

SEGMENTED AVERAGE-SUFFERAGE HEURISTIC FOR INDEPENDENT TASK SCHEDULING IN GRID

A DISSERTATION

*Submitted in partial fulfillment of the
requirements for the award of the degree*

of

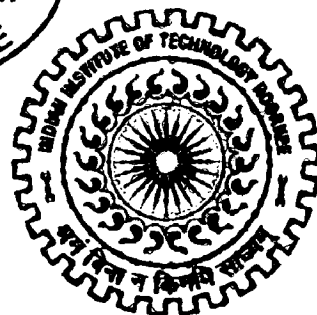
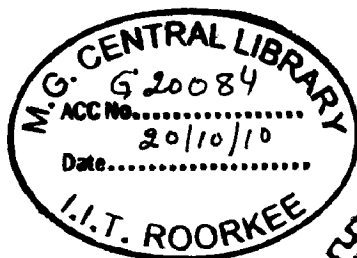
MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

ASHISH KUMAR



DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

ROORKEE -247 667 (INDIA)

JUNE, 2010

CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in the dissertation entitled "SEGMENTED AVERAGE-SUFFERAGE HEURISTIC FOR INDEPENDENT TASK SCHEDULING IN GRID" towards the partial fulfillment of the requirement for the award of the degree of **Master of Technology in Computer Science and Engineering** submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee (India) is an authentic record of my own work carried out during the period from July 2009 to June 2010, under the guidance of **Dr. A. K. Sarje**, Professor, Department of Electronics and Computer Engineering, IIT Roorkee.

The matter presented in this dissertation has not been submitted by me for the award of any other degree of this or any other Institute.

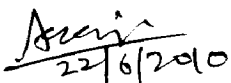
Date: 22/6/2010

Place: Roorkee


(ASHISH KUMAR)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.


22/6/2010
(Dr. A. K. Sarje)

Professor

Date

Department of Electronics and Computer Engineering

Place :- Roorkee

IIT Roorkee

ACKNOWLEDGEMENTS

First and foremost, I would like to extend my heartfelt gratitude to my guide and mentor **Dr. A. K. Sarje**, Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, for his invaluable advices, guidance, encouragement, and for sharing his broad knowledge. His wisdom, knowledge and commitment to the highest standards inspired and motivated me. He has been very generous in providing the necessary resources to carry out my research. He is an inspiring teacher, a great advisor, and most importantly a nice person.

I also wish to thank my friends, Sameer Singh Chauhan and Mukesh Sharma for their valuable suggestions and timely help regarding the domain knowledge. I am greatly indebted to all, who have graciously applied themselves to the task of helping me with ample moral supports and valuable suggestions.

I would also like to thank all faculty members of Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee for their kind help and support.

ASHISH KUMAR

Abstract

Grid Computing enables the secured, controlled and flexible sharing of resources among various dynamically created virtual organizations. These virtual organizations are setup for collaborative problem solving that requires a great number of processing cycles. In high throughput computing, the grid is used to schedule large number of task, with the aim of putting unused processor cycles to work. Grid computing provides highly scalable, highly secure and utmost high performance mechanisms for discovering and negotiating access to the computing resources among an infinite number of geographically distributed groups to solve complex scientific or technical problems.

Scheduling is a fundamental issue in achieving high performance on computational grids. An efficient grid scheduling system is an essential part of the grid. Even though middleware support for grid computing has been the subject of extensive research, scheduling policies for the grid context have not been much studied. In addition to processor utilization, it is important to consider the waiting time, throughput, and response times of tasks in evaluating the performance of grid scheduling strategies. The task scheduling problem for grid computing has been studied as a combinatorial optimization problem, which can be solved only using heuristic algorithms.

In this thesis, we consider the problem of allocating independent, heterogeneous tasks on grid environment. A heuristic namely, Segmented-Average Sufferage for batch mode independent task scheduling is proposed in this dissertation. The segmentation is done to give better makespan and load balancing. The heuristic is tested in GridSim simulator. The experiment results show that the Segmented Average-Sufferage heuristic gives significantly improvements in makespan, resource utilization and load balancing than existing Sufferage, Min-Min and Max-Min heuristics.

Table of Contents

Candidate's Declaration and Certificate.....	i
Acknowledgement.....	ii
Abstract.....	iii
Table of Contents.....	iv
List of Figures.....	vi
List of Tables.....	vii
Chapter 1. Introduction	1
1.1 Introduction.....	1
1.2 Motivation.....	2
1.3 Statement of the Problem.....	3
1.4 Organization of Report.....	3
Chapter 2. Grid Computing	4
2.1 Introduction.....	4
2.2 Types of Grid	4
2.2.1 Cluster Grids.....	5
2.2.2 Enterprise Grids.....	5
2.2.3 Global Grids.....	5
2.2.4 Compute Grids.....	6
2.2.5 Data Grids.....	6
2.3 Grid Application Areas.....	6
2.4 Grid Architecture	7
Chapter 3. Task Scheduling in Grid Computing	9
3.1 Grid Scheduling Process	12
3.1.1Phase 1: Resource Discovery.....	12
3.1.2 Phase 2: System Selection.....	13
3.1.3 Phase 3: Run the Job.....	14

3.2 Types of Grid Scheduling	15
3.3 Challenges in Grid Scheduling.....	18
3.3.1 Resource Heterogeneity.....	19
3.3.2 Site Autonomy.....	19
3.3.3 Local Priority.....	19
3.3.4 Resource Non-Dedication.....	20
3.3.5 Application Diversity.....	20
3.3.6 Dynamic Behavior.....	20
3.4 Batch Mode Heuristics for Independent Task.....	21
3.4.1 Min-Min.....	21
3.4.2 Max-Min.....	21
3.4.3 Sufferage.....	21
Chapter 4. Proposed Segmented Average-Sufferage Heuristic	23
4.1 Segmented Average-Sufferage Heuristic	23
4.2 Mapping Comparison of Sufferage and Segmented Average-Sufferage Heuristics.....	25
Chapter 5. Results and Discussions	33
5.1 Performance Metrics	33
5.2 Simulation Environment	34
5.3 Results	35
Chapter 6. Conclusions and Scope for Future Work	41
6.1 Conclusions.....	41
6.2 Scope for Future Work.....	41
REFERENCES	42
LIST OF PUBLICATIONS	45
Appendix A Introduction to GridSim.....	46

LIST OF FIGURES

Figure 2.1	Cluster Grids.....	5
Figure 2.2	Enterprise Grids.....	5
Figure 2.3	Global Grids.....	6
Figure 2.4	Grid Components.....	8
Figure 3.1	Grid Scheduling Process.....	13
Figure 3.2	Centralized Scheduler.....	17
Figure 3.3	Hierarchical Scheduler.....	17
Figure 3.4	Decentralized Scheduler.....	17
Figure 4.1	Segmented Average-Sufferage heuristic.....	24
Figure 4.2	Result of Sufferage heuristic.....	27
Figure 4.3	Result of Segmented Average-Sufferage heuristic.....	32
Figure 5.1	Makespan in Case I.....	36
Figure 5.2	Makespan in Case II.....	37
Figure 5.3	Makespan in Case III	37
Figure 5.4	Average resource utilization rate in Case I.....	38
Figure 5.5	Average resource utilization rate in Case II	38
Figure 5.6	Average resource utilization rate in Case III.	39
Figure 5.7	Load balancing level in Case I.....	39
Figure 5.8	Load balancing level in Case II	40
Figure 5.9	Load balancing level in Case III	40
Figure A.1	A modular architecture for Gridsim platform and components.....	47

LIST OF TABLES

Table 4.1	ETC Matrix of Ten Tasks on Four Resources	25
Table 5.1	Makespan Comparison of Segmented Average-Sufferage with Sufferage	36
Table 5.2	Makespan Comparison of Segmented Average-Sufferage with Sufferage	36
Table 5.3	Makespan Comparison of Segmented Average-Sufferage with Sufferage.....	36

Chapter 1

Introduction

1.1 Introduction

The rapid development in computing resources has enhanced the performance of computers and reduced their costs. This availability of low cost powerful computers coupled with the popularity of the Internet and high-speed networks has led the computing environment to be mapped from distributed to Grid environments [1]. In fact, recent researches on computing architectures are allowed the emergence of a new computing paradigm known as Grid computing. Grid is a type of distributed system which supports the sharing and coordinated use of geographically distributed and multi-owner resources, independently from their physical type and location, in dynamic virtual organizations that share the same goal of solving large-scale applications.

In order to fulfill the user expectations in terms of performance and efficiency, the Grid system needs efficient scheduling algorithms for the distribution of tasks. A scheduling algorithm attempts to improve the response time of user's submitted applications by ensuring maximal utilization of available resources. The main goal is to prevent, if possible, the condition where some processors are overloaded with a set of tasks while others are lightly loaded or even idle [2].

Although scheduling problem in conventional distributed systems has been intensively studied, new challenges in Grid computing still make it an interesting topic and many research projects are under way. This is due to the characteristics of Grid computing and the complex nature of the problem itself. Scheduling algorithms in classical distributed systems, which usually run on homogeneous and dedicated resources, cannot work well in the Grid architectures.

1.2 Motivation

A typical distributed system will have a number of interconnected resources which can work independently or in cooperation with each other [3]. Each resource has owner workload, which represents an amount of work to be performed and every one may have a different processing capability. To minimize the time needed to perform all tasks, the workload has to be evenly distributed over all resources based on their processing speed. The essential objective of a scheduling consists primarily in optimizing the average response time of applications, which often means maintaining the workload proportionally equivalent on the whole resources of a system. Job scheduling is a fundamental issue in achieving high performance in Grid computing systems. However, it is a big challenge for efficient scheduling algorithm design and implementation. Unlike scheduling problems in conventional distributed systems, this problem is much more complex as new features of Grid systems such as its dynamic nature. And the high degree of heterogeneity of jobs and resources must be tackled. The problem is multi-objective in its general formulation, the two most important objectives being the minimization of makespan and flow time of the system. Job scheduling is known to be NP-complete [4], therefore the use of non-heuristics is the de facto approach in order to cope in practice with its difficulty.

- **Heterogeneity:** Heterogeneity exists in both of computational and networks resources.
- **Autonomy:** Because the multiple administrative domains that share Grid resources, a site are viewed as an autonomous computational entity.
- **Scalability:** A Grid might grow from few resources to millions. This raises the problem of potential performance degradation as the size of a Grid increases.
- **Resource selection:** In traditional systems, executable codes of applications and input/output data are usually in the same site, or the input sources and output destinations are determined before the submission of an application [2, 3]. Thus the cost for data staging can be neglected or the cost is a constant determined before execution and load balancing algorithms need not consider it. But in a Grid the computation sites of an application are usually selected by the Grid scheduler according to resource status and some performance criterion.

1.3 Statement of the Problem

The main objective of this research work is to develop Segmented Average-Sufferage heuristic for independent task scheduling in Grid.

This main objective can be further divided into following sub problem.

- i) To design and propose algorithm for independent task scheduling in Grid.
- ii) To validate the proposed algorithm.

1.4 Organization of the Report

The remainder of the report is organized as follows:

The **second chapter** briefly introduces Grid computing concepts, types of Grid, Grid computing application areas and Grid architecture.

The **third chapter** explains Grid scheduler, Grid scheduling process, types of Grid scheduling, and various challenges in Grid scheduling. This section also gives a brief literature review of batch mode centralized heuristics for independent tasks.

The **fourth chapter** describes the proposed scheduling algorithm “Segmented Average-Sufferage Heuristic” and gives an example to briefly explain the proposed algorithm. Also, this section compares the proposed algorithm with sufferage heuristic.

The **fifth chapter** describes implementation details and provides the experimental results of the Segmented Average-Sufferage with Max-Min, Min-Min and Sufferage heuristics.

The **sixth chapter** concludes the report work, gives the contribution of report and what future work can done on it.

Chapter 2

Grid Computing

2.1 Introduction

In today's complex world of high speed computing, computers have become extremely powerful, even home-based desktops are powerful enough to run complex applications. But still we have numerous complex scientific experiments, advanced modeling scenarios, genome matching, astronomical research, a wide variety of simulations, complex scientific & business modeling scenarios and real-time personal portfolio management, which require huge amount of computational resources. To satisfy some of these aforementioned requirements, Grid computing is born.

The Grid is a wide-scale, distributed computing infrastructure that promises to support resource sharing and coordinated problem solving in dynamic, multi-institutional Virtual Organization [5]. Grid computing is applying the resources of many computers in a network for a single problem at the same time-usually to a scientific or technical problem that requires a great number of computer processing cycles or access to large amounts of data. Grid computing can be thought of as distributed and large-scale cluster computing and as a form of network-distributed parallel processing. Grid resources [6] fall into the categories of computation (i.e. a machine sharing its CPU), storage (i.e. a machine sharing its RAM or disk space), communication (i.e. sharing of bandwidth or a communication path), software and licenses and special equipment (i.e. sharing of devices).

2.2 Types of Grid

Grid computing can be used in a variety of ways to address various kinds of application requirements. Often, Grids may be a combination of two or more of these [7, 8]. Grids can be classified on the basis of two factors, scale and functionality. On basis of scale they can be further classified as.

- Global Grids
- Enterprise Grids
- Cluster Grids

2.2.1 Cluster Grids

Cluster Grids consist of one or more systems working together to provide a single point of access to users. Typically owned and used by a small number of users, such as a project or department, Cluster Grids support both high-throughput and high-performance jobs. Resources in the Grid can be focused on a narrow set of repetitive tasks, or made to work in true parallel fashion to execute a complex job.

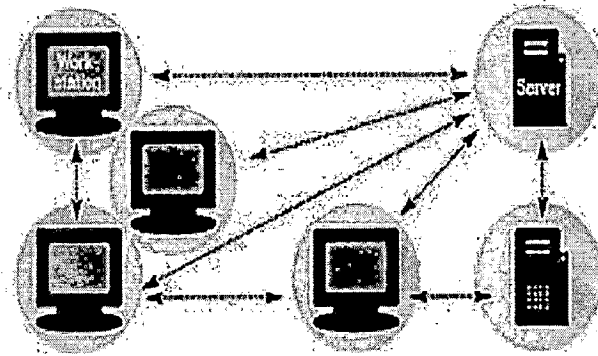


Figure 2.1: Cluster Grids [9]

2.2.2 Enterprise Grids

Enterprise Grids enable multiple project or department to share resources within an enterprise or campus and do not necessarily have to address security and other global policy management issues associated with global Grid.

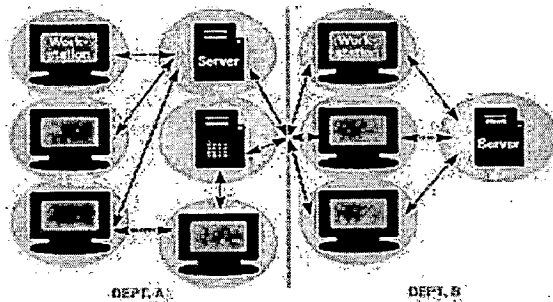


Figure 2.2: Enterprise Grids [9]

2.2.3 Global Grids

Global Grids are a collection of enterprise and cluster Grids as well as other geographically distributed resources, all of which are agreed upon global usage policies and protocols to enable resource sharing [10].

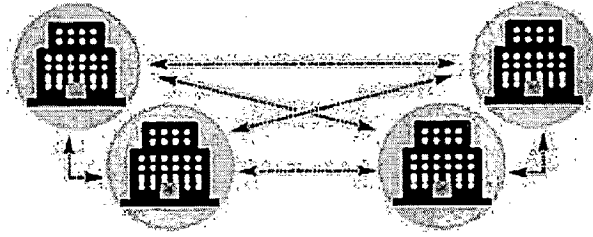


Figure 2.3: Global Grids [9]

On basis of Functionality they can be further classified as:

- Compute Grids
- Data Grids

2.2.4 Compute Grids

A compute Grid is essentially a collection of distributed computing resources, within or across locations that are aggregated to act as a unified processing resource or virtual supercomputer.

2.2.5 Data Grids

A data Grid provides wide area, secure access to current data. Data Grids enable users and applications to manage and efficiently use database information from distributed locations.

2.3 Grid Application Areas

Applications with heavy use of computing resources (e.g., simulations, number crunching, the so-called grand challenge applications), applications using large information resources (e.g., multimedia databases), applications using special sub applications (e.g., visualization), and applications using special devices (e.g., expensive scanners, laboratory equipment) are candidates for Grids. Especially we mention the following important application areas [9]:

- **Medical Applications:** In diagnostics huge amounts of data are generated at one place by specialized devices. These data have to be transported to the specialists, possibly located at several locations, while the patient might be at a third location. The task of a Grid in this scenario is to prepare and transport the medical data, so that they are available at the right location at the right time [11].

- **Support for multinational enterprises:** Multinational enterprises work at several locations in several time zones. Data, e.g., multimedia data from inspections, must be pre-processed and forwarded to specialists who can take decisions.
- **Multimedia Applications:** Several Multimedia Applications make use of a Grid for processing media streams within multimedia QoS control is very important. Applications often include the handling of Digital Rights Management, e.g., multimedia data can be watermarked scrambled etc.
- **Applications from bio-informatics, seismology, meteorology, etc.** are data – and computing-intensive, and need often other information resources [11].

2.4 Grid Architecture

Architecture identifies the fundamental system components, specifies purpose and function of these components, and indicates how these components interact with each other. Grid architecture is protocol architecture, with protocols defining the basic mechanisms by which Virtual Origination [12, 13] users and resources negotiate, establish, manage and exploit sharing relationships. Grid architecture is also a services standards based open architecture that facilitates extensibility, interoperability, portability and code sharing. The components that are necessary to form a Grid are shown in Figure 2.4 and they are briefly discussed below:

- **Grid Fabric:** It comprises all the resources geographically distributed (across the globe) and accessible from anywhere on the Internet. They could be computers (such as PCs or Workstations running operating systems such as UNIX or NT), clusters (running cluster operating systems or resource management systems such as LSF, Condor or PBS), storage devices, databases, and special scientific instruments such as a radio telescope.
- **Grid Middleware:** It offers core services such as remote process management, co allocation of resources, storage access, information (registry), security, authentication, and Quality of Service (QoS) such as resource reservation and trading.

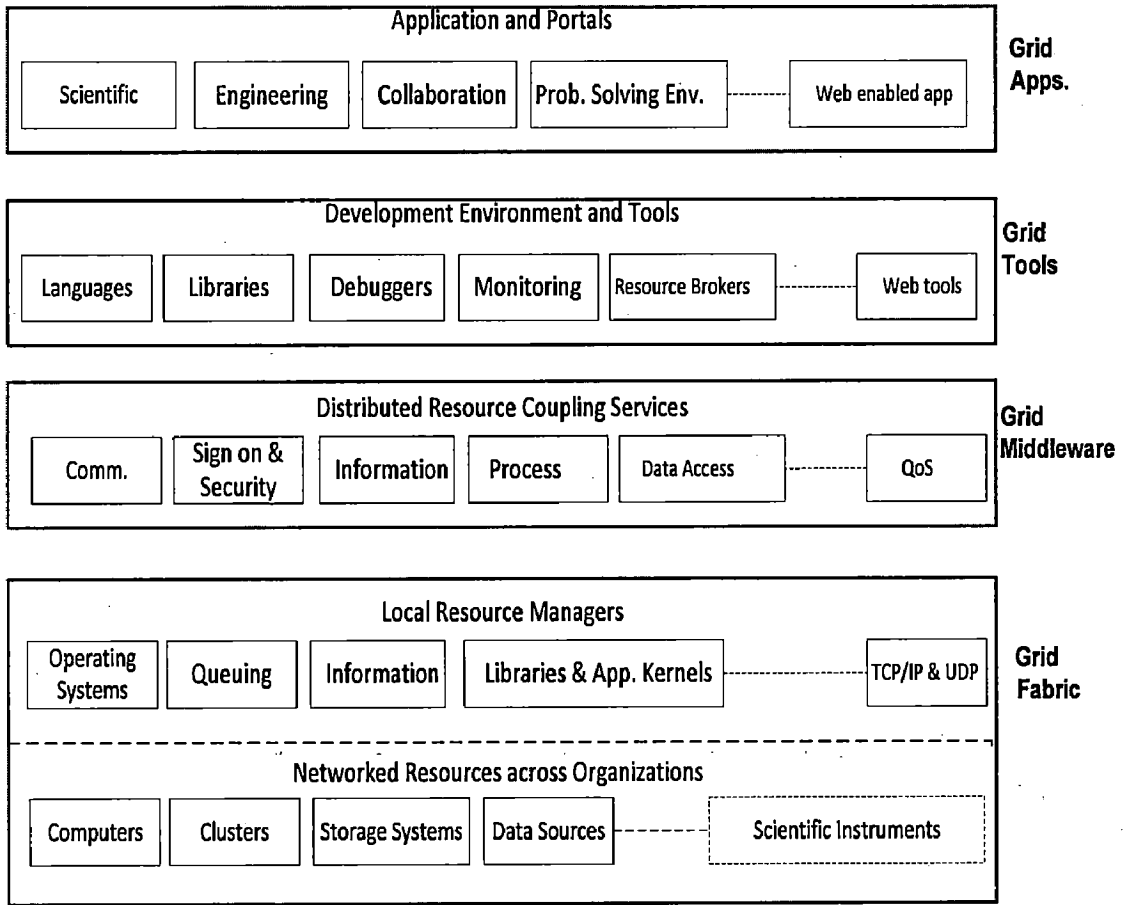


Figure 2.4: Grid Components [6]

- **Grid Development Environments and Tools:** These offer high-level services that allow programmers to develop applications and brokers that act as user agents that can manage or schedule computations across global resources.
- **Grid Applications and Portals:** They are developed using Grid-enabled languages such as HPC++, and message-passing systems such as MPI. Applications, such as parameter simulations and grand-challenge problems often require considerable computational power, require access to remote data sets, and may need to interact with scientific instruments. Grid portals offer web-enabled application services — i.e., users can submit and collect results for their jobs on remote resources through a web interface.

Chapter 3

Task Scheduling in Grid Computing

A precise definition of a Grid scheduler will much depend on the way the scheduler is organized (whether it is a super-scheduler, meta-scheduler, decentralized scheduler or a local scheduler) and the characteristics of the environment such as dynamics of the system. In a general setting, however, a Grid scheduler will be permanently running as follows: receive new incoming jobs, check for available resources, select the appropriate resources according to availability, performance criteria and produce a planning of jobs to selected resources.

Usually the following terminology is employed for scheduling in Grids.

- **Task:** Represents a computational unit (typically a program and possibly associated data) to run on a Grid node. Although in the literature there is no unique definition of task concept, usually a task is considered as an indivisible schedulable unit. Tasks could be independent (or loosely coupled) or there could be dependencies (Grid workflows).
- **Job:** A job is a computational activity made up of several tasks that could require different processing capabilities and could have different resource requirements (CPU, number of nodes, memory, software libraries, etc.) and constraints, usually expressed within the job description. In the simplest case, a job could have just one task.
- **Application:** An application is the software for solving a problem in a computational infrastructure; it may require splitting the computation into jobs or it could be a “monolithic” application. In the later case, the whole application is allocated in a computational node and is usually referred to as application deployment. Applications could have different resource requirements and constraints, usually expressed within the application description.
- **Resource:** A resource is a basic computational entity (computational device or service) where tasks, jobs and applications are scheduled, allocated and processed accordingly. Resources have their own characteristics such as CPU characteristics, memory, software, etc. Several parameters are usually associated with a resource, among them the processing speed and workload, which change

over time. Moreover, the resources may belong to different administrative domains, implying different policies on usage and access.

- **Specifications:** Task, job and application requirements are usually specified using high-level specification languages (meta-languages). Similarly, the resource characteristics are expressed using specification languages. One such language is the ClassAds language.
- **Resource pre-reservation:** Pre-reservation is needed either when tasks have requirements on the finishing time or when there are dependencies that require advance resource reservation to assure the correct execution of the workflow. The advance reservation goes through negotiation and agreement protocols between resource providers and consumers.
- **Planning:** A planning is the mapping of tasks to computational resources.
- **Grid scheduler:** Software components in charge of computing a mapping of tasks to Grid resources under multiple criteria and Grid environment configurations. Different levels within a Grid scheduler have been identified in the Grid computing literature, comprising super-schedulers, meta-schedulers, local/cluster schedulers and enterprise schedulers. As a main component of any Grid system, the Grid scheduler interacts with other components of the Grid system: Grid information system, local resource management systems and network management systems. It should be noted that, in Grid environments, all these kinds of schedulers must coexist, and they could in general pursue conflicting goals; thus, there is the need for interaction and coordination between the different schedulers in order to execute the tasks.
- **Super-scheduler:** This kind of scheduler corresponds to a centralized scheduling approach in which local schedulers are used to reserve and allocate resources in the Grid, while the local schedulers manage their job queue processing. The super-scheduler is in charge of managing the advance reservation, negotiation and service level agreement.
- **Meta-scheduler:** This kind of scheduler (also known as a metabroker) arises when a single job or application is allocated in more than one resource across different systems. As in the case of super-schedulers, a meta-scheduler uses local schedulers of the particular systems. Thus, meta-schedulers coordinate local schedulers to compute an overall schedule. Performing load balancing across multiple systems is a main objective here.

- **Local/cluster scheduler:** This kind of scheduler is in charge of assigning tasks to resources in the same local area network. The scheduler manages the local resources and the local job queuing system and is thus a “close to resource” scheduler type.
- **Enterprise scheduler:** This type of scheduler arises in large enterprises having computational resources distributed in many enterprise departments. The enterprise scheduler uses the different local schedulers belonging to the same enterprise.
- **Online mode scheduling:** In online mode scheduling, tasks are scheduled as soon as they enter the system.
- **Batch mode scheduling:** In batch mode scheduling, tasks are grouped into batches which are allocated to the resources by the scheduler. The results of processing are usually obtained at a later time.
- **Non-Preemptive/preemptive scheduling:** This classification of scheduling establishes whether a task, job or application can be interrupted or not, once allocated to the resource. In the non-preemptive mode, a task, job or application should entirely be completed in the resource (the resource cannot be taken away from the task, job or application). In the preemptive mode, preemption is allowed; that is, the current execution of the job can be interrupted and the job is migrated to another resource. Preemption can be useful if job priority is to be considered as one of the constraints.
- **Cooperative scheduling:** In cooperative scheduling, a feasible schedule is computed through the cooperation of procedures, rules, and Grid users.
- **High-throughput schedulers:** The objective of this kind of scheduler [14] is to maximize the throughput (average number of tasks or jobs processed per unit of time) in the system. These schedulers are thus task-oriented schedulers; that is, the focus is in task performance criteria.
- **Resource-oriented schedulers:** The objective of this kind of scheduler is to maximize resource utilization. These schedulers are thus resource-oriented schedulers; that is, the focus is in resource performance criteria.
- **Application-oriented schedulers:** This kind of scheduler is concerned with scheduling applications in order to meet a user's performance criteria. To this end, the scheduler have to take into account the application specific as well as system

information to achieve the best performance of the application. The interaction with the user could also be considered.

3.1 Grid Scheduling Process

A user goes through three stages to schedule a job when it involves multiple sites. Phase one is Resource Discovery, in which the user makes a list of potential resources to use. Phase two involves gathering information about those resources and choosing a best set to use. In phase three the user runs the job.

3.1.1 Phase 1: Resource Discovery

Resource discovery [15] involves the user selecting a set of resources to investigate in more detail; in phase two information gathering. At the beginning of this phase, the potential set of resources is empty set and at the end of this phase, the potential set of resources is some set that has passed a minimal feasibility requirement. Most users do this in three steps namely:

- **Authorization filtering:** It is generally assumed that a user will know which resources he has access to in terms of basic services. At the end of this step the user will have a list of machines or resources to which he has access.
- **Application Requirement Definition:** In order to proceed in resource discovery, the user must be able to specify some minimal set of job requirements in order to further filter the set of feasible resources. The set of possible job requirements can be very broad and vary significantly between jobs. It may include static details such as operating system or hardware for which a binary of the code is available. Or that the code is best suited to a specific architecture. Dynamic details are also possible e.g. a minimum RAM requirement, connectivity needed. This may include any information about the job that should be specified to make sure that the job could be matched to a set of resources.
- **Minimal Requirement Filtering:** Given a set of resources to which a user has access and the minimal set of requirements the job has, the third step in the resource discovery step is to filter out the resources that do not meet the minimal job requirements. The user generally does this step by going through the list of resources and eliminating the ones that do not meet the job requirements as much

as they are known. It could also be combined with the gathering more detailed information about each resource.

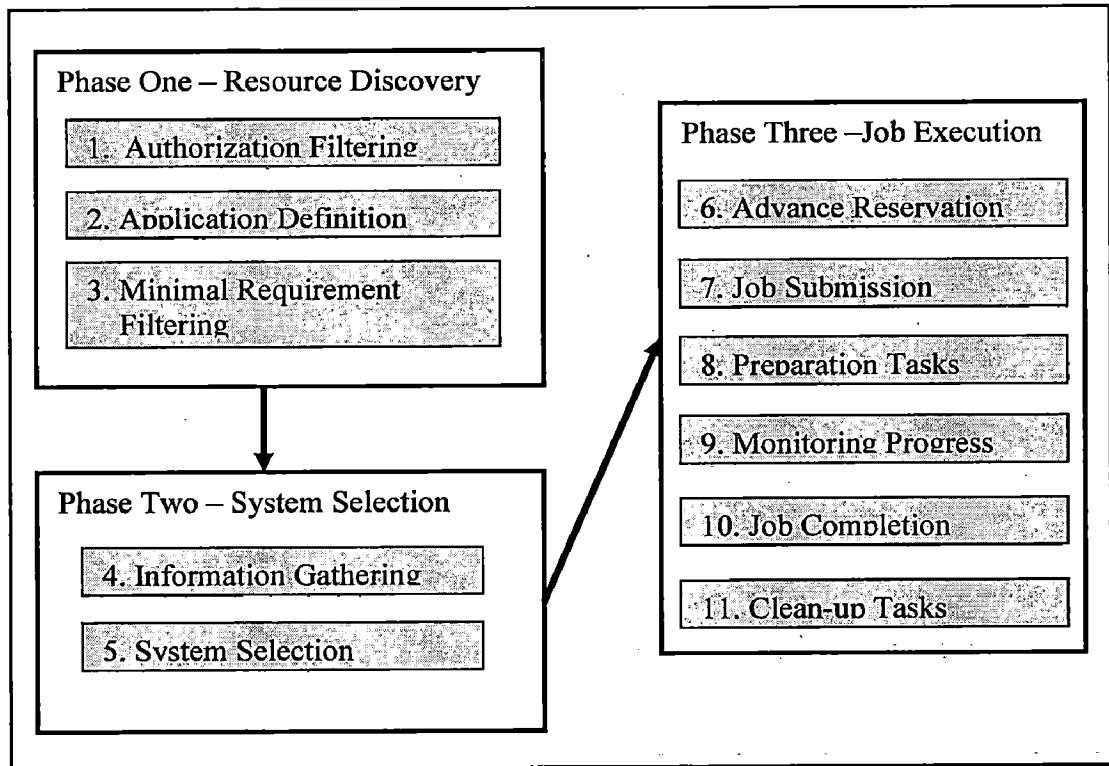


Figure 3.1: Grid Scheduling Process [15]

3.1.2 Phase 2: System Selection

Given a group of possible resources (or a group of possible resource sets), all of which meet the minimum requirements for the job, a single resource (or single resource set) must be selected on which to schedule the job. This is generally done in two steps [15]:

- **Gathering Information (QUERY):** In order to make the best possible resource match, a user needs to gather dynamic information about the resources in question. Depending on the application and resource in question, different information may be needed. Take for instance the simple case of finding the best single resource for a job to run on. A user might want to know the load on the various machine(s) and queue lengths if the machine has queues. In addition, physical characteristics and software requirements play a role, is the disk big enough for the data etc. then there are location/connectivity issues is the machine close enough to the data store. All of these issues are multiplied in the case of

multiple resources. Making an advance reservation may or may not be a part of this step.

- **Select the system(s) to run on:** Given the information gathered by the previous step, a decision of which resource (or set of resources) should the user submit a job is made in this step. This can be done in variety of ways. Note that this does not address the situation of speculative execution, where a job is submitted to multiple resources and when one begins to run the other submissions is cancelled.

3.1.3 Phase 3: Run the Job

The third phase of scheduling is running a job. This involves a number of steps [15]:

- **Make an Advance Reservation (Optional):** It may be the case that to make the best use of a given system, part or all of the resources will have to be reserved in advance. Depending on the resource, this can be easy or hard to do, may be done with mechanical means as opposed to human means, and the reservations may or may not expire with or without cost.
- **Submit Job to Resources:** Once resources are chosen the application must be submitted to resources. This may be easy as running a single command or as complicated as running a series of scripts, and may or may not include setup or staging.
- **Preparation Tasks:** The preparation stage may involve setup, claiming a reservation, or other actions needed to prepare the resource to run the application. One of the first attempts at writing a scheduler to run over multiple machines at America's National Aeronautics and Space Agency (NASA) was considered unsuccessful because it did not address the need to stage files automatically.
- **Monitor Progress:** Depending on the application and its running time, users may monitor the progress of their application.
- **Find out if Job is done:** When the job is finished, the user needs to be notified.
- **Completion Tasks:** After a job is run, the user may need to retrieve files from that resource in order to do analysis on the results, break down the environment and remove temporary settings etc.

3.2 Types of Grid Scheduling

Different types of scheduling are found in Grid systems as applications could have different scheduling needs such as batch or online mode, task independent or dependent; on the other hand, the Grid environment characteristics themselves impose restrictions such as dynamics, use of local schedulers, centralized or decentralized approach, etc. It is clear that in order to achieve the desired performance, both the problem specifics and Grid environment information should be “embedded” in the scheduler. In the following, we describe the main types of scheduling arising in Grid environments.

- ***Independent scheduling.*** Computational Grids are parallel in nature. The potential of a massive capacity of parallel computation is one of the most attractive characteristics of computational Grids. Aside from the purely scientific needs, the computational power is causing changes in important industries such as oil exploration, digital animation, aviation, financial fields, and many others. The common characteristic in these uses is that the applications are written to be able to be partitioned into almost independent parts (or loosely coupled), which can be scheduled independently.
- ***Grid workflows.*** Solving many complex problems in Grids requires the combination and orchestration of several processes (actors, services, etc.). This arises due to the dependencies in the solution flow (determined by control and data dependencies). This class of applications is known as Grid workflows. Such applications can take advantage of the power of Grid computing; however, the characteristics of the Grid environment make the coordination of its execution very complex [16, 17]. Besides the efficiency, Grid workflows should deal with robustness. Certainly, a Grid workflow could run for a long period, which in a dynamic setting increases the possibility of process failure, causing failure of the whole workflow, if failure recovery mechanisms are not used.
- ***Centralized, hierarchical and decentralized scheduling.*** Both centralized and decentralized scheduling is useful in Grid computing. Essentially, they differ in the control of the resources and knowledge of the overall Grid system. In the case of centralized scheduling, there is more control on resources: the scheduler has knowledge of the system by monitoring of the resource state, and therefore it is easier to obtain efficient schedulers. This type of scheduling, however, suffers

from limited scalability and is thus not appropriate for large-scale Grids. Moreover, centralized schedulers have a single point of failure. Another way to organize Grid schedulers is hierarchically, which allows one to coordinate different schedulers at a certain level. In this case, schedulers at the lowest level in the hierarchy have knowledge of the resources. This scheduler type still suffers from lack of scalability and fault tolerance, yet it scales better and is more fault tolerant than centralized schedulers. In decentralized or distributed scheduling there is no central entity for controlling resources. The autonomous Grid sites make it more challenging to obtain efficient schedulers. In decentralized schedulers, the local schedulers play an important role. The scheduling requests, either by local users or other Grid schedulers, are sent to local schedulers, which manage and maintain the state of the job queue. This type of scheduling is more realistic for real Grid systems of large scale, although decentralized schedulers could be less efficient than centralized schedulers.

- ***Static versus dynamic scheduling.*** There are essentially two main aspects that determine the dynamics of the Grid scheduling, namely: (a) The dynamics of job execution, which refers to the situation when job execution could fail or, in the preemptive mode, job execution is stopped due to the arrival in the system of high priority jobs; and (b) The dynamics of resources, in which resources can join or leave the Grid in an unpredictable way, their workload can significantly vary over time, the local policies on usage of resources could change over time, etc. These two factors decide the behavior of the Grid scheduler, ranging from static to highly dynamic. For instance, in the static case, there is no job failure and resources are assumed available all the time (e.g. in enterprise Grids). Although this is unrealistic for most Grids, it could be useful to consider for batch mode scheduling: the number of jobs and resources is considered fixed during short intervals of time (time interval between two successive activations of the scheduler) and the computing capacity is also considered unchangeable. Other variations are possible to consider: for instance, just the dynamics of resources but not that of jobs.

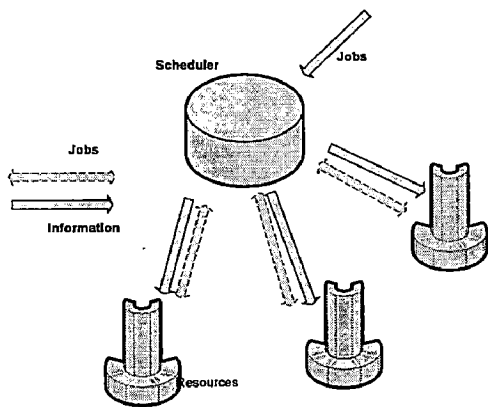


Figure 3.2: Centralized Scheduler

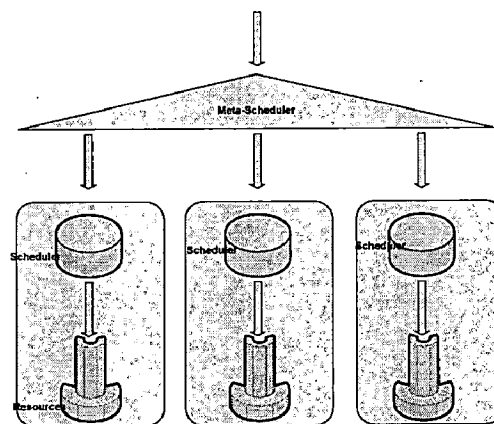


Figure 3.3: Hierarchical Scheduler

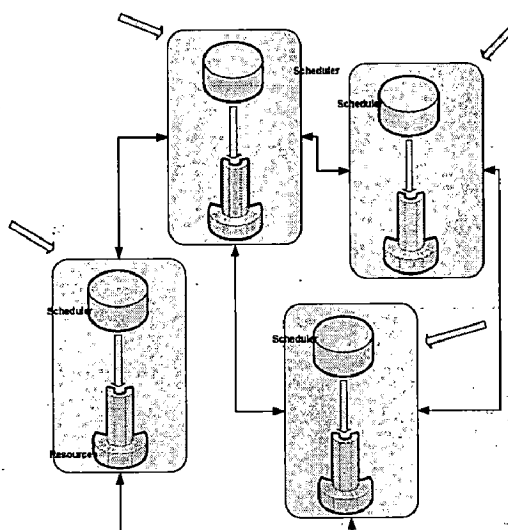


Figure 3.4: Decentralized Scheduler

- **Online versus batch mode scheduling.** Online and batch scheduling are well-known methods, largely explored in distributed computing. They are also useful for Grid scheduling. In online mode, jobs are scheduled as soon as they enter the system, without waiting for the next time interval when the scheduler will get activated or the job arrival rate is small having thus available resources to execute jobs immediately. In batch mode, tasks are grouped in batches and scheduled as a group. In contrast to online scheduling, batch scheduling could take better advantage of job and resource characteristics in deciding which job to map to which resource since they dispose of the time interval between two successive activations of the scheduler.

- ***Adaptive scheduling.*** The changeability over time of the Grid computing environment requires adaptive scheduling techniques [18], which will take into account both the current status of the resources and predictions for their future status with the aim of detecting and avoiding performance deterioration. Rescheduling can also be seen as a form of adaptive scheduling in which running jobs are migrated to more suitable resources. Casanova et al. [19] considered a class of Grid applications with large numbers of independent tasks (Monte Carlo simulations, parameter-space searches, etc.), also known as task farming applications. For these applications with loosely coupled tasks, the authors developed a general adaptive scheduling algorithm. The authors used NetSolve [20] as a test bed for evaluating the proposed algorithm. Othman et al. [21] stress the need for the Grid system's ability to recognize the state of the resources. The authors presented an approach for system adaptation, in which Grid jobs are maintained, using an adaptable Resource Broker. Huedo et al. [22] reported a scheduling algorithm built on top of the GridWay framework, which uses internally adaptive scheduling.
- ***Scheduling in Data Grids.*** Grid computing environments are making possible applications that work on distributed data and even across different data centers. In such applications, it is not only important to allocate tasks, jobs or application to the fastest and reliable nodes but also to minimize data movement and ensure fast access to data. In other terms, data location is important in such a type of scheduling. In fact, the usefulness of the large computing capacity of the Grid could be compromised by slow data transmission, which could be affected by both network bandwidth and available storage resources. Therefore, in general, data should be “close” to tasks to achieve efficient access.

3.3 Challenges in Grid Scheduling

Although Grids fall into the category of distributed parallel computing environments, they have a lot of unique characteristics, which make the Scheduling in Grids highly difficult. An adequate Grid scheduling system should overcome these challenges to leverage the promising potential of Grid systems, providing High-Performance services [23].

3.3.1 Resource Heterogeneity

A Grid has mainly two categories of resources: networks and computational resources. Heterogeneity exists in both of the two categories of resources. **First**, networks used to interconnect these resources may differ significantly in terms of their bandwidth and communicational protocols. A wide area Grid may have to utilize the best effort services provided by the internet. **Second**, computational resources are usually heterogeneous in that these resources may have different hardware, such as instruction set, computer architecture, number of processors, physical memory size, CPU speed and so on and also different software such as different operating systems, file systems, cluster management software and so on. The heterogeneity results in differing capability of processing jobs. Resources with different capacity cannot be considered uniformly. An adequate scheduling system should address the heterogeneity and further leverage different computing powers of diverse resources.

3.3.2 Site Autonomy

Typically a Grid may comprise multiple administrative domains. Each domain shares a common security and management policy. Each domain usually authorizes a group of users to use the resources in the domain. Thus applications from unauthorized users should not be eligible to run on the resources in some specific domains. Furthermore, a site is an autonomous computational entity. A shared site will result in many problems. It usually has its own scheduling policy, which complicates the prediction of a job on the site. A single overall performance goal is not feasible for a Grid system since each site has its own performance goal and scheduling decision is made independently of other sites according to its own performance goal.

3.3.3 Local Priority

It's another important issue. Each site within the Grid has its own scheduling policy. Certain classes of jobs have higher priority only on certain specific resources. For example, it can be expected that local jobs will be assigned higher priorities such that local jobs will be better served on the local resources.

3.3.4 Resource Non-Dedication

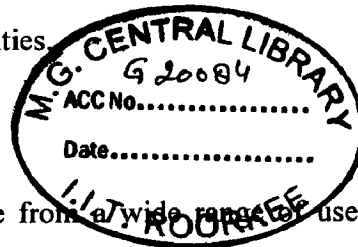
Because of non-dedication of resources, resource usage contention is a major issue. Competition may exist for both computational resources and interconnection networks. Due to the non-dedication of resources, a resource may join multiple Grids simultaneously. The workloads from both local users and other Grids share the resource concurrently. The underlying interconnection network is shared as well. One consequence of contention is that behavior and performance may vary over the time; Contention free at the guaranteed level schedulers must be able to consider the effects of contention and predict the available resource capabilities.

3.3.5 Application Diversity

This problem arises because the Grid applications are from a wide range of users, each having its own special requirements. For example, some applications may require sequential execution, some applications may consist of a set of independent jobs, and others may consist of a set of dependent jobs. In this context, building a general-purpose scheduling system seems extremely difficult. An adequate scheduling system should be able to handle a variety of applications.

3.3.6 Dynamic Behavior

In Traditional parallel computing environments such as a cluster, the pool of resources is assumed to be fixed or stable. In a Grid Environment, dynamics exists in both the networks and computational resources. **First**, a network shared by many parties cannot provide guaranteed bandwidth. This is particularly true when wide areas networks such as the internet are involved. **Second**, both the availability and capability of computational resources will exhibit dynamic behavior. On one hand new resources may join the Grid and on other hand, some resources may become unavailable due to problems such as network failure. The capability of resources may vary overtime due to the contention among many parties who share the resources. An adequate scheduler should adapt to such dynamic behavior. After a new resource joins the Grid, the scheduler should be able to detect it automatically and leverage the new resources in the later Scheduling decision making. When a computational resource becomes unavailable resulting from an unexpected failure, mechanisms such as check pointing or rescheduling should be used to guarantee the reliability of Grid



systems. These challenges pose significant obstacles on the problem of designing an efficient and effective scheduling system for Grid environments.

3.4 Batch Mode Heuristics for Independent Task

In the following discussion, m denotes number of machines, and t denotes the number of tasks in the meta-task. In the batch mode, tasks are collected into a set called metatask (MT). MT is mapped at prescheduled time called mapping events. Min-Min[24], Max-Min[24] and Sufferage[24] belongs to batch mode mapping.

3.4.1 Min-Min

Min-Min begins with the set MT of all unassigned tasks. It has two phases. In the first phase, the set of minimum expected completion time (such that task has the earliest expected completion time on the machine) for each task in MT is found. In the second phase, the task with the overall minimum expected completion time from MT is chosen and assigned to the corresponding resource. Then this task is removed from MT and the process is repeated until all tasks in the MT are mapped. This heuristic takes $O(t^2m)$ time.

3.4.2 Max-Min

Max-Min is very similar to Min-Min, except that in second phase. Max-Min assigns task with maximum expected completion time to the corresponding resource, the resource giving maximum completion time. It also takes $O(t^2m)$ time.

3.4.3 Sufferage

The Sufferage heuristic is based on the idea that better mappings can be generated by assigning a machine to a task that would “suffer” most in terms of expected completion time if that particular machine is not assigned to it. Sufferage gives each task its priority according to its sufferage value. For each task, its sufferage value is defined as the difference between its best completion time and its second best completion time. The sufferage value of each task varies over time because of the change of processor speed in a Grid. That is, it assigns the task t_i to machine m_j with earliest completion time for each task if machine m_j is available, otherwise it

calculates the sufferage value and assigns the task to that machine with high sufferage value. It also takes $O(t^2m)$ time.

The Min-Min heuristic, in most situations, maps as many tasks as possible to their first choice of service resources. In Min-Min, it is expected that a smaller makespan can be obtained if more tasks are assigned to the machines that completes them the earliest and also executes them fastest. However, the Min-Min algorithm is unable to balance the load well since it usually schedules small tasks first. The Max-Min algorithm may give a mapping with more balanced loads across the service resources in some environments. Max-Min attempts to minimize the penalties incurred from performing tasks with longer execution times. For example, let there are many tasks with shorter execution times and one task with larger execution time. Mapping the task with larger execution time to its best machine allows this task to be executed concurrently with the remaining tasks, having shorter execution time. In this case the Max-Min will give better mapping than Min-Min by executing larger task with parallel shorter tasks. In cases similar to this example, the Max-Min heuristic may give more balanced load and better makespan. The sufferage heuristic maps the task with highest sufferage value. It generally maps the short tasks first. This creates resource load unbalancing.

Chapter 4

Proposed Segmented Average-Sufferage Heuristic

The Sufferage heuristic is based on the idea that better mapping can be generated by assigning a machine to a task that would suffer most in terms of expected completion time if that particular machine is not assigned to it. For each task, its sufferage value is defined as the difference between its best minimum completion time and its second best minimum completion time. Tasks with high sufferage value take precedence. The proposed, Segmented Average-Sufferage heuristic divides the metatask in number of segments. The segment having larger tasks is mapped first. This way the task with larger execution time is executed first and finally, it results higher resource utilization and load balancing.

The following terminologies are used in this chapter. First is expected time to compute ETC_{ij} . It can be defined as time taken by the resource m_j to execute the task t_i , when there is no load with resource m_j . Second is expected completion time CT_{ij} . It can be defined as the wall-clock time when resource m_j completes the task t_i after finishing the previously assigned load.

4.1 Segmented Average-Sufferage Heuristic

Segmented Average-Sufferage heuristic first computes ETC matrix for all task t on m resources. It is a $t \times m$ matrix. Then, it computes average of each row of ETC. By taking this average value as key, it sorts these tasks in decreasing order of their respective key. The heuristic partitions the metatask into N segments and schedules each segment as their order in sorted list of task set by applying sufferage heuristic. Determining the optimal value of N is a trade-off. More segments result in better load balance. On the other hand, too many segments will lose advantages of the sufferage algorithm. Intuitively, as long as we partition the tasks into a few segments, such as large, medium, and small tasks, the load can be balanced fairly well. When t/m is large, sufferage performs well. For small t/m , which means the number of tasks per machine is not large, the optimal value of N is about 4 or 5. Therefore, we fix the

value of t/m to 4, which means that we always partition the tasks into four segments. The algorithm's steps are shown in figure 4.1.

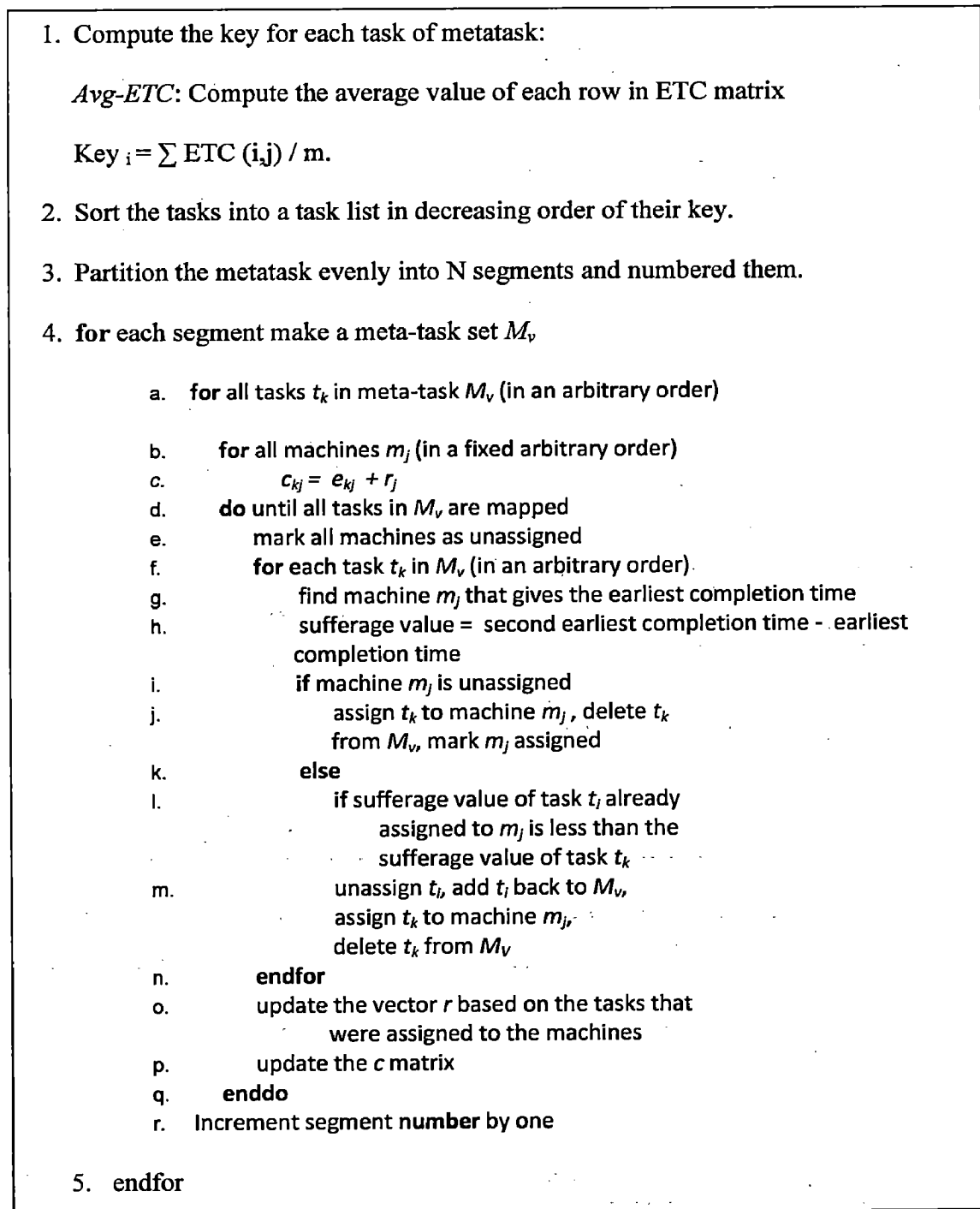


Figure 4.1: Segmented Average-Sufferage heuristic

Segmented Average-Sufferage performs task sorting before scheduling. Sorting implies that larger tasks are promoted to schedule earlier. Segmentation makes this algorithm faster than the existing Sufferage heuristic. Because after sorting them we

have N sets and we have to apply Sufferage in each segment. Now in each task set t/N tasks can be scheduled in short time by applying Sufferage.

4.2 Mapping Comparison of Sufferage and Segmented Average-Sufferage Heuristics

The expected execution times of ten tasks on four resources are shown in Table 4.1

Table 4.1 ETC Matrix for Ten Tasks on Four Resources

	R1	R2	R3	R4
t1	13.7	2.28	4.56	1.14
t2	11.75	1.95	3.91	0.97
t3	14.5	2.41	4.83	1.2
t4	11.84	1.97	3.94	0.98
t5	9.28	1.54	3.09	0.77
t6	16.53	2.75	5.51	1.37
t7	8.84	1.47	2.94	0.73
t8	16	2.66	5.33	1.33
t9	14.32	2.38	4.77	1.19
t10	8.66	1.44	2.88	0.72

a) Sufferage

The sufferage value of each task is computed. The task t6 is having the maximum sufferage value. It will be mapped on resource R4.

	R1	R2	R3	R4	Sufferage value
t1	13.7	2.28	4.56	1.14	1.14
t2	11.75	1.95	3.91	0.97	0.98
t3	14.5	2.41	4.83	1.2	1.21
t4	11.84	1.97	3.94	0.98	0.99
t5	9.28	1.54	3.09	0.77	0.77
t6	16.53	2.75	5.51	1.37	1.38
t7	8.84	1.47	2.94	0.73	0.74
t8	16	2.66	5.33	1.33	1.33
t9	14.32	2.38	4.77	1.19	1.19
t10	8.66	1.44	2.88	0.72	0.72

In next iteration, task t10 is having maximum sufferage value and it will be mapped on resource R2.

	R1	R2	R3	R4	Sufferage value
t1	13.7	2.28	4.56	2.51	0.23
t2	11.75	1.95	3.91	2.34	0.39
t3	14.5	2.41	4.83	2.57	0.16
t4	11.84	1.97	3.94	2.35	0.38
t5	9.28	1.54	3.09	2.14	0.6
t7	8.84	1.47	2.94	2.1	0.63
t8	16	2.66	5.33	2.7	0.04
t9	14.32	2.38	4.77	2.56	0.18
t10	8.66	1.44	2.88	2.09	0.65

In next iteration, task t8 is having maximum sufferage value and it will be mapped on resource R4.

	R1	R2	R3	R4	Sufferage value
t1	13.7	3.72	4.56	2.51	1.21
t2	11.75	3.39	3.91	2.34	1.05
t3	14.5	3.85	4.83	2.57	1.28
t4	11.84	3.41	3.94	2.35	1.06
t5	9.28	2.98	3.09	2.14	0.84
t7	8.84	2.91	2.94	2.1	0.81
t8	16	4.1	5.33	2.7	1.40
t9	14.32	3.82	4.77	2.56	1.26

Similarly, for the remaining tasks, we are finding first the sufferage value and selecting the task with maximum sufferage value for mapping. The complete result of mapping is shown in figure 4.2. We are getting 6.88 makespan using Sufferage.

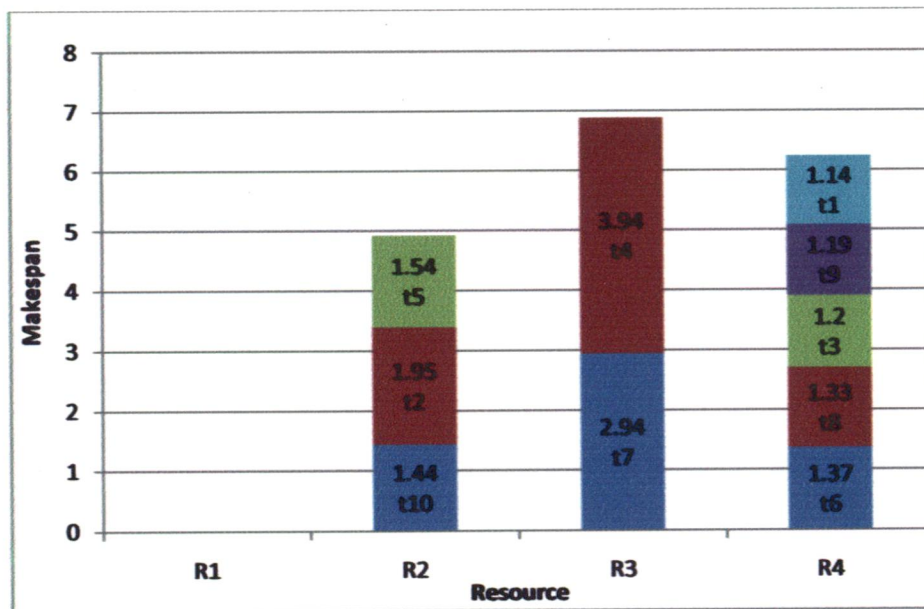


Figure 4.2: Result of Suffrage Heuristic

b) Segmented Average-Suffrage

First, the average execution time of each task on all resources is computed.

This is called the key.

	R1	R2	R3	R4	Key _i = $\sum \text{ETC} (i,j) / m$
t1	13.7	2.28	4.56	1.14	5.42
t2	11.75	1.95	3.91	0.97	4.64
t3	14.5	2.41	4.83	1.2	5.73
t4	11.84	1.97	3.94	0.98	4.68
t5	9.28	1.54	3.09	0.77	3.67
t6	16.53	2.75	5.51	1.37	6.54
t7	8.84	1.47	2.94	0.73	3.49
t8	16	2.66	5.33	1.33	6.33
t9	14.32	2.38	4.77	1.19	5.66
t10	8.66	1.44	2.88	0.72	3.42

Sort the task set into decreasing order of their key.

	R1	R2	R3	R4	Key _i = \sum ETC (i,j) / m
t6	16.53	2.75	5.51	1.37	6.54
t8	16	2.66	5.33	1.33	6.33
t3	14.5	2.41	4.83	1.2	5.73
t9	14.32	2.38	4.77	1.19	5.66
t1	13.7	2.28	4.56	1.14	5.42
t4	11.84	1.97	3.94	0.98	4.68
t2	11.75	1.95	3.91	0.97	4.64
t5	9.28	1.54	3.09	0.77	3.67
t7	8.84	1.47	2.94	0.73	3.49
t10	8.66	1.44	2.88	0.72	3.42

We created 2 segments. Task t6, t8, t3, t9 and t1 is assigned to segment1 and task t4, t2, t5, t7 and t10 is assigned to segment 2. For each segment, the sufferage heuristic is applied for mapping. First tasks from segment 1 are mapped. The detailed steps are shown below.

	R1	R2	R3	R4	
Segment 1	t6	16.53	2.75	5.51	1.37
	t8	16	2.66	5.33	1.33
	t3	14.5	2.41	4.83	1.2
	t9	14.32	2.38	4.77	1.19
	t1	13.7	2.28	4.56	1.14
Segment 2	t4	11.84	1.97	3.94	0.98
	t2	11.75	1.95	3.91	0.97
	t5	9.28	1.54	3.09	0.77
	t7	8.84	1.47	2.94	0.73
	t10	8.66	1.44	2.88	0.72

Before assigning any task to any resource all resources having ready time 0.0.

	R1	R2	R3	R4
Ready time	0.0	0.0	0.0	0.0

Now calculating sufferage value of each task {t6, t8, t3, t9, t1}.

	R1	R2	R3	R4	Sufferage value
t6	16.53	2.75	5.51	1.37	1.38
t8	16	2.66	5.33	1.33	1.33
t3	14.5	2.41	4.83	1.2	1.21
t9	14.32	2.38	4.77	1.19	1.19
t1	13.7	2.28	4.56	1.14	1.14

	R1	R2	R3	R4
Ready time	0.0	0.0	0.0	1.37

Task t6 have largest sufferage value 1.38, so it will be mapped on resource R4 and now the updated ready time of R4 is 1.37.

	R1	R2	R3	R4	Sufferage value
t8	16	2.66	5.33	2.7	0.04
t3	14.5	2.41	4.83	2.57	0.16
t9	14.32	2.38	4.77	2.56	0.18
t1	13.7	2.28	4.56	2.51	0.23

	R1	R2	R3	R4
Ready time	0.0	2.28	0.0	1.37

Task t1 have largest sufferage value 0.23, so it will be mapped resource R2 and now the updated ready time of R2 is 2.28.

	R1	R2	R3	R4	Sufferage value
t8	16	4.94	5.33	2.7	2.24
t3	14.5	4.69	4.83	2.57	2.12
t9	14.32	4.66	4.77	2.56	2.1

	R1	R2	R3	R4
Ready time	0.0	2.28	0.0	2.7

Task t8 have largest sufferage value 2.24, so it will be mapped on resource R4 and now the updated ready time of R4 is 2.7.

	R1	R2	R3	R4	Sufferage value
t3	14.5	4.69	4.83	3.9	0.79
t9	14.32	4.66	4.77	3.89	0.77

	R1	R2	R3	R4
Ready time	0.0	2.28	0.0	3.9

Task t3 have largest sufferage value 0.79, so it will be mapped on resource R4 and now the updated ready time of R4 is 3.9.

	R1	R2	R3	R4	Sufferage value
t9	14.32	4.66	4.77	5.09	0.11

	R1	R2	R3	R4
Ready time	0.0	4.66	0.0	3.9

Now the remaining task t9 having sufferage value 0.11, it will go to resource R2 and now updated ready time of R2 is 4.66.

Ready times of resources are 0.0, 4.66, 0.0 and 3.9. After completing with segment 1 now applying same steps from 4.a to 4.r of figure 4.1 on segment 2.

	R1	R2	R3	R4
t4	11.84	1.97	3.94	0.98
t2	11.75	1.95	3.91	0.97
t5	9.28	1.54	3.09	0.77
t7	8.84	1.47	2.94	0.73
t10	8.66	1.44	2.88	0.72

Ready times of all the resources are 0.0, 4.66, 0.0 and 3.9.

	R1	R2	R3	R4
Ready time	0.0	4.66	0.0	3.9

Now calculating sufferage value of each task {t4, t2, t5, t7, t10}.

	R1	R2	R3	R4	Sufferage value
t4	11.84	6.63	3.94	4.88	0.94
t2	11.75	6.61	3.91	4.87	0.96
t5	9.28	6.2	3.09	4.67	1.58
t7	8.84	6.13	2.94	4.63	1.69
t10	8.66	6.1	2.88	4.62	1.74

	R1	R2	R3	R4
Ready time	0.0	4.66	2.88	3.9

Task t10 have largest sufferage value 1.74, so it will be mapped to resource R3 and now the updated ready time of R3 is 2.88.

	R1	R2	R3	R4	Sufferage Value
t4	11.84	6.63	6.82	4.88	1.75
t2	11.75	6.61	6.79	4.86	1.75
t5	9.28	6.2	5.97	4.67	1.3
t7	8.84	6.13	5.82	4.63	1.19

	R1	R2	R3	R4
Ready time	0.0	4.66	2.88	4.88

Task t4 have largest sufferage value 1.75, so it will be mapped to resource R4 and now the updated ready time of R4 is 4.88.

	R1	R2	R3	R4	Sufferage Value
t2	11.75	6.61	6.79	5.84	0.95
t5	9.28	6.2	5.97	5.65	0.32
t7	8.84	6.13	5.82	5.61	0.21

	R1	R2	R3	R4
Ready time	0.0	4.66	2.88	5.84

Task t2 have largest sufferage value 0.95, so it will be mapped to resource R4 and now the updated ready time of R4 is 5.84.

	R1	R2	R3	R4	Sufferage value
t5	9.28	6.2	5.97	6.62	0.65
t7	8.84	6.13	5.82	6.58	0.76

	R1	R2	R3	R4
Ready time	0.0	4.66	5.82	5.84

Task t7 have largest sufferage value 0.76, so it will be mapped to resource R3 and now the updated ready time of R3 is 5.82.

	R1	R2	R3	R4	Sufferage value
t5	9.28	6.2	8.76	6.62	0.42

	R1	R2	R3	R4
Ready time	0.0	6.2	5.82	5.84

Now the remaining task t5 having sufferage value 0.42, it will go to resource R2 and now updated ready time of R2 is 6.2. Segmented Average-Sufferage heuristic gives 6.2 makespan. The results are shown in figure 4.3.

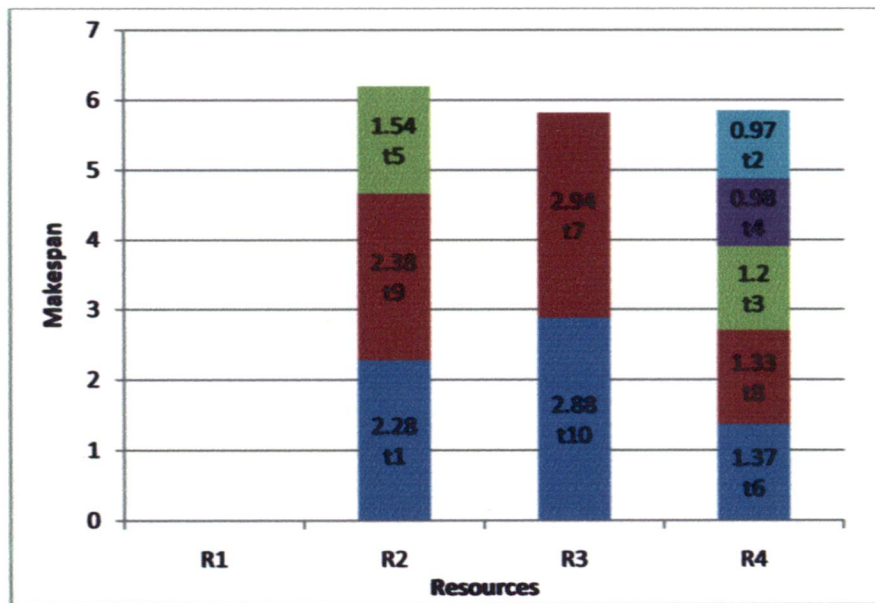


Figure 4.3: Result of Segmented Average-Sufferage

Above example explains Segmented Average-Sufferage heuristic achieves significant improvement in makespan compared to Sufferage heuristic.

Chapter 5

Results and Discussions

5.1 Performance Metrics

Depending on what scheduling performance is desired in Grid, there are different performance metrics for evaluating scheduling algorithms. Here, the results are evaluated on the basis of following performance matrices.

Makespan: - Makespan is the measure of the throughput of the Grid. It can be calculated using equation (5.1).

$$\text{makespan} = \max_{t_i \in \text{MT}}(\text{CT}_i) \quad (5.1)$$

Here CT_i is the total time taken by task t_i for execution. The less the makespan, the better is the algorithm.

Average Resource Utilization [25]:- Average resource utilization rate of all resources can be calculated through equation (5.2)

$$\text{ru} = \frac{\sum_{j=1}^m \text{ru}_j}{m} \quad (5.2)$$

Here ru_j is the resource utilization rate of resource r_j . It can be calculated using equation (5.3)

$$\text{ru}_j = \frac{\sum_{i \text{ where } t_i \text{ has been executed on } m_j} (\text{te}_i - \text{ts}_i)}{T} \quad (5.3)$$

Here te_i is the finish time and ts_i is the start time of task t_i on resource m_j . T is the total application time elapsed so far. It can be calculated using equation (5.4)

$$T = \max(\text{te}_i) - \min(\text{ts}_i) \quad (5.4)$$

Load Balancing Level [25]:- The mean square deviation of ru is given by equation (5.5)

$$d = \sqrt{\frac{\sum_{i=1}^m (\text{ru} - \text{ru}_i)^2}{m}} \quad (5.5)$$

The load balancing level, β , is determined through the relative deviation of d over ru .

$$\beta = 1 - \frac{d}{ru} \quad (5.6)$$

The best load balancing level is achieved if β reaches to 1 and d is close to 0.

5.2 Simulation Environment

To evaluate the Segmented Average-Sufferage heuristic, we have used the Gridsim Toolkit [26], a Grid Simulator as a simulation tool. We have compared the results given by our heuristic with the results given by Min-Min, Max-Min and Sufferage heuristics. For evaluating and comparing the results of Segmented Average-Sufferage heuristic we have used the three task cases as listed below.

Task length generation is done based on the formula given below.

$$\text{Task length} = \{ \text{value} * (1 - \text{lessFactor} + (\text{lessFactor} + \text{moreFactor}) * \text{randDouble}) \}$$

value - the estimated value

$0.0 \leq \text{lessFactor}$ and $\text{moreFactor} \leq 1.0$

randDouble - an uniformly distributed double value between 0.0 and 1.0

Case I: - A few short tasks (2% to 10%) along with many long tasks.

For short tasks: *value* = 100, *lessFactor* = 0.1, *moreFactor* = 0.9 and *randDouble* = An uniformly distributed random double value between 0.0 and 1.0

For long tasks: *value* = 9000, *lessFactor* = 0.1, *moreFactor* = 0.9 and *randDouble* = An uniformly distributed random double value between 0.0 and 1.0

Case II: - A few long tasks (2% to 10%) along with many short tasks.

For short tasks: *value* = 9000, *lessFactor* = 0.1, *moreFactor* = 0.9 and *randDouble* = An uniformly distributed random double value between 0.0 and 1.0

For long tasks: *value* = 100, *lessFactor* = 0.1, *moreFactor* = 0.9 and *randDouble* = An uniformly distributed random double value between 0.0 and 1.0

Case III: - Length of tasks is randomly determined.

For all task value = An uniformly distributed random double value between 100 and 9000, lessFactor, moreFactor and randDouble are uniformly distributed random double value between 0 and 1.

We have taken 10 resources and 1000 task for each case. Every resource has 15 to 20 machines and each machine has 2 to 4 processing elements. The arrival of tasks is modeled as Poisson random process.

5.3 Results

We have used the performance metrics given in section 5.1 for evaluating the results of Segmented Average-Sufferage (Avg-Suff), Sufferage, Min-Min and Max-Min heuristics. We have taken the task cases given in section 5.2. The results of makespan, average resource utilization and load balancing level are shown below.

a) Makespan Results

The makespan results are shown in figure 5.1, 5.2 and 5.3 for the task cases I, II, III, respectively. The results are compared with the results of Sufferage, Min-Min and Max-Min heuristics. Table 5.1 shows the comparison of Segmented Average-Sufferage and Sufferage heuristics. We can observe that the proposed heuristic gives 9.9%, 25.31% and 4.36% gain in makespan over sufferage for task cases I, II, III, respectively. Table 5.2 shows the comparison of Segmented Average-Sufferage and Min-Min heuristics. We can observe that the proposed heuristic gives 11.17%, 26.96% and 5.12% gain in makespan over Min-Min for task cases I, II, III, respectively. Table 5.3 shows the comparison of Segmented Average-Sufferage and Max-Min heuristics. We can observe that the proposed heuristic gives 12.78%, 26.56% and 4.97% gain in makespan over sufferage for task cases I, II, III, respectively. Over all, the proposed heuristic gives better makespan than Sufferage, Min-Min and Max-Min for each task case.

Table 5.1. Makespan Comparison of Segmented Average-Sufferage with Sufferage

Cases	Makespan (In Hundred Seconds)		Improvement over Sufferage
	Avg-Suff	Sufferage	
Case I	39.37	43.70	9.9%
Case II	29.45	39.43	25.31%
Case III	42.99	44.95	4.36%

Table 5.2. Makespan comparison of Segmented Average-Sufferage with Min-Min

Cases	Makespan (In Hundred Seconds)		Improvement over Min-Min
	Avg-Suff	Min-Min	
Case I	39.37	44.32	11.17%
Case II	29.45	40.32	26.96%
Case III	42.99	45.31	5.12%

Table 5.3. Makespan comparison of Segmented Average-Sufferage with Max-Min

Cases	Makespan (In Hundred Seconds)		Improvement over Max-Min
	Avg-Suff	Max-Min	
Case I	39.37	45.14	12.78%
Case II	29.45	40.1	26.56%
Case III	42.99	45.24	4.97%

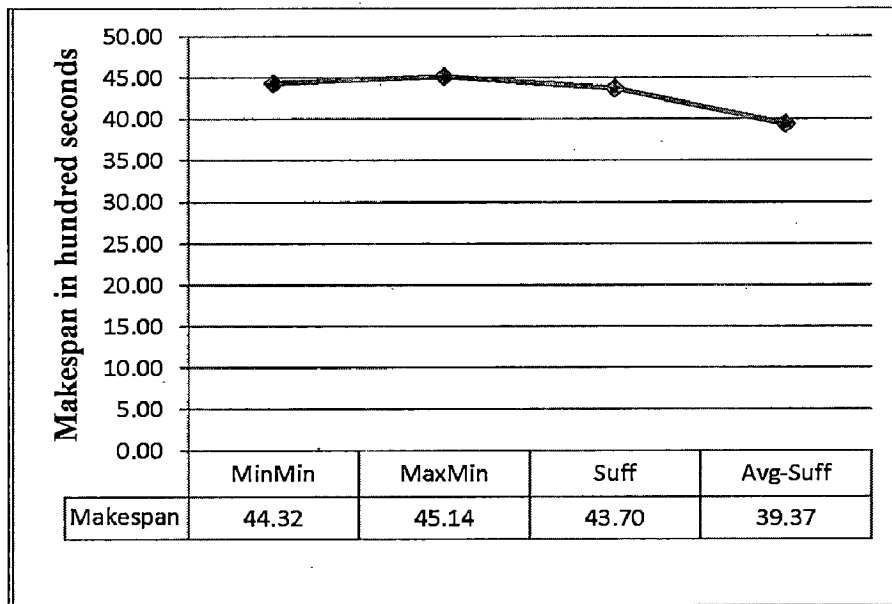


Figure 5.1: Makespan in Case I

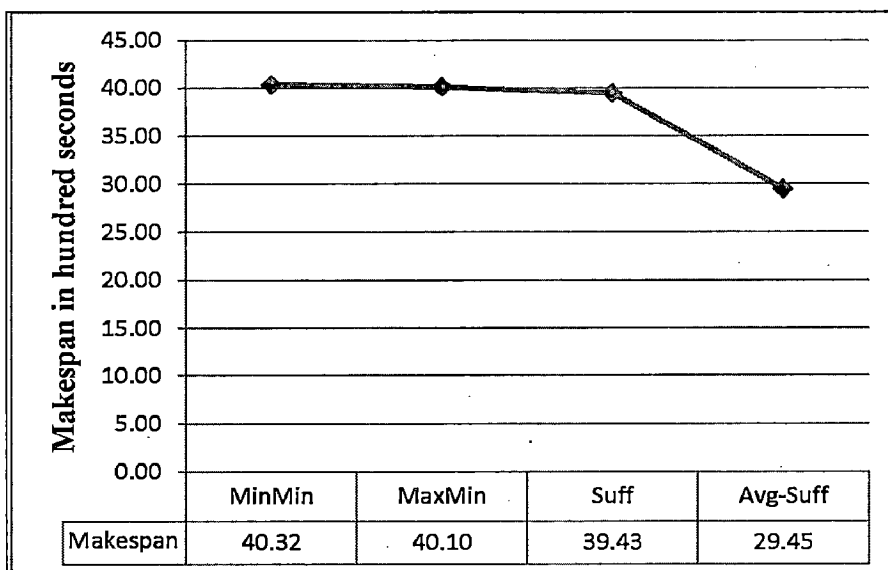


Figure 5.2: Makespan in Case II

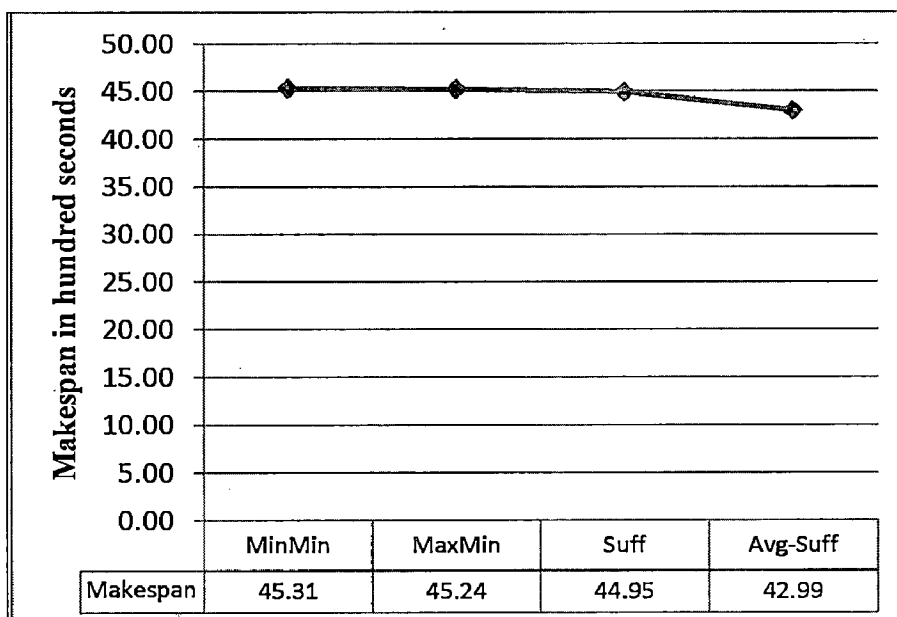


Figure 5.3: Makespan in Case III

b) Average Resource Utilization

Figures 5.4, 5.5 and 5.6 show the results of resource utilization rate for the case I, II and III, respectively. We can observe from them the proposed heuristic gives better resource utilization than others.

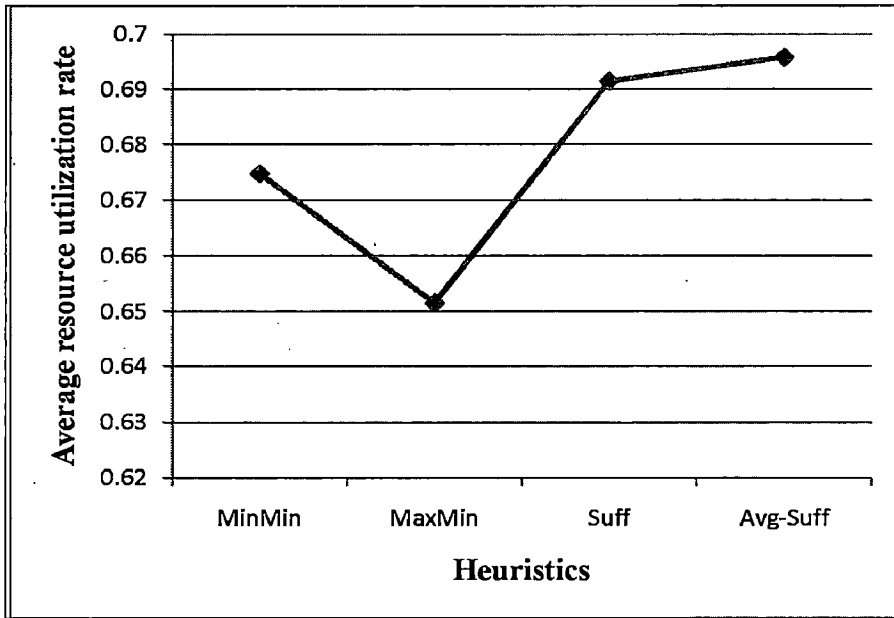


Figure 5.4: Average resource utilization in Case I

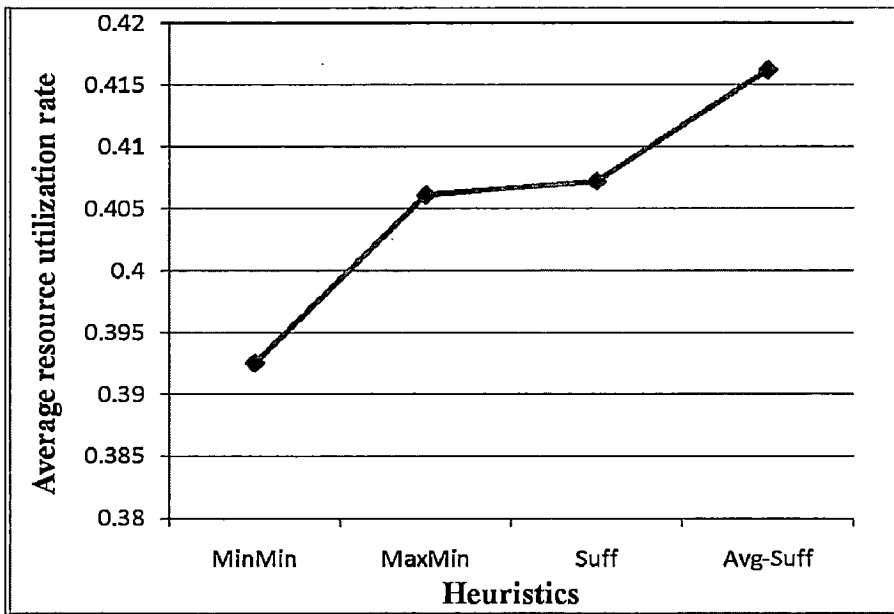


Figure 5.5: Average resource utilization in Case II

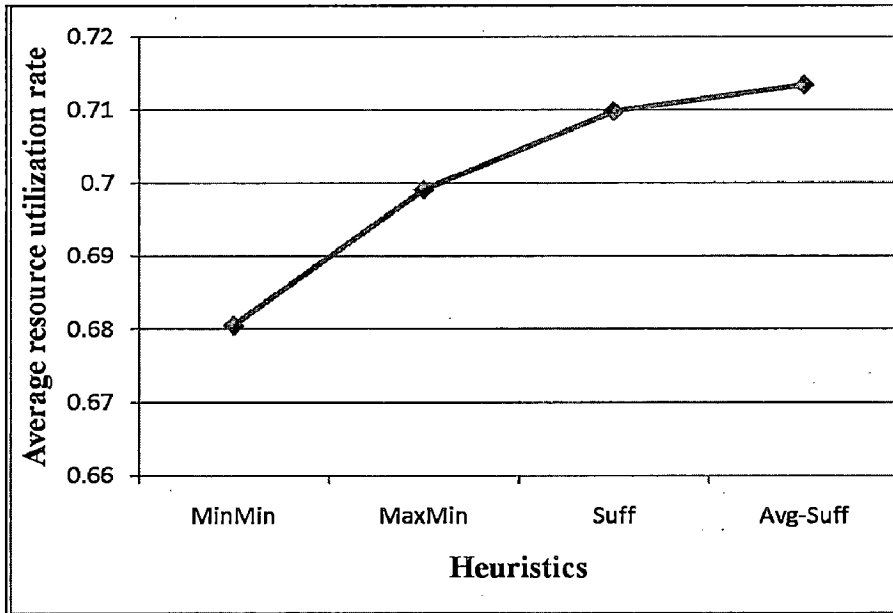


Figure 5.6: Average resource utilization in Case III

c) Load Balancing Level

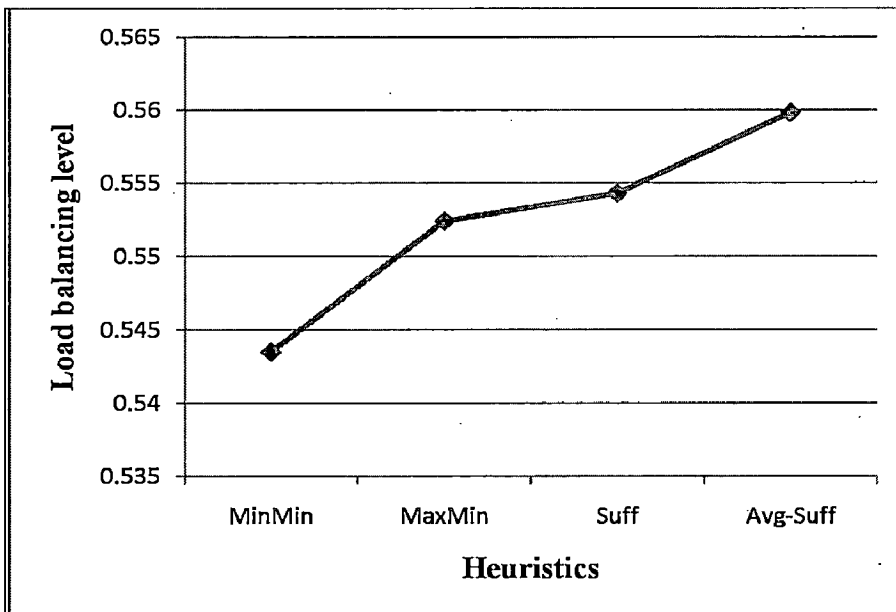


Figure 5.7: Load balancing level in Case I

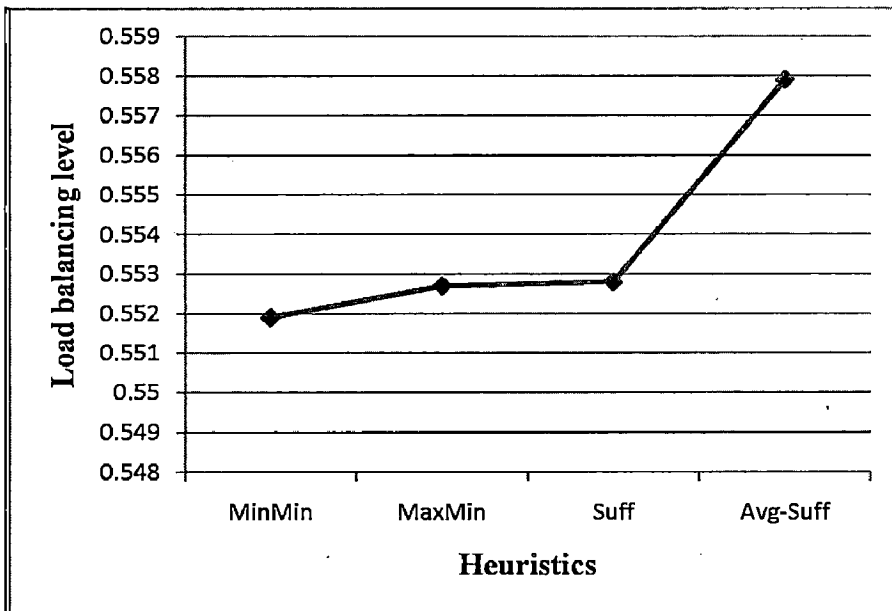


Figure 5.8: load balancing level in Case II

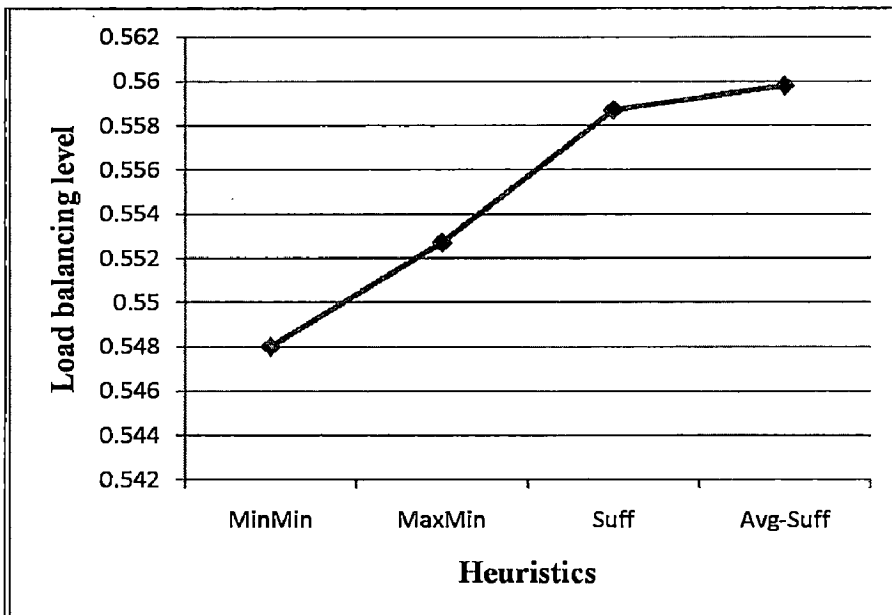


Figure 5.9: load balancing level in Case III

Figures 5.7, 5.8 and 5.9 show the results of load balancing level for the case I, II and III, respectively. We can observe from them the proposed heuristic gives better resource load balancing than others.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this dissertation, we have proposed Segmented Average-Sufferage heuristic to achieve high throughput in Grid computing. The study concentrates for batch mode independent task scheduling. We have used segmentation method to get better load balancing. It also helps to improve makespan.

The following improvements in makespan are obtained.

- The Segmented Average-Sufferage gives up to 26.96% improvement in makespan than Min-Min heuristic.
- The Segmented Average-Sufferage gives up to 26.56% improvement in makespan than Max-Min heuristic.
- The segmented Average-Sufferage gives up to 25.31% improvement in makespan than Sufferage heuristic.

The heuristic are also tested for resource utilization and load balancing. From the results given in chapter 5, we can conclude that the proposed heuristic, segmented average sufferage gives better resource utilization and resource load balancing than Min-Min, Max-Min and Sufferage heuristics.

6.2 Scope for Future Work

The proposed heuristic is tested for independent tasks batch mode scheduling in static environment. The following domains can be considered for future work.

- i) The heuristic can be implemented and tested in actual Grid environment.
- ii) The heuristic can be modified to consider the QoS demands of tasks.
- iii) The heuristic can be investigated in dynamic environment.

References

- [1] Krishnaram Kenthapadi, and Gurmeet Singh Mankuy, “Decentralized Algorithms using both Local and Random Probes for P2P Load Balancing,” Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures, Las Vegas, Nevada, USA, July 18 – 20, 2005, pp. 135 – 144.
- [2] B. Yagoubi and Y. Slimani, “Task Load Balancing Strategy for Grid Computing,” 22nd international symposium on Journal of Computer Science, vol. 3, no. 3, Ankara, Nov. 7 - 9 2007, pp. 186-194.
- [3] Dazhang Gu, Lin Yang, Lonnie R. Welch “A Predictive, Decentralized Load Balancing Approach”, Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium, April 2005 pp. 131b - 131b.
- [4] Ibarra O H and Kim C E. “Heuristic algorithms for scheduling independent tasks on non-identical processors”. Journal of the ACM, vol. 24, no. 2, 1977, pp. 280–289,
- [5] Foster, I., Kesselman, C. and Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing applications, vol. 15, no. 3, 2001, pp. 200-222.
- [6] Jarek Nabrzyski, Jennifer M. Schopf & Jan Weglarz, “GRID RESOURCE MANAGEMENT–STATE OF THE ART AND FUTURE TRENDS”, Kluwer Academic Publisher. Norwell, MA, USA Year of Publication: 2004.
- [7] Hai Zhuge, Xiaoping Sun, Jie Liu, Erlin Yao, and Xue Chen , “A Scalable P2P Platform for the Knowledge Grid”, IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 2, June 2002, pp. 1721 – 1736.
- [8] Scheduling and Resource Management in Computational Mini-Grids, www.parl.clemson.edu/~wjones/research/draft.pdf, July 1, 2002
- [9] Thierry Prioi, “Grid Middleware”, <http://www.gridatasia.net>, Advanced Grid Research Workshops through European and Asian Co-operation.
- [10] Yih-Jiun Lee and Peter Henderson, “A Practical Modelling Notation for Secure Distributed Computation”, Proceedings of 19th International Conference on Advanced Information Networking and Applications, 28-30 March 2005 pp.439 – 442.

- [11] I. Foster, et al., "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Technical Report, Glous Project; <http://www.globus.org/research/papers/ogsa.pdf> (current June 2010).
- [12] Foster, I., C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations". *International Journal of High Performance Computing Applications*, vol. 15, no. 3, 200-222, 2001.
- [13] Clovis Chapman¹, Paul Wilson², "Condor services for the Global Grid: Interoperability between Condor and OGSA", *Proceedings of the 2004 UK e-Science All Hands Meeting*, ISBN 1-904425-21-6, pp. 870-877, Nottingham, UK, August 2004 <http://www.cs.wisc.edu/condor/doc/condor-ogsa-2004.pdf>
- [14] E. Caron, V. Garonne, A. Tsaregorodtsev, "Definition, Modelling and Simulation of a Grid Computing Scheduling System for High Throughput Computing", *Future Generation Computer Systems*, vol. 23, no. 6, 2007, pp. 968-976.
- [15] J. M. Schopf, "A General Architecture for Scheduling in the Grid", *Journal of parallel and distributed computing*, special issue of Grid Computing, 2002.
- [16] J. Cao, S.A. Jarvis, S. Saini, G.R. Nudd, "GridFlow: Workflow Management for Grid Computing", in *Proceedings of the 3rd International Symposium on Cluster Computing and the Grid, CCGrid'03*, 2003, pp. 198-205.
- [17] J. Yu, R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing", *Journal of Grid Computing*, vol. 3, no. 2, 2006, pp. 171-200.
- [18] L. Lee, C. Liang, H. Chang, "An Adaptive Task Scheduling System for Grid Computing", in *Proceedings of the Sixth IEEE international Conference on Computer and information Technology, CIT'06*, 2006, p. 57.
- [19] H. Casanova, M. Kim, J.S. Plank, J.J. Dongarra, "Adaptive Scheduling for Task Farming with Grid Middleware", *International Journal of High Performance Computing Applications* vol. 13, no. 3, 1999, pp. 231-240.
- [20] H. Casanova, J. Dongarra, Netsolve: "Network Enabled Solvers, *IEEE Computational Science and Engineering*, vol. 5, no.3, 1998, pp. 57-67.
- [21] A. Othman, P. Dew, K. Djemame, K. Gourlay, "Adaptive Grid Resource Brokering", in *Proceedings of IEEE International Conference on Cluster Computing*, Hong Kong, 1 - 4 Dec. 2003, pp. 172-179.
- [22] E. Huedo, R.S. Montero, I.M. Llorente, "Experiences on Adaptive Grid Scheduling of Parameter Sweep Applications", in *Processings of 12th*

- Euromicro Conference on Parallel, Distributed and Network-based, PDP'04, 11-13 Feb. 2004, pp. 28 - 33.
- [23] N. Tonello, R. Yahyapour, "A Proposal for a Generic Grid Scheduling Architecture", Technical Report CoreGrid no TR-0015, Institute of Resource Management and Scheduling, Germany, 2006.
- [24] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. Freund, "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems", in proceedings of 8th IEEE Heterogeneous Computing Workshop (HCW '99), San Juan, Puerto Rico, April 1999, pp. 30 - 44.
- [25] Kobra Etminani, M. Naghibzadeh, "A Min-Min Max-Min Selective Algorithm for Grid Task Scheduling," 3rd IEEE/IFIP International Conference on Internet in Central Asia, 26 – 28 Sept, 2007, pp. 1 – 7.
- [26] R. Buyya, M. Murshed, "Gridsim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing," *Journal of Concurrency and Computation: Practice and Experience*, vol. 14, no. 13, Wiley Press, Nov. - Dec., 2002, pp. 1175–1220.

List of Publications

- [1] Ashish Kumar, Anil. K. Sarje, "*Segmented Average-Suffrage Heuristic for Independent Task Scheduling in Grid*", 4th International Conference on Information Processing – 2010, August 6-8, 2010, Bangalore, India. (Accepted)

Appendix A

Introduction to Gridsim

The Gridsim toolkit provides a comprehensive facility for simulation of different classes of heterogeneous resources, users, applications, resource brokers, and schedulers. It can be used to simulate application schedulers for single or multiple administrative domains distributed computing systems such as clusters and Grids. Application schedulers in the Grid environment, called resource brokers, perform resource discovery, selection, and aggregation of a diverse set of distributed resources for an individual user. This means that each user has his or her own private resource broker and hence it can be targeted to optimize for the requirements and objectives of its owner. In contrast, schedulers, managing resources such as clusters in a single administrative domain, have complete control over the policy used for allocation of resources. This means that all users need to submit their jobs to the central scheduler, which can be targeted to perform global optimization such as higher system utilization and overall user satisfaction depending on resource allocation policy or optimize for high priority users.

System Architecture of Gridsim

It employed a layered and modular architecture for Grid simulation to leverage existing technologies and manage them as separate components [26]. A multi-layer architecture and abstraction for the development of Gridsim platform and its applications is shown in Figure A-1. The first layer is concerned with the scalable Java interface and the runtime machinery, called JVM (Java Virtual Machine), whose implementation is available for single and multiprocessor systems including clusters. The second layer is concerned with a basic discrete-event infrastructure built using the interfaces provided by the first layer. One of the popular discrete-event infrastructure implementations available in Java is SimJava. Recently, a distributed implementation of SimJava was also made available. The third layer is concerned with modeling and simulation of core Grid entities such as resources, information services, and so on; application model, uniform access interface, and primitives application modeling and framework for creating higher level entities. The Gridsim toolkit focuses on this layer

that simulates system entities using the discrete event services offered by the lower-level infrastructure. The fourth layer is concerned with the simulation of resource aggregators called Grid resource brokers or schedulers. The final layer is focused on application and resource modeling with different scenarios using the services provided by the two lower-level layers for evaluating scheduling and resource management policies, heuristics, and algorithms. In this section, we briefly discuss the SimJava model for discrete events (a second-layer component) and focus mainly on the Gridsim (the third layer) design and implementation. Resource broker simulation and performance evaluation are highlighted in the next two sections.

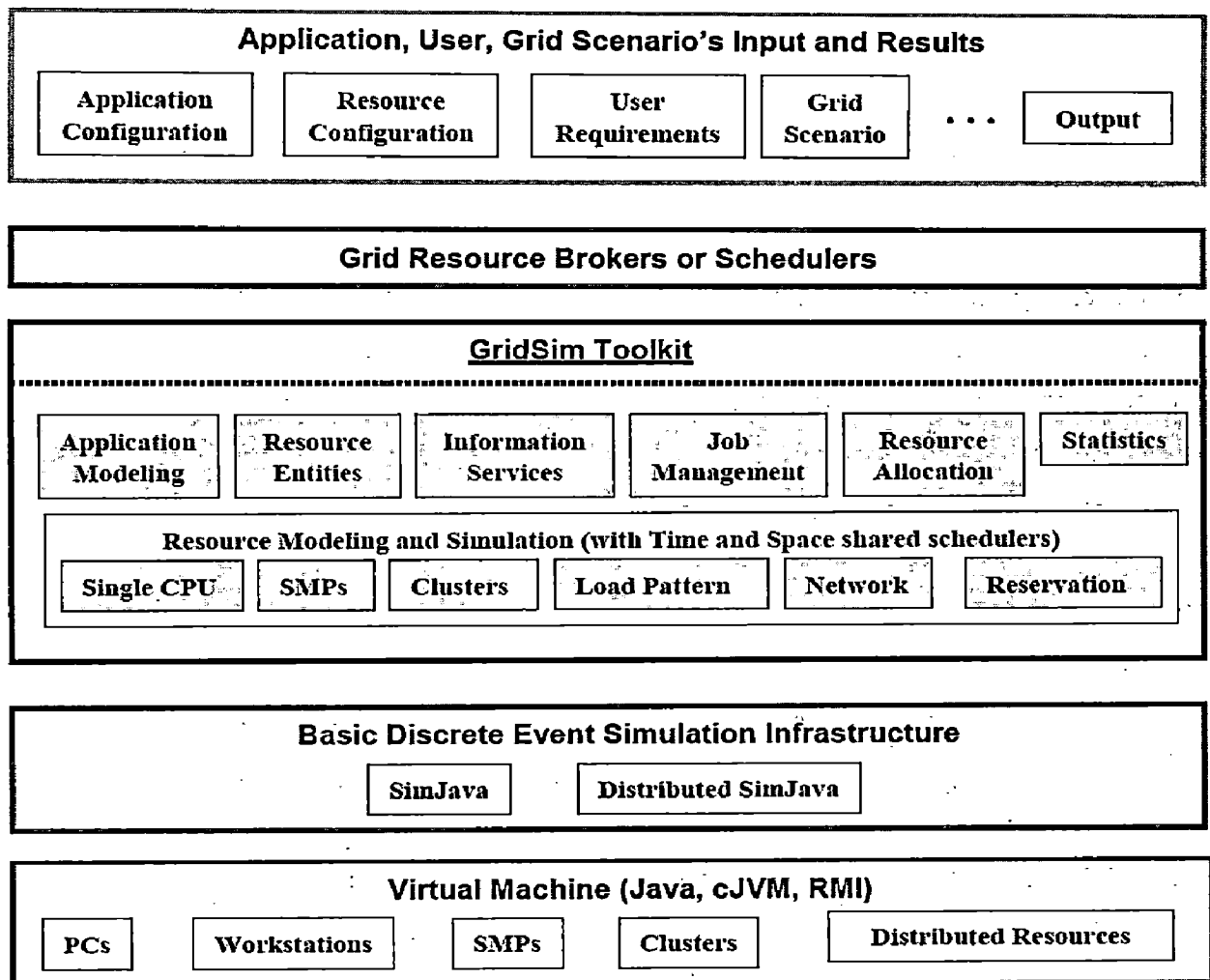


Figure A.1: A modular architecture for Gridsim platform and components.