Hierarchical Scheduling on Grid Environment

Dr. P. Varaprasada Rao
Professor
Department of Computer Science and Engineering
Gokaraju Rangaraju Institute of Engineering & Technology Jawaharlal Nehru Technological University, Hyderabad, India, 501301

Abstract— *Heterogeneous computing systems, such as Grid computing system requires powerful job scheduling algorithm. The major function of any job scheduling algorithm is to allocate the optimal resource to a job. Workflow scheduling is one of the key issues in the management of workflow execution. Scheduling introduces allocating suitable resources to workflow tasks so that the execution can be completed to satisfy objective functions specified by users. In this paper, we investigate existing workflow scheduling algorithms developed and deployed by various Grid projects. This paper will describe development of Quality of Service-constraint based workflow scheduling to analyze users' QOS requirements and map workflows on suitable resources such that the workflow execution can be completed to satisfy users' QOS constraints. In such "pay-per-use" Grids, workflow execution cost must be considered during scheduling based on users' QoS constraints. Here we propose a budget constraint based scheduling, which minimizes execution time while meeting a specified budget for delivering results. We also propose a deadline constraint based workflow scheduling algorithm that minimizes execution cost while meeting the deadline for delivering results. To evaluate the performance of grid resources, GridSim [1] is used. In this project a new type of genetic algorithm is developed to solve the scheduling optimization problem and we test the scheduling algorithm in a simulated Grid testbed.*

Keywords— Workflow scheduling, Inter-dependent tasks, Distributed resources, Heuristics

I. INTRODUCTION

Clusters, Grids, and peer-to-peer (P2P) networks have emerged as popular paradigms for next generation parallel and distributed computing. They enable aggregation of distributed resources for solving large scale problems in science, engineering, and commerce. In Grid and P2P computing environments, the resources are usually geographically distributed in multiple administrative domains, managed and owned by different organizations with different policies, and interconnected by wide-area networks or the Internet. This introduces a number of resource management and application scheduling challenges in the domain of security, resource and policy heterogeneity, fault tolerance, continuously changing resource conditions. The resource management and scheduling systems for Grid computing need to manage resources and application execution depending on either resource consumers' or owners' requirements, and continuously adapt to changes in resource availability. The management of resources and scheduling of applications in such large-scale distributed systems is a complex undertaking. In order to prove the effectiveness of resource brokers and associated scheduling algorithms, their performance needs to be evaluated under different scenarios such as varying number of resources and users with different requirements. In a Grid environment, it is hard and even impossible to perform scheduler performance evaluation in a repeatable and controllable manner as resources and users are distributed across multiple organizations with their own policies. To overcome this limitation, a Java-based discrete-event Grid simulation toolkit is developed called GridSim. The toolkit supports modeling and simulation of heterogeneous Grid resources (both time- and space-shared), users and application models. It provides primitives for creation of application tasks, mapping of tasks to resources, and their management. To demonstrate suitability of the GridSim toolkit, we have simulated a Nimrod-G like Grid resource broker and evaluated the performance of deadline and budget constrained cost- and time-minimization scheduling algorithms. The interest in coupling geographically distributed (computational) resources is also growing for solving large-scale problems, leading to what is popularly called the Grid [10] and peer-to-peer (P2P) computing [10] networks.

The Grid consists of four key layers of components: fabric, core middleware, user-level middleware, and applications [9]. The Grid fabric includes computers (low-end and high-end computers including clusters), networks, scientific instruments, and their resource management systems. The core Grid middleware provides services that are essential for securely accessing remote resources uniformly and transparently. The services they provide include security and access management, remote job, such as resource brokers, application development and adaptive runtime environment. Apart from the centralized approach, two other approaches that are used in distributed resource management are: hierarchical and decentralized scheduling or a combination of them [2], farming applications on geographically distributed resources. It supports deadline and budget-based scheduling driven by market-based economic models.

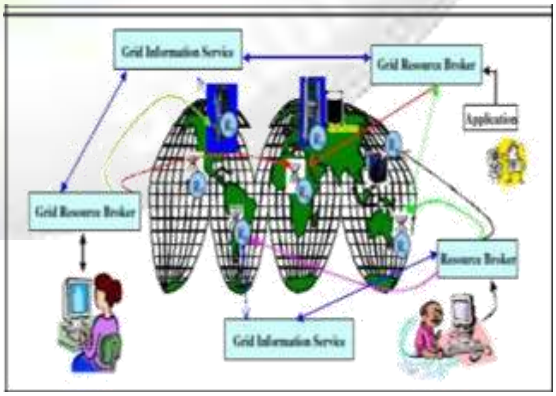


Fig. 1.1: A Generic View of the World-Wide Grid Computing Environment

To meet users 'quality of service requirements, our broker dynamically leases Grid resources and services at

runtime depending on their capability, cost, and availability. The ability to experiment with a large number of Grid scenarios was limited by the number of resources that were available in the WWG (World-Wide Grid) testbed[10]. It was impossible to create a repeatable and controlled environment for experimentation and evaluation of scheduling strategies. This is because of resources in the Grid span across multiple administrative domains, each with their own policies, users, and priorities.

To overcome these limitations, a Java-based Grid simulation toolkit called GridSim was developed. The Grid computing researchers and educators also recognized the importance and the need for such a toolkit for modeling and simulation environments [10]. The GridSim toolkit supports modeling and simulation of a wide range of heterogeneous resources, such as single or multiprocessors, shared and distributed memory machines such as PCs, workstations, SMPs, and clusters with different capabilities and configurations. It can be used for modeling and simulation of application scheduling on various classes of parallel and distributed computing systems such as clusters [10], Grids [9], and P2P networks [10]. The resources in clusters are located in a single administrative domain and managed by a single entity, whereas in Grid and P2P systems, resources are geographically distributed across multiple administrative domains with their own management policies and goals. The GridSim toolkit provides facilities for the modeling and simulation of resources and network connectivity with different capabilities, configurations, and domains. It supports primitives for application composition, information services for resource discovery, and interfaces for assigning application tasks to resources and managing their execution. These features can be used to simulate resource brokers or Grid schedulers for evaluating performance of scheduling algorithms or heuristics. GridSim toolkit is used to create a resource broker that simulates Nimrod-G for design and evaluation of deadline and budget constrained scheduling algorithms with cost and time optimizations.

The main resources to which Grid Computing gives access are

- Computing/processing power
- Data storage/networked file systems
- Communications and bandwidth
- Application software

A. Grid Computing Model

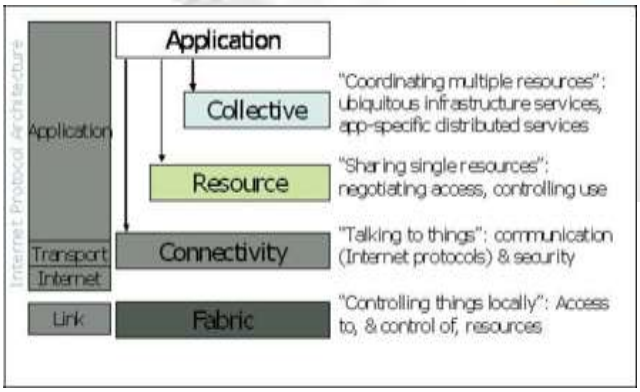


Fig. 1.2: Grid Computing Architecture Model

B. Grid Protocols

The protocols associated with each layer in the grid architecture are as follows

Security: Grid Security Infrastructure, Resource Management: Gird Resource Allocation Management Protocol, Data Transfer: GRIS File Transfer Protocol, Information Services: Grid Information Services.

II. SCHEDULING IN GRID COMPUTING

Workflow scheduling is one of the key issues in the workflow management. A scheduling is a process that maps and manages the execution of inter-dependent tasks on the distributed resources. It allocates suitable resources to workflow tasks so that the execution can be completed to satisfy objective functions imposed by users. Proper scheduling can have significant impact on the performance of the system. In general, the problem of mapping tasks on distributed services belongs to a class of problems known as NP-hard problems.

A. Workflow Management in Grid

Workflow is concerned with the automation of procedures, whereby files and other data are passed between participants according to a defined set of rules to achieve an overall goal. A workflow management system defines, manages and executes workflows on computing resources.

There are three major components in a workflow enactment engine: the workflow scheduling, data movement and fault management. Workflow scheduling discovers resources and allocates tasks on suitable resources to meet users' requirements, while data movement manages data transfer between selected resources and fault management provides mechanisms for failure handling during execution. Workflow scheduling is one of the key issues in the workflow management [10]. A scheduling is a process that maps and manages the execution of inter-dependent tasks on the distributed resources. It allocates suitable resources to workflow tasks so that the execution can be completed to satisfy objective functions imposed by users. Proper scheduling can have significant impact on the performance of the system. In general, the problem of mapping tasks on distributed services belongs to a class of problems known as NP-hard problems. For such problems, no known algorithms are able to generate the optimal solution within polynomial time. Even though the workflow scheduling problem can be solved by using exhaustive search the complexity of the methods for solving it is very high. In Grid environments, scheduling decisions must be made in the shortest time possible, because there are many users competing for resources, and time slots desired by one user could be taken by another user at any moment.

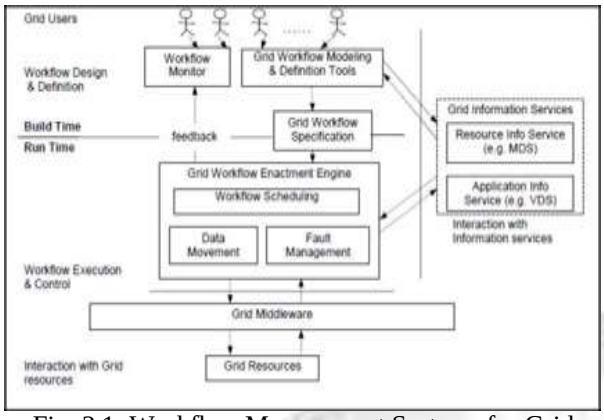


Fig. 2.1: Workflow Management Systems for Grid Computing

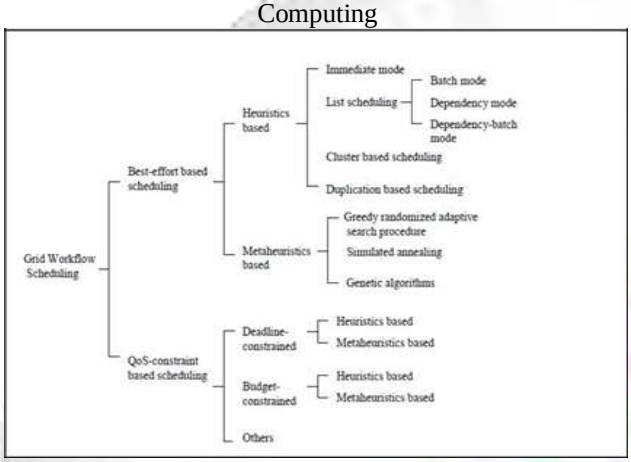


Fig. 2.2: Taxonomy of Grid Workflow Scheduling Algorithms

B. Best effort Based Workflow Scheduling

In general, best-effort based scheduling algorithms are derived from either heuristics based or metaheuristics based approach. The heuristic based approach is to develop a scheduling algorithm which fit only a particular type of problem, while the metaheuristic based approach is to develop an algorithm based on a metaheuristic method which provides a general solution method for developing a specific heuristic to fit a particular kind of problem.

C. Heurostics

In general, there are four classes of scheduling heuristics for workflow applications, namely individual task scheduling, list scheduling, and cluster and duplication based scheduling.

D. Individual Task Scheduling

The individual task scheduling is the simplest scheduling method for scheduling workflow applications and it makes schedule decision based only on one individual task.

E. List Scheduling

A list scheduling heuristic prioritizes workflow tasks and schedules the tasks based on their priorities. There are two major phases in a list scheduling heuristic, the task prioritizing phase and the resource selection phase. The task prioritizing phase sets the priority of each task with a rank value and generates a scheduling list by sorting the tasks according to their rank values. The resource selection phases select tasks in the order of their priorities and map each selected task on its optimal resource.

F. Qos Constrained based Workflow Scheduling

Many workflow applications require some assurances of quality of services (QoS). For example, a workflow application for maxillo-facial surgery planning [10] needs results to be delivered before a certain time. For these applications, workflow scheduling is required to be able to analyze users' QoS requirements and map workflows on suitable resources such that the workflow execution can be completed to satisfy users' QoS constraints. However, completing the execution within a required QoS not only depends on the global scheduling decision of the workflow scheduler, but also depends on the local resource allocation model of each execution site. If the execution of every single task in the workflow cannot be completed as expected by the scheduler, it is impossible to guarantee the entire workflow execution. Instead of scheduling tasks on community Grids, QoS-constraint based schedulers should be able to interact with service-oriented Grid services to ensure resource availability and QoS levels. It is required that the scheduler can negotiate with service providers to establish a service level agreement (SLA) which is a contract specifying the minimum expectations and obligations between service providers and consumers. Users normally would like to specify a QoS constraint for entire workflow. The scheduler needs to determine a QoS constraint for each task in the workflow, such that the QoS of entire workflow is satisfied. To date, supporting QoS in scheduling of workflow applications is at a very preliminary stage. Most QoS constraint based workflow scheduling heuristics are based on either time or cost constraints. Time is the total execution time of the workflow (known as deadline).

III. DEADLINE CONSTRAINED SCHEDULING

Three major categories of workflow scheduling architecture are centralized, distributed and hierarchical scheduling schemes.

A. Centralized Scheduling

In a centralized scheduling environment, a central machine (node) acts as a resource manager to schedule jobs to all the surrounding nodes that are part of the environment. This scheduling paradigm is often used in situations like a computing centre where resources have similar characteristics and usage policies. Figure 3.1 shows the architecture of centralized scheduling. In this scenario; jobs are first submitted to the central scheduler, which then dispatches the jobs to the appropriate nodes. Those jobs that cannot be started on a node are normally stored in a central job queue for a later start. One advantage of a centralized scheduling system is that the scheduler may produce better scheduling decisions because it has all necessary, and up-to-date, information about the available resources. However, centralized scheduling obviously does not scale well with the increasing size of the environment that it manages. The scheduler itself may well become a bottleneck, and if there is a problem with the hardware or software of the scheduler's server, i.e. a failure, it presents a single point of failure in the environment.

B. Distributed Scheduling

In this, there is no central scheduler responsible for managing all the jobs. Instead, distributed scheduling involves multiple localized schedulers, which interact with each other in order to dispatch jobs to the participating nodes. There are two mechanisms for a scheduler to communicate with other schedulers – direct or indirect communication. Distributed scheduling overcomes scalability problems, which are incurred in the centralized paradigm; in addition it can offer better fault tolerance and reliability. However, the lack of a global scheduler, which has all the necessary information on available resource, usually leads to sub-optimal scheduling decisions.

C. Heirachial Scheduling

In hierarchical scheduling, a centralized scheduler interacts with local schedulers for job submission. The centralized scheduler is a kind of a meta-scheduler that dispatches submitted jobs to local schedulers. Below figure shows the architecture of this paradigm. Similar to the centralized scheduling paradigm, hierarchical scheduling can have scalability and communication bottlenecks. However, compared with centralized scheduling, one advantage of hierarchical scheduling is that the global scheduler and local scheduler can have different policies in scheduling jobs.

D. Deadline Constrained Scheduling

Users normally would like to specify a QoS constraint for entire workflow. The scheduler needs to determine a QoS constraint for each task in the workflow, such that the QoS of entire workflow is satisfied. Most QoS constraint based workflow scheduling heuristics are based on either time or cost constraints. Time is the total execution time of the workflow (known as deadline). Cost is the total expense for executing workflow execution including the usage charges by accessing remote resources and data transfer cost (known as budget). In this section, Deadline constrained scheduling is proposed. Some workflow applications are time critical and require the execution can be completed within a certain timeframe. Deadline constrained scheduling is designed for these applications to deliver results before the deadline.

1) Back-Tracking:

The heuristic developed by Menasce and Casalicchio assigns available tasks to least expensive computing resources. An available task is an unmapped task whose parent tasks have been scheduled. If there is more than one available task, the algorithm assigns the task with the largest computational demand to the fastest resources in its available resource list. The heuristic repeats the procedure until all tasks have been mapped. After each iterative step, the execution time of current assignment is computed. If the execution time exceeds the time constraint, the heuristic back-tracks the previous step and remove the least expensive resource from its resource list and reassigns tasks with the reduced resource set. If the resource list is empty the heuristic keep back-tracking to the previous step, reduces corresponding resource list and reassign the tasks.

2) Deadline Distribution:

Instead of back-tracking and repairing the initial schedule, the TD heuristic partitions a workflow and distributes overall deadline into each task based on their workload and dependencies. After deadline distribution, the entire workflow scheduling problem has been divided into several sub-task scheduling problems.

3) Example:

A workflow application is modeled as a Directed Acyclic Graph (DAG). Let Γ be the finite set of tasks T_i ($1 <= i <= n$). Let Λ be the set of directed arcs of the form (T_i ,T_j) where T_i is called a parent task of T_j, and T_j the child task of T_i. We assume that a child task cannot be executed until all of its parent tasks are completed. Let D be the time constraint (deadline) specified by the users for workflow execution. Then, the workflow application can be described as a tuple Ω (Γ, Λ, D).

Let m be the total number of services available. There is a set of services $S_i^{\ j}$: cond ($1 <= i <= n$, $1 <= j <= m_i$, $m_i <= m$) capable of executing the task T_i, but only one service can be assigned for the execution of a task. Services have varied processing capability delivered at different prices. $t_i^{\ j}$ is denoted as the sum of the processing time and data transmission time, and $c_i^{\ j}$ as the sum of the service price and data transmission cost for processing Ti on service $S_i^{\ j}$. The scheduling problem is to map every T_i onto a suitable $S_i^{\ j}$ to minimize the execution time of the workflow and complete it within the deadline D. The scheduling problem can be solved by following the divide-and-conquer technique and the methodology is listed below:

- Step. 1. Discover available services and predict execution time for every task.
- Step. 2. Group workflow tasks into task partitions.
- Step. 3. Distribute users' overall deadline into every task partition.
- Step. 4. Query available time slots, generate optimized schedule plan and make advance reservations based on the local optimal solution of every task partition.
- Step. 5. Start workflow execution and reschedule when the initial schedule is violated at run-time.

E. Workflow Task Partioninig

In workflow task partitioning, workflow tasks are categorized as either synchronization tasks or simple tasks. A synchronization task is defined as a task which has more than one parent or child task. For example, T_1, T_{10} and T_{14} are synchronization tasks. Other tasks which have only one parent task and child task are simple tasks. For example, T_2 - T_9 and T_{11} - T_{13} are simple tasks. Simple tasks are then clustered into a branch. A branch is a set of interdependent simple tasks that are executed sequentially between two synchronization tasks. For example, the branches in the example are {T2, T3, T4} and {T_5, T6}, {T_7}, {T_8, T9}, {T_{11}} and {T_{12}, T13}.

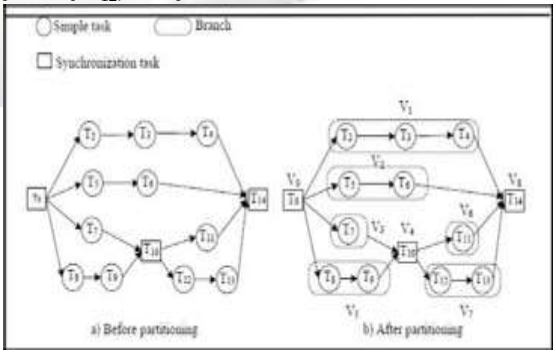


Fig. 3.1: Workflow Task Partition

IV. DEADLINE CONSTRAINED SCHEDULING

Simulation of deadline constrained scheduling uses Windows XP OperatingSystem, Jcreator editor, Gridsim Toolkit, jdk1.6. Here Jcreator is installed and include JDK files and simJava and distributedSimJava packages to run the simulated code.

A. Simulation of Deadline Constrained Scheduling

Simulation of Deadline constrained scheduling has the following steps.

- Creating grid resources (one or more): A Grid resource contains one or more Machines. Similarly, a Machine contains one or more PEs (Processing Elements or CPUs).
- Creating number of users: Create a Grid user containing one or more Gridlets.
- Constructing a Hierarchical model: Construct a hierarchical model for maintaining the network with the help of routers and links between users and resources.
- Execution of model: Finally jobs allocated to grid resource are executed in space-shared environment.

The system model is shown in Figure 4.1. It contains many user(s) that are submitting Gridlets (jobs) that are executed on different Gridresources. The users and gridresources are connected by routers. The Gridlets are executed on different Gridresources based on the priorities. The gridlets that are having priorities (<=4) are executed on Gridresource0 and the others are executed on Gridresource1. The gridlets at Gridresources are executed locally using Space shared algorithm.

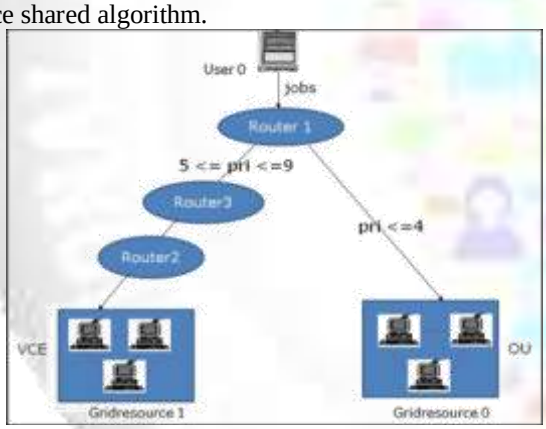
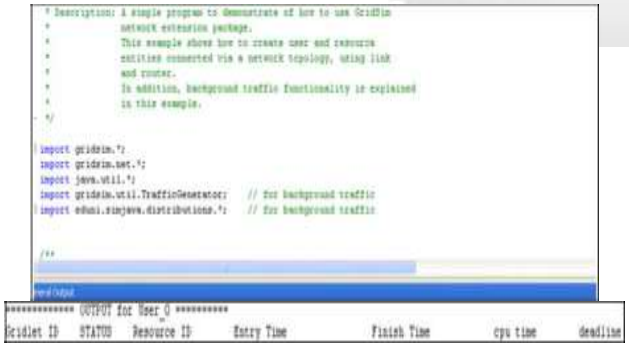


Fig. 4.1: System Model

To use the GridSim for the implementation and simulation of the Deadline Constrained Scheduling follow the steps mentioned in Appendix, which when together coded in JAVA programming language to simulate a heterogeneous Grid Computing Environment.

B. Experimental Results



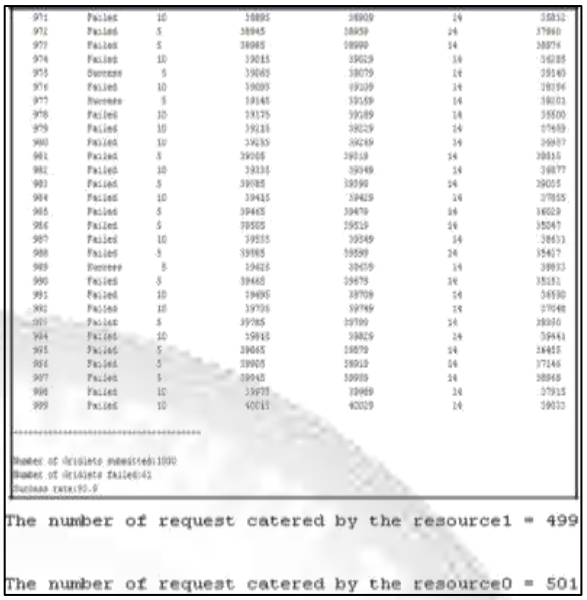


Fig. 4.1: Output for 1000 Gridlets

1) Graphs:

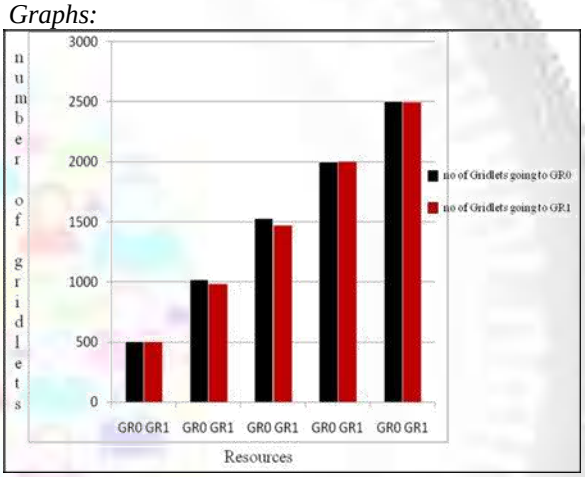


Fig. 4.2: No of Gridlets Vs Gridlets Received by Resources

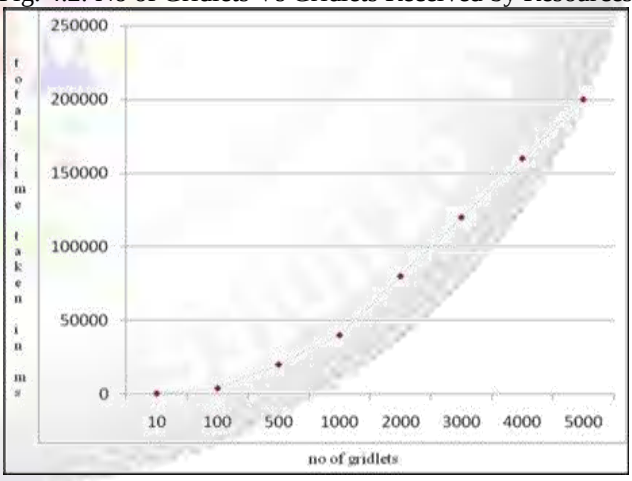


Fig. 4.3: Total Time Taken (ms) vs No of Gridlets

Figure 4.2 shows the graph between no of gridlets and those that are received by resources (GR0, GR1). Resources are taken on x-axis and no of gridlets are taken on y-axis.The gridlets that are executed on GR0, GR1 are based on the priorities of gridlets.

Figure 4.3 shows the graph between no of gridlets and total time taken to execute those grilets. X-axis represents no of gridlets between 10-5000 and Y-axis

represents totaltime in ms. If the no. of gridlets are increased total time taken also increases.

V. CONCLUSION AND FUTURE SCOPE

This work primarily focuses on scheduling the jobs in an optimize manner using deadlines and priorities to make it hierarchical. The user submits the numbers of jobs, which in this thesis, are created in a simulated environment using Gridsim.

This work discuss about an object-oriented toolkit, called GridSim, for resource modeling and scheduling simulation. GridSim simulates time- and space-shared resources with different capabilities, time zones, and configurations. It supports different application models that can be mapped to resources for execution by developing simulated application schedulers. The implementation of the GridSim toolkit in Java is an important contribution since Java provides a rich set of tools that enhance programming productivity, application portability, and a scalable runtime environment.

In future, the improvements can be done by also taking into account the dynamic behavior of the grid resources. This thesis is carried out in a simulated environment. In future the work done here can be used in an actual Grid environment. A further extension to this work would be use to enhance the different scheduling algorithm to improve the performance, quality of service etc.

ACKNOWLEDGMENT

Our work is supported by the H.O.D of Computer Science and Engineering and Vice Principal of University College of Engineering, Osmania University, Hyderabad. So we wish to thank and express deep sense of gratitude to our guide Prof. S. Ramachandram and other faculty members for their consistent guidance, inspiration and sympathetic attitude throughout this research.

REFERENCES

[1] D. Abramson, R. Buyya, J. Giddy, Nimrod/G: An architecture for aresource management and scheduling system in a global computational Grid, Proceedings 4th International Conference and Exhibition on High Performance Computing in Asia-Pacific Region (HPC ASIA 2000), Beijing, China, 14–17 May 2000. IEEE Computer Society Press, 2000.

[2] D. Abramson, R. Buyya, J. Giddy, An economy driven resource management architecture for global computational power Grids, Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), Las Vegas, NV, 26–29 June 2000,Las Vegas, USA, CSREA Press, 2000.

[3] D. Abramson, R. Buyya, J. Giddy, An evaluation of economy-based resource trading and scheduling on computational power Grids for parameter sweep applications, Proceedings of the 2nd International Workshop on Active Middleware Services(AMS 2000), Pittsburgh, PA, 1 August 2000. Kluwer Academic Press, USA, 2000.

[4] D. Abramson, J. Giddy, and L. Kotler, High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?, Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2000), May 1-5, 2000, Cancun, Mexico, IEEE Computer Society (CS) Press, USA, 2000.

[5] D. Abramson, R. Buyya, H. Stockinger, and J. Giddy Economic models for management of resources in peer-to-peer and Grid computing, SPIE International Conference on Commercial Applications for High-Performance Computing, 20–24 August 2001, Denver, USA.

[6] R. Sakellariou and H. Zhao, A Low-Cost Rescheduling Policy for Efficient Mapping of Workflows on Grid Systems, Scientific Programming, 12(4):253-262, Dec.2004.

[7] E. Casalicchio and D. A. Menasce, A Framework for Resource Allocation in Grid Computing, The 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04), Volendam, The Netherlands, Oct. 5-7 2004.

[8] E. Deelman et al., Pegasus: Mapping scientific workflows onto the grid, European Across Grids Conference, pp. 11-20, 2004.

[9] F. Howell and R. McNab, SimJava: A Discrete Event Simulation Package For Java With Applications In Computer Systems Modelling, First International Conference on Web-based Modelling and Simulation, San Diego, CA, Society for Computer Simulation, January 1998.

[10] Oram (editor), Peer-to-Peer: Harnessing the Power of Disruptive Technologies, O'Reilly Press, USA, 2001.