International Journal of Computer Applications (0975 – 8887)
Volume 107 – No 4, December 2014

Comparative Analysis of Scheduling Algorithms in Computational Grid Environment

Uttaran Bhattacharya
Dept. of Computer Science and Engineering,
Institute of Engineering & Management

Dipannita Dey
Dept. of Computer Science and Engineering,
Institute of Engineering & Management

ABSTRACT

This paper tracks the development of Grid Computing since its inception in the late 1990s to its dominance in today's world of Distributed Computing and Information Technology. It focuses on the recent developments that spurred our interest to take up this field of research with emphasis on the algorithms we are researching for job scheduling and load balancing in the Grid Environment. The entire structure of the Grid Environment is dynamic and hybrid by nature, changing with the availability and the capability of resources or hosts that perform user tasks and the Quality of Service requirements of the tasks themselves. This makes the problem of developing an optimal task-to-resource schedule that ensures proper load balancing and also produces the minimum overall makespan (time to complete scheduled tasks) an NP-Hard Problem. Our attempt in this research is not to find the optimal solution for this problem, but to analyze and test various algorithms that produce acceptable performance in the commonly occurring practical scenarios.

The algorithms considered for research and analysis include the classical First Come First Served Algorithm that schedules jobs to the best possible resources on arrival basis, Adaptive Workload Balancing Algorithm that essentially considers the job pool and the resource pool at a given point of time to give the optimal load-balanced schedule, and a brief insight into Genetic Algorithms that apply heuristics to perform optimization for some job criteria. Lastly, we include two algorithms, Fastest Processor to Largest Task First and Nearest Deadline First Served, which we have indigenously developed and are currently testing for performance.

General Terms

Grid Computing, Grid Environment, Scheduling Algorithms.

Keywords

Clients, Users, Resources, Resource Domains, Hosts, Processing Elements (PEs), Jobs, Tasks, Workload, Makespan, FCFS, AWLB, FPLTF, NDFS, Globus Toolkit, Unicore.

1. INTRODUCTION

The focus of development in the field of Information Technology has, in the recent years, been largely concentrated to producing, maintaining, manipulating and accessing data on a global scale. The digital community of today is moving away from the confines of locally produced and consumed data and is trying to bring the entire world's data under a common roof, popularly named the 'Cloud' [1].

With the advent of more and more advanced technologies in this regard, researchers are designing more and more complex systems to manage and process large sets of data and balance workload allocation for parallel and distributed resources over the world.

Grid computing is possibly one of the most significant developments from this area of research. It burst to prominence in the early 2000s, especially after American scientist-duo Ian Foster and Carl Kesselman published their seminal work, "The Grid: Blueprint for a New Computing Infrastructure" (1999) [2]. Since then, the Grid environment has developed and advanced remarkably to dominate the world of computer science and Information Technology today.

The Grid is a collection of computer resources that are used to perform various kinds of computational and communicational jobs among others, as per incoming requests [3][4]. These resources are often distributed over diverse geographical locations and have dynamic working capabilities (i.e., performance level for the same job can vary with time depending on hardware state, software availability and other local delimiting factors). Job scheduling in the Grid environment focuses on the requests from the clients and it is the responsibility of the Grid Manager or Grid Broker to distribute the workload amongst the various remote resources based on various characteristics, namely, computational power (generally measured in Millions of Instructions Per Second (MIPS) or Floating-point Operations Per Second (FLOPS)), memory capacity of worker nodes, network links bandwidth, I/O activities, etc for efficient execution of the requests.

2. THE GRID SYSTEM

The Grid can be visualized as a network of computing resources that ideally have unbounded computing capabilities and are perpetually available. This notion of a Grid network is vital to understanding the philosophy it strives to follow – that given any nature of job at any point of time, the resources will be able to complete the job within the given constraints and return the result to the requestor. This is somewhat analogous to the electric Power Grid that we see today – electricity produced from Generator Stations is available perpetually on the physical grid from where Consumers can draw electricity according to their need – and hence the use of the name.

Figure 1 shows a basic logical architecture of a simple Grid system [5]. Grid Broker acts as a mediator between the Users and the Resources and as such is the extra "third" layer to the Grid Architecture. The advantage of adding this third layer is, of course, division of labor, which makes the overall architecture easier to implement. The disadvantage, however, is that the Broker becomes the single point of failure in the architecture and so any downtime of the Broker is not acceptable to the point of being fatal. Grid Broker interacts mainly with 3 components – Grid Applications, which maintains a list of all available Resource Domains connected to the Grid, Grid Information

Service, which collects all necessary information regarding the resources required for scheduling, and Job Launching and Monitoring, which delegates the jobs to the resources, monitors performance and sends feedback to the Broker.

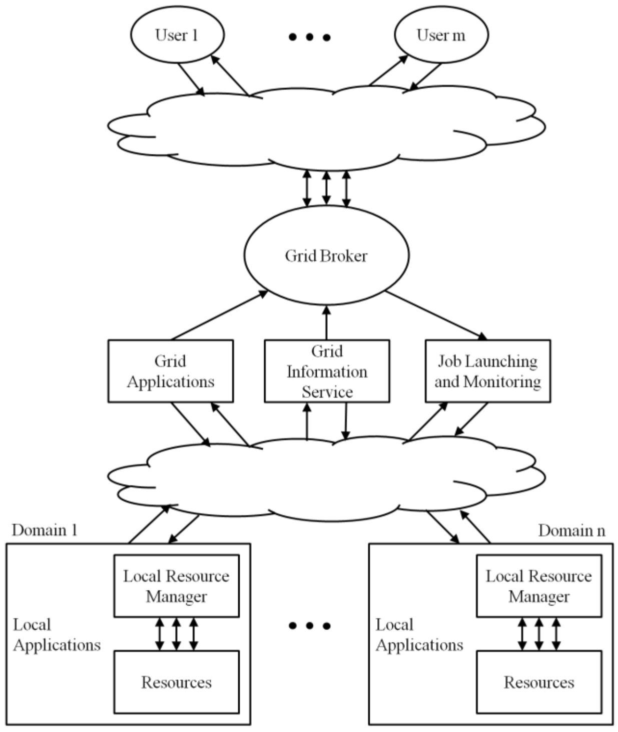

Figure 1: Basic Logical Architecture of a Simple Grid System

The Grid System is a natural evolution of the Cluster Computing paradigm with more dynamicity and variety in the resource pool, and as a result, can cater to a larger and more complex pool of job requests. What distinguishes Grid Computing from conventional cluster systems is that Grids tend to be more loosely coupled, heterogeneous, and geographically dispersed [7]. An efficient Grid system should take care of several facts as given under.

a. Employing dynamic parallel algorithms that control distributed resources.

b. Utilization of resources in a domain-specific fashion to increase throughput, or to reduce execution costs.

c. Balancing the entire workload among all the available resources to extract maximum performance from the system as a whole.

d. Maintaining proper interconnection between processing elements and Grid Broker, and between Grid Broker and the users so everyone is at a common level of understanding.

e. Heterogeneity, autonomy, scalability and adaptability, resource selection and computation-data separation of a grid bring obstacles to design efficient and effective load balancing for Grid environments.

f. Carefully monitoring the performance of the resources so that if a job is interrupted, or a resource breaks down, then the job rescheduled at minimum additional expense.

g. Integrating and coordinating multiple schemes involved in managing different parts of the Grid Environment, thus promoting inter-organizational collaborations.

From the above discussions, it is abundantly clear that job scheduling in the Grid environment is the single most important feature of the system. The entire efficiency and usability of the Grid paradigm is dependent on the underlying algorithms that receive, analyze and schedule jobs to the available resources. For without proper scheduling, the resources will tend to miss the Quality of Service requirements set by the requester, which will ultimately render the entire system useless. Even more concerning is that if jobs are, for example, delay-sensitive (e.g., processing of Air Traffic Control data) then improper scheduling will tend to cause adverse effects on a mass scale.

In the following articles, we elaborate on the scheduling algorithms that we are currently researching for the Grid Environment. The algorithms include First Come First Served (FCFS), Adaptive Workload Balancing (AWLB), Genetic Algorithms (GAs) (brief account), Fastest Processor to Largest Task First (FPLTF), and Nearest Deadline First Served (NDFS). For the ease of discussion, we are defining the most frequently used keywords in the text as under:

- **Task, Job and Application**: A Task is the smallest unit of work done by a resource (e.g., performing a mathematical computation), whereas a Job or an Application is a collection of related tasks and sub-jobs [5]. However, for our practical purpose, we use the terms interchangeably.

- **Resource, Node and Site**: A Resource is the smallest computing unit that performs a task (e.g., a data processor), whereas a Node or a Site is an autonomous collection of resources [5]. However, for our practical purpose, we use the terms interchangeably.

3. SCHEDULING ALGORITHMS FOR THE GRID ENVIRONMENT

3.1. First Come First Served (FCFS)

Here the task has the probability of being allocated to any of the resources or Processing Elements (PEs). So, the FCFS should consider all these possibilities and identify which node will complete the task earliest [8][9]. As a result, the aim of FCFS is to find the earliest possible time for each task to complete, according to the sequence of the task arrivals. Figure 2 shows a flowchart for the FCFS implementation.

Whenever a task arrives to the Scheduler, a function *f* is used determine its processing requirements (**Step 1**). *f* gives a quantified measure of whatever job parameter(s) that we are trying to prioritize, e.g., the total number of calculations to perform measured in Flops/s. The Scheduler then estimates the corresponding processor capability μ_f for every PE in the resource pool and chooses 2 PEs that best map to the task requirements defined by *f* (**Step 2**). The better of the two, the 'Primary', is given the task to perform and the other, the 'Secondary' is kept as backup in case the Primary encounters some unprecedented fault and/or error conditions (**Step 3**). In case both the PEs are equally placed to perform the task, the Primary and the Secondary can be chosen randomly. The resource pool is then updated to reflect this allocation (**Step 4**).

This routine can be made more efficient by having the Primary insert 'Checkpoints' while performing the task. A Checkpoint denotes a point in course of the task where the System is in a stable state. These Checkpoints are transferred to the Secondary after creation, so that if the Primary fails, the Secondary can pick up task execution from the most recent Checkpoint and bypass all the work done by the Primary up to that Checkpoint.

Although the FCFS algorithm is proved to be efficient under some circumstance, it is clear that if the number of PEs in the Grid increases, the overhead incurred by the FCFS algorithm will increase considerably since the agent should probe every node to collect the required information. In addition, the detection of the sequence of the task arrivals determines the order of task execution and limits this algorithm from gaining more desirable performance.

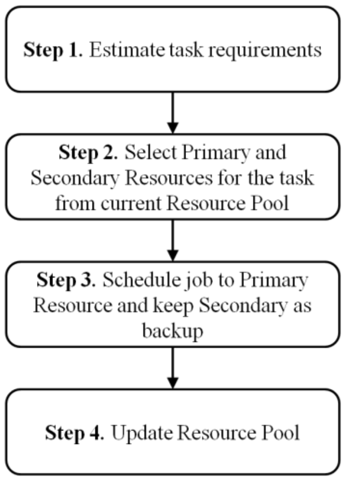
Step 1. Estimate task requirements
Step 2. Select Primary and Secondary Resources for the task from current Resource Pool
Step 3. Schedule job to Primary Resource and keep Secondary as backup
Step 4. Update Resource Pool

Figure 2: FCFS

3.2. Adaptive Workload Balancing (AWLB)

The adaptive workload balancing technique [10][11] takes into account the heterogeneity and the dynamicity of the Grid resources, estimates and predicts the initially unknown application requirements and provides resource selection and optimal mapping of parallel processes to the available resources.

The load balancing steps aim at optimizing the load distribution among the resources after performing the check against the restricting factors such as memory deficiency. Therefore the theory is given under the assumption that the resources are fixed for a single load-balancing loop. Another prerequisite is that the application is already implemented as a parallel program and is able to distribute the workload by chunks of controllable size.

Figure 3 shows a flowchart for the AWLB implementation. It is broadly divided into two levels: the Resource Level and the Application or Job Level.

At the **Resource Level**, the Scheduler benchmarks and ranks the available resources in the Resource Pool according to the metric μ (**Steps R1 through R3**), defined as:

$$\mu = p / n,$$

where, p = processing capability of the node (in Flops/s)

and n = network bandwidth available at this node (in bits/s).

At the **Application Level**, the application or job requirements are estimated and parameterized according to the metric 'f_c' (**Step A1**), defined as:

$$f_c = N_{comm} / N_{calc}$$

where, N_{comm} = total data to be exchanged by the application (in bits)

and N_{calc} = total processing required by the application (in Flops)

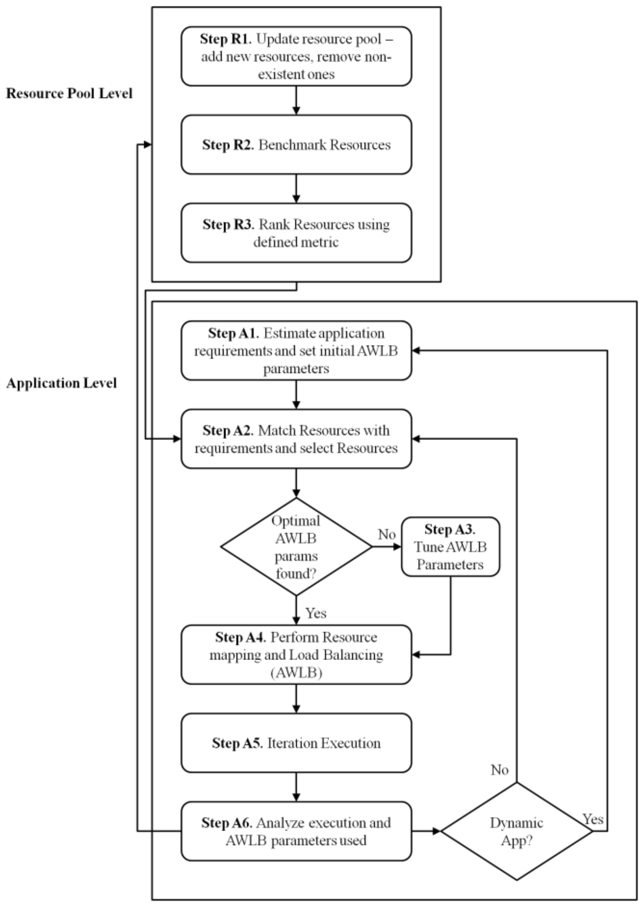
Resource Pool Level
Step R1. Update resource pool – add new resources, remove non-existent ones
Step R2. Benchmark Resources
Step R3. Rank Resources using defined metric
Application Level
Step A1. Estimate application requirements and set initial AWLB parameters
Step A2. Match Resources with requirements and select Resources
Optimal AWLB params found?
No
Step A3. Tune AWLB Parameters
Yes
Step A4. Perform Resource mapping and Load Balancing (AWLB)
Step A5. Iteration Execution
No
Step A6. Analyze execution and AWLB parameters used
Dynamic App?
Yes

Figure 3: AWLB

The Scheduler then selects the most promising resource (**Step A2**) and tests the performance of the selected resource in the successive iterations (**Steps A3 through A5**). If performance degrades, or if the requirements of the application change, it performs ad-hoc load balancing and selects new resources to continue with running the application (**Step A6**), thus 'adapting' dynamically to the change in the environment.

3.3. Genetic Algorithms (GAs)

The Genetic Algorithms (GAs) [8] focus on an overall performance of the tasks in the job queue. Rather than looking for an earliest completion time for each task individually, the main objectives of the GA are to achieve the minimum of makespan, which represents the latest completion time among all the tasks involved, maximum node utilization and a well-balanced load across all the PEs. GAs have gained more application in load balancing for the Grid Environment in recent years.

The GA is generally brought into effect when the queue of pending tasks reaches a certain limit. It starts off by choosing a set of random solutions to the scheduling problem, called a **population**. The individual solutions in the population are called **chromosomes**. These chromosomes are then evaluated using a **heuristic** or **fitness function**. The fitness function is so designed in our case that the chromosomes giving lesser overall makespan have higher fitness values over others. The chromosomes giving the best fitness values are chosen as the '**good parents**' and are either (a) merged in pairs using a **crossover operator** or (b) modified individually using a **mutation operator** to create a new set of chromosomes, called their '**offspring**'. A new generation is then formed by (a) **selecting**, according to fitness values, some of the parents and offspring and (b) **rejecting** others so as to keep the population size constant. After several generations, the Algorithm converges to the best chromosome, which hopefully represents the optimum or suboptimal solution to the problem.

3.4. Fastest Processor to Largest Task First (FPLTF)

Fastest Processor to Largest Task First (FPLTF) [12] schedules tasks to resources in the Grid according to the load of the tasks and available capability of the resources, such that the most capable resource gets the 'heaviest' task. The Algorithm is subdivided into 3 major parts – the Resource Level, the Task Level and the Execution Level, as depicted in Figure 4.

At the **Resource Level**, the capabilities of all available resources are measured using the metric μ (**Steps R1 and R2**). μ can be defined according to the system design; one such definition is:

$$\mu = \mu_p / \mu_n,$$

where μ_p gives a measure of the available processing power and μ_n gives a measure of the available network bandwidth of the node, relative to some standard measures. The Scheduler maintains a list of all nodes as a priority queue based on their values of μ (**Step R3**). This priority queue gets updated time-to-time and also when a node is allocated to a task.

At the **Task Level**, the Scheduler maintains a queue of the incoming tasks. When the queue reaches a certain size, the Scheduler picks the tasks from the queue (**Step T1**) and analyzes the load requirement of the tasks, determined by the metric f that should correspond to the metric μ (**Step T2**). Considering our example, f can be:

$$f = f_p / f_n$$

where, f_p and f_n respectively give measures of the total processing and bandwidth requirements of the task relative to some standard measures. It then forms a priority queue of the tasks according to their values of f (**Step T3**).

Finally, at the **Execution Level**, the Scheduler starts scheduling in order: the first task in the task queue to the first node in the resource queue, the second task in the task queue to the second node in the resource queue and so on till all the jobs are scheduled (**Steps E1 through E5**).

3.5. Nearest Deadline First Served (NDFS)

Nearest Deadline First Served (NDFS) [13] is a dynamic process for task scheduling in Grid Environment where tasks with nearer

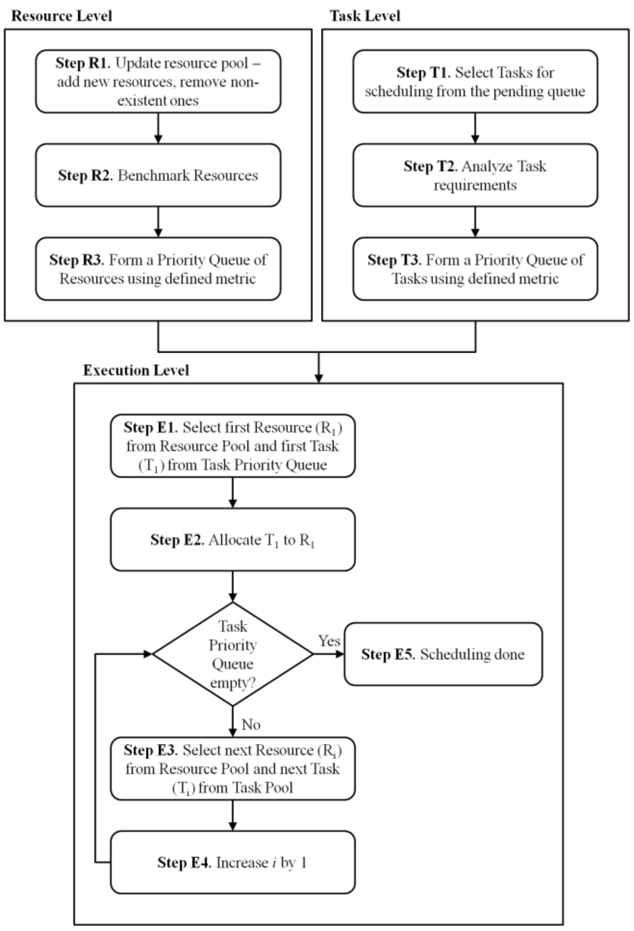
Resource Level
Task Level
Step R1. Update resource pool – add new resources, remove non-existent ones
Step R2. Benchmark Resources
Step R3. Form a Priority Queue of Resources using defined metric
Step T1. Select Tasks for scheduling from the pending queue
Step T2. Analyze Task requirements
Step T3. Form a Priority Queue of Tasks using defined metric
Execution Level
Step E1. Select first Resource (R_1) from Resource Pool and first Task (T_1) from Task Priority Queue
Step E2. Allocate T_1 to R_1
Task Priority Queue empty?
Yes
Step E5. Scheduling done
No
Step E3. Select next Resource (R_i) from Resource Pool and next Task (T_i) from Task Pool
Step E4. Increase i by 1

Figure 4: FPLTF

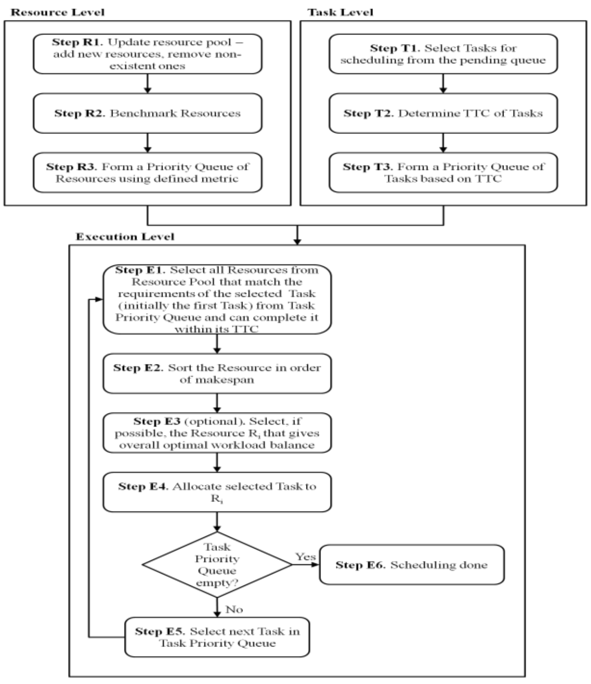
Resource Level
Task Level
Step R1. Update resource pool – add new resources, remove non-existent ones
Step R2. Benchmark Resources
Step R3. Form a Priority Queue of Resources using defined metric
Step T1. Select Tasks for scheduling from the pending queue
Step T2. Determine TTC of Tasks
Step T3. Form a Priority Queue of Tasks based on TTC
Execution Level
Step E1. Select all Resources from Resource Pool that match the requirements of the selected Task (initially the first Task) from Task Priority Queue and can complete it within its TTC
Step E2. Sort the Resource in order of makespan
Step E3 (optional). Select, if possible, the Resource R_i that gives overall optimal workload balance
Step E4. Allocate selected Task to R_i
Task Priority Queue empty?
Yes
Step E6. Scheduling done
No
Step E5. Select next Task in Task Priority Queue

Figure 5: NDFS

deadlines (measured by the parameter '**time to complete**' or **TTC**) are scheduled before others. The Algorithm is also subdivided into 3 major parts – the Resource Level, the Task Level and the Execution Level, as depicted in Figure 5.

At the **Resource Level**, the Scheduler maintains a priority queue of the available resources based on the values of μ as before (**Steps R1 through R3**). In this case, however, μ is determined by the usage still empty in the node under consideration, in addition to its μ_p and μ_n values. In other words, we take into account the workload currently being handled by the nodes, as this will affect the processing speed of new tasks loaded to those nodes. The current total usage of a node is measured by the parameter k that ranges from 0 (no usage) to 1 (full usage). The revised value of μ is now given as:

$$\mu = (1 - k)(\mu_p / \mu_n)$$

At the **Task Level**, the Scheduler maintains a priority queue of tasks based on their TTC. It also checks the pending queue of tasks as soon as the size of the queue exceeds a certain limit (called the '**critical size**') and adds tasks to the priority queue according to TTC (**Steps T1 through T3**).

Once the tasks are placed in the priority queue, the Scheduler moves on to the **Execution Level**. It picks the first task from the priority queue and selects all the resources from the available resource pool that match the requirements of the job (given by mapping f toμ) and can complete this task within its TTC (**Step E1**). It then sorts the selected resources in decreasing order of makespan to determine who can complete the task fastest (**Step E2**). Next, it attempts to determine selecting which resource will give an overall optimal workload balance by using a suitable GA (**Step E3**). This step is optional and is carried out only if the task at hand has sufficient TTC. Finally, it allocates the task to the best possible resource (the Primary) and keeps the second best (the Secondary) as a standby. We can again use the concept of 'Checkpoint' here to ensure that no redundant work is done is the Primary fails while executing the task. The Scheduler now repeats this entire procedure for all the remaining tasks in the priority queue till all of them are scheduled (**Steps E5 and E6**).

The estimation of the critical size of the queue is vital for the successful running of this Algorithm. Otherwise, tasks with nearer deadlines might miss their deadlines while waiting in the queue, resulting in the violation of QoS. The critical size is estimated heuristically by checking the **miss-to-Met (m2M) ratio**, which determines how many deadlines are 'missed' for every deadline 'met'. The critical size begins with a predefined value that increases additively till the m2M ratio exceeds an upper limit (e.g., 0.1). After that critical size is repeatedly halved till m2M ratio falls below the upper limit and then the entire process is repeated. The main challenge to implement this heuristic is the dynamic workload and deadline demand of the incoming tasks, which can make the m2M ratio highly dispersed with time. However, for practical implementations, an overall uniformity in the nature of the tasks can be established by considering all the tasks arriving over a sufficiently large period of time.

NDFS can guarantee that all the deadlines are met provided that the total CPU utilization is not more than 100%. However, when the system is overloaded, the set of processes that will miss deadlines becomes unpredictable and Algorithm starts losing its efficiency.

4. COMPARATIVE STUDY OF THE SCHEDULING ALGORITHMS

Table 1 gives a comparative study of the Algorithms discussed so far. The data for the table have been obtained through experimentation and extrapolation of the results.

Table 1: Comparative Study of the Scheduling Algorithms

Algorithm	**AWLB**	**FCFS**	**GA**	**FPLTF**	**NDFS**
Time Complexity	High	Low	High	Low	Low
Makespan	Application-specific	Application-specific	Optimal	Application-specific	Optimal
Connection Bandwidth	Application-specific	Low	High	Low	Low
Workload Balance	Better for Communication-intensive Applications	Not Optimal	Varies with Nature of Consecutive Applications	Better for Computation-intensive Applications	Application-specific

5. MAJOR TOOLKITS FOR THE GRID ENVIRONMENT

5.1. Globus Toolkit

5.1.1. Overview

• *Developed by:* Globus Alliance

• *Type:* Open Source

• *Compatible Operating Systems:* Linux, Solaris, Mac OS X, FreeBSD, HP-UX, AIX

• *License:* Apache License

• *First Release:* 1998

• *Last Stable Release:* Version 5.2.2, July 24, 2012

The Globus Toolkit is one of the foremost and widely used toolkits developed to work in the Grid Environment [14]. It was originally built on the Open Grid Services Architecture (OGSA), which has now been superseded by the Web Services Resource Framework (WSRF). It also employs the WS-Management and has implementations for the following protocols recommended by the Open Grid Forum (OGF), a community of users, developers and vendors for standardization of Grid Computing.

a. *Resource Management:* Grid Resource Allocation and Management (GRAM) Protocol,

b. *Information Services:* Monitoring and Discovery Service (MDS),

c. *Security Services:* Grid Security Infrastructure (GSI), which uses asymmetric encryption for data security and digital signature technology for user authentication,

d. *Data Movement and Management:* Global Access to Secondary Storage (GASS) and GridFTP.

The resource manager of Globus Toolkit, GRAM, officially supports the following job schedulers (batch-queuing systems):

a. *Portable Batch System*

b. *Condor High-Throughput Computing System*, for distributed parallelization of computationally intensive tasks,

c. *Platform Load Sharing Facility (LSF)*

d. *Sun Grid Engine*, integrated by third-party methods,

e. *Simple Linux Utility for Resource Management (SLURM)*, used via shell wrappers.

We have used Globus Toolkit version 5.2 to set up the Grid Environment and test and analyze all the scheduling algorithms.

5.1.2. Download and Install

The complete Globus repository is available at http://globus.org and can be downloaded and installed directly from there using the following command in Linux:

```
rpm -i Globus-repo-config.fedora-<version>.noarch.rpm
yum groupinstall globus-gridftp globus-gram5
```

5.1.3. Setting Up Secured Environment

Globus requires a secured environment to be set between the client (or user) and the server (or host) before it can function. The following command is used in Linux to set up the secured environment:

```
wget http://globus.org/toolkit/docs/5.2/5.2.0/admin/

quickstart/setup-simplecash setup-simpleca -y
```

5.1.4. Creating Certificates

The user and the host need to hold valid security certificates issued by a common Certification Authority (CA) so that they can communicate through the secure environment.

The following commands are used in Linux to create the Host Certificates:

```
grid-cert-request -host 'hostname'

grid-ca-sign -in hostcert_request.pem –out hostsigned.pem
```

The following commands are used in Linux to create the User Certificates:

```
grid-cert-request

grid-ca-sign -in usercert_request.pem -out signed.pem
```

5.1.5. Job Posting

Once the secure environment is set up between the user and the host, we can post jobs to the host from the user and analyze performance of the algorithms in real time. The Globus Resource Allocation Manager (GRAM) is used to perform the posting tasks.

5.2. UNICORE – A Brief Overview

• *Developed by:* German Ministry for Education and Research (BMBF)

• *Type:* Open Source

• *Compatible Operationg Systems:* Linux, Solaris, Mac OS X

• *License:* BSD License

• *First Release:* 1998

• *Last Stable Release:* UNICORE 6, August 28, 2007

UNICORE (UNiform Interface to COmputing REsources) was primarily built for resources like supercomputers, cluster systems and information stored in databases, and served as an alternative for the Globus Toolkit in European countries [15]. It evolved to a middleware in many supercomputer centers and served as the basis for numerous research projects.

The UNICORE architecture consists of three layers:

a. *User Layer*, represented by various clients like the UNICORE Rich Client (a GUI based on the Eclipse Framework) and the UNICORE Commandline Client (UCC).

b. *Server Layer*, with whom the user interacts using Simple Object Access Protocol (SOAP) Web services. XML documents are used to transmit platform and site-independent data and computational related tasks. The servers can be accessed using only the Secure Socket Layer (SSL) protocol.

c. *Target System Layer*, which contains the necessary resources to perform the tasks requested by the user. UNICORE supports many different system architectures, provided and maintained by various organizations. An additional server, called UNICORE/X, can be used to access resources like supercomputers, Linux clusters, or even single PCs. An important feature of UNICORE/X is that it creates target-system specific jobs from the XML description of user requests.

The services provided by UNICORE include submission and management of user jobs, storage, transfer and accessing of resource files to execute the jobs, and submission and management of workflow to maintain the resources at an optimal level of performance.

Security within UNICORE is maintained using permanent X.509 certificates issued by a trusted Certification Authority (CA). These X.509 certificates provide single sign-on in the UNICORE client and specify standard formats for public key certificates, certificate revocation lists, attribute certificates and certification path validation algorithm.

We have used UNICORE in our initial phases of the research but later on switched to Globus Toolkit for the latter's better support in the Linux environment. However, UNICORE remains a viable alternative for working in the Grid environment.

6. CONCLUSION

We have successfully implemented the algorithms described so far. An exhaustive analysis of the algorithms with varying nature of incoming jobs and varying population of the resource pool, both qualitatively and quantitatively shows that no algorithm gives a perfectly optimal solution to the scheduling problem, as we proposed at the very beginning of this report. However, all the algorithms perform satisfactorily within their own defined constraints. Hence, the next step in course of this research is to identify the ranges of best performance of each of these algorithms in terms of QoS requirements of the incoming jobs and the size and heterogeneity of the resource pool, and form a Master Scheduling Algorithm that will be able to decide which of these algorithms to implement under what circumstances.

7. ACKNOWLEDGEMENTS

We would sincerely like to thank all the Professors of the Computer Science Department for their continued patience and support during the course of this ongoing research, and also for providing the inspiration to make this paper a reality.

We would also like to sincerely thank our respective family members for their encouragement, their enthusiasm in the subject matter and their ever-present faith in us.

8. REFERENCES

[1] Thomas B Winans and John Seely Brown, *Cloud computing A collection of working papers*, Delloite Development LLC, 2009.

[2] I. Foster and C. Kesselman (editors), *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.

[3] R. Buyya, D. Abramson, J.Giddy and H.Stockinger, *Economic Models for Resource Management and Scheduling in Grid Computing, in J of concurrency and computation: Practice and Experience*, Volume 14, Issue 13-15, PP. 1507-1542, Wiley Press, December 2002.

[4] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, *Grid Information Services for Distributed Resource Sharing*, Proceedings of the 10th IEEE International Symposium on High- Performance Distributed Computing (HPDC-10), pp. 181-194, San Francisco, California, USA, August 2001.

[5] Fangpeng Dong and Selim G. Akl, *Scheduling Algorithms for Grid Computing: State of the Art and Open Problems*, Technical Report No. 2006-504, School of Computing, Queen's University, Kingston, Ontario, January 2006.

[6] Pinky Rosemarry, Payal Singhal and Ravinder Singh, *A Study of Various Job & Resource Scheduling Algorithms in Grid Computing*, International Journal of Computer Science and Information Technologies, Vol. 3 (6), 2012, 5504-5507.

[7] Rajkumar Buyya, David Abramson and Jonathan Giddy, *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid*, High Performance Computing Asia 2000, Beijing, China, May 14-17, 2000, pp. 283-289.

[8] Yajun Li, Yuhang Yanga, Maode Mab, Liang Zhou, *A hybrid load balancing strategy of sequential tasks for grid computing Environments*, Future Generation Computer Systems 25 (2009) 819-828.

[9] Raksha Sharma, Vishnu Kant Soni et al., *An Agent Based Dynamic Resource Scheduling Model with FCFS-Job Grouping Strategy in Grid Computing*, World Academy of Science, Engineering and Technology, 2010, pp. 467-471.

[10] Vladimir V. Korkhova, Jakub T. Moscicki, Valeria V. Krzhizhanovskaya, *Dynamic workload balancing of parallel applications with user-level scheduling on the Grid*, Future Generation Computer Systems 25 (2009) 28–34.

[11] D. Daniel, Mrs. S.P. Jeno Lovesum M.E., D. Asir and A. Catherine Esther Karunya, *Adaptive Job Scheduling with Load Balancing for Workflow Application in Grid Platform*, International Journal of Computer Engineering and Technology, Volume 2 Number 1, January 2011, pp. 09-21.

[12] Jairam Naik K., K. Vijaya Kumar and N. Satyanarayana, *Scheduling Tasks on Most Suitable Fault tolerant Resource for Execution in Computational Grid*, International Journal of Grid and Distributed Computing, Vol. 5, No. 3, September, 2012, pp. 121-132.

[13] Sukalyan Goswami and Ajanta De Sarkar, *A Comparative Study of Load Balancing Algorithms in Computational Grid Environment*, Proceedings of the Fifth International Conference on Computational Intelligence, Modelling and Simulation by IEEE Computer Society, 2013, pp. 99-104.

[14] Luis Ferreira, Viktors Berstis, Jonathan Armstrong et al., *Introduction to Grid Computing with Globus*, First Edition, IBM RedBooks, December 2002.

[15] Bernd Schuller and the UNICORE team, *General Introduction to UNICORE 6*, Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, March 17, 2010, OGF28 Munich.