Volume 2, Issue 6,June2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Review of Various Grid Middleware Technologies

Priyanka Arora
Department of Information Technology,
Guru Nanak Dev Engineering College, Ludhiana

Harneet Arora
Department of Computer Science & Engineering
Sri Guru Granth Sahib World University, Fathegarh Sahib

Abstract— **In the past decade, the demand for computing and storage power has increased tremendously. The huge amount of data is being generated by the real time systems; say for example the Largest Hadron Collider (LHC) built at the CERN physics laboratory generates about 10 Petabytes or 10^7 Gigabytes of data a year [1]. To manage such a large amount of data one needs to have huge storage and computational capacity. So, as meet such growing needs Grid Computing Middlewares are designed, on top of which one can develop a Grid Environment and manage this growing demand in a distributive fashion. This paper explains gives a review of various grid middlewares which can be used to build a powerful Grid Environment.**

Keywords— **Grid Computing, Globus, UNICORE, Legion, Alchemi**

I. INTRODUCTION

Computing is being transformed to a model consisting of services that are commoditized and delivered in a manner similar to utilities such as water, electricity, gas, and telephony. In such a model, users access services based on their requirements without regard to where the services are hosted [2]. Computers spend a lot of their time doing nothing. With the recent popularity of the Internet, distributed computing has become popular [3]. Distributed computing can be deemed as an attempt to produce a virtual supercomputer out of hundreds or thousands of individual computers [4].

Grids are usually heterogeneous networks. Grid nodes, generally individual computers, consist of different hardware and use a variety of operating systems, and the networks connecting them vary in bandwidth. Grid computing refers to the combination of computer resources from multiple administrative domains to reach a common goal. Grid computing uses middleware to divide and apportion pieces of a program among several computers, sometimes up to many thousands. Grid computing involves computation in a distributed fashion, which may also involve the aggregation of large-scale cluster computing-based systems. This strategy helps to develop a "virtual supercomputer" system within an organization. Grids also address various important issues of security, discovery of resources at appropriate time, uniform access, dynamic aggregation of the data and maintaining quality of service. Various projects are undergoing that efficiently utilize the benefits of grid computing such as NorduGrid[5] ,GridPP[6], PRAGMA[7], EGEE[8], EU-IndiaGrid[9], etc. Also, Grids have the capability of allowing sharing of various scientific instruments such as at the CERN Large Hadron Collider (LHC), Australian radio telescope; synchrotron and many more [10].

The further section discusses various middlewares. In section 2 the Globus middleware is discussed with its features, section 3 discusses about Alchemi Grid and its components, section 4 and 5 explains about various features of Unicore and Legion.

II. GLOBUS

While developing a grid, the middleware is used to hide the heterogeneous nature and provide users and applications with a homogeneous and seamless environment by providing a set of standardized interfaces to a variety of services. Globus Toolkit is a middleware technology developed by the Globus Alliance, provides a software infrastructure that enables applications to handle distributed heterogeneous computing resources as a single virtual machine. It defines basic services and capabilities required to construct a computational grid. The Globus Toolkit is a community-based, open-architecture, open-source set of services and software libraries that support Grids and Grid applications. The toolkit addresses issues of security, information discovery, resource management, data management, communication, fault detection, and portability [11]. The latest version of the toolkit i.e. GT4 is explored in the next sections.

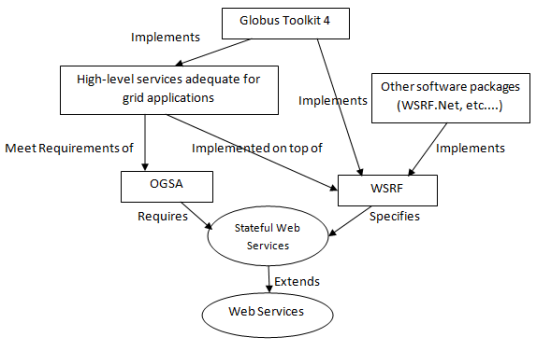

Figure 1: Relationship between OGSA, WSRF and GT4 [12]

A. Exploring GT4

The Globus Toolkit4 is a software toolkit, developed by The Globus Alliance, which can be used to program grid-based applications. The toolkit includes a few high-level services that we can use to build Grid applications. Globus Toolkit includes a resource monitoring and discovery service, a job submission infrastructure, a security infrastructure, and data management services. It is a realization of the OGSA [12] requirements and a de facto standard for the Grid community. Most of these services are implemented on top of WSRF [12]. The toolkit also includes some services that are not implemented on top of WSRF and are called the non-WS components. The Globus Toolkit includes a complete implementation of the WSRF specification.

GT4 primarily consists of five main components as:-

- **Common Runtime**: The Common Runtime components provide a set of fundamental libraries and tools which are needed to build both WS and non-WS services.
- **Security**: Using the Security components, based on the Grid Security
- Infrastructure (GSI), we can make sure that our communications are secure.
- **Data management services**: These components will allow us to manage large sets of data in our virtual organization.
- **Information services**: The Information Services, commonly referred to as the Monitoring and Discovery Services (MDS) [13], includes a set of components to discover and monitor resources in a virtual organization. GT4 also includes a non-WS version of MDS (MDS2) for legacy purposes.
- **Execution management**: Execution Management components deal with the initiation, monitoring, management, scheduling and coordination of executable programs, usually called jobs, in a Grid.
- So, GT4 is a set of software components for building distributed systems: systems in which diverse and discrete software agents interact via message exchanges over a network to perform some tasks. Distributed systems face particular challenges relating to sometimes high and unpredictable network latencies, the possibility of partial failure, and issues of concurrency. In addition, system components may be located within distinct administrative domains, thus introducing issues of decentralized control and negotiation.

B. Predefined GT4 Services

GT4 provides a set of predefined services. Nine GT4 services implement Web services interfaces: Job management (GRAM); Reliable file transfer (RFT); Delegation; MDS-Index, MDS-Trigger, and MDSArchive (collectively termed the Monitoring and Discovery System, or MDS); Community authorization (CAS); DAI data access and integration; and GTCP Grid TeleControl Protocol for online control of instrumentation.

For two of these services, GRAM and MDS-Index, pre-WS legacy implementations are also provided. For three additional GT4 services, WS interfaces are not yet provided. They are GridFTP data transport, replica location service, and My Proxy online credential repository. Other libraries provide powerful authentication and authorization mechanisms, while the eXtensible I/O (XIO) library provides convenient access to a variety of underlying transport protocols. SimpleCA is a lightweight certification authority [12][13][16].

C. GT4 Architecture Overview

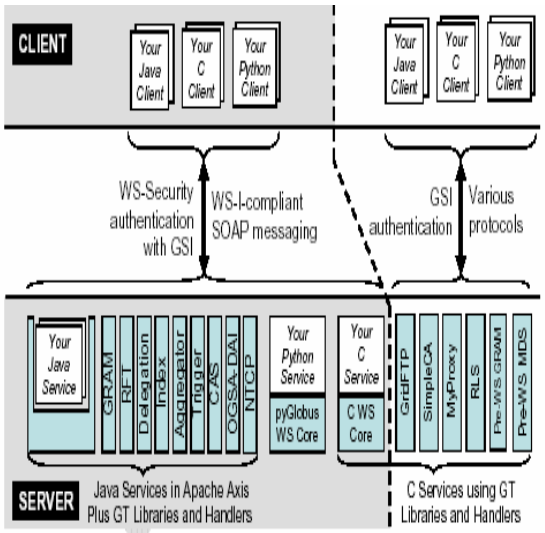

Figure 2: Schematic View of GT4 Components [12]

As shown in Figure 2, GT4 comprises both a set of service implementations i.e. server code and associated client libraries. GT4 provides both Web services components and non-WS components. All boxes in the client domain denote custom applications and/or third-party tools that access GT4 services or GT4-enabled services. All GT4 WS components use WS-Interoperability-compliant transport and security mechanisms, and can thus interoperate with each other and with other WS components. In addition, all GT4 components, both WS and non-WS, support X.509 end entity certificates and proxy certificates. Thus a client can use the same credentials to authenticate with any GT4 WS [12] or non-WS component.

III. **ALCHEMI**

Alchemi is an open-source .Net based Enterprise Grid computing framework developed by researchers at the GRIDS lab, in the Computer Science and Software Engineering Department at the University of Melbourne, Australia. It allows you to painlessly aggregate the computing power of networked machines into a virtual supercomputer and to develop applications to run on the grid with no additional investment and no discernible impact to users. It has been designed with the primary goal of being easy to use without sacrificing power and flexibility. It has been designed for the Microsoft Windows operating system, which is seen as key

factor in industry adoption of grid computing technology since more than 90% of machines worldwide run variants of Windows [14]

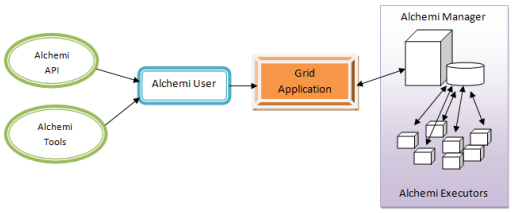

Figure 3: An Alchemi Grid [14]

All paragraphs must be indented. All paragraphs must be justified, i.e. both left-justified and right-justified.

A. Alchemi Architecture

Alchemi follows the master-worker parallel programming paradigm in which a central component dispatches independent units of parallel execution to workers and manages them. This smallest unit of parallel execution is a grid thread, which is conceptually and programmatically similar to a thread object (in object-oriented sense) that wraps a "normal" multitasking operating system thread. A grid application is defined simply as an application that is to be executed on a grid and that consists of a number of grid threads. Grid applications and grid threads are exposed to the grid application developer via the object oriented Alchemi .NET API.

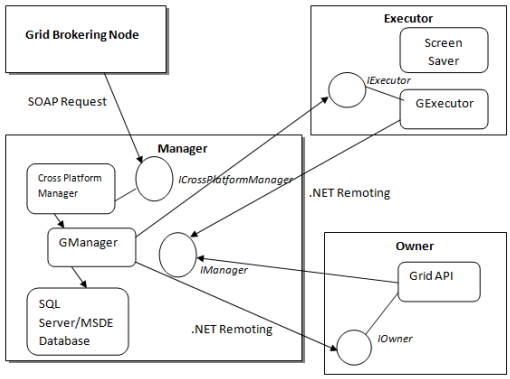

Figure 4: Alchemi Architecture [14]

Alchemi components

Alchemi has the following components designed for the grid construction:

- Manager
- Executor
- Owner
- Cross Platform Manger

Manager

The Manager manages the execution of grid applications and provides services associated with managing thread execution. The Executors register themselves with the Manager, which in turn keeps track of their availability. Threads received from the owner are placed in a pool and scheduled to be executed on the various available Executors. A priority for each thread can be explicitly specified when it is created within the Owner, but is assigned the highest priority by default if none is specified.

Threads are scheduled on a Priority and First Come First Served (FCFS) basis, in that order. The Executors return completed threads to the Manager, which are subsequently passed on or collected by the respective Owner.

Executor

The Executor accepts threads from the Manager and executes them. An Executor can be configured to be dedicated, meaning the resource is centrally managed by the Manager, or non-dedicated, meaning that the resource is managed on a volunteer basis via a screen saver or by the user. For non-dedicated execution, there is one-way communication between the Executor and the Manager. In this case, the resource that the Executor resides on is managed on a volunteer basis since it requests threads to execute from the Manager. Where two-way communication is possible and dedicated execution is desired the Executor exposes an interface (IExecutor) so that the Manager may communicate with it directly. In this case, the Manager explicitly instructs the Executor to execute threads, resulting in centralized management of the resource where the Executor resides. Thus, Alchemi's execution model provides the dual benefit of:

- Flexible resource management i.e. centralized management with dedicated execution vs. decentralized management with non-dedicated execution; and
- Flexible deployment under network constraints i.e. the component can be deployment as non dedicated where two-way communication is not desired or not possible (e.g. when it is behind a firewall or NAT/proxy server).

Thus, dedicated execution is more suitable where the Manager and Executor are on the same Local Area Network while non-dedicated execution is more appropriate when the Manager and Executor are to be connected over the Internet.

Owner

Grid applications created using the Alchemi API are executed on the Owner component. The Owner provides an interface with respect to grid applications between the application developer and the grid. Hence it "owns" the application and provides services associated with the ownership of an application and its constituent threads. The Owner submits threads to the Manager and collects completed threads on behalf of the application developer via the Alchemi API.

Cross-Platform Manager

The Cross-Platform Manager, an optional sub-component of the Manager, is a generic web services interface that exposes a portion of the functionality of the Manager in order to enable Alchemi to manage the execution of platform independent grid jobs(as opposed to grid applications utilizing the Alchemi grid thread model). Jobs submitted to the Cross-Platform

Manager are translated into a form that is accepted by the Manager (i.e. grid threads), which are then scheduled and executed as normal in the fashion described above. Thus, in addition to supporting the grid enabling of existing applications, the Cross-Platform Manager enables other grid middleware to interoperate with and leverage Alchemi on any platform that supports web services (e.g. Gridbus Grid Service Broker).

IV. UNICORE

UNICORE (UNiform INterface to COmputing REsources) is a Grid computing technology that provides seamless, secure, and intuitive access to distributed Grid resources such as supercomputers or cluster systems and information stored in databases. UNICORE was developed in two projects funded by the German ministry for education and research (BMBF). In various European-funded projects UNICORE has evolved to a full-grown and well-tested Grid middleware system over the years. UNICORE is used in daily production at several supercomputer centers worldwide. Beyond this production usage, UNICORE serves as a solid basis in many European and international research projects. The UNICORE technology is open source under BSD license and available at SourceForge, where new releases are published on a regular basis. The design goals for UNICORE include a uniform and easy to use GUI, an open architecture based on the concept of an abstract job, a consistent security architecture, minimal interference with local administrative procedures, exploitation of existing and emerging technologies, a zero administration user interface through a standard Web browser and Java applets, and a production ready prototype within two years. UNICORE is designed to support batch jobs, it does not allow for interactive processes. At the application level asynchronous metacomputing is supported, allowing for independent and dependent parts of a UNICORE job to be executed on a set of distributed systems. The user is provided with a unique UNICORE user-ID for uniform access to all UNICORE sites [14].

A. Features of UNICORE

UNICORE has special characteristics that make it unique among Grid middleware systems. The UNICORE design is based on several guiding principles that serve as key objectives for further enhancements.

- Open source under BSD license.
- Standards-based, conforming to the latest standards from the Open Grid Forum (OGF), W3C, OASIS, and IETF, in particular the Open Grid Services Architecture (OGSA) and the Web Services Resource Framework (WS-RF 1.2).
- Open and extensible realized with a modern Service-Oriented Architecture (SOA), which allows to easily replacing particular components with others.
- Interoperable with other Grid technologies to enable a coupling of Grid infrastructures or the user's needs
- Seamless, secure, and intuitive following a vertical, end-to-end approach and offering components at all levels of a modern Grid architecture from intuitive user interfaces down to the resource level. Like previous versions UNICORE 6 seamlessly integrates in existing environments.
- Mature security mechanisms adequate for the use in supercomputing environments and Grid infrastructures. X.509 certificates form the basis for authentication and authorisation, enhanced with a support for proxy certificates and virtual organisations (VO) based access control.
- Workflow support tightly integrated into the stack while being extensible in order to use different workflow languages and engines for domain-specific usage.
- Application integration mechanisms on the client, services and resource level for a tight integration of various types of applications from the scientific and industrial domain.
- Different clients serving the needs of various scientific communities, e.g. graphical clients to define complex workflows, command line tool, web-based access.
- Quick and simple to install and configure to address requirements from operational teams and to lower the barrier of adopting Grid technologies. Similar the configuration of the various services and components is easy to handle.
- Various operating and batch systems are supported on all layers, i.e. clients, services and systems; Windows, MacOS, Linux, and Unix systems as well as different batch systems are supported such as LoadLeveler, Torque, SLURM, LSF, OpenCCS, etc.
- Implemented in Java to achieve platform independence.

V. LEGION

Legion, an object-based meta-systems software project at the University of Virginia, is designed for a system of millions of hosts and trillions of objects tied together with high-speed links. Users working on their home machines see the illusion of a single computer, with access to all kinds of data and physical resources, such as digital libraries, physical simulations, cameras, linear accelerators, and video streams. Groups of users can construct shared virtual work spaces, in order to collaborate research and exchange information. This abstraction springs from Legion's transparent scheduling, data management, fault tolerance, site autonomy, and a wide range of security options.

As new requirements and new opportunities for distributed computing emerge and future users make unforeseen demands on resources and software, the demands placed on a virtual computer will evolve and grow. What works today or even tomorrow will soon be worse than useless, and we strongly believe that Legion should be a flexible tool that can adapt to new needs. Legion is therefore an open system, designed to encourage third party development of new or

updated applications, run-time library implementations, and core components.

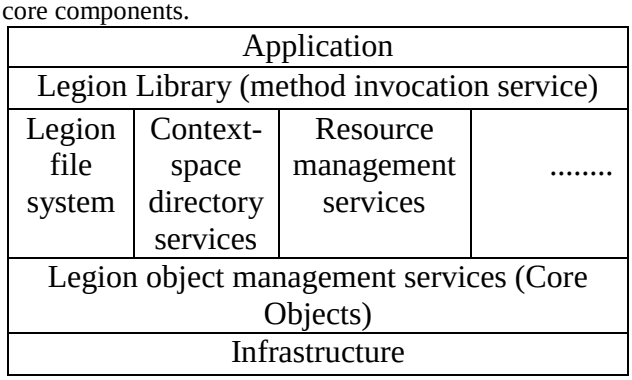

Figure 5: Legion Architecture [15]

Legion sits on top of the user's operating system, acting as liaison between its own host(s) and whatever other resources are required. The user isn't bogged down with time-consuming negotiations with outside systems and system administrators, since Legion's scheduling and security policies act on his or her behalf. Conversely, it can protect its own resources against other Legion users, so that administrators can choose appropriate policies for who uses which resources under what circumstances. To allow users to take advantage of a wide range of possible resources, Legion offers a user-controlled naming system called *context space*, so that users can easily create and use objects in far-flung systems. Users can also run applications written in multiple languages, since Legion supports interoperability between objects written in multiple languages.

A. Features of Legion

A.1 Scheduling Features

- Resource reservations for Host and Vaults.
- Collection objects providing resource information for schedulers, using data collection agents that push information.
- Enactor objects to implement schedules, by obtaining resource reservations and starting objects.
- Support for application-level, per-object schedulers.
- Per-class default external schedulers and placements (these may be overridden at user's behest).
- Intelligent scheduling for stateless objects, which balances the workload across available hosts.
- A pull model for Collection data gathering will be added in future releases, as well as additional monitoring support and sample schedulers.

B.1 Security Features

- Public-key cryptography based on RSAREF 2.0.
- Three message-layer security modes: private (encrypted communication), protected (fast digested communication with unforgettable secrets to ensure authentic replies to message calls), and no security.
- Caching secret-keys for faster encryption of multiple messages between communicating parties.
- Auto-encrypted bearer credentials with free-form rights. Propagation of security modes and certificates through calling trees (e.g., if a caller demands encryption, all downstream calls will use it automatically).
- Persistent *authentication objects* that serve as the representation for users in a trust domain.
- Secure legion shell to allow users to login to their authentication objects and obtain associated credentials and environment information.
- Isolation and protection of objects using local OS accounts.
- Easily checked Process Control Daemon for granting limited OS privileges to Legion Host Objects

VI. CONCLUSION

In this paper, we discussed about grid computing and various grid middleware technologies that help efficiently develop grid applications. The emerging Grid Technology enables the users to develop more and more advanced systems where the major requirements of computation and storage are met. Grid Computing technologies is a cost effective solution to these needs by utilizing the waste CPU cycles for calculations which otherwise requires expensive supercomputers. Globus is widely used and accepted as a *de-facto* standard grid middleware for developing grid applications. UNICORE along with gLite is used as a part of EGEE project for developing grid environment.

REFERENCES

[1] Sotomayer B., Childers L., "Globus toolkit 4 : Programming Java Services", Morgan Kaufmann, edition 2006
[2] Buyya R.K., "Market-Oriented Cloud Computing: Vision, Hype, and Reality of Delivering Computing as the 5th Utility.", International Symposium on Cluster Computing and the Grid ,IEEE, 2009
[3] Kumar R., Kaur N., " Job Scheduling in Grid Computers", National Conference on Challenges & Opportunities, RIMT Mandi Gobindgarh, pg111, 2007
[4] Manjulaka A., Karthikeyan P.," Distributed Computing Approaches for Scalability and High Performance", International Journal of Engineering Science and Technology, vol. 2(6), pg2328, 2010
[5] Nordu Grid, Available: http://www.nordugrid.org/
[6] GridPP, Available: http://www.gridpp.ac.uk/
[7] Pragma, Available: http://www.pragma-grid.net
[8] EGEE, Available: http://www.eu-egee.org/
[9] EU-IndiaGrid, Available: http://www.euindiagrid.eu/

[10] Asadzadeh P., Buyya R., et. al, "Global Grids and Software Toolkits: A Study of Four Grid Middleware Technologies", High Performance Computing: Paradigm and Infrastructure, Wiley Press, USA, June 2005
[11] Amoon M., Mowafy M., Altameem T., "A Multiagent-Based System for Scheduling Jobs in Computational Grids", ICGST- AIML journal, 2009
[12] Amoon M., Mowafy M., Altameem T., "A Multiagent-Based System for Scheduling Jobs in Computational Grids", ICGST- AIML journal, 2009
[13] Sotomayer B., Childers L., "Globus toolkit 4 : Programming Java Services", Morgan Kaufmann, edition 2006
[14] Luther A., Buyya R., et al, "Alchemi: A .NET-Based Enterprise Grid Computing System", International Conference on Internet Computing, 2005, Las Vegas, USA, Available: http://www.gridbus.org/%7Eraj/papers/alchemi_icomp05.pdf
[15] Natrajan A., Nguyen-Tuong A., et. al, "The Legion Grid Portal", Concurrency and Computation, vol. 14, part 13/15, pages 1365-1394, ISSN 1532-0626, Wiley Publisher Britain, 2002
[16] Ferreira L., Bieberstein N., et al "Introduction to Grid Computing with Globus," Redbook, IBM Corp., Available: http://www.redbooks.ibm.com/redbooks/pdfs/sg246895.pdf

