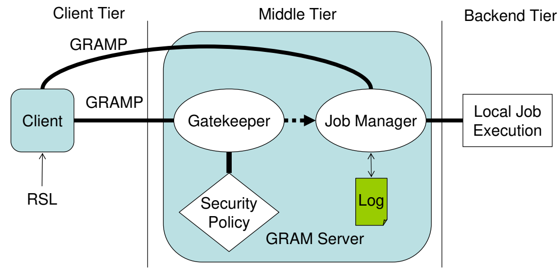

Figure 1. The Gram Architecture

Middle Tier. Internally, GRAM consists of a gatekeeper and a job manger. The gatekeeper is responsible for authentication with the client, performing a simple authorization based on mapping the authentication information into a local security context (e.g., a Unix login). After this initial security check, it starts up a job manager that interacts thereafter with the client based on the GRAM protocol (from now on referred to as GRAMP). Each job submitted by a client to the same GRAM will start its own job manager.

Backend Tier. Once the job manager is activated, it handles the communication between the client and the backend system on which the job is executed. The backend tier is easily portable to various scheduling systems. The Globus Toolkit services provide scheduling interfaces such as PBS, LSF, Condor, and Unix process fork [21, 18].

The GRAM service can be accessed with the help of a C or a Java application interface. This interface includes the ability to specify a job runable on a particular resource with the help of a uniform Resource Specification Language (RSL). The RSL makes it possible to quickly and uniformly specify jobs to be run as part of a Globus enabled Grid. Simple tools are available to access the basic functionality also from the command line.

Although, we have in the past demonstrated mechanisms and protocols for application states and notification, such advanced functionality [32] has not yet been included in the Globus Toolkit.

3. Information Service

The basic structure of a Grid information service is defined in [31] and was further refined in [8]. A Grid information service requires:

- access to static and dynamic information regarding system components and services,

- a framework that fits well with the heterogeneous and dynamic nature of Grids, including decentralized maintenance and operation,

- scalability and performance,

- integration of a variety of information providers.

The Globus Project has developed a basic information service that addresses these requirements. The Globus Grid information service, MDS, contains two fundamental entities: distributed information providers and information aggregates. An information provider is a service that provides a subset of useful information about resources exploited by Grid users or Grid services. Examples of information that may be accessed through such an information provider is CPU, operating system, network, and file system information.

Additionally, the aggregate service is used to integrate a set of information providers that may be part of a virtual organization [14]. To increase the scalability of a distributed information service, the MDS provides an information caching function that allows viewing and querying the information about a resource from a cache. Furthermore, the newest implementation of a Grid information service that implements the framework proposed by the MDS concept integrates GSI to perform authentication.

The information contained within MDS can be queried and used to enable more sophisticated Grid services. More details about the protocols, the services, and the newest nomenclature can be found in [17, 7].

The research within this paper concentrates on the information provider itself, as we can create information aggregates through reuse of information providers to improve scalability. Furthermore, we argue that it is worthwhile to provide *google-like* services, as have been used in many previous Grid like projects [28, 29, 11].

4. Using GRAM and MDS in Production Grids

Figure 2 shows how the GRAM and the MDS services may be used in a simple production Grid. Our Grid consists of one virtual organization that maintains a number of compute resources. Each compute resource has the Globus GRAM and the Globus Resource Information Service (GRIS) that returns information related to the local resource installed.

In order for a client to perform a job execution and an information query, two different mechanisms for contacting these services must be used. Not only do the services operate through different ports, but they also use different protocols making the amount of code sharing for interpreting return values more complex. The installation of both services requires additional sophistication. We feel that the

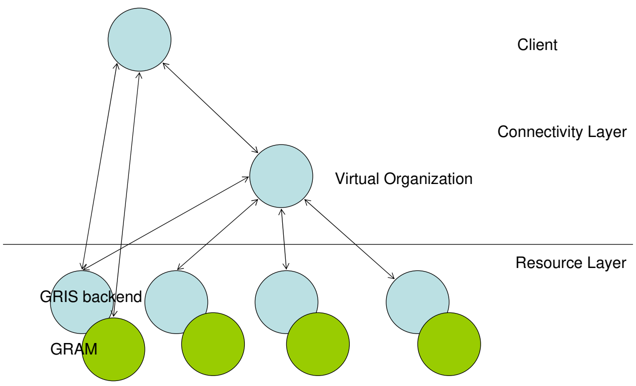

Figure 2. A sample interaction between a client, GRAM, and MDS

use of different technologies is in contrast with the desire to provide a minimal set of protocols and services for Grids as promoted by the Global Grid Forum and the Globus Project [14]. If we think abstractly about *job execution and an information* service, we must recognize that they *are based on* the same principle: *A query formulated and submitted to a server followed by a stream of information that returns the result based on the query.*

5 Addressing Requirements for the InfoGram Service

We have designed our InfoGram service according to a set of requirements determined by general software engineering practices which include factors such as quality, performance, reliability, security, and portability. All of these factors must be addressed within the realm of Grids. Nevertheless, we concentrate our efforts on the following issues.

5.1. Performance

An information and Job execution service must perform their tasks quickly. The elapsed time between job request and job submission must be as short as possible. At the same time, information within the system must be accessible quickly. For example, it may be inefficient to execute each time a user requests data the program creating the data or a query relayed to an external information service. A simple example will illustrate our point. Assume we have a large number of clients that need to know the CPU load of a remote compute resource. It would be wasteful to execute the command requesting the load every single time. Instead, it can be more efficient to cache this value within the information service, and only refresh this cache value periodically. In order to prevent staleness of information we attach a time to live (TTL) value with the information. This value will tell us when a refresh of the information in the cache is necessary.

5.2. Quality of Information

Information within Grids may become quickly inaccurate. We often observe two cases. Case One: In the simplest form the information can be describe as binary system where the information is either accurate or inaccurate. Case Two: In many other situations the information may degrade over time in a discrete fashion. Thus, it is not unreasonable to attach a degradation function with the actual value of information that reflects the degree of degradation. This function may be influenced by time, system state, or prediction functions to derive a quality of information assessment. Often it is possible to attempt to derive such degradation information through observation or through mathematical models while performing self correction based on observation data. This is not unlike sophisticated data assimilation as used in climate research that corrects its values based on a comparison between observations and prediction models. The quality of information becomes important in case more sophisticated resource management strategies are developed. If I obtain an attribute such as "mean CPU load" from a Grid information service, it is beneficial to have the quality of the information attached. Knowing the standard deviation or knowing that the accuracy of the value is valid over the last hour or the last day is an important factor to create more sophisticated Grid services.

5.3. Security

Access to services such as the information and job execution needs to be performed securely. The Grid Security Infrastructure (GSI) provides us with an elementary framework for authentication. Nevertheless, authentication is only one problem to be addressed within Grids. In our framework, we strive to include authorization that allows us to specify contracts such as "allow access to this resource from 3 to 4 pm to user X." Additions to GSI and the use of more sophisticated authentication frameworks [27] may provide them in the future.

5.4. Portability

Protocol compatibility of these services is preserved with the Globus Toolkit while using the GRAM, and Grid Security Infrastructure (GSI) protocols. Future activities will include the integration of commodity protocols (such as SOAP) to provide interoperability to Web services and greater acceptance outside of the Grid community [15, 36, 38, 37].

5.5. Flexible and Extensible Information Model

One of the issues we face with information providers is the lack of a standard that is uniformly adhered by the community. We observe the use of CIM, MIB, MDS, or non standard or unorthodox display of information in tables. Although we believe that the creation of a consistent information model is an important one, we focus within this paper on the mechanism of delivering that information to the user. The reasoning for this strategy is that our InfoGram service provides the necessary mechanisms for delivering the information according to the information model used within the information provider. Our positive experience with the use of XML schemas as basis for the next generation of Information services makes us believe that it provides a viable alternative to the currently used LDAP schemas. Compatibility can be maintained while developing strict guidelines for the object definition by the Global Grid Forum.

Nevertheless, we believe that an additional requirement must be fulfilled to enhance the use and acceptance of Grids. We believe that the execution of untrusted applications in trusted environments is important to enable the use of Grids. We hope that through this feature the user community will increase dramatically based on software that is developed as part of our activities.

Providing such software will enable the creation of infrastructures that will promote Grids in new communities, which previously did not have the luxury to access high end resources. Besides making access to supercomputer centers for outside users much more feasible, we foresee that resource providers may be more willing to contribute resources otherwise not part of the national-scale Grids.

6. InfoGRAM Architecture

As pointed out earlier, we modified the architecture of the GRAM server and enhanced it substantially in order to fulfill the requirements described earlier. We added to the original architecture additional components, as shown by the shaded components in Figure 3, and describe these enhancements based on the functionality they are providing. These functionalities are centered on client interaction, logging and check pointing, job execution, information management, and configuration.

Logging and check pointing is enabled through a logging service. This service can receive logging events from several components. The log can either be stored in the middle tier, or on the backend tier. In either case the log can be used to restart our InfoGRAM service in case it needs to be restarted (e.g. the machine was shut down). In the same way it would be possible to use the logging service for check pointing of applications. Presently, we only record minimal information such as the command used and arguments

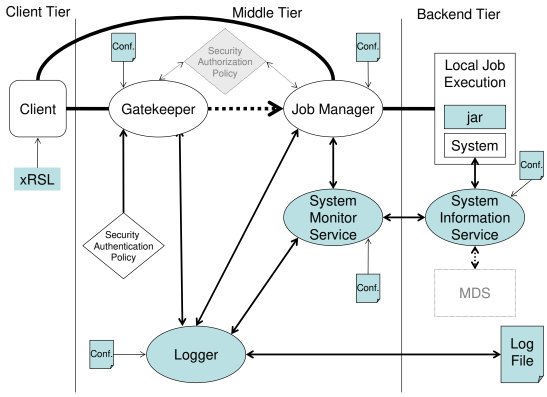

Figure 3. The InfoGram architecture that combines a GRAM service with an Information Service using only one protocol between client and server.

executed. We intend to use this logging service to provide simple Grid accounting.

6.1. Job execution

The execution of jobs is made more robust while integrating a logging and fault tolerance mechanism that allows to restart a job upon failure.

6.2. Information Service

As mentioned previously, the InfoGram Service contains several novel features in regards to the information service part it provides. These features are: An Information service that is integrated with GRAM providing backwards compatibility to MDS, and support of information caching and the retrieval of elementary information associated with the remote resource. Additionally, we are integrating in our service the feature of information degradation and self adaptation of information updates as discussed earlier.

Our information service is architected with two components (see Figure 3): the system monitor and the system information service. The monitor service controls initializing and caching the results requested by the clients. The system information service returns relevant information about the system resources, through either (a) calls to a system command via the Java runtime exec (b) a query to a function exposing Java runtime information such as load, memory, or disk space (c) or a read function from a file that is used by an information provider. A good example for an information provider is the Linux proc file system. As we have chosen an object oriented framework for our implementation, the integration of new information providers can be performed through the implementation of interfaces. This will allow us to be able to provide a flexible and extensible information services framework.

```
class SystemInformation interface {
    String getKeyword();
    void setKeyword();
    Object queryState();
    Object updateState();
    Time ttl ();
    int validity();
    Public void setDelay(Time time);
    String setFormat(Format format);
    Time getAverageUpdateTime();
}
```

This interface allows us to generate new information providers in a fashion very similar to the current MDS model and its implementation. The method `queryState` is non blocking and returns valid information only when the information has been queried previously and the time to live (ttl) value has not expired. Otherwise, it throws an exception. Upon invocation of the `updateState` method, a blocking method is called that returns the appropriate information while also updating the time to live value. If multiple `updateState` methods are invoked, monitors are used to perform only one such update at a time. Additionally, we provide a delay that controls how many milliseconds must pass between consecutive calls of `updateState` before the actual information is obtained through a runtime exec call. This is useful in cases where users ask for information more frequently than it can be produced by the system.

6.3. Configuration

We provided a configuration component that allows us to setup the InfoGram service with ease. This component includes the possibility to configure the system monitor service with customized information providers similar to the MDS. This configuration file contains the following parameters:

TTL: the lifetime in millisecond of each data generated by the specific key word; 0 specifies execution of the keyword every time it is requested.

Keyword: the keyword that will be used in an RSL string to identify the mapping to a real program or a Java application to be executed in the background.

Executable Path: the full executable path and name with arguments, machine dependant that is associated with the keyword.

Table 1. The InfoGram configuration file provides a mapping between keywords and information providers

TTL	Keyword	Command
60	Date	date -u
80	Memory	/sbin/sysinfo.exe -mem
100	CPU	/sbin/sysinfo.exe -cpu
0	CPULoad	/usr/local/bin/cpuload.exe
1000	list	/bin/ls /home/gregor

We provide an example for the information represented within such a configuration file in Table 1. As the keyword identifies the information obtained with the program we will refer to it from now on as a key information provider. Each attribute within a key information provider is augmented with a namespace conform to the keyword. Thus the attribute "total" in the "Memory information provider" would be referred to as `Memory:total`.

6.4. Caching and Information Degradation

The caching functionality is similar to that of the MDS 2.0. Nevertheless, queries to the information service are simple all-or-nothing queries based on the keywords used within the configuration files. That means, all attributes that are obtained through the command associated with a keyword will be returned. Based on this simple model. the caching of information is easily possible. Additionally, we have the option to augment each attribute that is returned within a key information provider with a degradation function or a quality of information value. Selecting similar information attributes can then be performed on the quality of the information provided.

6.5. Service Reflection

Each information service can be queried and a client may inspect the schema that is returned by the information service. Thus it will allow developers to design programs that can be flexible to the actually used information schema. We believe that reflection and introspection of the capabilities of an execution and information service will become increasingly important with the increased number of available Grid services.

6.6. Client Interaction through xRSL

Although we developed our first prototype architecture as a Web service, we believed at the time that it would provide to big of a departure from the existing Globus Toolkit. We thought that most important for the acceptance of our information service, is the recognition that the Globus Toolkit reached ubiquity within the community, and that the Globus protocols should be reused.

Thus, as we wanted to maintain a degree of backwards compatibility, we decided not to chose a pure Web services-based implementation that uses only WSDL [36], XML-schema [39], and XML query. We felt that such an effort could be performed in a second step (as it is now performed as part of the Open Grid Service Architecture [1]). Instead of using URIs to formulate job submission and information queries, we argued that users of the Globus Toolkit are sufficiently familiar with RSL. Therefore, it was most natural to extend RSL with the more advanced features we have introduced so far. We added the following tags to the Globus RSL: *schema, info, filter, response, performance, quality*, and *format*. We call the result xRSL.

Info. The *info* tag is followed by the *key* as specified in the configuration file, defining a mapping between the keyword and the command to be executed. If it is set to `(info=all)`, all commands are executed. Commands can be selectively queried while concatenating multiple info tag queries, for example, `(info=Memory)(info=CPU)`. A special value for the info tag is `(info=schema)`. This returns a hierarchical schema that contains all objects associated with the keywords and lists properties of their attributes.

Response. The *response* tag defines the behavior with respect to the information caching. Thus, with `(response=immediate)` the commands associated with the info tag are executed immediately regardless of the time to live. This will also update the cached values. Using `(response=cached)` will return the information from the cache value if it is valid; otherwise it will update the cache first. Using `(response=last)` will return the value stored last in the cache without updating it.

Quality. The *quality* threshold tag provides the possibility to specify a percentage number that gives additional guidance if a cached value should be returned or if the information needs to be refreshed before return. Currently, we define the following semantic. If the degradation function of any of its returned attributes is below that threshold, this attribute is regenerated by the associated command.

Performance. The *performance* tag returns the number of seconds and the standard deviation about how long it takes to obtain a particular information value. The performance of a command and its attributed values is measured and catalogued during runtime.

Format. The *format* tag defines the format in which the information is returned. The supported formats are LDIF and XML. Nevertheless, it is straightforward to support other formats such as DSML.

Extensions. We are planning to extend our existing timeout tag with an additional action tag upon reaching this timeout. For example, the RSL `(executable=command)(timeout=1000)` `(action=cancel)` would cancel the command specified through the RSL, while `(action=exception)` would throw an exception if the command has not completed is execution, but the execution of the command itself would be continuing.

Advantages. The advantages of this information stem from the simplification of the architecture bound to the delivery of an integrated job submission and information service. Querying the information is handled by clients much as the execution of jobs. Moreover, this information service can easily be integrated into the Globus MDS information service architecture.

In summary, we have explored changes to MDS at the protocol and the implementation level. At the protocol level we have replaced an LDAP search query with a "query" cast as a simple job submission through RSL. At the implementation level, we have replaced the modular, configurable MDS information provider architecture with a less complex, even more modular, configurable architecture that we believe fulfills, in a straightforward fashion, the Grid designers quest for an easy to use and maintain information service. As part of this implementation effort we have also explored more advanced features for dealing with caching of the information based on quality augmentations to the data itself. The result of our simplified architecture is presented in Figure 4 and contrasts our earlier Figure 2. We believe that although the number of the components within our Info Gram service increased the overall complexity of the combined service is lower that the current provided solution.

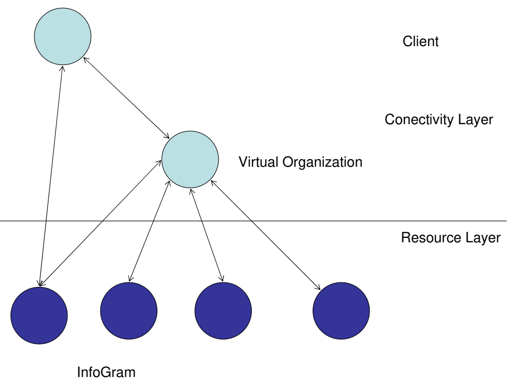

Figure 4. The new InfoGram service reduces the number of protocols and components in a Grid.

7. Implementation

Although our services can be implemented in any other language we have chosen to prototype them in Java. It is a straightforward engineering exercise to implement them in C.

The Java platform enhances the functionality of our service based on the use of additional features that are otherwise not available in C. Thus, we were able to achieve:

- Delivery of a pure Java Information and GRAM service providing cross-platform portability, which includes the Windows Operating System.
- Delivery of a Web-enabled installation service that can deploy the InfoGram service with low overhead on installation time and administrative burden.
- Execution of untrusted applications in trusted environments on remote machines as part of the Java Virtual Machine model.

To support the development of the previously outlined service, we have performed significant enhancements to the Java CoG Kit that is maintained as part of the Globus Project. These enhancements are focusing on the job submission, deployment, logging, security, and information service. Whenever possible, we use standard Java packages to reduce the amount of codebase that must be maintained by us. This includes logging [20, 25] and security [19, 26].

In a first step, we have implemented a pure Java implementation a Globus GRAM [16, 9] service that provides much the same functionality than its C-based counterpart. In order to support interoperability and compatibility, we based the design directly on the architecture of the C GRAM service. It contains a gatekeeper, job manager, and a local job execution process. We name this service J-GRAM.

Job Submission.

This Job Execution service within J-GRAM is protocol-compatible with the "C-GRAM" distributed with the Globus Toolkit. At present, we investigate the implementation of major GRAM functionality, such as the support for gridmaps, which map user certificates to local user IDs, as well as the possibility to interfaces easily to schedulers. We learned from this prototype that it is possible to provide a service in Java that mimics the behavior of C-GRAM.

Besides the invokation of executables from precompiled native code, our J-GRAM service enhances the normal Globus GRAM service by being able to execute pure Java code submitted as Java jar files. To enable the execution of jar files as part of the J-GRAM service, a variety of changes were necessary. We extended the functionality of the job manager to start up the code embedded in a jar file that was submitted through an RSL call such as, `(executable=myJavaApplication.jar)`

In order to run Java applications, one method is to execute the code in the same JVM as the rest of the components are running. An alternative is to separate the execution of the job into a JVM to increase security [19, 24]. We provide the ability to configure the job manager to run in either of these modes. The Grid administrator must decide which mode should be run. The execution of system commands is performed through the runtime.exec() call. It is possible to redirect I/O to and from the client. The functionality is equivalent to the one from the C GRAM service with exception that DUROC is not supported. As the Globus Project will replace it in the near future, we have decided to refer to full delegation to a C Globus GRAM in order to provide this functionality . Therefore, it is still possible to start up MPICH-G2 jobs [22].

Deployment. We have demonstrated this service at SC2001 and featured the ease of installation of such a service while using the Java framework deployment methods known as Web Start. Using this advanced deployment protocol, we are also able to maintain the upgradeability with more ease and to provide future solutions for automatically upgrading such services in production Grids. This feature is naturally supported while choosing Java as an implementation and deployment platform. Such sophisticated approaches require much more effort in traditional operating systems.

Logging. We are in the process of refining a logging mechanism for the execution of jobs assists in the fault recovery abilities of GRAM, as well as the possibility of logging authenticated information queries to guide the use as part of intelligent scheduling services.

Secure Sandboxing. In traditional programming languages, such as C, C++, and FORTRAN, it is difficult to execute untrusted applications in a trusted environment similar to the one the Grid provides. With a JVM, however, we are able to enable a trust relation between an untrusted client application to be executed in a trusted environment. Additionally, we were able to package a gatekeeper with non-root access rights in a jar file that can be easily installed in one environment. J-GRAM can be configured in various ways. We can either execute each job in the already running JVM or start up a number of external JVM to execute a jar file in an even more restrictive environment.

Portability. Other advantages (that are based on the use of Java) are the immediate availability of an information service on the Windows operating system. Other benefits are introduced by providing authorization mechanisms as part of this service, which can be supported by the Java platform.

InfoGram. In a second step we have prototyped much of the functionality described within this paper to enable the InfoGram service. We have obtained good experience to return information queries in LDIF and XML.

8. Application

Currently, the J-GRAM service has already been used in several projects, one of which is the emerging OGSA framework [12] that has been developed after our investigations.

We have tested our InfoGram prototype on an application that we have termed a sporadic Grid. Such a Grid is created just for a short period of time during sophisticated experiments at synchrotrons or photon sources [35, 34]. To implement such a service we need a simple architecture that contains a set of advanced Grid services that are useful for supporting the creation and maintenance of sporadic Grids. Our InfoGram service provides such a service. As we are able to distribute it as a pure Java application, it will be easy to install it on a number of machines or access it through Web-browsers.

We will extend our efforts to support computationally mediated sciences [40]. In this technique, a focused electron probe is sequentially scanned across a two dimensional field of view a thin specimen, and at each point on the specimen a two dimensional electron diffraction pattern is acquired and stored. The analysis of the spatial variation in the electron diffraction pattern allows a researcher to study the subtle changes resulting from microstructural differences, such as ferro and electro magnetic domain formation and motion at unprecedented spatial scales. We will provide the computational Grid infrastructure for these classes of experiments.

9. Related Work

Parallel to the research described in this paper, modifications to GRAM1.0 were performed by colleagues within the Globus Project together with the Condor team at the University of Wisconsin. This modified version of GRAM is available as part of the Globus 2.0 release. We are protocol compatible to that version. Most recently, the Globus Project, started together with IBM on the Open Grid Services Architecture. Our work was performed before OGSA. Lessons learned from our activities should have influence on the OGSA work. The current OGSA prototype implementation uses the J-GRAM service, as well as the GSI security provided through the Java CoG Kit [33].

10. Status and Future Plans

The work performed within this research activity explored new concepts that we expect to be considered in future Globus Toolkit developments. Future research activities will include exploration of conceptual issues identified within this paper, as well as their implementation as part of prototype and Globus toolkit developments. On the conceptual level, we will investigate the explicit guidelines for system designers to choose the right configuration for setting up the InfoGram Service with the appropriate parameters and configuration files. We will perform further simplifications on the J-GRAM architecture while using only one port to communicate between job mangers and clients. For compatibility reasons, we have not yet been able to perform this change. Improved fault tolerance will allow for automatic restart capabilities enabled through checkpointing. We are improving our code and hope to integrate it in either the Globus Toolkit or the OGSA framework. Several features, such as the use of the performance tag and the information degradation, are integrated at the moment. We are also experimenting with integration of our framework in Web services and JXTA [23].

11. Discussion and Conclusion

We feel that we have contributed to several areas within Grid computing. First, we identified that it is possible to design an Information system and a Job submission service that simplifies the architecture of the services provided by the Globus Toolkit. Through the extension of the RSL it will be easy for current Globus Toolkit users to adapt their code to use this information query. Second, we provide the possibility of being protocol compatible to the Globus Toolkit, while being able to integrate our information provider in the existent MDS. Therefore, we provide the option to move to a different Information provider while enabling a gradual transition. New information providers could be integrated easily in this information service framework. Third, we already integrated in the current Java CoG Kit our J-GRAM service that allows executing untrusted applications in trusted environments. This service is naturally able to run on Windows platforms and can be used to support sporadic Grids as defined in the paper. Forth, we set the stage for a multi protocol support for Grid information services that may export their data in LDIF or XML-schema.

We presented suggestions for enhancing the Globus Toolkit and believe that future development on Globus GRAM can benefit from our research on sporadic Grids. We believe that the Open Grid Services Architecture will benefit from this work performed over the last year. In particular the simplified InfoGram service can be used as an elementary replacement for a lightweight job execution and information service. It is straight forward to cast the Info Gram in WSDL.

Acknowledgments

This work was supported by the Mathematical, Information, and Computational Science Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. DARPA, DOE, and NSF support Globus Project research and development. The work of Carlos J. Peña was partially funded by the DOE ERULF program. We thank Trilok Velingetti for his help during the implementation. We thank Ian Foster, Dennis Gannon, Peter Lane, Nell Rehn, Mike Russell for the valuable discussions during the course of the ongoing development. This work would not been possible without the help of the Globus team. Globus Toolkit and Globus Project are trademarks held by the University of Chicago.

References

[1] The physiology of the grid: An open grid services architecture for distributed systems integration. Available from http://www.globus.org/research/papers/ogsa.pdf.

[2] EUROGRID: Application Testbed for European Grid Computing, 2001. http://www.eurogrid.org/.

[3] Information Power Grid Engeneering and Research Site, 2001. http://www.ipg.nasa.gov/.

[4] Neesgrid homepage. http://www.neesgrid.org/, March 2002.

[5] G. Allen, W. Benger, T. Goodale, H.-C. Hege, G. Lanfermann, A. Merzky, T. Radke, E. Seidel, and J. Shalf. The cactus code: A problem solving environment for the grid. In *High-Performance Distributed Computing, 2000. Proceedings. The Ninth International Symposium on*, pages 253 – 260, San Fransisco, August 2000.

[6] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch. Design and deployment of a national-scale authentication infrastructure. *IEEE Computer*, 33(12):60–66, 2000.

[7] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *10th IEEE International Symposium on High Performance Distributed Computing*, pages 181–184, San Fransisco, CA, August 7-9 2001. IEEE Press. www.globus.org.

[8] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. pages 181–184, 2001.

[9] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In *Proceedings of IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing,*, Lecture Notes on Computers Science, 1998.

[10] Doe science grid. http://www.doesciencegrid.org/.

[11] G. Fagg, K. Moore, and J. Dongarra. Scalable networked information processing environment (snipe). *International Journal on Future Generation Computer Systems*, 15:595–605, 1999.

[12] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Technical report, The Globus Project, Jan. 2002. http://www.globus.org/ogsa.

[13] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[14] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001. www.globus.org/research/papers/anatomy.pdf.

[15] D. Gannon, R. Bramley, G. Fox, S. Smallen, A. Rossi, R. Ananthakrishnan, F. Bertrand, K. Chiu, M. Farrellee, M. Govindaraju, S. Krishnan, L. Ramakrishnan, Y. Simmhan, A. Slominski, Y. Ma, C. Olariu, and N. Rey-Cenvaz. Programming the grid: Distributed software components, p2p and grid web services for scientific applications. http://www.extreme.indiana.edu/an/papers/ProgGrids.pdf.

[16] The Globus GRAM Web Page. www.globus.org/gram.

[17] The Globus MDS Users Guide. www.globus.org/mds/mdsusersguide.pdf.

[18] The Globus Project Web Page. www.globus.org.

[19] IAIK Java Cryptology, 2001. http://jcewww.iaik.at/.

[20] Logging Toolkit for Java, July 2001. http://www.alphaworks.ibm.com/tech/loggingtoolkit4j.

[21] J. Kaplan and M. Nelson. A comparison of queueing, cluster and distributed compuing systems, 1994.

[22] N. Karonis. MPICH-G2 Web Page, 2001. http://www.hpclab.niu.edu/mpi/.

[23] N. Krishnan. The jxta solution to p2p. October 10 2001.

[24] P. Lipp. *Sicherheit und Kryptographie in Java . Einfhrung, Anwendung und Lsungen*. Addison-Wesley, 2000.

[25] LogKit Developer Documentation, 2001. http://jakarta.apache.org/avalon/logkit/whitepaper.html.

[26] S. Oaks and N. Inc. *Java Security*. Java series. O'Reilly, Cambridge, 1998. http://www.netlibrary.com/.

[27] M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiari. Certificate-based Access Control for Widely Distributed Resources. In *Proc. 8th Usenix Security Symposium*, Washington, DC, August 23-26 1999. http://www-itg.lbl.gov/Akenti/papers.html.

[28] G. von Laszewski. *A Parallel Data Assimilation System and its Implications on a Metacomputing Environment*. PhD thesis, Syracuse University, Nov. 1996.

[29] G. von Laszewski. A loosely coupled metacomputer: Cooperating job submissions across multiple supercomputing sites. *Concurrency: Experience, and Practice*, 11(15):933–948, 1999. www.cogkits.org.

[30] G. von Laszewski. Grid Computing: Enabling a Vision for Collaborative Research. In *Conference on Applied Parallel Computing, 3rd CSC Scientific Meeting*, Lecture Notes, Espoo, Finland, 15 - 18 June 2002. Springer.

[31] G. von Laszewski, S. Fitzgerald, I. Foster, C. Kesselman, W. Smith, and S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. In *Proc. 6th International Symposium on High-Performance Distributed Computing*, pages 365–375, Portland, OR, August 5-8 1997. IEEE. ftp://ftp.globus.org/pub/globus/papers/hpdc97-mds.pdf.

[32] G. von Laszewski and I. Foster. Grid infrastructure to support science portals for large scale instruments. In *Proc. of the Workshop Distributed Computing on the Web*. University of Rostock, Germany, June 21-23 1999. http://www.mcs.anl.gov/ laszewsk/papers/rostock.pdf.

[33] G. von Laszewski, I. Foster, J. Gawor, and P. Lane. A Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience*, 13(8-9):643–662, 2001. http://www.globus.org/cog/documentation/papers/cog-cpe-final.pdf.

[34] G. von Laszewski, I. Foster, J. A. Insley, J. Bresnahan, C. Kesselman, M. Su, M. Thiebaux, M. L. Rivers, I. McNulty, B. Tieman, and S. Wang. Real-time analysis, visualization, and steering of microtomography experiments at photon sources. In *SIAM99*. SIAM, 1999.

[35] G. von Laszewski, M. L. Westbrook, C. Barnes, I. T. Foster, and E. M. Westbrook. Using computational Grid capabilities to enhance the capability of an X-ray source for structural biology. *Cluster Computing*, 3(3):187–199, 2000.

[36] Web services description language (wsdl) 1.1. http://www.w3.org/TR/wsdl. W3C.

[37] www-4.ibm.com/software/solutions/webservices/.

[38] Web services inspection language (wsil). http://xml.coverpages.org/IBM-WS-Inspection-Overview.pdf.

[39] XML Schema, Primer 0, 1, and 3, 2001. http://www.w3.org/XML/Schema.

[40] N. J. Zaluzec and G. von Laszewski. Computationally mediated sciences. in preparation, September 2002.