Decentralized Job Scheduler in Desktop Grid

Entisar S. Alkayal
Faculty of Computing and Information Technology
King Abdul Aziz University
Jeddah, Saudi Arabia
entisar_alkayal@hotmail.com

Prof.Dr. Fathy A. Essa
Faculty of Computing and Information Technology
King Abdul Aziz University
Jeddah, Saudi Arabia
fathy55@yahoo.com

Abstract

Desktop Grid computing are developed using different desktop Computers to benefits from idles computers in an efficient way to solve complex problems. Desktop Grid system, which connects a number of personal desktop computers, can achieve the same computing power as a supercomputer does, also with a lower cost. The goal of desktop grid system to aggregate idle computers on the network to utilize them in an efficient way. In this research we present desktop computational grid system that used to utilize desktop computers in Intranet to execute jobs in an efficient way than using a single computer. We design and implement Service Oriented Decentralized Job Scheduler (SODJS) framework based on web services technology. SODJS schedules and balances the jobs among available resources to minimize jobs response time. We evaluate SODJS performance by applying different tests. These tests show that SODJS system provides good performance compared with using single computer in terms of job execution time and system throughput.

1. Introduction

In Desktop grid computing , a wide variety of Desktop computational resources, storage systems and databases, special class of scientific instruments (such as radio telescopes), computational kernels, and so on are logically coupled together and presented as a single integrated resource to the user. Desktop Grid management system is more complicated. Grid development involves the efficient management of heterogeneous, geographically distributed, and dynamically available resources. In this environment, the job scheduler become one of the most critical components of the Grid management middleware, since it has the responsibility of selecting resources and scheduling jobs in such a way that the system provide good performance in term of execution time.

Job scheduling is an important part of grid management system. The efficiency and acceptability of resource management mainly depends on its scheduling strategy. Job scheduling allocates needed resources to job requests, including cooperation allocation through different systems. However, resource scheduling is becoming a complicated problem because of the dynamic and heterogeneous characteristics of grid system as well as the different needs for the resources of the applications applied in grid system [1].

In this research we study, design, implement and evaluate prototype of job scheduler that used to manage jobs in desktop grid based on service oriented technology. SODJS is a scheduling system which provides an easy way to manage desktop computers in an efficient way to process a large number of jobs.

The research is organized as follows: **section 1** introduces the research subject and it also addresses the objectives and motivation of the research. **Section 2** presents briefs background about the desktop Grid system and it discusses different job scheduling architecture. **Section 3** discusses the design of proposed Job Scheduler middleware based on web services architecture and provides a model for Service Oriented Decentralized Job Scheduler (SODJS). **Section 4** discusses the results that are obtained from testing SODJS and evaluate the system. **Section 5** presents the conclusion and direction for future works.

2. Background

A job scheduler can be implemented in different structures, which determine the architecture of the resource management system and the scalability of the system. Grid resource management systems can be classified as having ***centralized, hierarchical*** or ***decentralized*** job scheduler *[*2] as shown in Figure 1.

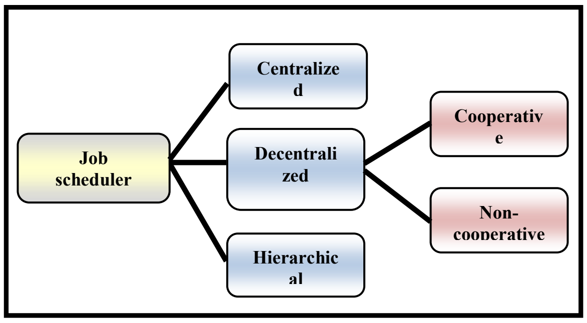

Figure 1: Categorization of Job Scheduler

- In a ***centralized*** job scheduler, all jobs are submitted to a single manager that is responsible for scheduling them on the available resources. Since all the scheduling information is available at one single position, the scheduling decisions are optimal but this approach is not very scalable in large grid systems [1].

- In a ***decentralized*** *job* scheduler there is no central manager, scheduling is done by the resource requestors and owners independently. This approach is scalable and suits large grid systems. However, individual schedulers should cooperate with each other in making scheduling decisions and the schedule generated may not be the optimal schedule.
- In a ***hierarchical*** job scheduler, the managers are organized in a hierarchy. High-level resource entities are scheduled at higher levels and lower level. The smaller sub-entities are scheduled at lower levels of the manager hierarchy. This model is a combination of the above two models [1].

Scheduling policies used by the grid resource management system can be also classified into two major categories: *system-oriented scheduling* or *user-oriented scheduling*. System oriented scheduling often strives to maximize over system throughput , average response time , fairness or a combination of these , whereas user-oriented scheduling try to optimize the performance for an individual user , typically by minimizing the response time for each job submitted by the user [3] . A centralized manager typically performs system–oriented scheduling, whereas a decentralized manager often uses a user-oriented policy. The comparison between a centralized job scheduler and job decentralized scheduler is summarized in table 1.

A central job scheduler needs current knowledge about the entire state of the system at each point in time. This makes it scale badly with the growth in the size of the system [5]. Failure of the job scheduling results in failure of the whole system, while in a distributed approach only some of the work will be lost.

Table 1: Comparison between Centralized and Decentralized scheduling

	Centralized Scheduling	**Decentralized Scheduling**
Scalability	Not scalable	Scalable
Fault tolerance	Single point of failure	More fault tolerance
Architecture	Client-Server architecture	Peer-to-Peer and dynamic environments
Information Storage	Keep information about all resources and jobs	Don't keep information about all resource and jobs
Performance	System-oriented	User-oriented

Many solutions have been developed to solve the job scheduling problem in grid. Local schedulers, like Condor project [4], were developed to schedule jobs in a single compute farm. Meta-schedulers were then introduced to allow multiple sites to cooperate. The difficulty of allocating jobs larger than the maximum size of the biggest resources and the need for a more efficient organization of the Grid led to the creation of other solutions.

3 .Service Oriented Decentralized Job Scheduler (SODJS)

In this section, we introduce the architecture of Service Oriented Decentralized Job Scheduler (SODJS) that is built based on web services technology to manage desktop grid systems and scheduling jobs.

Manager in the SODJS structure can be implemented in any structure depending on numbers of managers in the network. We allow distributed manager architecture to avoid the disadvantages in a centralized manager and provide more scalability in the management system. Distributed managers are much more dynamic with respect to changes than centralized approaches, because they do not need the state of the system at each step to do their job.

The architecture model of SODJS consists of three types of modules: A ***Manager*** module that should be installed in a server machine. many ***Executer*** modules that are used to execute jobs that are submitted by users and a ***User*** module that can be registered and connected to the system to submit their jobs and retrieve their results (as shown in figure 2). ***Manager*** provides the management middleware that manage the SODJS system while an executer node is any machine that can be registered or connected to SODJS system through executer interface. Any user on the Local Network Area (LAN) can register to the system, connect to the manager and send jobs to the manager.

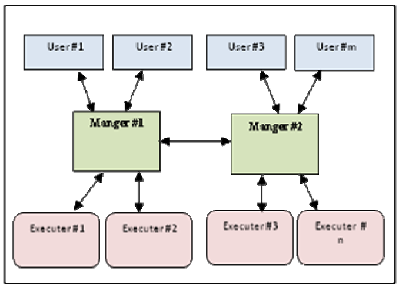

Figure 2: The General Structure of Centralized SODJS Model

SODJS structure is organized into three layers for more simplicity and flexibility. Each layer has certain tasks as shown in Figure 3. The details for each layer are as follows:

- **Interface layer**: provides resource registration, authentication and connection. It also provides job submission to allow users to submit their jobs to the system through user interface.

- **Scheduling layer**: selects resources for jobs depending on the load on each resource. The main functions in scheduling layer are resource selection and job scheduling.

- **Execution layer**: provides the execution service for jobs. This layer executes and monitors jobs, stage job files to mapped resources, release resources after job completion and return results after execution.

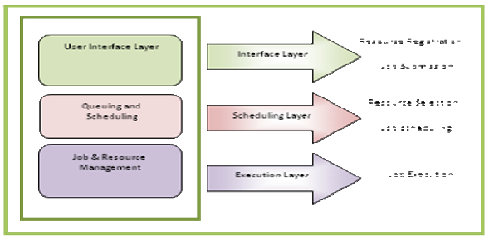

Figure 3: Grid Scheduling Layers

The manager is responsible for scheduling jobs among different available executers. In general, each executer may have single or multiple processors and the processors at executers can be either homogeneous or heterogeneous. Upon arrival, jobs must be assigned to exactly one executer for processing immediately by instantaneous scheduling or wait to be scheduled by the scheduler services. We assume that jobs can be executed on any executer and no parallelism and migration is allowed among executers. The load monitor in each executer is responsible for probing the current state of each executer. Arrived jobs will be placed in a waiting queue at the manager. As the system workload grows heavy, there are more and more jobs waiting in the queue, the scheduler performs load balancing over available executer nodes. The details structure of model is shown in figure 4.

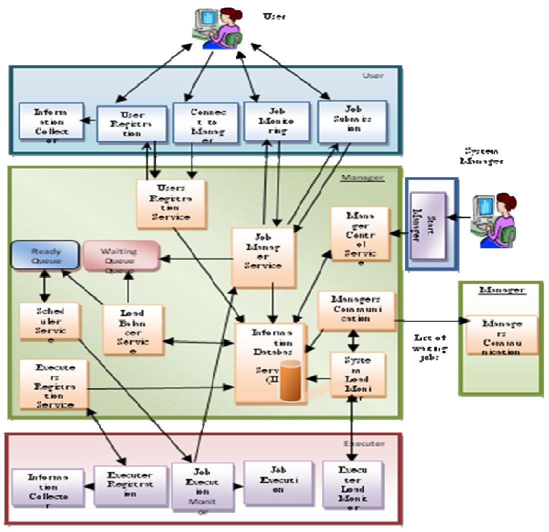

Figure 4: The Architecture of SODJS Model

3.1 Manager Services in SODJS

SODJS Manager manages the machines in the system and it is responsible for naming machines and registering them in the system. It also accepts the jobs from users and schedules them depending on the load-balancing algorithm, then migrate the job to selected Executer machines to execute it and to accept the results. Also, it monitors execution of jobs and reschedule jobs if failure occurs. SODJS Manager consists of numbers of services each of which performs a specific function of the manager, and cooperates with each other to satisfy manager goals and responsibilities. The descriptions of these services are presented as follow:

- **Executers Registration Service :**

This service is responsible for managing executer's requests. It is used to register the new executer machines to the system database, connects executers to the system after verifying authentication and disconnects them from the system. The registration service accepts information about executer machines. After the function of registration completes successfully, the manager generates unique ID to that executer to identify the machine in the system. The Executer ID is used for connecting the machines to the system or disconnecting them.

- **Users Registration Service :**

This service task is managing user's requests to register them into the system. It is used to register the new user machine to the system database, connects users to the system after verifying authentication and disconnects them from the system. The users submit information about their machines to the manager machine. After the function of registration completes successfully, the manager sends the generated unique ID to the user to identify this machine into the system .The user Machine ID is used for connecting the machine to the system or for disconnecting.

- **Job Manager Service** :

The Job manager service collects the information about jobs in the SODJS and stores them in the system database. Job manager service takes a job request from user, stores job files in the manager machine, and stores the job into waiting queue. After executing a job, the job manager service also takes the responsibility of collecting results and sends it back to the user. When the job failed, the job manager restores the job to waiting queue for new scheduling.

- **Load Balancer Service**

The load balancer service is responsible for selecting a specific executer depending on executer availability and load. When the job execution failure in the selected executer, reschedule the fails job on that executer machine to another executer machine to ensure load balancing of global grid system. If it failed to receive executer machine information in a period of validity, re-enter all the jobs scheduled to this executer into job queue. The load balancer accesses executer information in the system database that satisfies load-balancing strategies to maximize the executer utilization and minimize the total job execution time. Then allocate the job to one of the available executer machines. The Load balancer selects a job from waiting queue as First Come First serve (FCFS) algorithm and assigns it to an appropriate executer depending on the executer load and the number of running jobs in each executer. The algorithm of executer selector by load balancer services is shown in Figure 5.

```
Input: Queue of Waiting Jobs J, List of Available Executer nodes E
Begin
For each i job in J
Sort List E depending on Executer Load and Numbers of Running jobs
If E is not empty
Remove i from waiting jobs J List
Queue i job in Ready Queue with e Executer
Else if E is empty
Queue i job in Waiting Queue
End
```

Figure 5: Resource selector algorithm

- **Job Scheduler Service**

Job scheduler service retrieves jobs ready for execution from job ready queue and then dispatches the job to the selected executer. The job scheduler selects the job from the ready queue depending on FCFS algorithm. In addition, the job scheduler is responsible for dispatching the job files to a specific executer.

- **System Load Monitor Service**

This Service monitors the Load of the system by getting the load of each available executer from Executer Load Monitor service in them. This service determines if the system is loaded or not. The service stores information for each executer machine in the database. This information is CPU utilization, Available Memory size, Available hard disk size, and numbers of completed jobs, numbers of failed jobs and numbers of running jobs in the system.

- **Managers Communication Service**

This service is used when architecture of the system has more than one manager. This service is responsible for communicating managers with each other. When one system manager is loaded, it communicates with an unloaded manager and sends waiting jobs to it. A manager is considered loaded when its load is greater than 85% of its maximum load. This service makes our system more scalable, so it can serve many users and jobs at the same time with good performance.

- **Information Database Service (IDS)**

Information database service provides services to enable resource registration while keeping track of a list of available executers in the SODJS system. The manager can query this entity for executers contact, configuration, characteristics and status information. Information contains static information and dynamic information about executer machines and jobs in SODJS.

3.2 Executer Services in SODJS

The Executer module in SODJS system consists of a number of services, each of which performs specific tasks. Executer functions are: request Register machine to SODJS, connect and disconnect executer from SODJS, monitor Executer Status and execute jobs for users and send back the result to manager.

3.3 User Module Services in SODJS

The User module in SODJS structure consists of a number of services, each of which performs specific tasks. The user module functions are: request Register user machine to SODJS, connect and disconnect to the SODJS, submit jobs and require input file to SODJS, manage and monitor submitted jobs.

4. SODJS IMPLEMINTATION AND EVALUATION

In order to validate our approach, we implemented a services oriented job scheduler by using web service technologies. Our prototype is a Microsoft .NET application developed using C Sharp (C#) and Windows Foundation Communication (WCF), which implement web services with Visual Studio 2008.

To evaluate the SODJS system, we made various experiments to measure the efficiency of SODJS scheduling factors. We tested that our system performs its functions by scheduling jobs over available executer machines to balance the load over them. We preformed experiments test on desktop computers in the Computer Laboratory in Computing and Information Faculty in king Abdul Aziz University. The computers that were used in the test are 16 computers connected with each other in a local area network (LAN).

The jobs in SODJS denote applications executable files that the SODJS system schedules and runs them. Jobs are computational jobs that need high performance requirement in processing. Jobs in our experiments are chosen from scientific problems such as security algorithms and cryptography.

To test the efficiency of the decentralized scheduler over the centralized job scheduler we compared the load of the system when the numbers of jobs and executers nodes in the centralized job scheduler and in decentralized scheduler are increased. In addition, we compared the average waiting time and the average execution time.

Table 2: System Load in centralized and decentralized scheduler

# Jobs	Number of Managers	
	1M and 6E	2M and 6E
2	36.5	36.3
3	42.6	41.5
4	43.7	42.9
5	45.25	43.2
6	48.5	46
7	49.9	46.8
8	52.3	48.7

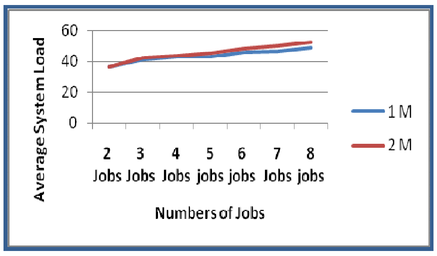

Figure 6: System load in Centralized and Decentralized

We perform experiment using two managers with six executer nodes and compute the execution time for jobs. The results of experiments are shown in figure 6, 7. Comparing the results from these experiments with result when using one manager, we find that using decentralized scheduler gives less execution time comparing with using centralized manager especially when numbers of jobs increased.

Table 3: Execution Time in Centralized and Decentralized

# Jobs	Number of Managers	
	1M and 6E	2M and 6E
2	7.05	7.94
3	11.18	11.14
4	12.37	12.06
5	16.5	16.31
6	31.17	30
7	39.4	37.9
8	40.34	38.13

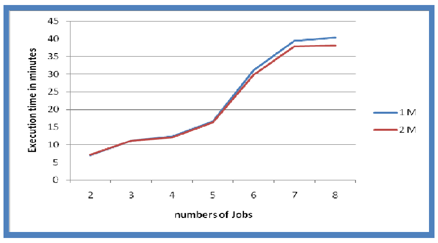

Figure 7: Execution time in Centralized and Decentralized

The waiting time is decreased when using more a decentralized scheduler because the jobs are distributed over managers and this makes management of jobs is easier than in a centralized manager.

Table 4: Waiting Time in centralized and decentralized

# Jobs	Number of Managers	
	1M and 6E	2M and 6E
2	0.35	0.33
3	0.4	038
4	1.39	1.25
5	5.5	4.1
6	9.31	7.90
7	15.25	12.18
8	24.26	18.37

As shown in Table 4, using a decentralized scheduler in a large number of jobs makes the system to run jobs with a smaller execution time i.e. it gives good performance in minimizing execution time compared to centralized manager.

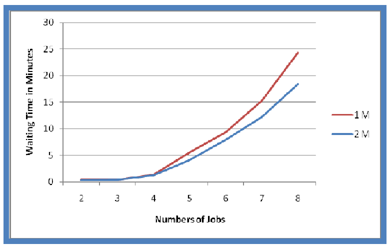

Figure 8: Waiting time in Centralized and Decentralized

4. CONCLOUSION

In this research, we introduced a new Service Oriented Decentralized Job Scheduler (SODJS) that has been built based on web services technology for managing desktop grid systems. SODJS has been designed, implemented and evaluated. The benefits of SODJS are: increasing available computing power, maximizing the use of new/existing resources, minimizing jobs execution time. Finally, SODJS hides the complexity of the grid to users.

SODJS can be installed as centralized job scheduler on a single machine or each web service of the manager can be deployed on a different machine to increase SODJS performance and scalability if the numbers of both users and jobs are increased. This means that the performance of SODJS is not affected (decreased) by jobs scalability. SODJS can be used as centralized manager architecture or decentralized manager architecture. This means that SDGM has decentralized manager advantages, which include scalability, fault tolerance and peer-to peer functionality. SODJS manages jobs and executer nodes with good performance by minimizing job execution time and waiting time.

5. REFERENCES

[1] Krauter, K., Buyya, R., and Maheswaran, M. (2002), "A taxonomy and survey of grid resource management systems for distributed computing". *Int. J. of Software Practiceand Experience*, 32(2): PP. 135–164.
[2] [Foster, I., Kesselman, C., and Tuecke, S. (2001), "The Anatomy of the Grid: Enabling ", Scalable Virtual Organizations. *Int. J. Supercomp. App.*, 15(3): PP.200–222.
[3] Rawat, S. and Rajamani, L. (2009), "Experiments with CPU Scheduling Algorithm on a Computational Grid ", IEEE International Advance Computing Conference (IACC 2009), PP. 71-75
[4] Chapman, C., et al. (2004), "Condor Services for the Global Grid ", National Environment Research Council, 2004.
[5] Schopf, J. (2003), "TEN ACTIONS WHEN GRID SCHEDULING"
[6] Veldman, E. et al. (2009), "Technical Benefits of Distributed Storage and Load Management in Distribution Grids", IEEE Bucharest Power Tech Conference, June 28th - July 2nd, Bucharest, Romania 2009.