

Grid-JQA : Grid Java based Quality of service management by Active database

L. Mohammad Khanli

Ph.D. student
C.E. Dept. IUST
Tehran, Iran

Khanli@iust.ac.ir

M. Analoui

Assistant professor
C.E. Dept. IUST
Tehran, Iran

Analoui@iust.ac.ir

Abstract

Task scheduling is an integrated component of computing. With the emergence of grid and ubiquitous computing, newer challenges have arisen in task scheduling. Unlike traditional parallel computing, grid is a shared enterprising environment where is no central control. The goal of grid task scheduling is to achieve high system throughput and to match the application needs with the available computing resources. This matching of resources in a non-deterministically shared heterogeneous environment leads to concerns on Quality of Service (QoS). The paper presents a novel solution based on Active database to assure Quality of Services in grid computing systems. Active database accommodates some actions to be done automatically by insert, delete or update transactions. In this reason the best matching is obtained by using the most up to date information. Some comparison with other solutions had been done. As a result, our solution is safe factory and acceptable in the limits of performance.

Keywords: Grid-JQA, QoS, Active database.

1 Introduction

Quality of service management has been explored in various contexts, particularly for computer networks, and multimedia applications; network QoS and end system QoS, such as memory and CPU. QoS management has also been explored in the context of Grid computing.

QoS management covers a range of different activities, from resource selection and allocation through to resource release. Regardless of the context, a QoS management system should address the following needs [1]:

- Specifying QoS requirements.
- Mapping QoS requirements to resource capabilities.
- Negotiating QoS with resource owners, where a requirement cannot be exactly met.
- Establishing SLA with clients.
- Reserving and allocating resources.
- Monitoring parameters associated with a QoS session.
- Adapting to varying resource quality characteristics.
- Terminating QoS sessions.

The Grid Java based Quality of service management by Active database (Grid-JQA) is a framework that aims to address the above mentioned requirements. Grid-JQA provides reservations and end-to-end management for quality of service on different types of resources, including networks, CPUs, and disks. It also encourages Grid customers to specify their quality of service needs based on their actual requirements.

It has been envisioned that Grids enable the creation of computing market places. In a typical market-based model, Grid Resources (GR) publish their offerings in a market directory and Grid Customers (GC) employ a Grid Resource Broker (GRB) that identifies GRs through the market directory (Active Database) and map the request to suitable resources that meet their QoS requirements. (Figure 1)

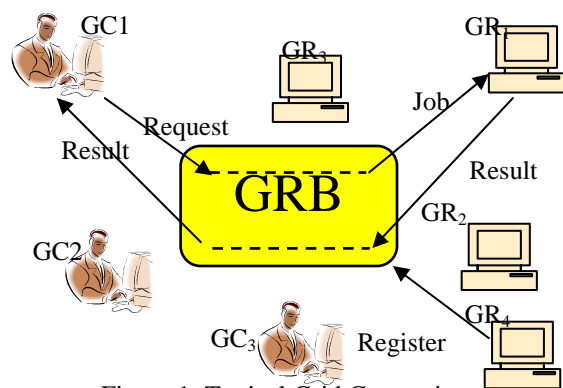


Figure 1: Typical Grid Computing market-based model

To realize this vision, Grids need to support diverse infrastructures/services, including an infrastructure that allows (a) the creation of one or more GRB registries; (b) resources to publish themselves in one or more GRBs along with capabilities; (c) customers to request their jobs for execution from exactly one GRB with their requirements; (d) GRBs to discover the best match of resources to requests, that meet customers QoS requirements; (e) GRBs to send jobs to GRs; (f) GRBs to monitor execution of jobs; (g) GRs to return results to GRBs; (h) GRBs to return results to GCs. In this paper, we propose an infrastructure called the Grid Java based QoS management by using Active database (G-JQA) that supports these requirements.

The rest of this paper is organized as follows: the related work within Grid Computing is presented in section 2.

The detailed system architecture and features of G-JQA are described in section 3. Section 4 describes technologies of matching that are used in the current implementation. We conclude in section 5 with a discussion of current system status and future work.

2 Related Work

There are many existing Grid computing projects currently underway but a few of them guarantee QoS, this section provides a brief description of these systems and compares with our project.

Foster et al. [2] propose a framework for QoS in Grid computing, called the Globus Architecture for Reservation and Allocation (GARA), which enables programmers and users to specify and manage end-to-end QoS for Grid-based applications. It also provides a uniform mechanism for making QoS reservations for different types of Grid resources, such as processors, networks and storage devices. The main drawback of GARA is its inability to support subtask management, which is one of Grid's main goals. There are two more drawbacks in GARA: the topology of domain should be known in advance and also the resources can not publish themselves. Grid-JQA is going to address the subtask management while the topology information and resource publishing are handled dynamically.

Javelin [3] is a Java based infrastructure for internet-wide parallel computing. The three key components of Javelin system are the clients or applications, hosts, and brokers. A client is a process seeking computing resources, a host is a process offering computing resources, and a broker is a process that coordinates the allocation of computing resources. The Javelin system can be considered a computational Grid for high-throughput computing. It has a hierarchical machine organization where each broker manages a tree of hosts. Resources are simple fixed objects with a tree namespace organization. The resources are simply the hosts that are attached to a broker. Any host that wants to be part of Javelin contacts JavelinBNS system, a Javelin information backbone that maintains the list of available brokers. The host then communicates with brokers, chooses a suitable broker, and then becomes part of the broker-managed resources. Thus the information store is a network directory implemented by JavelinBNS. Hosts and brokers update each other as a result of scheduling work. Thus, Javelin uses demand resource dissemination. The broker manages the host-tree or resource information through a heap-like data structure. Resource discovery uses the decentralized query based approach since queries are handled by the distributed set of brokers. In Javelin the burden of the subtask management and monitoring is layered on the client side. The client should also look for a broker that matches its requirements. Thus the clients are thick. Grid-JQA tries to keep the clients as thin as possible and refers the subtask management and monitoring, and the matching to the broker of the system.

Legion [4] is an object-based metasystem or Grid operating system developed at the University of Virginia. Legion provides the software infrastructure so that a

system of heterogeneous, geographically distributed, high performance computers can seamlessly interact. Legion provides application users with a single, coherent, virtual machine. The Legion system is organized into classes and metaclasses. It uses an object based information store organization through the Collection objects. The Collections periodically pull resource state information from host objects. Host objects track load and users can call the individual host directly to get the resource information. The users or their agents can obtain information about resources by issuing queries to a Collection. In Legion, collection has a main role. Collection must pull resource state information. Legion cannot replace host operating systems, Legion cannot legislate changes to the interconnection network, and Legion cannot expect to run as root (or the equivalent).

Currently available scheduling models include: AppLeS [5,6], Nimrod [7], and Condor[8]. The scheduling algorithm in AppLeS focuses on efficient co-location of data and experiments as well as adaptive scheduling. In addition to the prediction model adopted, our approach differs from AppLeS in that our work considers QoS in scheduling. The scheduling in Nimrod is based on deadline and Grid economy model that is different from what we use in Grid-JQA. Condor is designed for high throughput computing in a controlled load network environment. Its matchmaker scheduler targets only single processor tasks which are scheduled independently.

3 Grid-JQA Architecture

Looking on the matching problem, we bring about the following solutions:

1. Resources introduce themselves to all clients and each client decides where to send its request, thus matching is accomplished by the client.
2. The clients send their request to all resources and each resource decides which request to reply, therefore matching is done in the resources.
3. The resources introduce themselves to a broker and each client sends its request to the broker, then the broker decides how to match the requests with the resources. Here there are two possible methods, (a) after the broker finds the best matched resource, the resource and the client negotiate with each other directly. (b) The broker interferes in the process until end of job, in order to guarantee the quality of service and perform subtask management.

In this work, we implement the last mentioned solution 3 (b), and compare it with other works. In the proposed method there are the following steps:

1. The resources introduce themselves to the broker and through initial introduction send their specifications of CPU, memory capacity, disks and I/O devices. The broker computes other specifications such as bandwidth, delay and loss rate.

2. The broker inserts this information into Active Database.
3. The client sends his/her request together QoS parameters to the broker.
4. The broker inserts the client's request into the Active Database.
5. Then the Active Database matches the resources and the requests, and finds the best resource for the request. When there is not a proper match, the request must wait till a suitable resource is found.
6. Finally broker sends results to the client.

Figure 2 shows Grid-JQA architecture. The brokers use exclusive Active database for registries and matching.

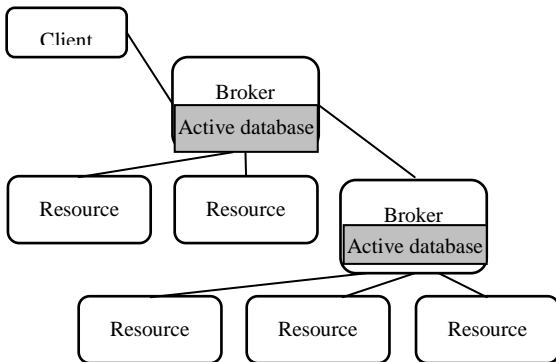


Figure2: Grid-JQA Architecture for a 2-level Grid

For extending environment, we use multi-level Grids. Multi-level Grids are created by connecting brokers hierarchically. The key to accomplishing this is in Grid-JQA's inherent architecture, which allows a broker to behave like a resource towards a higher level manager. The client which goes to a higher level broker has access to the entire computing power of the Grid, while the client connected into a lower-level broker has only access to the computing power managed by the lower-level broker. In this fashion Grids can be scaled to an infinite number of levels.

Compare Grid-JQA with Alchemi[9] architecture, there are four differences that are given in Table 1.

	QoS	Database	Implementation	Platforms
Alchemi	No QoS	SQL	C#, .Net	Windows
Grid-JQA	Guarantee QoS	Active Database	Java	Windows Unix Other Platforms

Table 1: Difference between Alchemi and Grid-JQA

4 Matching

Using Active Database in the broker is a novel technology in Grid computing environment. Active Database automatically reacts to events. In the following events, and Active Database reactions are described.

- New resource connection

In this event, new record inserts into Database that shows resource specifications. Thus Active

Database must find waiting job for executing in this new resource. The ECA (Event Condition Action) Rule for this event is:

```

On insert
  If have a job
    Then Match_Resource
      (new_resource)
  
```

Match_Resource searches waiting jobs and finds matched job with the new resource. If it found, sends the job to the new resource.

- Resource disconnection

If disconnected resource has an incomplete job, the Broker must match this job immediately. In this reason ECA rule for deleting the resource from Database is:

```

On delete
  If not finish job
    Then Match_Job (job)
  
```

Match_Job function finds the best resource for the job that met the job requirements.

- Job Arrival

Job arrival is equal with inserting new record in Database. If Broker has an idle resource that met the job requirements, it must send the job to this resource immediately.

```

On insert
  If have a free resource
    Then Match_Job (job)
  
```

- Job Completion

Resource status changes from busy to idle, when its job has finished. If the job has a result, it must send to client. ECA rule for updating Database is:

```

On update
  If have a result
    Then return (result)
  
```

- Resource Failed

When resource fails, it is deleted from Database and if it has an incomplete job, broker must match this job immediately with idle resources.

```

On delete_resource
  If state=busy
    Then Match_Job (job)
  
```

Implementation Scenario

As proof of concept, a reference implementation for the proposed Grid-JQA architecture is designed and implemented. The implementation highlights the rule execution and demonstrate best matching between resources and clients. Figure 3 shows the implemented architecture.

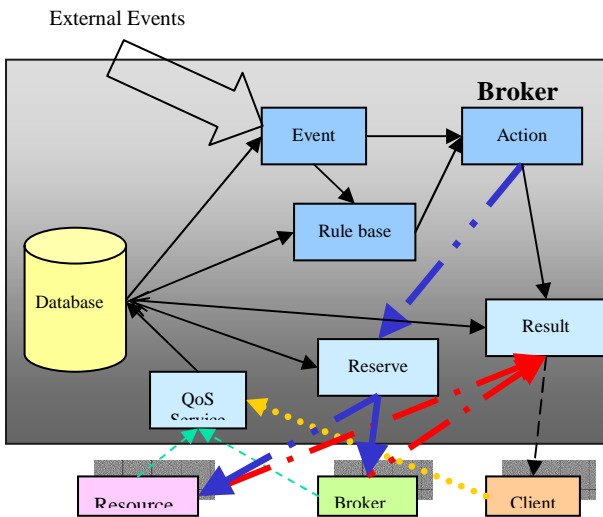


Figure 3.: Implemented Architecture

In this scenario a client utilizes GUI component tools to invoke the QoS Service to request its requirements, in order to execute its job. Resources register their specification in the broker. Also other brokers can register themselves in the broker. Reserve Manager reserves resources and sends the jobs. Result Manager gets subtasks results and sends final result to the client. Event Detector detects internal and external events. Rule base includes ECA rules. Action Generator executes actions on the bases of happened event.

Figure 4 is a UML interaction sequence diagram showing the interaction between the various components of the architecture.

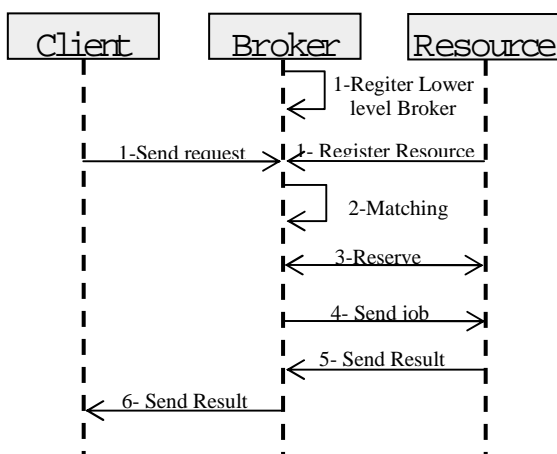


Figure 4: A UML sequence diagram for executing

We have used Java which is a platform independent. Therefore our Grid computation can be used in all platforms such as windows, Linux, etc. Implementation is in early steps and only supports simple events such as insert, delete and update. In the future, we want to expand rules for implementing fault tolerance, load balancing and accounting.

5 Conclusions

This paper contributes the following points into the Grid computing.

1. The use of Active Database. It accommodates some actions to be done automatically by insert, delete or update transactions. As a result the best matching is obtained by using the most up to date information.
2. Decreasing the layers and therefore negotiation becomes minimal. Thus one job is accomplished with minimum negotiation and decrease overhead. If the job isn't done or has failed, matching is accomplished again by the broker, without informing the client. Consequently the client has no interference until he/she receives the final results.
3. Usually the expansion of the domain of the Grid is done by using the domain concept. For the sake of simplicity, we do not use that concept and try to achieve the expansion by expanding the number of the brokers in a hierarchical fashion.
4. We have used Java which is a platform independent. Therefore our Grid computation can be used in all platforms such as Windows, Linux, etc.

6 References

- Rashid J. Al-Ali, Abdelhakim Hafid, Omer F. Rana, David W. Walker, QoS adaptation in service oriented grids, in proceeding of the 1st international workshop on middleware for Grid Computing (MGC 2003) at ACM/IFIP/USENIX, middleware 2003, Rio de Janeiro, Brazil, 2003.
- I.Foster, C.Kesselman, et al. A Distributed resource management architecture that supports advance reservation and co-allocation. In proceedings of the international workshop on Quality of Service, page 27-36, 1999.
- M.Neary, A.Phipps, S. Richman, P. Cappello, Javelin 2.0: Java-Based Parallel Computing on the Internet, Proceedings of European Parallel Computing Conference (Euro-Par 2000), Germany, 2000.
- S. Chapin, J.Karpovich, and A. Grimshaw, The Legion Resource Management System, Proceedings oh the 5th Workshop on Job Scheduling Strategies for Parallel Processing, April 16, 1999, San Juan, Puerto Rico, USA, Springer Verlag Press, Germany, 1999.
- H. Casanova, G. Obertelli, F. Berman and R. Wolski, The AppLeS Parameter Sweep Template: User_level Middleware for the Grid, Proceedings of the super computing conference (SC'2000), Nov.2000.
- H. Csanova, A. Legrand, D. Zagorodnov, F. Berman, Heuristics for Scheduling Parameter Sweep applications in Grid environments, Proceedings of the 9th heterogeneous Computing Workshop (HCW'2000), 2000.

- R. Buyya, M. Murshed, D. Abramson, A Deadline and Budget Constrained Cost-Time Optimization Algorithm for Scheduling Task Farming Applications on Global Grids, The 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02), Las Vegas, Nevada , USA, June 2002.
- R. Raman, M. Livny, M. Solomon, Matchmaking: Distributed Resource Management for high Throughput Computing, Proceedings of the seventh IEEE International Symposium on High Performance Distributed Computing, July 28-31, 1998, Chicago.

Alchemi Documentation, www.alchemi.net