



# **JOB SCHEDULING IN HETEROGENEOUS GRID ENVIRONMENT**



By

**Harikarthik P C**

**Reg. No: 0920108005**

of

**KUMARAGURU COLLEGE OF TECHNOLOGY**

**(An Autonomous Institution Affiliated to Anna University of Technology, Coimbatore)**

**COIMBATORE – 641 049**

**A PROJECT REPORT**

**Submitted to the**

**FACULTY OF COMPUTER SCIENCE AND ENGINEERING**

*In partial fulfillment of the requirements  
for the award of the degree  
of*

**MASTER OF ENGINEERING**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**APRIL 2011**

## BONAFIDE CERTIFICATE

Certified that this project report titled, “**Job Scheduling in Heterogeneous Grid Environment**” is the bonafide work of **HARIKARTHIK P C (0920108005)** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.



**GUIDE**

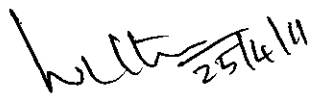
(Ms. P. DEVAKI, M.E., (Ph.D))




**HEAD OF THE DEPARTMENT**

(Ms. P. DEVAKI, M.E., (Ph.D ))

The candidate with **University Register No. 0920108005** was examined by us in Project Viva-Voce examination held on 25<sup>th</sup> April, 2011



**Internal Examiner**



**External Examiner**

**DECLARATION**

I affirm that the project work titled “**Job Scheduling in Heterogeneous Grid Environment**” being submitted in partial fulfillment for the award of **M.E** degree is the original work carried out by me. It has not formed the part of any other project work submitted for the award of any degree or diploma, either in this or any other University.



(Harikarthik P C)

(Reg. No: 0920108005)

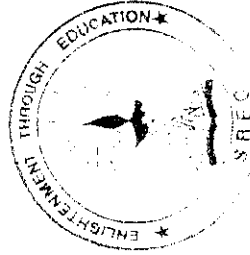
I certify that the declaration made above by the candidate is true



**Ms. P. DEVAKI, M.E., (Ph.D)**

Head of the Department,

Department of Computer Science and Engineering,  
Kumaraguru College of Technology,  
(An Autonomous Institution)  
Coimbatore-641 049.



Sri Ramakrishna Engineering College

Vattamalaipalayam, NGGO Colony (Po), Coimbatore – 641 022

Department of Computer Applications

International Conference on

Emerging Trends in Computing

17-18 March, 2011



CERTIFICATE

This is to certify that B./Mr./Ms. P. C. Hanikarthik

has participated / presented a paper entitled Weighted Gos and Ant Colony

Optimization Algorithm [WGAACO]

in ICETC-2011 held at Sri Ramakrishna Engineering College during 17 – 18 March 2011.

  
Convener

  
Director – MCA

  
Principal

Sponsored by  
SNR Sons Charitable Trust



## ACKNOWLEDGEMENT

First and foremost, I would like to thank the Almighty for enabling me to complete this project.

I whole-heartedly like to express my special thanks to our Chairman, Management, and Director for giving this great opportunity to pursue this course. I thank **Dr. S. Ramachandran**, Principal, for providing me with the necessary facilities and infrastructure to work on this project.

I would like to take this opportunity to thank **Dr.S.Thangasamy**, R&D Dean, for his everlasting support and useful suggestions throughout this project.

I would like to extend my thanks to my Project Guide **Ms. P. Devaki, M.E., (Ph.D.)** Head, Department of Computer Science and Engineering for her valuable suggestions and support right from the beginning to perform my project work extremely well.

I thank our Project Coordinators, **Ms. V.Vanitha, M.E., (Ph.D.)**, Associate Professor and **Mr.V.Subramani, M.Tech**, Associate Professor, Department of Computer Science and Engineering, for their valuable advice and guidance throughout this project.

I would like to convey my honest thanks to all **Teaching and Non-Teaching** staff members of the department for their support.

I would to extend my heartfelt thanks to all my beloved friends and my family members for their moral support and encouragements.

## ABSTRACT

Grid computing, a next leap in communication technology, a new trend in distributed computing system that enables utilization of idle resources existing worldwide, to solve data intensive and computational intensive problems. The resources may either be homogeneous or heterogeneous in nature and they are shared from multiple administrative domains. The problem is divided into independent tasks and the tasks are executed by the resources available in grid. Scheduling these tasks to various resources in a grid is a very important problem and it is NP Complete. Hence we need a good task scheduling strategy to utilize the grids effectively such that overall completion time (make span) is minimized. In literature, many heuristic approaches for scheduling are available that give near optimal solution.

In this project we propose a weighted QoS (Quality of Service) factor enabled ant colony algorithm for scheduling independent tasks on heterogeneous grid resources. The main contributions of our work are to minimize the makespan (overall completion time) with QoS satisfaction and the results are compared with hybrid genetic algorithm and other heuristic algorithms such as min-min, max-min, minimum expected time, minimum completion time.

## ஆய்வுச் சுருக்கம்

தொலைத்தொடர்பு, பகிர்செய் தொழில் நுட்பங்களின் அடுத்த கட்ட முயற்சியின் பயனே வலைச்சூழல் ஆகும். வலைச்சூழல் உலகில் பயனில்லாத நிலையில் உள்ள மூலங்களை உபயோகித்து அதீத செறிவு மிகு கணக்கிடுதல்களையும், அதீத தரவு மிகு கணக்கிடுதல்களையும் எளிதில் செய்யத்தக்கதாகும். வலைச்சூழலின் மூலங்கள் பலவகைப்பட்டதாகவும் மற்றும் மாறுபடும் தன்மையுடைய ஒரு அமைப்பாகும். வலை என்பது ஒரு வளர்ந்து வரும் கூட்டமைப்பாகும். மூலங்களைப் பங்கிடுதல் மற்றும் நிறுவனங்களின் செயல்பாடு ஒருங்கிணைப்பு போன்ற செயல்பாடுகளில் வலை முக்கிய பங்களிக்கின்றது. மூலங்களை சமன் செய்யும் கணக்கீடுகளில் வலையானது மூலங்களை தகுந்த பணிகளுக்கு அளிக்கும் வேலையில் பணிகளின் தேவைகள் மற்றும் மூலங்களின் கோட்பாடுகள் ஆகியவற்றை கருத்தில் கொண்டு செயல்படுகிறது.

இந்தப் பணி வலை மூலங்களை கையாளுதலுக்கான செயல் முறை பற்றிய விளக்கத்தை அளிப்பதற்காக கொடுக்கப்பட்டுள்ளது. இது மூலங்களை கையாளுதல் மற்றும் பங்கிடுதல் போன்ற பணிகளை செய்கின்றது. இந்தப் பங்கீடு முறையானது பணித் தேவைகளை மற்றும் மூலங்களின் இருப்பு ஆகியவற்றை கருத்தில் கொண்டு QoS சார்ந்த மரபுவழி நெறிமுறைப்படியும் செய்யப்பட்டு, அதன் திறன் கணக்கிடப்பட்டுள்ளது. இப்பங்கீடு குறுகிய காலத்தில் அனைத்து சார்பற்ற பணிகளையும் முடிக்கிறது. இதுவே இந்த ஆய்வின் முக்கிய நோக்கமாகும். இம்முறை பற்றிய பகுப்பாய்வுகளும், செய்முறைகளும் இதனை கட்டமைப்பான, கட்டமைப்பற்ற மற்றும் பகுதி கட்டமைப்பான சூழல்களிலும் இதன் செயல்திறனை உறுதி செய்கின்றன.

**TABLE OF CONTENTS**

<b>CONTENTS</b>	<b>PAGE NO.</b>
<b>ABSTRACT</b>	v
<b>ABSTRACT (TAMIL)</b>	vi
<b>LIST OF FIGURES</b>	ix
<b>LIST OF ABBREVIATIONS</b>	xi
<b>1. INTRODUCTION</b>	
1.1 Overview of Grid Computing	1
1.2 Applications of Grid Computing	13
1.3 Issues in Grid Computing	15
1.4 Job Scheduling in Grid Computing	16
<b>2. LITERATURE REVIEW</b>	
2.1 An Approach to Grid Scheduling by using Condor-G	25
2.2 Heuristic Techniques for Job Scheduling	30
2.3 Existing Algorithms	33
<b>3. METHODOLOGY</b>	
3.1 Overview of ACO	35
3.2 Architecture of the System	35
3.3 Problem Definition	37
3.4 ETC Matrix Generation	37
3.5 Weighted QoS Ant Colony Optimization Algorithm	39
3.6 Genetic Algorithm Overview	42
3.7 Hybridization of Genetic Algorithm with ACO as initial seed	44



<b>4. EXPERIMENTATION RESULTS</b>	
4.1    Makespan Graphs	47
4.2    Resource Utilization Graphs	53
<b>5. CONCLUSION AND FUTURE OUTLOOK</b>	
5.1    Conclusion	59
5.2    Future Outlook	59
<b>6. APPENDIX</b>	
6.1    Source Code	60
6.2    Snapshots	85
<b>7. REFERENCES</b>	88

## LIST OF FIGURES

FIGURE NO.	CAPTION	PAGE NO.
1.1	Grid Architecture	5
1.2	OGSA Platform Architecture	7
1.3	Web service and grid service in OGSI	9
1.4	Basic Grid Model	20
1.5	Classification of Static Task-Scheduling algorithms	24
2.1	Grid Scheduling Architecture	26
2.2	Condor Architecture	28
2.3	Condor-G Matchmaking in CRO-Grid	30
3.1	General Ant Behavior	35
3.2	System Architecture	36
3.3	Ant – Grid Mapping	36
4.1	Makespan Graphs	
4.1.1	Low Task Low Machine Heterogeneity – Consistent	47
4.1.2	Low Task Low Machine Heterogeneity – Inconsistent	47
4.1.3	Low Task Low Machine Heterogeneity – Partial Consistent	48
4.1.4	Low Task High Machine Heterogeneity – Consistent	48
4.1.5	Low Task High Machine Heterogeneity – Inconsistent	49
4.1.6	Low Task High Machine Heterogeneity – Partial Consistent	49
4.1.7	High Task Low Machine Heterogeneity – Consistent	50
4.1.8	High Task Low Machine Heterogeneity – Inconsistent	50
4.1.9	High Task Low Machine Heterogeneity – Partial Consistent	51
4.1.10	High Task High Machine Heterogeneity – Consistent	51
4.1.11	High Task High Machine Heterogeneity – Inconsistent	52
4.1.12	High Task High Machine Heterogeneity – Partial Consistent	52

4.2	Resource Utilization Graphs	
4.2.1	Low Task Low Machine Heterogeneity – Consistent	53
4.2.2	Low Task Low Machine Heterogeneity – Inconsistent	53
4.2.3	Low Task Low Machine Heterogeneity – Partial Consistent	54
4.2.4	Low Task High Machine Heterogeneity – Consistent	54
4.2.5	Low Task High Machine Heterogeneity – Inconsistent	55
4.2.6	Low Task High Machine Heterogeneity – Partial Consistent	55
4.2.7	High Task Low Machine Heterogeneity – Consistent	56
4.2.8	High Task Low Machine Heterogeneity – Inconsistent	56
4.2.9	High Task Low Machine Heterogeneity – Partial Consistent	57
4.2.10	High Task High Machine Heterogeneity – Consistent	57
4.2.11	High Task High Machine Heterogeneity – Inconsistent	58
4.2.12	High Task High Machine Heterogeneity – Partial Consistent	58

## LIST OF ABBREVIATIONS

Quality of Service	QoS
Ant Colony Optimization	ACO
Weighted QoS Ant Colony Optimization	WQACO
Genetic Algorithm	GA
Minimum Execution Time	MET
Minimum Completion Time	MCT
Pheromone Indicator	PI

# CHAPTER 1

## INTRODUCTION

### 1.1. Overview of Grid Computing

Over the past few years the popularity of the Internet has been growing by leaps and bounds. The growth of Internet along with availability of powerful computers and high speed networks as low cost commodity components is changing the way the scientists and engineers do computing and how society in general manages the information. Often called the “next big thing” in global Internet technology, Grid Computing takes collective advantage of the vast improvements in microprocessor speeds, optical communications, raw storage capacity, World Wide Web and the Internet that have occurred over the last few years . The last few years we have witnessed the emergence of Grid Computing as an innovative extension to distributed computing technology, for computing resource sharing among participants in a *virtualized* collection of organizations. These new technologies have enabled the clustering of a wide variety of geographically distributed resources such as super computers, storage systems, data sources at the exact time it is needed (on demand) for solving computation-intensive and data-intensive problems.

#### **Grid Computing**

Grid computing is a form of distributed computing that involves coordinating and sharing computing, application, data storage or network resources across dynamic and geographically dispersed organizations. Grid technologies promise to change the way organizations tackle complex computational problems.

Grid computing enables the virtualization of distributed computing and data resources such as processing, network bandwidth and storage capacity to create a single system image, granting users and applications seamless access to vast IT capabilities.

The emergence of open standards has a great influence on this computing technology, especially in providing seamless Grid interoperability and Grid integration facilities. We could find that technologies of Grid computing are still evolving; however the alignment with industry-wide open standards and the commercial interests quickly placed this technology

into a forerunning state for infrastructure and technology development. The most notable standard we have seen in the area of Grid is the Global Grid Forum's Open Grid Services Architecture (OGSA) which enables communication across heterogeneous, geographically dispersed environments.

### **What is Grid Computing?**

“Grid computing is a term referring to the combination of computer resources from multiple administrative domains to reach a common goal.”- Wikipedia

“A Grid is a collection of distributed computing resources available over a local or wide area network that appears to an end user or application as one large virtual computing system.” – IBM

“Conceptually, a grid is quite simple. It is a collection of computing resources that perform tasks. In its simplest form, a grid appears to users as a large system that provides a single point of access to powerful distributed resources.” – Sun

“Grid computing is computing as a utility – you do not care where data resides or what computer processes your requests. Analogous to the way utilities work, clients request information or computation and have it delivered – as much as they want and whenever they want.” – Oracle

The Grid can be thought of as a distributed system with non-interactive workloads that involve a large number of files. Grids are a form of distributed computing whereby a “super virtual computer” is composed of many networked loosely coupled computers acting together to perform very large tasks. “Distributed” or “grid” computing in general is a special type of parallel computing that relies on complete computers (with onboard CPUs, storage, power supplies, network interfaces, etc.) connected to a network (private, public or the Internet) by a conventional network interface, such as Ethernet. This is in contrast to the traditional notion of a supercomputer, which has many processors connected by a local high-speed computer bus.

Grid computing represents an enabling technology that permits the dynamic coupling of geographically dispersed resources for performance-oriented distributed applications in science, engineering, medicine and e-commerce. However, it is difficult task to agree on a

concrete definition of Grid Computing, as different commercial and academic implementation use the word for a fairly wide spectrum of architectures. It is generally agreed in the literature that there are two important goals which are the driving force behind grid computing.

The first goal is to build up a **computational and networking infrastructure** that is designed to provide pervasive, uniform and reliable access to data, computational and human resources distributed over wide area environments.

The second and more distant goal behind grid computing is the **delivery of computing power** as a utility, like the electrical system.

Actually the name 'Grid' comes from an analogy from power grids that supply electricity. When somebody needs electricity, he plugs in a device to the system which uses as much resources as it needs. The end user is not concerned with the details like which power plant is supplying the electricity at that moment or lack of power if he buys a hi-fi system. By analogy the home computer in the future will have only Human Computer Interface (HCI) and the computing power will be provided by the grid.

**Computational Grid** is a collection of distributed, possibly heterogeneous resources which can be used as an ensemble to execute large-scale applications. What distinguishes grid computing from conventional high performance computing systems such as cluster computing is that grids tend to be more loosely coupled, heterogeneous and geographically dispersed. Although a grid can be dedicated to a specialized application, it is more common that a single grid will be used for a variety of different purposes. Grids are often constructed with the aid of general-purpose grid software libraries known as middleware.

Grid technology has been applied to computationally intensive scientific, mathematical, and academic problems through volunteer computing, and it is used in commercial enterprises for such diverse applications as drug discovery, economic forecasting, seismic analysis, and back office data processing in support for e-commerce and Web services.

Grid applications include

- **Distributed Supercomputing**
  - Distributed Supercomputing applications couple multiple computational resources- supercomputers and workstations
- **High-Throughput Applications**
  - Grid used to schedule large numbers of independent or loosely coupled tasks with the goal of putting unused cycles to work. High-throughput applications include RSA key cracking, deletion of extra telecommunication.
- **Data-Intensive Applications**
  - Focus is on synthesizing new information from large amounts of physically distributed data. Examples include European Union Data Grid Project for real data intensive computing applications such of High Energy Physics, Biology and Medical Image Processing and Earth Observation

## **Grid construction**

There are three main issues that characterize computational grids:

**Heterogeneity:** a grid involves a multiplicity of resources that are heterogeneous in nature and might span numerous administrative domains across wide geographical distances.

**Scalability:** a grid might grow from few resources to millions.

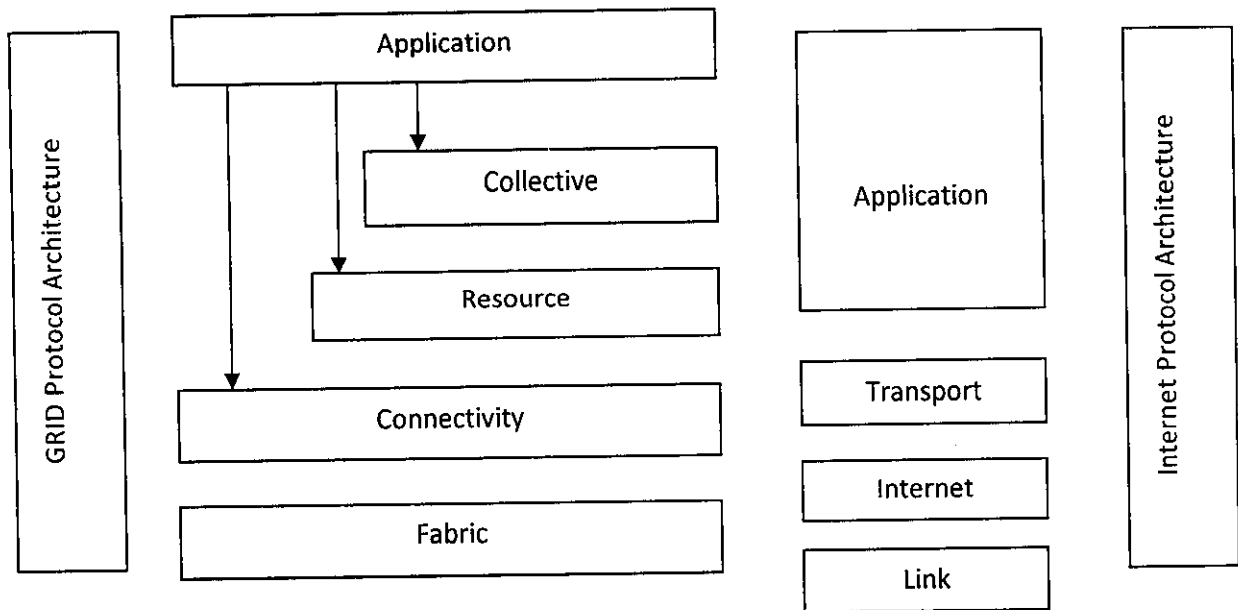
**Dynamicity or Adaptability:** with so many resources in a Grid, the probability of some resources is naturally high.

The steps necessary to realize a computational grid include:

- The integration of individual software and hardware components into a combined networked resource.
- The implementation of middleware to provide a transparent view of the resources available.



- The development of tools that allows the management and control of grid applications and infrastructure.



**Figure 1.1 Grid Architecture**

**A layered grid architecture and its relationship to the Internet protocol architecture.**

The above diagram illustrates the component layers of the architecture with specific capabilities at each layer. Each layer shares the behavior of the component layers. Each of these component layers is compared with their corresponding Internet protocol Layers, for purposes of providing more clarity in their capabilities.

**Fabric Layer: Interface to Local Resources**

This defines the resources that can be shared. This could include computational resources, data storage, networks, catalogs and other system resources. These resources can be physical resources or logical resources by nature. Example for logical resources are distributed file systems, computer clusters etc.,

Basic capabilities are

- Provide an “inquiry” mechanism whereby it allows for the discovery against its own resource capabilities, structure and state of operations.

- Provide appropriate “resource management” capabilities to control the QoS the grid solution promises or has been contracted to deliver.

### **Connectivity Layer: Manages Communications**

This defines the core communication and authentication protocol required for grid-specific networking services transactions. It includes networking transport, routing and naming. Characteristics to be considered are Single sign on, Delegation, User-Based trust relationships and Data Security.

### **Resource Layer: Sharing of a Single Resource**

This utilizes the communication and security protocol defined by the networking communications layer, to control the secure negotiation, initiation, monitoring, metering, accounting, and payment involving the sharing of operations across individual resources.

### **The Collective Layer: Coordinating Multiple Resources**

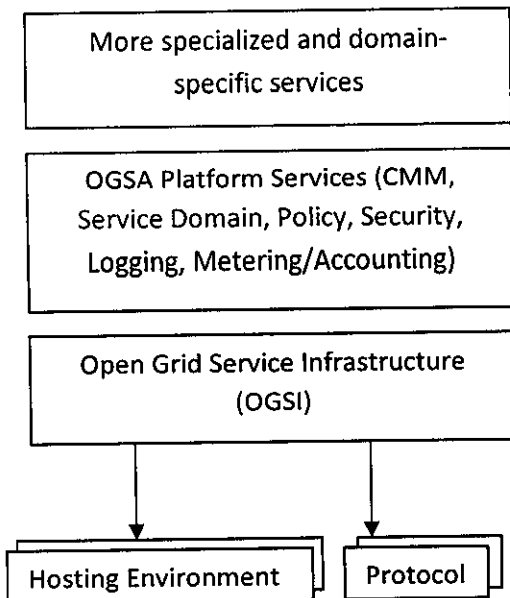
While the Resource layer manages an individual resource, the Collective layer is responsible for all global resource management and interaction with a collection of resources. Collective services are Discovery Services, Co allocation, Scheduling and Brokering Services, Monitoring and Diagnostic Services, Data Replication Services etc.,

### **Application Layer: User- Defined Grid Applications**

These are user applications, which are constructed by utilizing the services defined at each lower layer. Such an application can directly access the resource, or can access the resource through the Collective service interface APIs (Application Provider Interface)

### **OGSA Architecture and Goal:**

OGSA architecture is a layered architecture with clear separation of the functionalities at each layer. The core architecture layers are OGSi, which provides the base infrastructure and OGSA core platform services which are a set of standard services including policy, logging, service-level management and so on. The high level applications and services use these lower layer core platform components and OGSi that become part of resource sharing grid.



**Figure 1.2 OGSA Platform Architecture**

The major OGSA goals are

- Identify the use cases that can drive OGSA platform components.
- Identify and define the core OGSA platform components
- Define hosting and platform-specific bindings
- Define resource models and resource profiles with interoperable solutions

### **OGSA Basic Services**

Some of the most notable and interesting basic services of OGSA are

- Common Management Model
- Service Domains
- Distributed data access and replication
- Policy
- Security
- Provisioning and resource management

- Accounting/metering
- Common distributed logging
- Monitoring
- Scheduling

## **Open Grid Services Infrastructure (OGSI)**

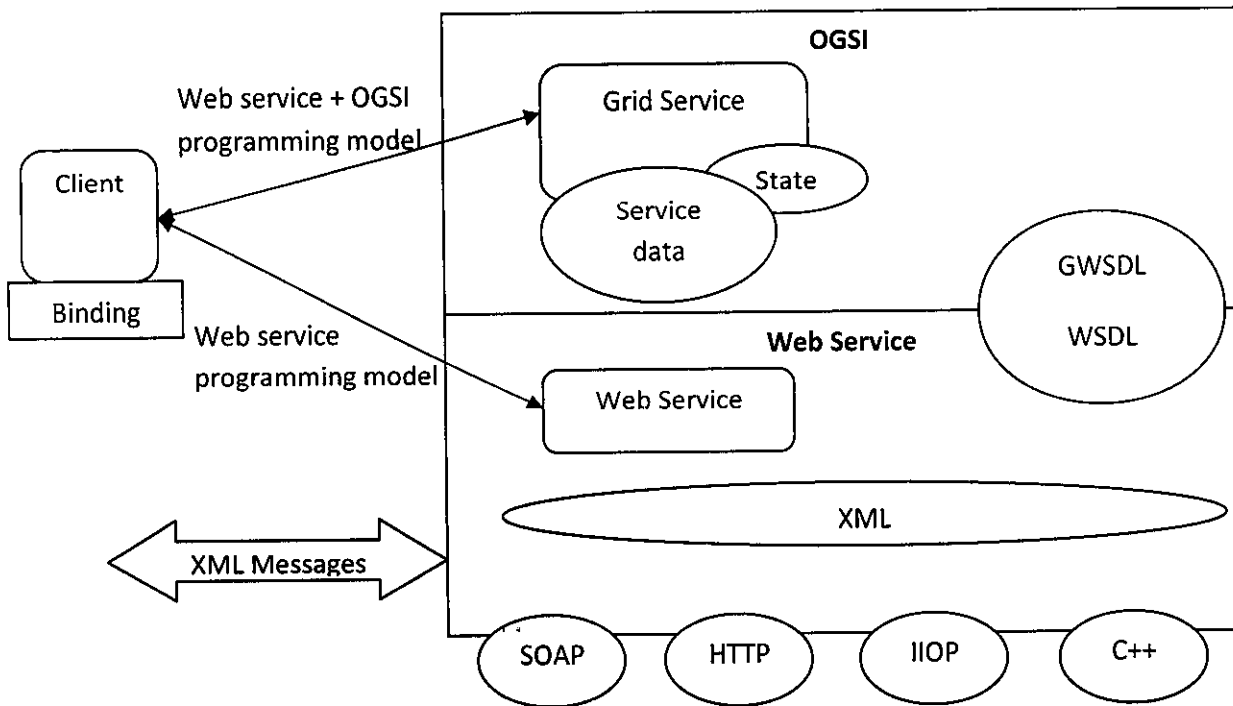
The base component of OGSA architecture is the OGSI. The OGSI is a grid software infrastructure standardisation initiative, based on emerging Web services standards that are intended to provide maximum interoperability among OGSA software components.

Based on the OGSI specification, a grid service instance is a web service that conforms to a set of conventions expressed by the WSDL as service interfaces, extensions and behaviours. A grid service provides the controlled management of the distributed and often long-lived state that is commonly required in sophisticated distributed applications. According to the definition of OGSI, every grid service is a web service; however, the converse need not be true.

The OGSI specification defines:

- How grid services instance are named and referenced
- How the interfaces and behaviours are common to all GRID services
- How to specify additional interfaces, behaviours and their extensions

The grid services specification does not address the common service hosting behaviours, including how grid services are created, managed and destroyed in a hosting environment. Rather, the specification recommends message-level interoperability, whereby any grid service client following the OGSI standard can interact with every grid service hosted by any hosting environment. This message-level interoperability is the core feature of this standard and it is achieved by using XML as the core message format and schema.



**Figure 1.3 Web service and grid service in OGSi**

The above figure introduces a number of concepts surrounding OGSi, and its relation to Web Services as

- Grid services are layered on top of web services.
- Grid services contain application state factors and provide concepts for exposing the state, which is referred to as the service data element.
- Both grid services and web services communicate with its client by exchanging XML messages.
- Grid services are described using GWSDL, which is an extension of WSDL.

### Grid Components

**Grid Fabric:** It comprises all the resources geographically distributed and accessible from anywhere on the Internet. They could be computers, clusters, 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, security, authentication, and Quality of Service (QoS) such as resource reservation and trading.

**Grid Development Environment 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 server they are accessing on a UNIX or NT platform.

## **Types of grids**

### **National-Grids**

National Grids provide a strategic “computing reserve” and will allow substantial computing resources to be applied to large problems in times of crisis, such as to plan responses to a major environmental disaster, earthquake or terrorist attack. They will act as a “national collaborators”, supporting collaborative investigations of complex scientific and engineering problems, such as global climate change, space station design and environmental cleanup.

### **Private-Grids**

Private Grids can be useful in many institutions (hospitals, corporations, small firms, etc). They are characterized by a relatively small scale, central management and common purpose and in most cases; they will probably need to integrate low-cost commodity technologies.

### **Project-Grids**

Project Grids will likely be created to meet the needs of a variety of multi-institutional research groups and multi-company “virtual teams”, to pursue short- or medium-term projects (scientific collaborations, engineering projects). A Project Grid will typically be built ad hoc from shared resources for a limited time and focus on a specific goal.

## **Goodwill-Grids**

Goodwill Grids are for anyone owning a computer at home who wants to donate some computer capacity to a good cause.

## **Peer-to-peer-Grids**

Peer-to-peer technology depends on people sharing data between their computers. The name Peer-to-peer suggests that there is no central control.

## **Consumer Grids**

In a Consumer Grid, resources are shared on a commercial basis, rather than on the basis of goodwill or mutual self-interest. A big issue in such Grids will be “resource marketing”: a user has to find the resources needed to solve his particular problem and the supplier must make potential users aware of the resources he has to offer.

## **Key benefits of the Grid computing model**

### **Consolidation**

From servers to applications to whole sites, consolidation is a key benefit of the Grid computing model, especially in the data center. Consolidation not only minimizes the infrastructure necessary to meet an enterprise’s business demands, but also reduces costs by migrating from proprietary or single-use systems to commercial off-the-shelf (COTS) –based systems that can be shared by multiple applications.

### **Modular Computing**

Modular computing, especially in the data center, minimizes and simplifies the infrastructure using building blocks that address higher density, lower power, lower thermals, simplified cabling and ease of upgrading and management. Blade servers are an excellent example of modularity.

### **Virtualization**

By creating pools of resources enabled by highly automated management capabilities, virtualization can enable an IT system administrator to utilize far more of the resources in the



data center, making the resources accessible to more than a single application sitting on a single physical server.

### **Utility Computing**

Utility Computing allows an infrastructure to be managed analogously to an electric utility, applying a pay-per-use model, thereby optimizing and balancing the computing needs of an enterprise and allowing it to run at maximum efficiency.

The following summarizes the merits of Grid computing

1. Exploiting underutilized resources
2. Parallel CPU capacity
3. Virtualization
4. Scalability
5. Reliability
6. Resource balancing
7. Management
8. Cost-effective use of computing resources
9. Reduces upfront cost
10. Pay per use (on need basis)

### **End users of Grid computing**

There are hundreds of computer grids around the world. Many grids are used for e-science: enabling projects that would be impossible without massive computing power.

- **Biologists** are using grids to simulate thousands of molecular drug candidates on their computer, aiming to find a molecule able to block specific disease proteins.
- **Earth scientists** are using grids to track ozone levels using satellites, downloading hundreds of Gigabytes of data every day (the equivalent of about 150 CDs a day).



- **High energy physicists** are using grids in their search for a better understanding of the universe, relying on a grid of tens of thousands of desktops to store and analyze the 10 Petabytes of data (equivalent to the data on about 20 million CDs!) produced by the Large Hadron Collider each year. Thousands of physicists in dozens of universities around the world want to analyse this data.
- **Engineers** are using grids to study alternative fuels, such as fusion energy.
- **Artists** are using grids to create complex animations for feature films (check out Kung Fu Panda for example).
- **Social scientists** are using grids to study the social life of bees, the makeup of our society, the secrets of history.

Grid computing not only provides the resources that allow our scientists to cope with vast collections of data, it also allows this data to be distributed all over the world, which means scientific teams can work on international projects from the comfort of their own laboratories.

Grid computing is powering science from around the globe, providing the technology to explore new ways of doing science. Scientists can now share data, data storage space, computing power, and results. Together, researchers can tackle bigger questions than ever before: from disease cures and disaster management to global warming and the mysteries of the universe.

## 1.2. Applications of Grid Computing

### Opportunities for Grid computing in Bio- and Health-Informatics

Biology provides some of the most important, as well as most complex, scientific challenges of our times. These problems include understanding the human genome, discovering the structure and functions of the proteins that the genes encode and using this information efficiently for drug design. Most of these problems are extremely intensive from a computational perspective.

One of the principal design goals for the Grid Framework is the effective logical separation of the complexities of programming a massively parallel machine from the complexities of bioinformatics computations through the definition of appropriate interfaces.

Encapsulation of the semantics of the bioinformatics computations methodologies means that the application can track the evolution of the machine architecture and explorations of various parallel decomposition schemes can take place with minimal intervention from the domain experts or the end users.

For example, understanding the physical basis of protein function is a central objective of molecular biology. Proteins function through internal motion and interaction with their environment. An understanding of protein motion at the atomic level has been pursued since the earliest simulations of their dynamics. When simulations can connect to experimental results, the microscopic examinations of the different processes (via simulation) acquire more credibility and the simulation results can then help interpret the experimental data. Improvements in computational power and simulation methods facilitated by the Grid framework could lead to important progress in studies of protein structure, thermodynamics, and kinetics.

### **Grid computing in Petroleum exploration**

Grid will become the information infrastructure of workflow in the petroleum industry. Advances in grid computing will meet the tremendous and scalable demands of petroleum E & P (Exploration & Production) for high performance computing. The petroleum industry needs the grid computing to set up the information infrastructure of workflow, enable information, data, knowledge, storage, computing power, software sharing and collaboration, etc.

Application of grid computing in the petroleum industry would bring the following effects and benefits:

- Improving the utilization of the existing resources by sharing and integrating of them
- Improving the available computing performance and supporting the new computing intensive technology widely applied reducing the processing cycle
- Improving the processing quality
- Improving the flexibility to realize the on-demand computing and form the flexible data processing service cost and price system

- Forming new operating mode and service mode

Some petroleum companies such as Royal Dutch Shell, Venezuela PDVSA, and Latin America petroleum companies have developed the research of the grid application, and have achieved some practical application effects. When the petroleum industry met some problems and challenge, the Royal Dutch Shell petroleum company carried out the research and development of the grid application to increase the data processing ability, increase the production efficiency, reduce the cost and the time of seismic data processing, realize the cooperation across the organizations in order to meet the demand of high performance power and managing the computing resource, and increase the accuracy of seismic data processing and oil reservoir modelling.

### 1.3. Issues in Grid Computing

A grid is a **distributed and heterogeneous** environment. A heterogeneous environment involves dynamic arrival of tasks where the tasks and resources can be from various administrative domains. Both of these issues require are the source of challenging design problems.

Being heterogeneous inherently contains the problem of managing multiple technologies and administrative domains. The computers that participate in a grid may have different hardware configurations, operating systems and software configurations. This makes it necessary to have right management tools for finding a suitable resource for the task and controlling the execution and data management.

A grid may also be distributed over a number of administrative domains. Two or more institutions may decide to contribute their resources to a grid. In such cases, security is a main issue. The users who submit their tasks and their data to the grid wish to make sure that their programs and data is not stolen or altered by the computer in which it is running. Of course the problem is reciprocal. The computer administrators also have to make sure that harmful programs do not arrive over the grid.

Another important issue is **scheduling**. Scheduling a task to the correct resource requires considerable effort. The picture is further complicated when we consider the need to access the data. In this project, we have assumed that the capacity of the machines and the

execution time of the tasks are known in advance and no jobs arrive dynamically. In case of a dynamic scenario, the chances of failure are high.

Grid computing environment may also involve the service level agreements (SLA) which are service based agreements rather than customer based agreements. SLA is a negotiation mechanism between resource providers and task submitting sources.

#### **1.4. Job Scheduling in Grid Computing**

The job scheduling system is responsible to select best suitable machines in a grid for user jobs. The management and scheduling system generates job schedules for each machine in the grid by taking static restrictions and dynamic parameters of jobs and machines into consideration.

##### **Job Scheduling in Grids:**

A grid scheduler should:

- Find suitable execution site(s) possibly at multiple locations, i.e., co-allocation
- Transfer the application and if required input files to the sites and run it
- Return results

In a Grid system

1. It schedules the resources for higher utilization.
2. Complex as many machines with local policies involved.
3. Resources are either fixed or resources may join or leave randomly
4. One job scheduler or two job schedulers.

##### **Types of Job Scheduling Infrastructures**

There are two different types of scheduler systems namely,

- The local scheduler and
- The global scheduler (Meta-Scheduler).

The local scheduler schedules the jobs within its own managed site. Typically, these local schedulers cannot schedule jobs to some other available sites, rather it has localized control. The most popular local schedulers are: Load Sharing Facility (LSF), the Open Portable Batch System (PBS), Sun Grid Engine (SGE) and Condor.

On the other hand Meta-Scheduler manages jobs among available sites in the grid environment. Although there are already available methods for Meta-scheduler, there is still a need to improve the scheduling techniques because the number of jobs running on grid has increased that cause the system to degrade.

Therefore, our focus is to optimize the efficiency of the Meta-Scheduler by providing more enhanced techniques. The task of the scheduler is to dynamically identify and characterize the available resources and to allocate the most appropriate resources for the given jobs. The resources are typically heterogeneous, locally administered and accessible under different local access policies and hence there is a issue in determining how to select the best site for submitting the underlying jobs and how many jobs the system should be submit each time.

Few other types of scheduler are,

**Centralized:** Single job scheduler on one instance, all information collected here.

**Hierarchical:** Two job schedulers, one at global and other at local level.

**Decentralized:** No central instance, distributed schedulers interact and commit resources.

### **Centralized Job scheduling**

#### **Multi Site Scheduling**

A job can be executed on more than one machine in parallel. As job-parts are running on different machines, latency is important, different job-parts are started synchronously on all machines.

#### **Single Site Scheduling**

A job is executed on a single parallel machine. This means that system boundaries are not crossed. Efficient as communication inside a machine is fast.

## **Advantages of centralized job scheduling**

Efficiency: The scheduler is conceptually able to produce very efficient schedules, because the central instance has all necessary information on the available resources. Centralization is useful e.g. at a computing center, where all resources are used under the same objective. Due to this fact even communication bottlenecks can be ignored.

## **Hierarchical job scheduling**

Jobs are submitted to the central scheduler; in turn jobs are submitted to low level machines. In addition, every machine uses a local job scheduler. Basically they are centralized as there is one global instance.

Main advantage of this scheduling is that different policies can be used for global and local scheduling. Meta-scheduler redirects submitted jobs to local schedulers for resources based on some policy.

## **Decentralized Scheduling**

No central instance is responsible. Distributed schedulers interact with each other and decide the allocations for each job to be performed. Information about state of all systems is not collected at a single point. Local job schedulers may have different but compatible scheduling policies.

### **Advantages of Decentralized Scheduling**

- No communication bottleneck.
- Scalable to greater extent.
- Failure of single component doesn't affect whole metasystem.
- Better fault tolerance and reliability than centralized systems which have no back-ups.
- Site-autonomy for scheduling can be achieved easily as the local schedulers can be specialized on the needs of the resource provide or the resource itself.

## Disadvantages of Decentralized scheduling

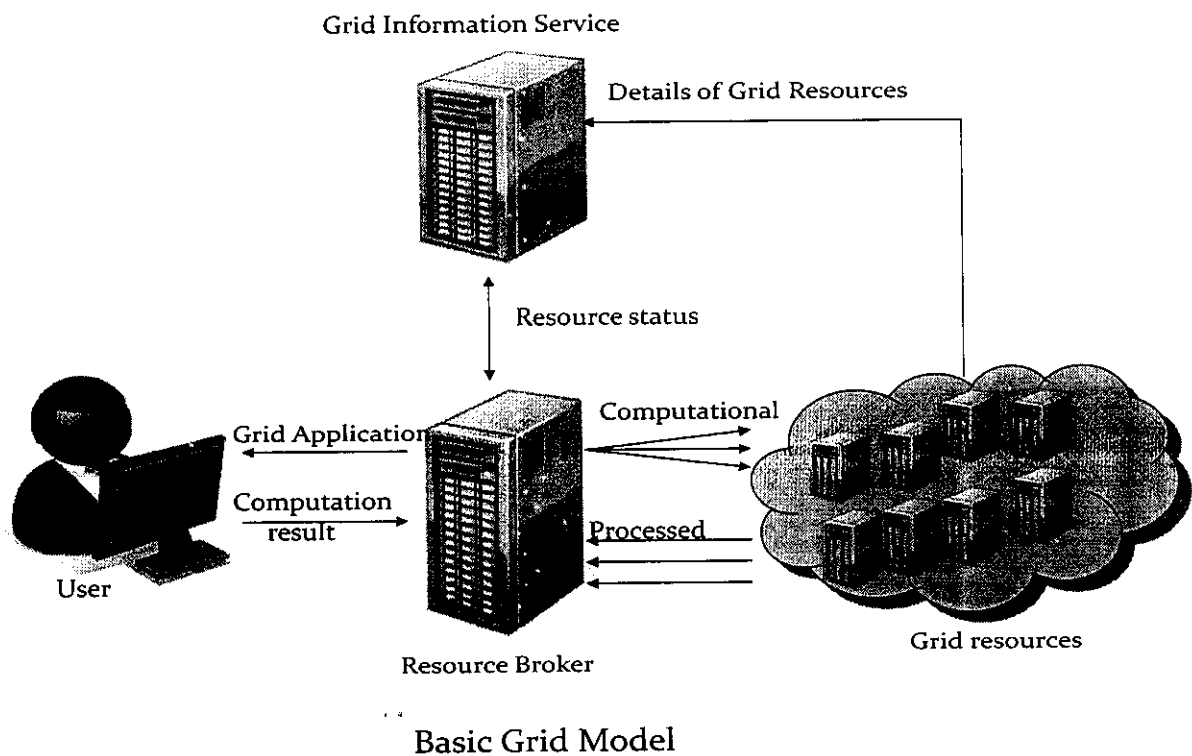
- The lack of a global scheduler, which knows all job and system information at every time instant, usually leads to sub-optimal schedules.
- The support for multi-site applications is rather difficult to achieve.
- As all parts of a parallel program must be active at the same time, the different schedulers must synchronize the jobs and guarantee simultaneous execution decentralized scheduling with direct communication.
- The local schedulers can send/receive jobs to/from other schedulers directly. Either schedulers have a list of remote schedulers they can contact or there is a directory that provides information of others systems. If a job start is not possible on the local machine immediately, the local scheduler is searching for an alternative machine.
- If a system has been found, where an immediate start is possible, the job and all its data is transferred to the other machine/scheduler.
- It can be parameterized which jobs are forwarded to another machine, this affects the local queue. This can also affect the performance of some scheduling algorithms.

Scheduling onto the Grid is NP-Complete, so there is no best scheduling algorithm for all grid computing systems. An alternative is to select an appropriate scheduling algorithm to use in a given grid environment because of the characteristics of the tasks, machines and network connectivity. Job scheduling is one of the key research area in grid computing. The goal of scheduling is to achieve highest possible system throughput and to match the application need with the available computing resources.

For running applications, resource management and job scheduling are the most crucial problems in grid computing systems. In recent years, the researchers have proposed several efficient scheduling algorithms that are used in grid computing to allocate grid resources with a special emphasis on job scheduling. With further development of grid technology, it is very likely that corporations, universities and public institutions will exploit grids to enhance their computing infrastructure.

## Basic Grid Model

The basic grid model generally composed of a number of hosts, each composed of several computational resources, which may be homogeneous or heterogeneous. The four basic building blocks of grid model are user, resource broker, Grid Information Service (GIS) and lastly resources. When user requires high speed execution, the job is submitted to the broker in grid. Broker splits the job into various tasks and distributes to several resources according to user's requirements and availability of resources. GIS keeps the status information of all resources which helps the broker for scheduling.



## Job scheduling algorithms

Job scheduling is the process of mapping jobs into specific available physical resources, trying to minimize some cost function specified by the user. This is a NP-complete problem and different heuristics may be used to reach an optimal or near optimal solution. Jobs are submitted to scheduler either in batch/offline or online mode. Effective computation and job scheduling is rapidly becoming one of the main challenges in grid computing and is seen as being vital for its success.



## **The state of the art: A review of traditional scheduling methodologies**

Grid scheduling could benefit from several traditional scheduling methodologies. These methodologies have achieved successful results in a wide range of scheduling applications. Therefore, it worth to start to investigate their performance in Grid scheduling. These methodologies are described below.

### **1.4.1. Heuristics**

A Heuristic is a technique that seeks good solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is. Dispatching rules are example of heuristics; they are used to select the next job to process on the resource whenever the resource becomes free. Dispatch rules include EDD (Earliest Due Date) and FCFS ( First Come First Served).

#### **1.4.1.1. Meta-heuristics: tabu search, simulated annealing and evolutionary algorithms**

Meta-heuristics are high-level heuristics that guide local search heuristics to escape from local optima. Meta-heuristics such as tabu search, simulated annealing and genetic algorithms improve the local search algorithms to escape local optima by either accepting worse solutions, or by generating good starting solutions for the local search in a more intelligent way than just providing random initial solutions

#### **1.4.1.2. Knowledge-based systems**

Knowledge-based systems focus on capturing the expertise or the experience of the scheduling expert and an inference mechanism is used to derive conclusions or recommendations regarding the scheduling problem.

#### **1.4.1.3. Case-based reasoning**

Case-Based Reasoning (CBR) is an artificial intelligence methodology in which a new problem is solved by reusing knowledge and experience gained in solving previous problems. A case contains a description of the problem, and its solution. Cases are stored in a case base. The CBR process is divided into four phases: retrieval of the case most similar to the new problem, reuse and revision of its solution, and inclusion of the new case in the case base.

#### 1.4.1.4. Dynamic scheduling

Dynamic scheduling is the problem of scheduling in dynamic environments. Grid scheduling systems operate in dynamic environments subject to various unforeseen and unplanned events that can happen at short notice. Such events include the breakdown of computers, arrival of new jobs, processing times are subject to stochastic variations, etc. It turns out that the performance of a schedule is very sensitive to these disturbances, and it is difficult to execute a predictive schedule generated in advance. These real-time events not only interrupt system operation but also upset the predictive schedule that was previously established. Consequently the resulting schedule may neither be feasible nor nearly optimal anymore. Dynamic scheduling is arguably of practical importance in Grid Scheduling to generate robust schedules.

#### 1.4.1.5. Fuzzy methodologies

Fuzzy systems consist of a variety of concepts and techniques for representing and inferring knowledge that is imprecise, uncertain, or unreliable. Scheduling models based on fuzzy methods have recently attracted interest among the scheduling research community. A fuzzy set is a very general concept that extends the notion of a standard set defined by a binary membership to accommodate gradual transitions through various degrees. Since the original introduction of fuzzy sets by Lotfi Zadeh in 1965, the notion has been extended to a complete framework of fuzzy methodology incorporating aspects such as fuzzy numbers, fuzzy arithmetic and fuzzy relations, and fuzzy reasoning. Previous work has investigated the representation of uncertainty in processing time and due time by fuzzy numbers, the representation of flexible constraints by fuzzy measures, fuzzy job precedence relations or machine breakdowns, but these have been in isolation.

Even once an SLA has been agreed, there are many ways in which it might need renegotiation:(compute and other) resources may fail unpredictably, sub-jobs may fail due to user error, more important (high-priority) jobs may be submitted, user-requirements might change, etc. In a busy Grid environment, SLAs would be constantly being added, altered or withdrawn, and hence scheduling would need to be a continual, dynamic and uncertain process. Some preliminary work has been carried out to examine whether fuzzy methods can be used in the evaluation of *Grid performance contract violations*, but this has been very simplistic (based on 2 fuzzy rules with 2 variables).

#### **1.4.1.6. Agents and multi-agent systems**

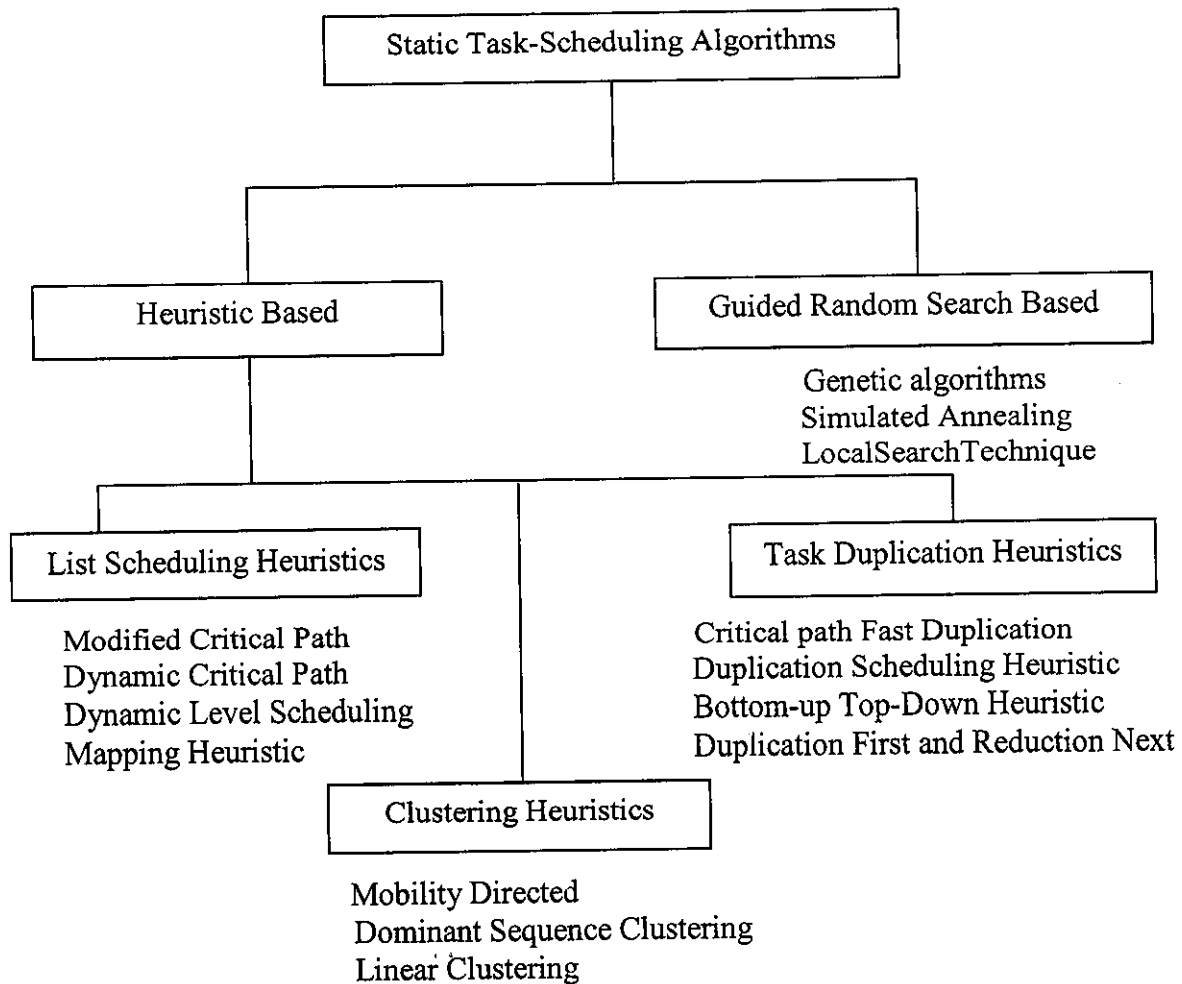
Recently, multi-agent systems are one of the most promising approaches to building complex, robust, and cost-effective next-generation manufacturing scheduling systems because of their autonomous, distributed and dynamic nature, and their robustness against failures. An agent is a computer system that is situated in some environment, and that is capable of flexible and autonomous action in this environment in order to meet its design objectives. By flexible we mean that the system must be responsive, proactive, and social. A Multi-Agent System is a system composed of a population of autonomous agents, which interact with each other to reach common objectives, while simultaneously each agent pursues individual objectives

#### **Job Scheduling in a Heterogeneous Grid Environment**

Computational grids have the potential for solving large-scale scientific problems using heterogeneous and geographically distributed resources. However, a number of major technical hurdles must be overcome before this potential can be realized. One problem that is critical to effective utilization of computational grids is the efficient scheduling of jobs.

One of the primary goals of grid computing is to share access to geographically distributed heterogeneous resources in a transparent manner. There will be many benefits when this goal is realized, including the ability to execute applications whose computational requirements exceed local resources and the reduction of job turnaround time through workload balancing across multiple computing facilities. The development of computational grids and the associated middleware has therefore been actively pursued in recent years. However, many major technical (and political) hurdles stand in the way of realizing these benefits. Although numerous researchers have proposed scheduling algorithms for parallel architectures, the problem of scheduling jobs in a heterogeneous grid environment is fundamentally different.

## Classification of Static Task-Scheduling algorithms



### Optimisation criteria

Various optimisation criteria such as

- Minimization of overall computation time
- Minimization of the maximum waiting time
- Minimization of the cost to the user
- Maximization of the profit to vendor and user
- Maximization of resource utilization
- Minimization of broken SLA's

can be taken as the objective function for scheduling jobs in grids.

## CHAPTER 2

### LITERATURE SURVEY

#### 2.1. An Approach to Grid Scheduling by Using Condor-G Matchmaking Mechanism

Emir Imamagic, Branimir Radic, Dobrisa Dobrenic. In Journal of Computing and Information Technology –CIT 14,2006, 4, pages 329-336

Grid middleware (GMW) is a set of services and protocols that enables seamless integration of resources in grid. It provides a layer of abstraction that hides differences in underlying technologies (e.g. computer clusters, storage managers, application servers). Numerous standards are being defined for grid protocols and services majority of them within Global Grid Forum (GGF) organization. Basic functionalities of grid middleware are security, information, job and data management. Most widely used solutions that provide these basic functionalities are Globus Toolkit and UNICORE.

Execution of jobs in grid consists of following activities

- Users submit their jobs described by using a description language to the grid scheduler(1)
- Scheduler uses grid middlewares' information systems to discover and evaluate resources (2)
- Once the scheduler has defined where the job will be executed, execution is started and managed by using available gridmiddleware components (e.g. job and data management systems) (3).

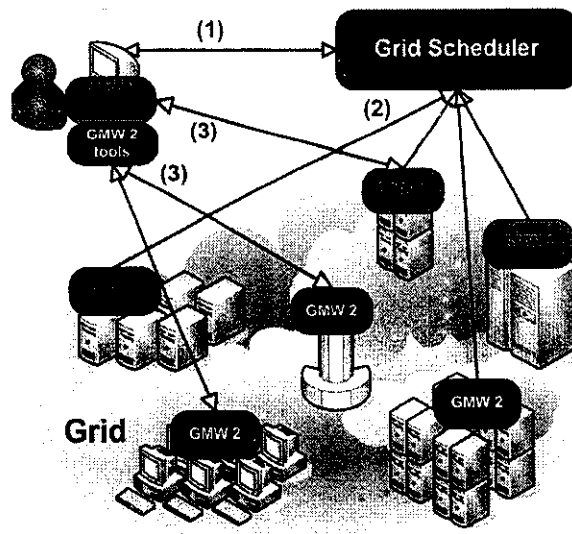


Figure 2.1 Grid Scheduling Architecture

### Grid scheduling issues

Grid scheduling differentiates from classical cluster batch scheduling in many ways. In case of grid, a scheduler does not have full control over resources, information about resources is usually unreliable and stale, application behaviour is difficult to predict. Due to the complexity of grid environment, grid scheduling raises numerous issues.

Today, there are many **grid middlewares** that provide basic set of functionalities. Some of them are implemented in accordance with grid standards (e.g. Globus Toolkit 4.0) and some have become de facto standards (e.g. UNICORE and Globus Toolkit 2.4). Grid scheduling system should be able to utilize as many grid middlewares as possible.

### Few existing solutions for particular subsystems

- Security management: Globus Grid Security Infrastructure (GSI), UNICORE Security, Virtual Organization Membership Service (VOMS)
- Information systems: Globus Monitoring and Discovery System (MDS) 2 and MDS 4, Ganglia, Network Weather Service.
- Job management: Globus Grid Resource Allocation and Management (GRAM) 2 and GRAM), UNICORE, Condor
- Data management: GlobusGridFTP and Reliable FileTransfer (RFT) , UNICORE File Management.

Ability to refresh proxy certificates in environments that utilize Globus Toolkit as grid middleware is preferred. The most widely used solution to this problem is using MyProxy credential repository.

Grid scheduler should support various *job types*. Basic types are serial and parallel jobs. Serial job is an application that demands single processor for execution. Parallel job requires more than one processor for execution (typically MPI applications). Some of the complex job types are job arrays and workflows. Job array comprises multiple executions of the same job (usually initialized with different parameters) and the workflow is a set of dependent tasks.

The functionalities of a job management system that should be supported are

- *Check pointing* – ability to store job's current state, which can be used afterwards to reconstruct the job.
- *Pre-emption* – ability to evict one job from one resource, in favour of another with higher priority.
- *Job migration* – ability to dynamically move active job from one resource to another.
- *Rescheduling* jobs on alternative resources. This ability is important in cases when job does not execute properly on specific resources.
- *Fault tolerance* – ability to automatically recover from job or resource failure.
- *Advance reservation* of resources for a specific period of time.

A scheduler should be capable of assigning *priorities* to jobs and resources. In addition, resource owners should be able to define which jobs they prefer. In the same way, users should be able to rank the resources and request specific features (e.g. libraries or hardware capabilities). Users should be able to define *custom scheduling algorithms*. Besides support for custom algorithm development, most common algorithms implementations should be provided.

Scheduling system should make use of local job management system's *advance reservations* mechanism. By using advance reservations, grid execution system enables users to run their applications in explicitly defined timeframe. In addition, by using advance

reservations, parallel applications distributed over multiple clusters can be guaranteed synchronous start-up of processes on all clusters.

A scheduler should take into account data location and movement (*dataaware scheduling*) and characteristics of network links. An example of data-aware scheduling is assigning jobs to resources closer to the data instead of moving large data over network. System *scalability* is very important due to the nature of grid system. For example, grid scheduling system should be capable of handling continuous and massive load. Condor-G satisfy most of the features

## Condor

Condor is a distributed environment developed at the University of Wisconsin, designed for High Throughput Computing (HTC) and CPU harvesting. CPU harvesting is a process of exploiting nondedicated computers (e.g. desktop computers) when they are not used. Condor architecture as shown below.

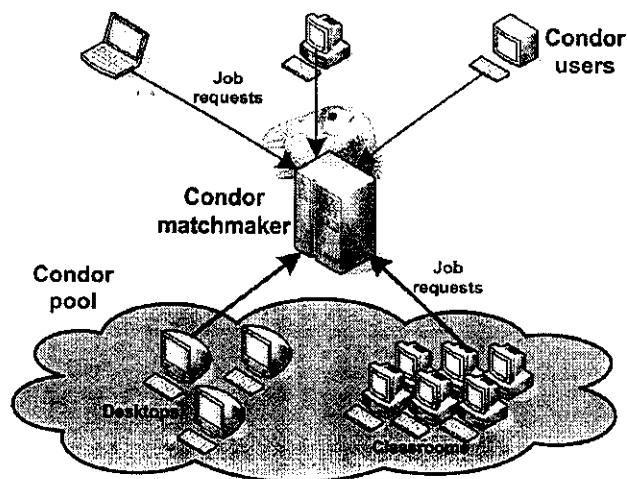


Figure 2.2 Condor architecture

Heart of the system is Condor Matchmaker. Users describe their applications with ClassAds language and submit them to Matchmaker. ClassAds allows users to define custom attributes for resources and jobs. On the other side, resources publish information to Matchmaker. Condor Matchmaker then matches job requests with available resources. Condor provides numerous advanced functionalities, such as job arrays and workflows support, checkpointing, job migration, rescheduling, fault recovery. It enables users to define



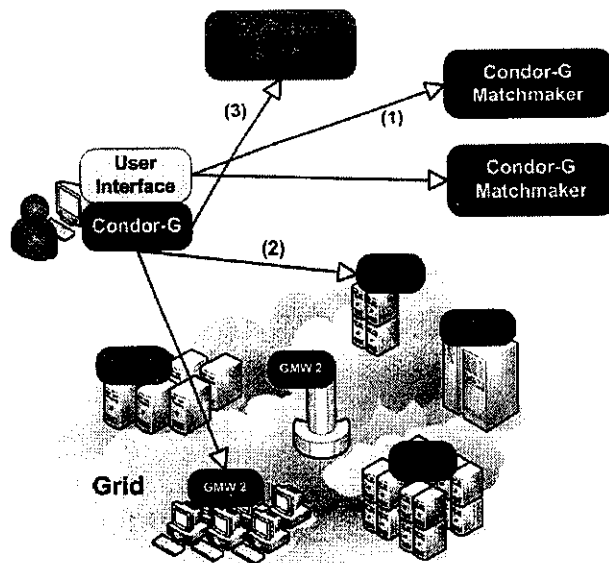
resource requirements and rank resources and mechanism of transferring files to/from remote machines.

Although utilized for integration of resources, Condor is intended to be used in smaller scale environments and should be seen more as a local job management system than a grid middleware.

### **Condor-G**

Condor-G is the Condor extension that enables using Condor tools for submitting jobs to grid. Currently supported gridmiddlewares are: Globus Toolkit versions 2.4, 3 and 4, UNICORE and NorduGrid. Additionally, Condor-G can use MyProxy server for storing user credentials in case of long running jobs. However, Condor-G cannot schedule jobs; it simply executes the job on remote resource by using grid middleware. Condor-G Matchmaking mechanism extends Condor-G with the ability to use matchmaking algorithm to schedule jobs to grid. Condor-G Matchmaking inherits almost all advantages of standard Condor system and Condor-G. It enables job scheduling based on complex job requirements. For example, a user can easily define specific requirements, such as storage space required, required libraries, etc. In addition, a user can prefer specific set of resources to others, by giving them a higher rank. Furthermore, using multiple Condor-G Matchmakers are supported. This feature enables distribution of workload, which is important for achieving greater scalability of the system.

One of the disadvantages of Condor-G Matchmaking is lack of support for parallel jobs. In current version users can submit parallel jobs, however Condor-G Matchmaker will not be aware of jobs parallelism. It will simply assume that it is dealing with serial jobs. Furthermore, there is no implicit data-aware scheduling, although users can explicitly define resources, closer to input data are preferred. Also, in current version, it is not possible to define custom scheduling algorithm. Another issue is that Condor-G Matchmaking lacks integration with grid information systems. In order to use Condor-G Matchmaking, one has to develop a custom system that will provide information about resources to Condor-G Matchmaker.



**Figure 2.3 Condor-G matchmaking in CRO-Grid (Croatian Grid)**

1. The user describes the job with ClassAds as in standard Condor system and uses Condor tools to submit it.
2. Condor-G matchmaker assigns set of grid resources to the job for execution. Condor-G utilizes appropriate underlying grid middleware components to execute the job.
3. If needed, Condor-G enables refreshing of user's certificate by using MyProxy server. Our User interface component enables the user to retrieve additional, grid-specific information about the job.

## 2.2. Heuristic Techniques for Job Scheduling

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. Job scheduling in computational grids is a multi-objective optimization problem.

## Scheduling Algorithms

**Genetic algorithms (GA)** maintain a pool of solutions rather than just one. The process of finding superior solutions mimics that of evolution, with solutions being combined or mutated to alter the pool of solutions, with solutions of inferior quality being discarded.

**Simulated annealing (SA)** is a related global optimization technique which traverses the search space by generating neighboring solutions of the current solution. A superior neighbor is always accepted. An inferior neighbor is accepted probabilistically based on the difference in quality and a temperature parameter. The temperature parameter is modified as the algorithm progresses to alter the nature of the search.

**Tabu search (TS)** is similar to simulated annealing in that both traverse the solution space by testing mutations of an individual solution. While simulated annealing generates only one mutated solution, tabu search generates many mutated solutions and moves to the solution with the lowest fitness of those generated. To prevent cycling and encourage greater movement through the solution space, a tabu list is maintained of partial or complete solutions. It is forbidden to move to a solution that contains elements of the tabu list, which is updated as the solution traverses the solution space.

**Artificial immune system (AIS)** algorithms are modeled on vertebrate immune systems. It is a source of constant inspiration to various computing systems. It is concerned with abstracting the structure and function of the immune system to computational systems, and investigating the application of these systems towards solving computational problems from mathematics, engineering, and information technology

**Particle Swarm Optimisation (PSO)**, a Swarm intelligence method. Particle Swarm Optimization in its basic form is best suited for continuous a variable, that is the objective function can be evaluated for even the tiniest increment. The method has been adapted as a binary PSO to also optimize binary variables which take only one of two values.

**Ant Colony Optimization (ACO)**, a Swarm intelligence method. Ant Colony optimization algorithms have been applied to many combinatorial optimization problems, ranging from quadratic assignment to fold protein or routing vehicles and a lot of derived methods have been adapted to dynamic problems in real variables, stochastic problems, multi-targets and

parallel implementations. It has also been used to produce near-optimal solutions to the travelling salesman problem.

### **ACO for Grid scheduling**

Ant Colony algorithm is a new heuristic algorithm; it is based on the behaviour of ants. When the blind insects, such as ants look for food, every moving ant lays the pheromone on the path, then the pheromone on the shorter path will be increased quickly, the quantity of the pheromone on every path will effect the possibility of other ants to select path. At last all the ants will choose the shortest path.

Ant algorithm has been successfully used to solve many NP-Problems. The algorithm has inherent parallelism and we can validate its scalability. So it's obvious that ant colony algorithm is suitable to be used in Grid computing task scheduling. The factors that affects the state of resources can be described by pheromone and we can get the predictive results very simple and quickly.

#### **Steps in ACO:**

1. The initial pheromone value of each resource for each job is equal to the pheromone indicator. The pheromone indicator of each resource for each job is calculated by adding the estimated transmission time and execution time of a given job when assigned to this resource.
2. The estimated transmission time can be easily determined by  $M_i/\text{Bandwidth}_j$  where  $M_i$  is the size of a given job<sub>i</sub> and  $\text{Bandwidth}_j$  is the bandwidth available between the scheduler and the resource.
3. The other parameter, job execution time, is hard to predict. Depending on the type of programs, many methods can be used to estimate the program execution time. The method used here is generation of Expected Time to Compute (ETC) matrix,  $E[i,j]$  with the pheromone indicator defined by

$$PI_{ij} = [M_i/\text{Bandwidth}_j + T_i/\text{CPU\_speed}_i]^{-1}$$

Where  $PI_{ij}$  is the pheromone indicator for job  $i$  assigned to resource  $j$ ,

$M_i$  is the size of a given job  $i$ ,

$T_i$  is the CPU time needed of job  $i$ ,

$CPU\_speed_j$  and  $Bandwidth_j$  are the status of the resource.

4. The pheromone indicator tells that when a job is assigned to a resource, we consider the resource status, the size of the jobs and the program execution time in order to select a suitable resource for execution. The larger the value of  $PI_{ij}$  is, the more efficient it is for resource  $j$  to execute this job  $i$ .

Assume that there are  $n$  resources and  $m$  jobs. We have the PI matrix as follows

$$PI = \begin{pmatrix} PI_{11} & PI_{12} & \dots & PI_{1n} \\ \cdot & \cdot & \dots & \cdot \\ PI_{m1} & PI_{m2} & \dots & PI_{mn} \end{pmatrix}$$

5. In each iteration, we need to select the largest entry from the matrix. Assuming  $PI_{ij}$  is selected, then job  $i$  assigned to a resource  $j$ ; we apply the formula to the resource selected for each unassigned jobs in the PI matrix. This step is done to recalculate the entire PI matrix. When a job is completed we apply the formula along with which we multiply  $(1-P_j)$  further where  $1 > P_j = 0$

Ant colony optimization algorithms have been applied to many combinatorial optimization problems, ranging from quadratic assignment to fold protein or routing vehicles and a lot of derived methods have been adapted to dynamic problems in real variables, stochastic problems, multi-targets and parallel implementations. It has also been used to produce near-optimal solutions to the travelling salesman problem.

They have an advantage over simulated annealing and genetic algorithm approaches of similar problems when the graph may change dynamically; the ant colony algorithm can be run continuously and adapt to changes in real time. This is of interest in network routing and urban transportation systems.

### 2.3 Existing Algorithms

There are various algorithms proposed in literature for scheduling jobs in grids such as, Min-Min, Max-Min, MCT (Minimum Completion Time) and MET (Minimum Execution Time) etc.

### **2.3.1. MET (Minimum Execution Time)**

MET assigns each task to the resource that performs it in the least amount of execution time, no matter whether this resource is available or not at that time. This heuristic can cause a severe load imbalance across the resources. However, this is one of the heuristics that is implemented in SmartNet.

### **2.3.2. MCT (minimum Completion Time)**

MCT assigns each task to the resource which obtains earliest completion time for that task. This causes some tasks to be assigned to resources that do not have minimum execution time for them. This heuristic is also implemented in SmartNet.

### **2.3.3. Min – Min algorithm**

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 corresponding 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.

### **2.3.4 Max- Min algorithm**

Max-Min is very similar to Min-Min, except in phase 2. Max-Min assigns task with maximum expected completion time to the corresponding resource, in phase 2.

# CHAPTER 3

## METHODOLOGY

### 3.1. Overview of ACO

#### General ant behaviour:

The ants in an ant colony go in search of food. It secretes a pheromone fluid in its path. When any one of the ants finds the food resource, the other ants follow the ant's path by its pheromone. When another ant finds another path which is shorter than this path, more pheromone is secreted in that path than any other path and every ant follows that shorter path and the pheromone in the other parts gets evaporated.

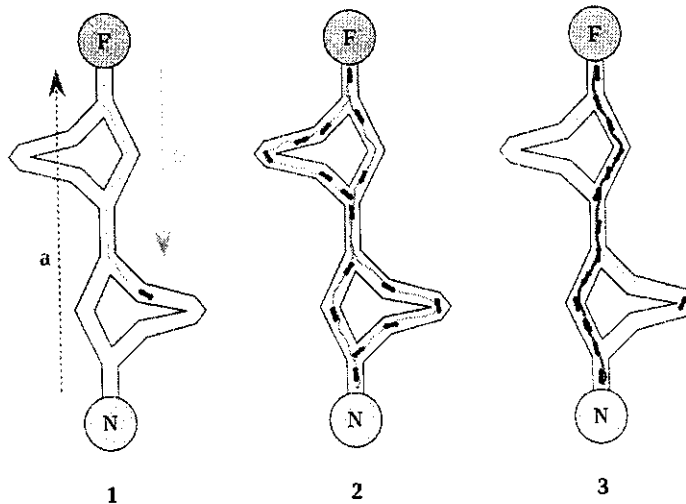


Figure 3.1 General ant behaviour

### 3.2 Architecture of the System

The clients use the portal interface for job execution. The Network Weather Service reports system information to the Information server periodically. The job scheduler selects the most appropriate resources to execute the request according to the proposed ACO algorithm. Finally the results will be sent back to the user.

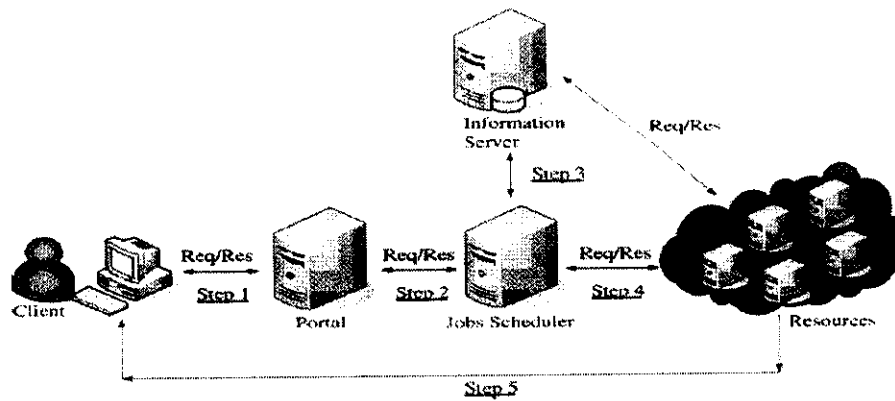


Figure 3.2 System Architecture

**Ant system –Grid System Mapping:**

In order to map the ant system to the grid system

1. An ant - an ant in the ant system is a job in the grid system
2. Pheromone - pheromone value on a path in the ant system is equivalent for a weight for the resource in the grid system

A resource with a larger weight value means the resources has a better computing power. The scheduler collects data from the information server and uses the data to calculate a weight value of resource. The (weight) of each resource is stored in the scheduler and the scheduler uses it as a parameter for ACO algorithm and sends the job to the selected resource.

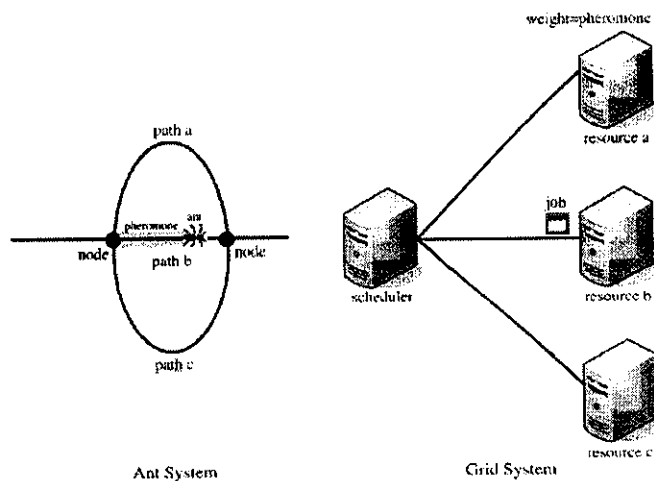


Figure 3.3 Ant – Grid Mapping



### 3.3. Problem Definition

Given a set of tasks( $n$ ) and a set of heterogeneous machines( $m$ ) such that( $m < n$ ), the main objective is to use ACO approach to allocate tasks to the machines based on the QoS satisfaction parameters, so that the overall completion time (makespan) is minimized and the resource or the machine utilization is maximized.

### 3.4. ETC Matrix Generation

It is assumed that an accurate estimate of the expected execution time for each task on each resource is known prior to execution and contained within an Expected Time to Compute (ETC) matrix. One row of the ETC matrix contains the estimated execution times for a given task on each machine. Similarly, one column of the ETC matrix consists of the estimated execution times of a given machine for each task in the meta-task. Thus, for an arbitrary task  $t$ , and an arbitrary machine  $m$ ,  $ETC(t_i, m)$  is the estimated execution time of  $t_i$  on  $m$ .

For cases when inter-machine communications are required.  $ETC(t_i, m_j)$  could be assumed to include the time to move the executables and data associated with task  $t$ , from their known source to machine  $m$ . For cases when it is impossible to execute task  $t$ , on machine  $m_j$  (e.g., if specialized hardware is needed), the value of  $ETC(t_i, m)$  can be set to infinity, or some other arbitrary value. For this study, it is assumed that there are inter-task communication each task it can execute on each machine, and estimated expected execution time of each task on each machine following method are known. The assumption that these estimated expected execution times are known is commonly made when studying mapping heuristics for HC systems.

For the simulation studies, characteristics of the ETC matrices were varied in an attempt to represent a range of possible HC environments. The ETC matrices used were generated using the following method. Initially, a  $t \times 1$  baseline column vector,  $W$ , of floating point values is created. The baseline column vector is generated by repeatedly selecting random numbers  $x_w^i$  and multiplying them by a constant 'a' letting  $W(i) = (x_w^i \times a)$  for  $0 = i < t$ . Next, the rows of the ETC matrix are constructed. Each element  $ETC(t_i, m_j)$  in row  $i$  of the ETC matrix is created by taking the baseline value,  $W(i)$ , and multiplying it by a vector  $X(j)$ . The vector  $X(j) = (x_r^j \times b)$  is created similar to the way  $W(i)$  is created. Each row  $i$  of the

ETC matrix can then be described as  $ETC(t_i, m_j) = B(i) \times X(j)$  for  $0 = j < m$ . (The baseline column itself does not appear in the final ETC matrix). This process is repeated for each row until the  $t \times m$  ETC matrix is full.

The variation along a column of an ETC matrix is referred to as the task heterogeneity. This is the degree to which the task execution times vary for a given machine. Task heterogeneity was varied by changing the value of constant 'a' used to multiply the elements of vector  $W(i)$ . The variation along a row is referred to as the machine heterogeneity; this is the degree to which the machine execution times vary for a given task [4]. Machine heterogeneity was varied by changing the value of constant 'b' used to multiply the elements of vector  $X(j)$ . The ranges were chosen in such a way that there is less variability across execution times for different tasks on a given machine than the execution time for a single task across different machines.

To further vary the ETC matrix in an attempt to capture more aspects of realistic mapping situations. Different ETC matrix consistencies were used. An ETC matrix is said to be consistent if whenever a machine  $m_j$  executes any task  $t_i$  faster than machine  $m_k$ , then machine  $m_j$  executes all the task faster than  $m_k$ . Consistent matrices were generated by sorting each row of the ETC matrix independently, with machine  $m_0$  always being the fastest and machine  $m_{(m-1)}$  the slowest. In contrast: inconsistent matrices characterize the situation where machine  $m_j$  may be faster than the machine  $m_k$  for some tasks, may be slower for others. These matrices are left in the unordered, random state in which they were generated (i.e., no consistence is enforced). Partially-consistent matrices are inconsistent matrices that include a consistent sub matrix. For the partially-consistent matrices used here, the row elements in column positions  $\{0,2,4,\dots\}$  of row  $I$  are extracted sorted, and replaced in order, while the row elements in column positions  $\{1,3,5,\dots\}$  remain unordered (i.e., the even columns are consistent and odd columns are in general inconsistent).

A system's machine heterogeneity is based on a combination of the machine heterogeneities for all tasks (rows). A system comprised mainly of workstations of similar capabilities can be said to have "low" machine heterogeneity. A system consisting of diversely capable machines, e.g., a collection of SMP's, workstations, and supercomputers, may be said to have "high" machine heterogeneity. A system's task heterogeneity is based on a combination of the task heterogeneities for all machines (columns). "High" task

heterogeneity may occur when the computational needs of the tasks vary greatly, e.g., when both time-consuming simulations and fast compilations of small programs are performed.

“Low” task heterogeneity may typically be seen in the jobs submitted by users solving problems of similar complexity (and hence have similar Execution times on a given machine). Based on the above idea, four categories were proposed for the ETC matrix in : (a) high task heterogeneity and high machine heterogeneity, (b) high task heterogeneity and low machine heterogeneity, (c) low task heterogeneity and high machine heterogeneity, and (d) low task Heterogeneity and low machine heterogeneity.

**Sample ETC Matrix (for 8 Tasks and 8 Machines – Low Task Low Machine Heterogeneity Inconsistent)**

1.097707	2.989389	3.004404	0.68733	2.280924	2.081497	2.415987	0.738158
0.642505	1.749737	1.758525	0.402305	1.335061	1.218333	1.414115	0.432056
1.013353	2.759668	2.773529	0.634512	2.105646	1.921543	2.230329	0.681434
3.517587	9.579454	9.627568	2.20254	7.30919	6.670126	7.741994	2.365418
0.162561	0.442702	0.444925	0.101787	0.337784	0.308251	0.357786	0.109315
1.55419	4.232531	4.253789	0.973158	3.22945	2.94709	3.420678	1.045122
1.74766	4.759408	4.783312	1.094299	3.631461	3.313952	3.846493	1.175222
3.570314	9.723048	9.771883	2.235556	7.418752	6.770109	7.858045	2.400874

**3.5. Weighted QoS Ant Colony Optimisation algorithm (WQACO)**

WQACO inherits the basic ideas from ACO algorithm to minimise makespan of jobs in grid environment and it also considers the job allocation to a particular resource satisfies the QoS factors such as cost, RAM and deadline.

**3.5.1. Formulation of WQACO**

Generally, the WQACO is described as follows. There are n tasks to be processed by m machines where  $n > m$ . The following assumptions are made

- All jobs are independent of each other and no priorities are among them.

- All machines and jobs are simultaneously available at the initial time
- One machine can only process an operation of a job at the same time and the processing cannot be interrupted before an operation is completed.
- The transportation time of a job from one machine to another is negligible and the machine setup time for an operation is included in its processing time

### 3.5.2. WQACO Definition and Methodology

The pheromone indicator of the ant system represents the weight of resource in the grid system, which is the capability of the resource; the value represents the QoS satisfaction of the resource. A resource with a larger weight value means that the resource has better QoS satisfaction.

The scheduler or resource broker collects data from GIS or the information server and uses the data to calculate the weight value of a resource. The pheromone (weight) value is stored in the scheduler and the scheduler uses it as the parameter of WQACO algorithm. At last, the scheduler selects a resource by scheduling algorithm and sends the task to the selected resource.

Let us assume  $n$  tasks are scheduled in  $m$  machines where  $n > m$ . The ETC (Expected Time to Compute) matrix is of form,  $n \times m$ , where  $ETC_{ij}$  represents accurate estimate of expected execution time for each task <sub>$i$</sub>  on machine <sub>$j$</sub> .

Machine Capability matrix for each machine <sub>$j$</sub> , indicates the QoS factors associated with machine <sub>$j$</sub>  for each job <sub>$i$</sub> . Let  $k$  indicates the number of QoS factors. Hence the machine capability matrix for a machine <sub>$j$</sub>  will be of the order  $n \times k$

$$(\text{MachineCapability}_{ik})_j = \begin{pmatrix} QoS_{11} & QoS_{12} & \dots & QoS_{1k} \\ \vdots & \vdots & & \vdots \\ QoS_{n1} & QoS_{n2} & \dots & QoS_{nk} \end{pmatrix}$$

For each machine the  $\text{MachineCapability}_{ik}$  values are calculated. Job requirements matrix represents the user requirements of QoS factors for executing the particular job.

$$\text{JobRequirements}_{ik} = \begin{pmatrix} QoS_{11} & QoS_{12} & \dots & QoS_{1k} \\ \vdots & \vdots & & \vdots \\ QoS_{n1} & QoS_{n2} & \dots & QoS_{nk} \end{pmatrix}$$

Since we manipulate multiple QoS factors, we assign a guided probability value for each QoS factor. Hence,

$$w = \langle w_1, \dots, w_k \rangle \quad 0 \leq w_k \leq 1 \quad \sum_{i=1}^k w_i = 1$$

We introduce *satisfy operator*  $\bowtie$ .  $R_j \bowtie J_i$  means that the resource  $R_j$  can satisfy the job  $J_i$  and guarantees QoS parameters.

$$R_j \bowtie J_i = \sum_{l=1}^k \frac{QoS^{JobRequirements}_l}{QoS^{MachineCapability}_j} \times w_l \geq 1$$

( $k$  = the number of QoS parameters) ----- 1

If all the QoS factors are satisfied then, the initial pheromone indicator value is calculated based on the QoS factor values in machine capability and Job requirements matrix and guided probability value as,

$$PI_{ij} = \sum_{l=1}^k \left( \frac{QoS^{JobRequirements}_l}{QoS^{MachineCapability}_j} \times w_l \right)^* \frac{1}{ETC_{ij} + MachineAvailability_j} \quad \text{----- 2}$$

Where  $PI_{ij}$  indicates the pheromone indicator value for job<sub>*i*</sub> assigned to machine<sub>*j*</sub>.  $JobRequirements_{il}$  represents the value of user expectation of QoS factor<sub>*l*</sub> (Cost, RAM and deadline) for each job<sub>*i*</sub>.  $(MachineCapability_{jl})_k$  represents the value of machine capability matrix, for the particular machine *j*, the  $MachineCapability_{jl}$  indicates the QoS factor<sub>*l*</sub> of machine for executing job<sub>*i*</sub>. Hence for *m* jobs and *n* machines, PI matrix will be of the order (*n* x *m*).

In each iteration we select the largest entry from the matrix, for instance if  $PI_{11}$  is the largest entry in the matrix and if no other job<sub>*i*</sub> is allocated to the machine<sub>*1*</sub> already, then the job *i* is allocated to machine *j* else the job is allocated to the next machine that has the next highest pheromone value. After a job is allocated to a machine a local pheromone (column) update is made. The local pheromone update is given by the formula,

$$PI_{ij} = (1-\rho) PI_{ij} + \rho \cdot 1/N_r \quad \text{----- 3}$$

Where  $PI_{ij}$  indicates the PI values on the machine *j* where the job is allocated.  $\rho$  indicates the evaporation rate of the pheromone ( $0 < \rho < 1$ ),  $N_r$  indicates the number of resources or

machines in the grid. Local update is made in order avoid same machine to be over loaded with multiple jobs and to avoid stagnation.

After the job is completed on the machine, global update is made to the entire PI matrix. The PI matrix is modified with an update function based on the availability value of the machine<sub>j</sub> after completing the job<sub>i</sub>,

$$PI_{\text{modified}} = (1 - \alpha) PI_{\text{initial}} * 1 / \text{MachineAvailability}_j \quad \text{-- 4}$$

### 3.5.3. Algorithm

1. *Input:* ETC matrix of size  $n \times m$

*Job Requirement matrix, Machine capability matrix for K QoS parameters*

2. *Method:* First check for the QoS Requirement satisfaction for all K QoS parameters using the equation 1.

*Calculate the initial pheromone value using the equation 2*

*Select the machine which has highest pheromone value and QoS satisfied Assign that machine to the corresponding job.*

*Update the local pheromone value using the equation 3*

*After completion of the job update the global pheromone value using the equation 4*

3. *Repeat the step 2 until all jobs are assigned*

4. *Calculate the makespan*

5. *End*

### 3.6. Overview of Genetic Algorithm

Genetic algorithms are a part of evolutionary computing, which is a rapidly growing area of artificial intelligence. GAs are excellent for all tasks requiring optimization and is highly effective in any situation where many inputs (variables) interact to produce a large number of possible outputs (solutions). It can quickly scan a vast solution set. Genetic algorithms are a class of search techniques inspired from the biological process of evolution by means of natural selection. GA is an iterative procedure that consists of a constant-size population of individuals, each one represented by a finite string of symbols, known as the genome, encoding a possible solution in a given problem space. This space, referred to as the search space, comprises all possible solutions to the problem at hand. Generally speaking, the genetic algorithm is applied to spaces which are too large to be exhaustively searched.

A genetic algorithm (GA) is an iterative search procedure widely used in solving optimization problems, motivated by biological models of evolution. In each iteration, a population of candidate solutions is maintained. Genetic operators such as mutation and crossover are applied to evolve the solutions and to find the good solutions that have a high probability to survive for the next iteration.

Start with a set of possible solutions (represented by chromosomes) the population. Solutions from one population are taken and used to form a new population. This is motivated by a hope that the new population will be better than the old one. New solutions (offspring) are selected according to their fitness – the more suitable they are the more chances they have to reproduce by mating (crossover). Repeat the cycle until some condition is satisfied.

### 3.6.1. Steps in Genetic Algorithm

- Generate a random population of  $n$  chromosomes which are suitable solutions.
- Establish a method to evaluate the fitness  $f(x)$  of each chromosome  $x$  in the population
- Create a new population by repeating the following steps until the new population is complete
  - - **Selection** – select from the population according to some fitness scheme.
    - **Crossover** – New offspring formed by a crossover with the parents.
    - **Mutation** – With a mutation probability mutate new offspring at each locus (position in chromosome).
- Use the newly generated population for a further run of algorithm

### 3.7. Hybridization of Genetic Algorithm with ACO as initial seed

In our hybrid genetic algorithm, the steps followed are as

#### 3.7.1. Chromosome Presentation

Chromosome presentation or Order Vector (OV) [11] of length equal to  $n$ , (The number of tasks to be scheduled) composed of collection of genes. The efficiency of GA depends on chromosome presentation. In this paper each gene represents the job $_i$ , executed on resource $_j$  and the corresponding ETC $_{ij}$

$$OV[i] = j$$

#### 3.7.2. Population Initialization, cross over and mutation

Initial seeds are generated based on the WQACO and Max-Min heuristic. Single point cross over process is applied with the probability  $\mu_c=0.9$  for the initial seeds in order to get the global minimum makespan. The resulting chromosomes undergoes mutation process with probability  $\mu_m = 0.05$  in order to avoid locally minimum makespan.

#### 3.7.3. Selection with Fitness Function

For the selection process, we introduce a criterion called QoS factors satisfaction. Resource Capability matrix for each resource $_j$ , indicates the QoS factors associated with resource $_j$  for each job $_i$ . Hence the resource capability matrix for a resource $_j$  is given by,

$$(\text{ResourceCapability})_{ij} = \begin{pmatrix} QoS_{11} & QoS_{12} & QoS_{1k} \\ \vdots & \vdots & \vdots \\ QoS_{n1} & QoS_{n2} & QoS_{nk} \end{pmatrix} \quad -- 1$$

$n$  indicates the number of jobs and  $k$  indicates the number of QoS factors

Job requirements matrix represents the user requirements of QoS factors for executing the particular job.

$$\text{JobRequirements}_{ij} = \begin{pmatrix} QoS_{11} & QoS_{12} & QoS_{1k} \\ \vdots & \vdots & \vdots \\ QoS_{n1} & QoS_{n2} & QoS_{nk} \end{pmatrix} \quad -- 2$$

A resource is said to be satisfying a QoS Parameter for a particular job only when,



$$\frac{QoSResourceCapability_{ij}}{QoSJobRequirements_{ij}} > 1 \quad -- 3$$

Let w indicates the guided probability value for each QoS factor,

$$w = \langle w_1 \dots w_k \rangle > 0 \leq w_k \leq 1 \quad \sum_{i=1}^k w_i = 1 \quad --4$$

The QoS Satisfaction matrix will be generated based on the formula,

$$QoSSatisfaction_{ij} = \begin{cases} \sum_{l=1}^k \left( \frac{QoSTaskRequirements_{il}}{QoSMachineCapability_{jl}} \times w_l \right) \\ \text{(if eqn 3 is satisfied for all the QoS factors)} \\ 0 \quad \text{Otherwise} \end{cases} \quad -- 5$$

In each chromosome, for each job<sub>i</sub> we identify the resource<sub>j</sub> that has highest QoS satisfaction and replace the gene value for that job<sub>i</sub> pointing to that particular resource<sub>j</sub> and the makespan is calculated.

Fitness function is given by,

$$f(x) = \frac{1}{makespan} \quad -- 6$$

The top 2 chromosomes that have the highest f(x) are given as the initial seeds to the next step in the evolutionary process until the chromosomes converge. After this evolutionary process, the makespan and resource utilisation factor for WQACO and Hybrid GA were calculated and the results are plotted in the graph.

### 3.7.4. Hybrid GA algorithm

1. *Input:* ETC matrix of size n x m, Job Requirement matrix, Resource capability matrix for k QoS parameters, initial seed (orderVector) of size n from Min-Min and Min-Max algorithm.

Constants: cross over probability  $\mu_c=0.9$ , mutation probability  $\mu_m=0.05$

2. *Method:*

*Step1:* Perform crossover process to the initial seeds based on the crossover probability value.

*Step2:* Perform mutation process to each of the seeds thus obtained from step1 based on the mutation probability value. After mutation process, each of the seed undergoes selection and fitness function

*Step3:* For the selection criterion, generate the QoS satisfaction matrix as given in the equation 5. From the QoS satisfaction matrix for each job<sub>i</sub> identify a resource<sub>j</sub> that has the highest QoS satisfaction value and allocate the particular job<sub>i</sub> to that resource<sub>j</sub>. If all the resources didn't satisfy QoS for a particular job, then we assign that job to the resource that has the lowest ETC for that job. For each seed the makespan is calculated.

*Step4:* Fitness function,  $f(x)$  as given in equation 6 is calculated for each seed and the top 2 order vectors will be given to the step1 as a initial seed and the process is repeated until the seeds converges.

*Step5:* calculate the makespan.

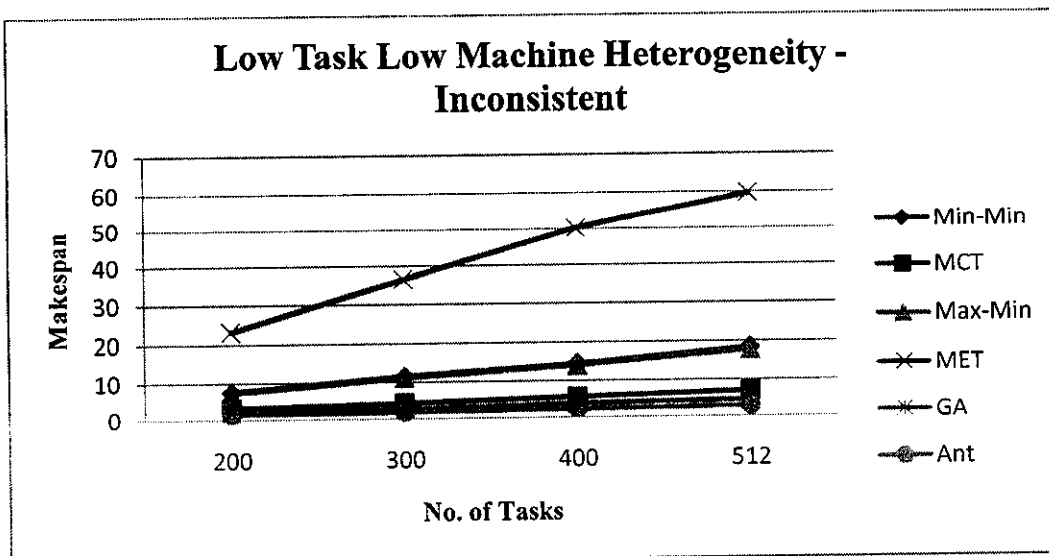
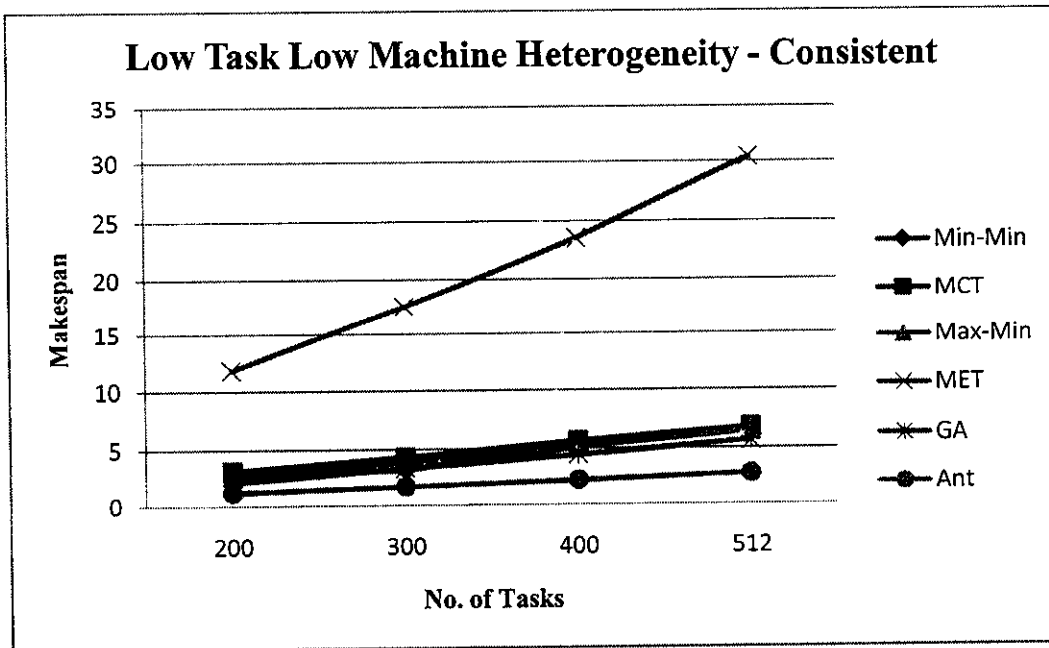
*Step6:* End

## CHAPTER 4

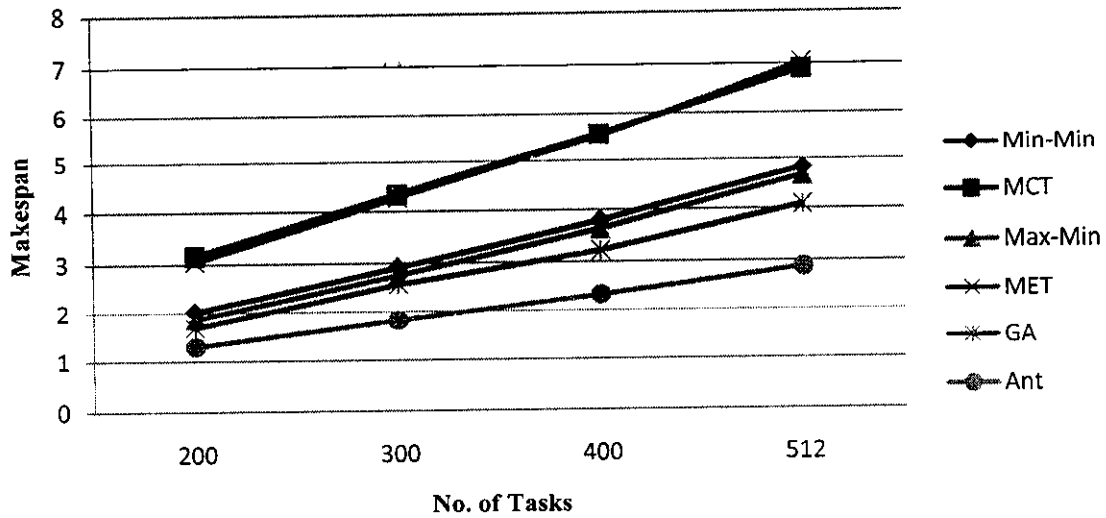
### EXPERIMENTATION RESULTS

The experimentation is done with 12 different ETCs based on the combinations of heterogeneity of task (low, high), heterogeneity of machines (low, high) and consistency (consistent, inconsistent and partial) and are repeated by varying the number of tasks such as 200,300,400 etc and the makespan and the resource utilization factors are calculated for each of them and they are plotted in the graph as shown below.

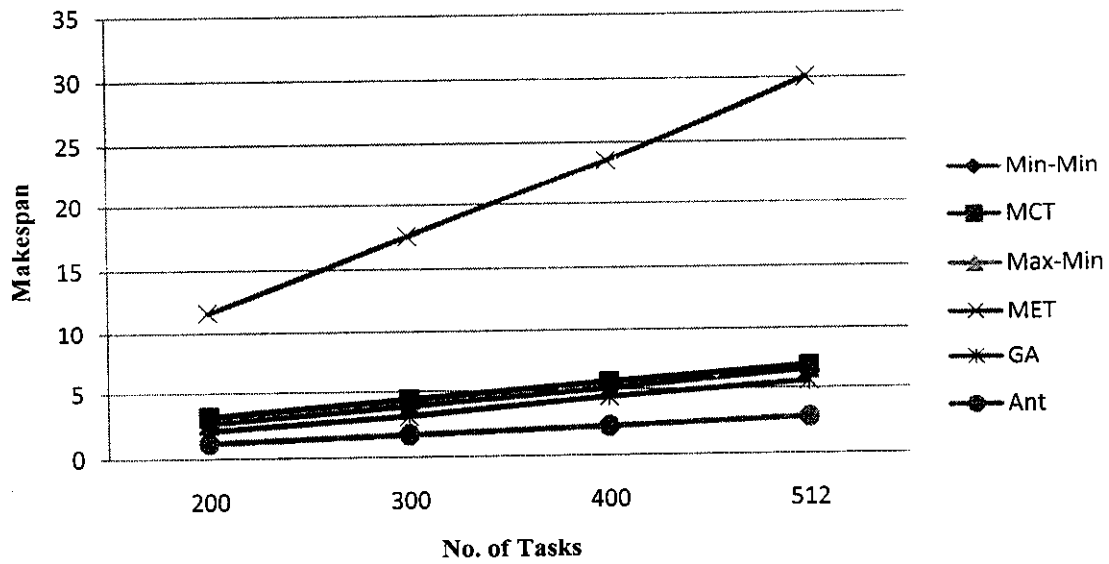
#### Makespan Graphs:



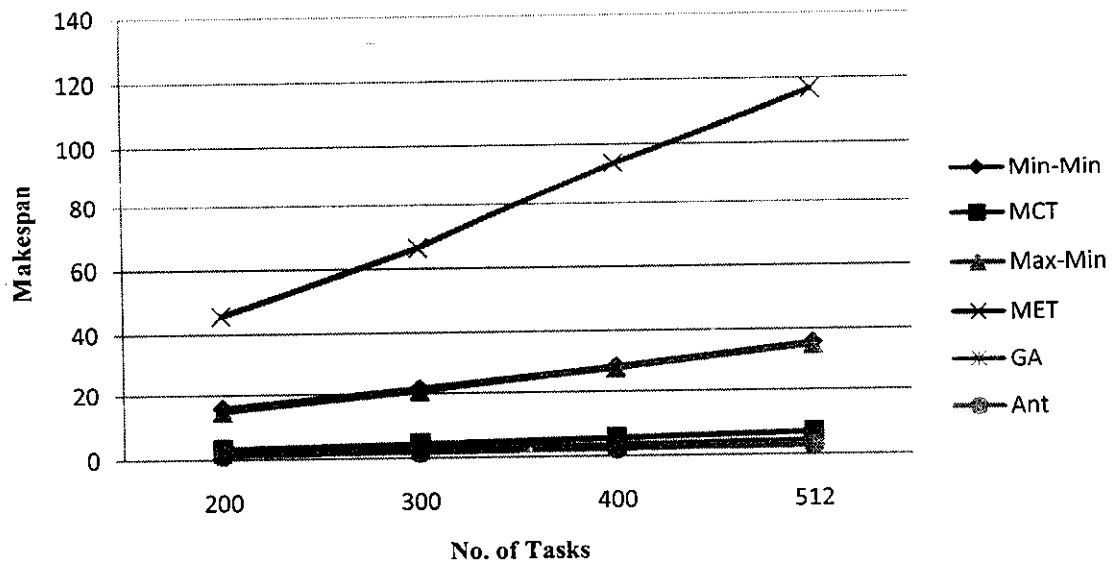
**Low Task Low Machine Heterogeneity - Partially Consistent**



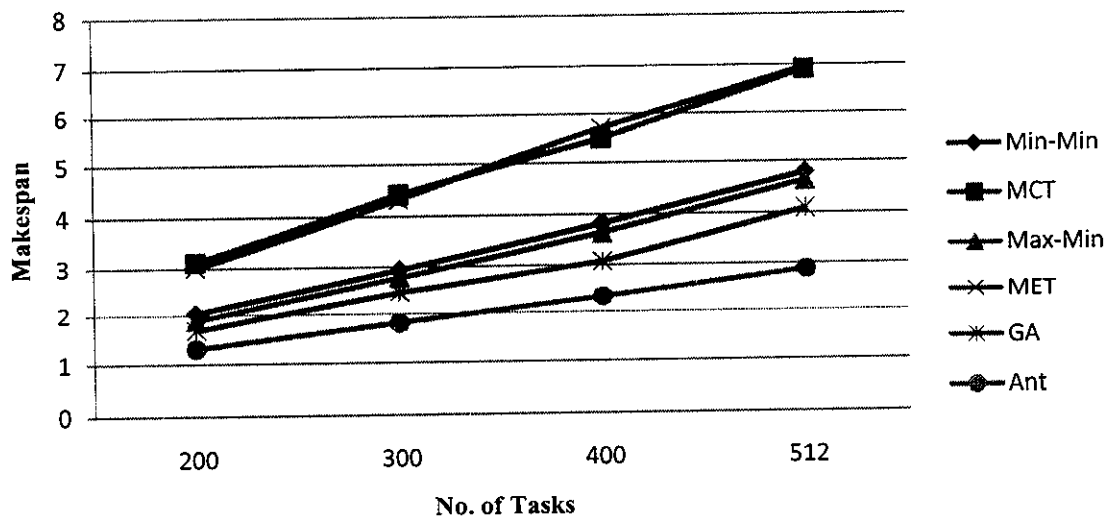
**Low Task High Machine Heterogeneity - Consistent**



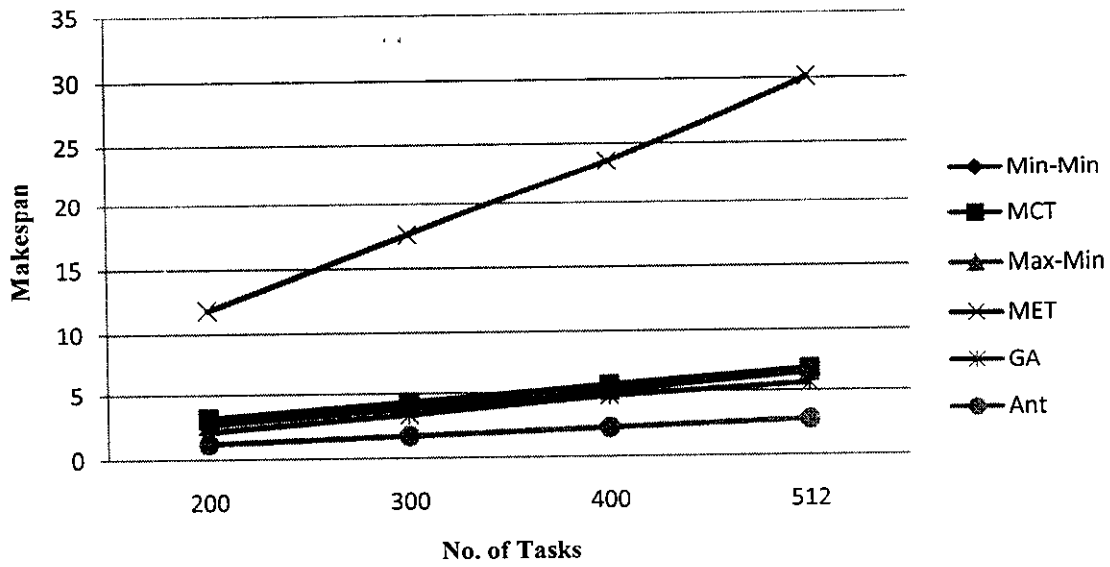
**Low Task High Machine Heterogeneity - Inconsistent**



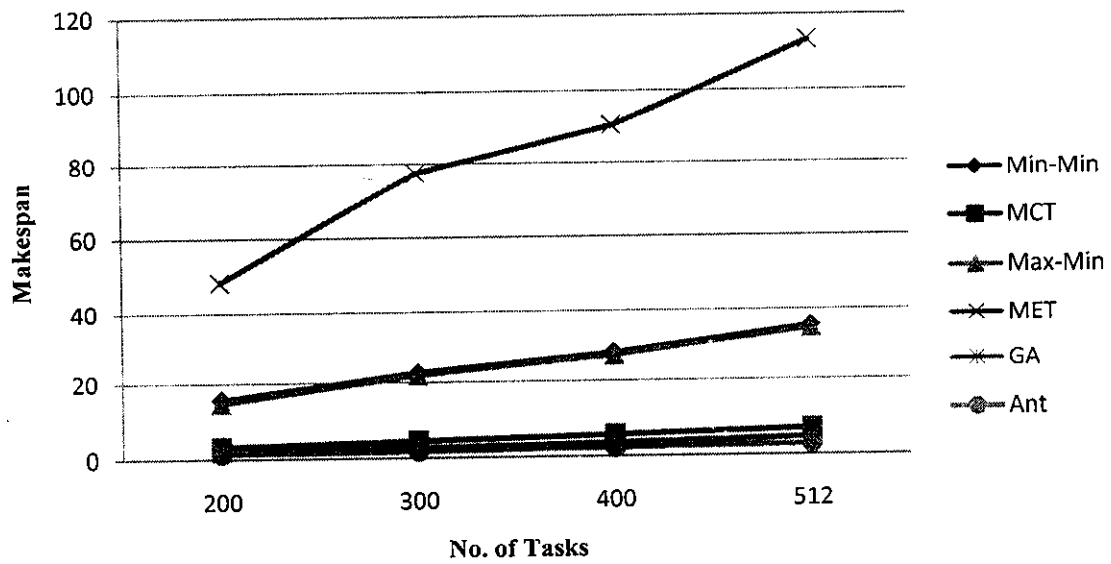
**Low Task High Machine Heterogeneity - Partial Consistent**



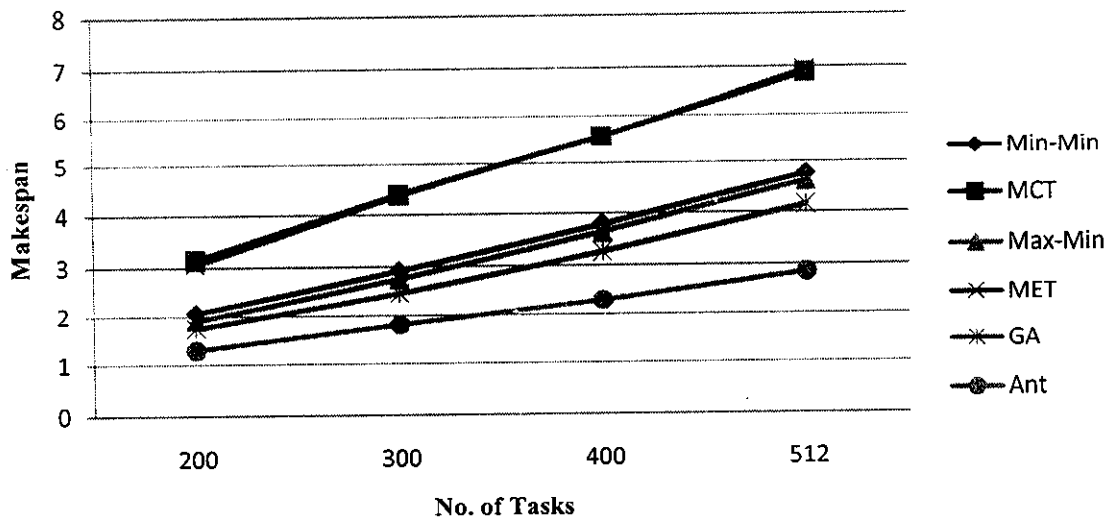
### High Task Low Machine Heterogeneity - Consistent



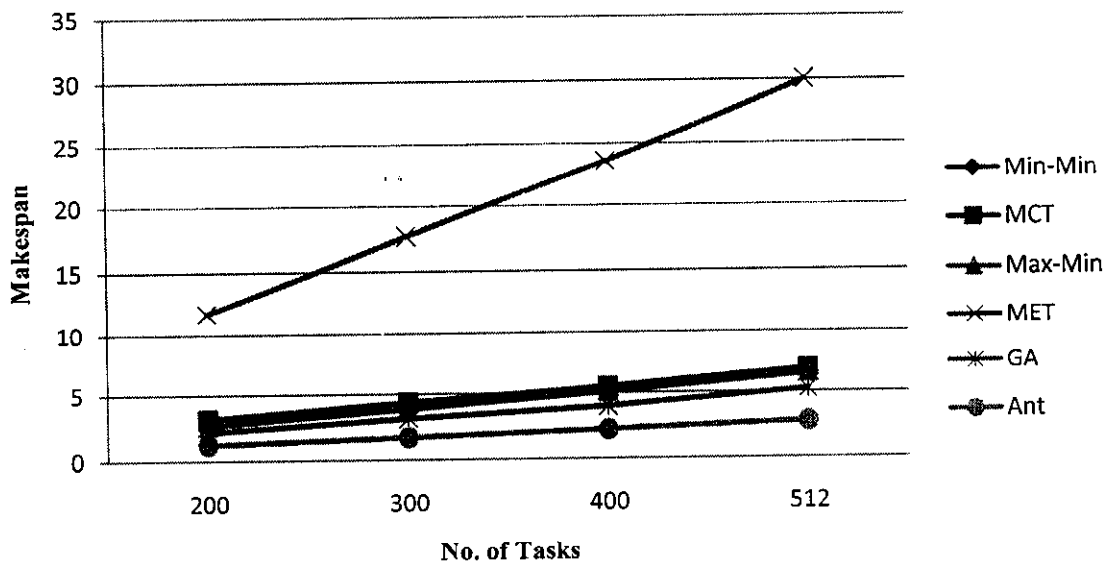
### High Task Low Machine Heterogeneity - Inconsistent



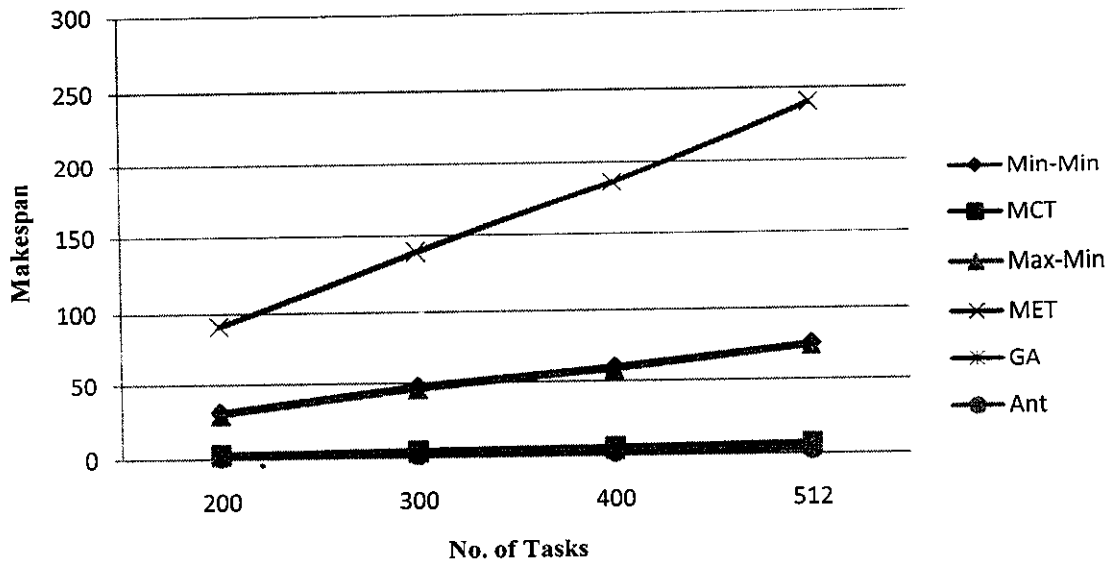
### High Task Low Machine Heterogeneity - Partial Consistent



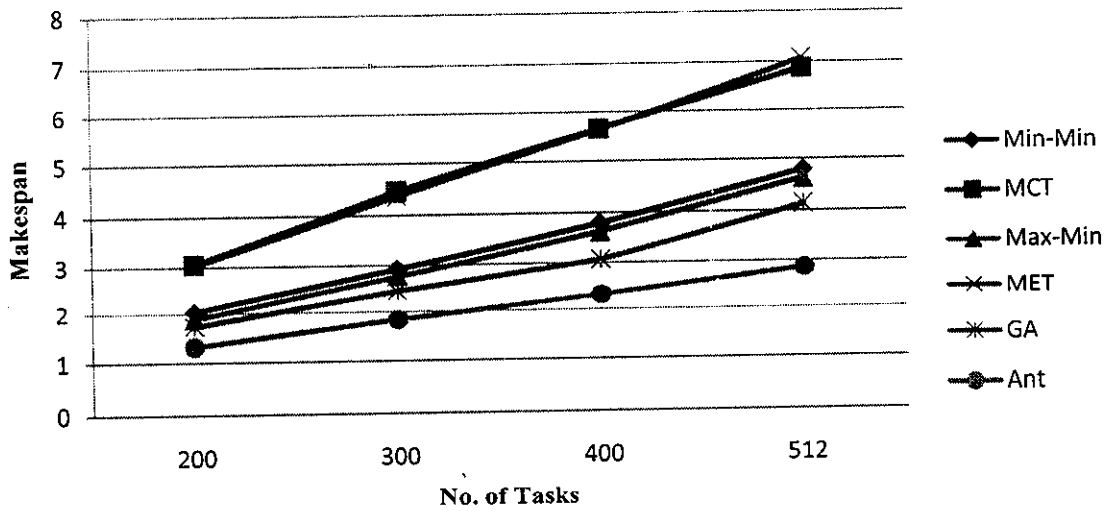
### High Task High Machine Heterogeneity - Consistent



**High Task High Machine Heterogeneity -Inconsistent**

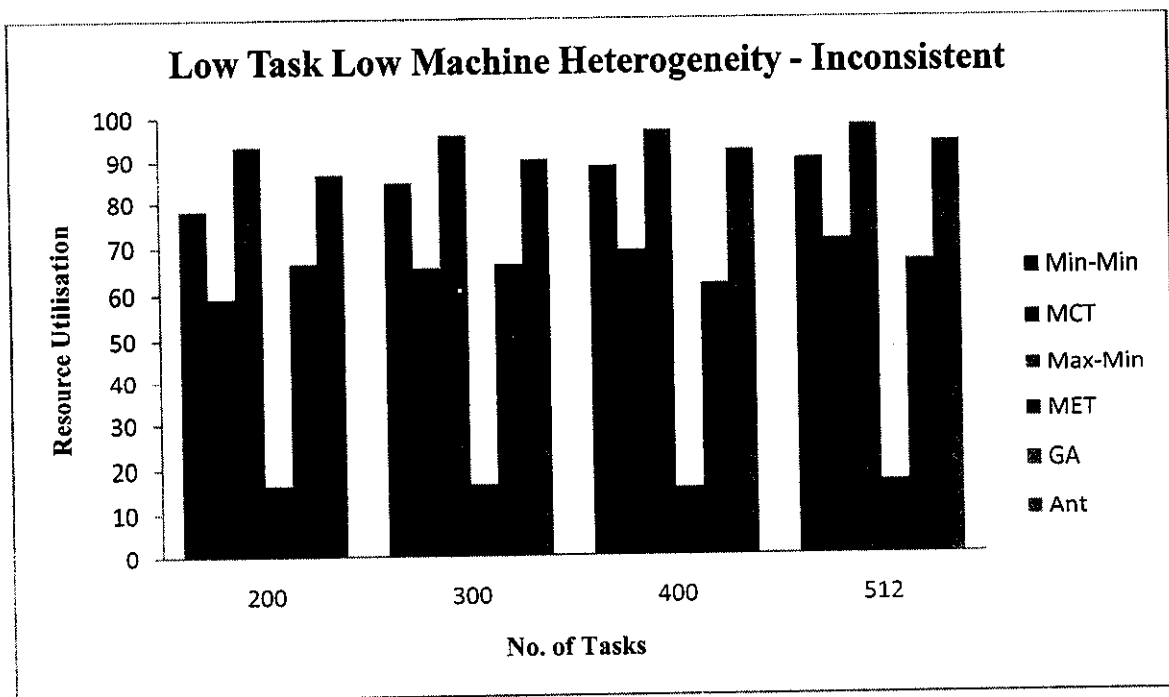
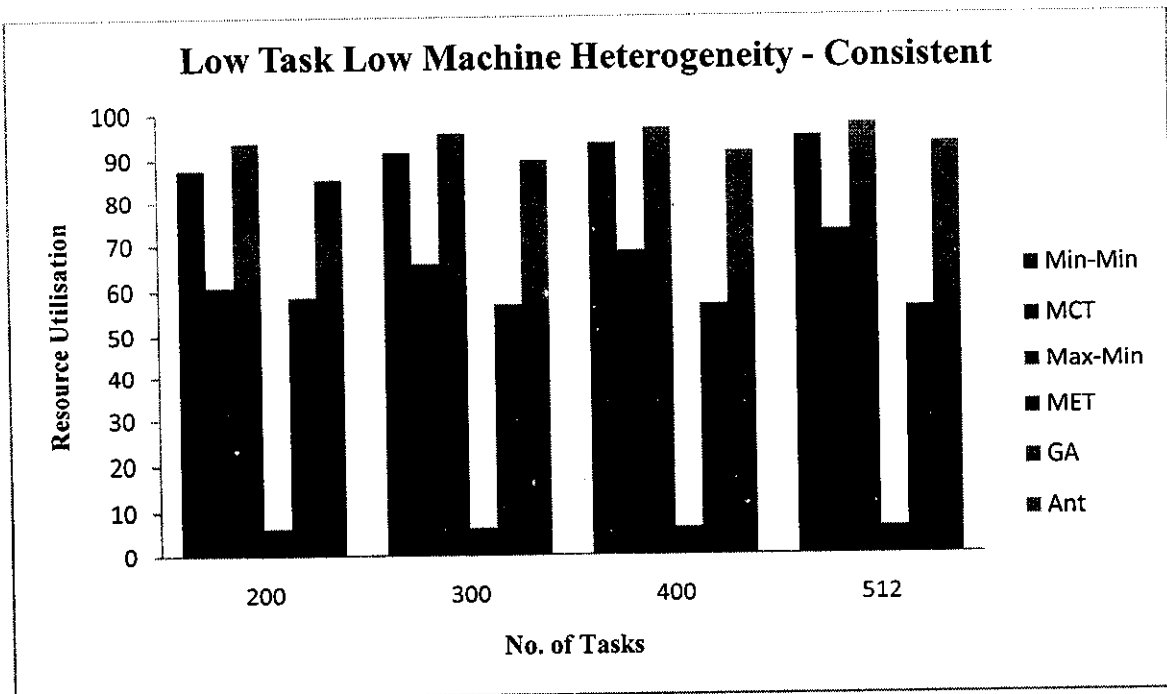


**High Task High Machine Heterogeneity - Partial Consistent**

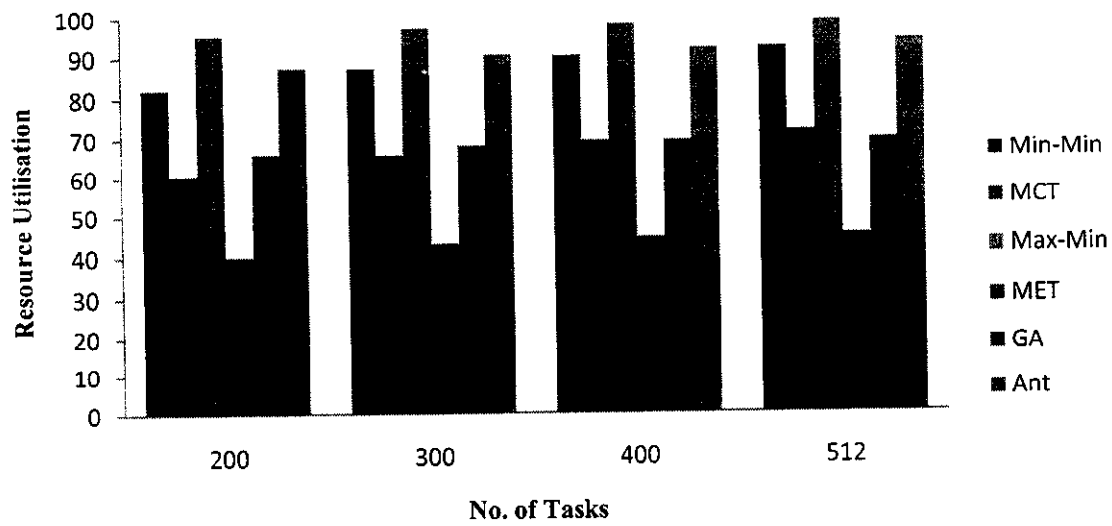




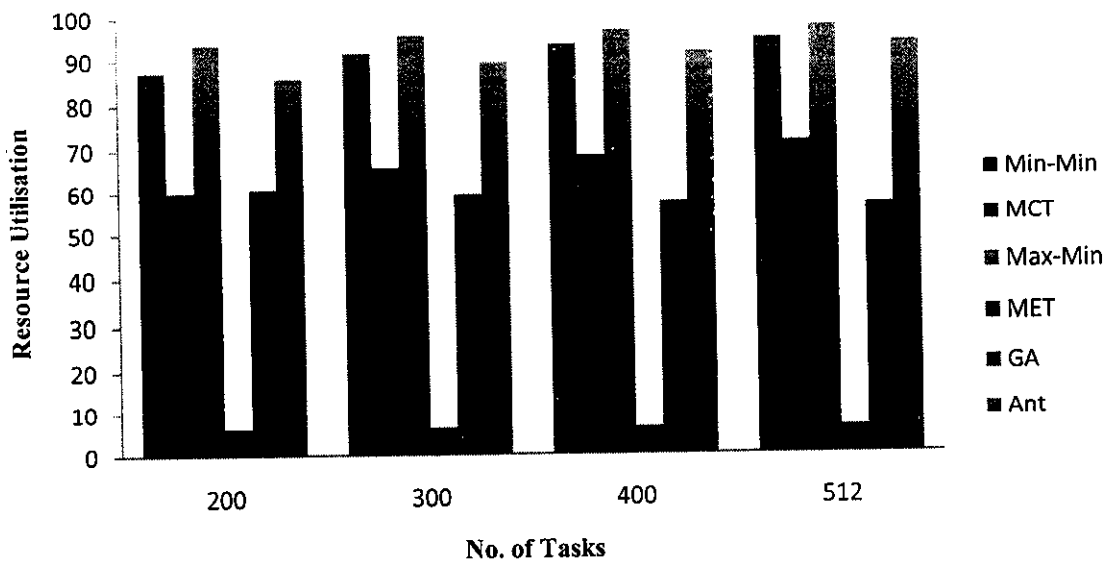
**Resource Utilisation Graphs:**

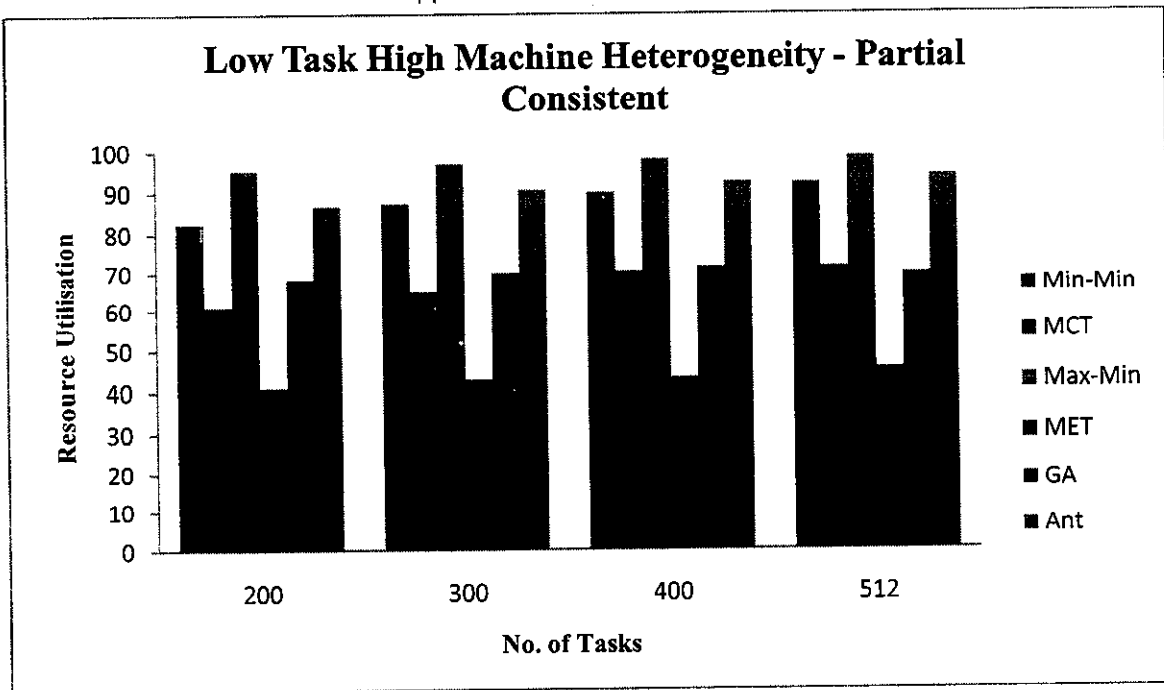
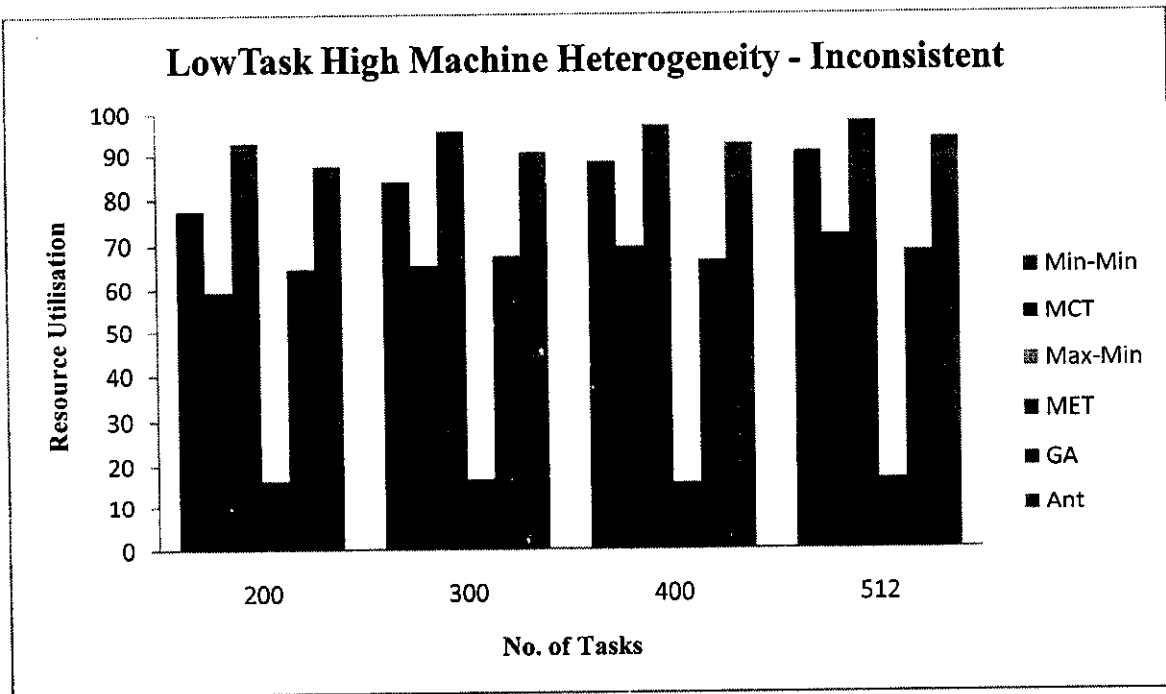


**Low Task Low Machine Heterogeneity - Partial Consistent**

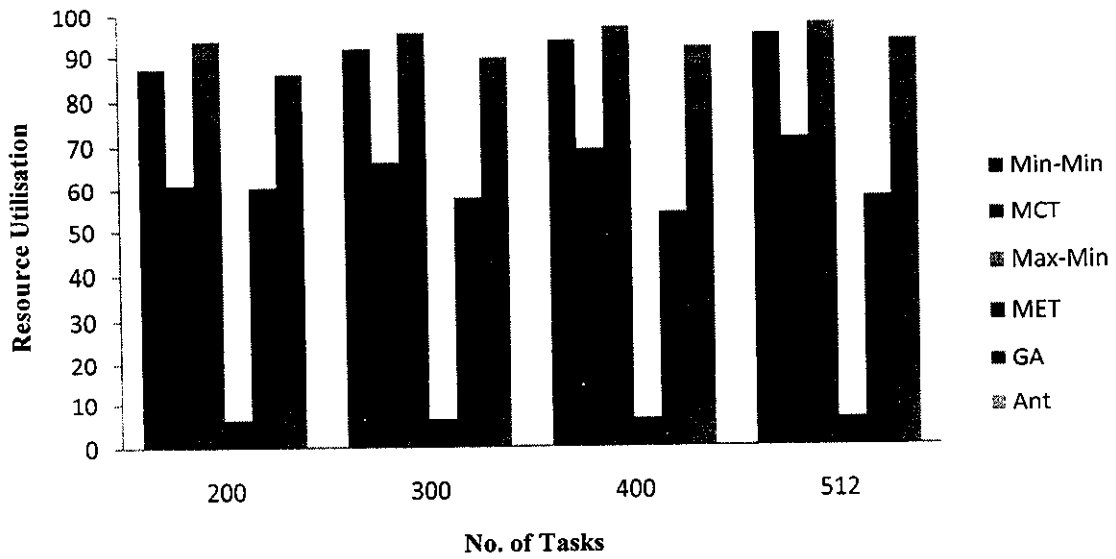


**Low Task High Machine Heterogeneity - Consistent**

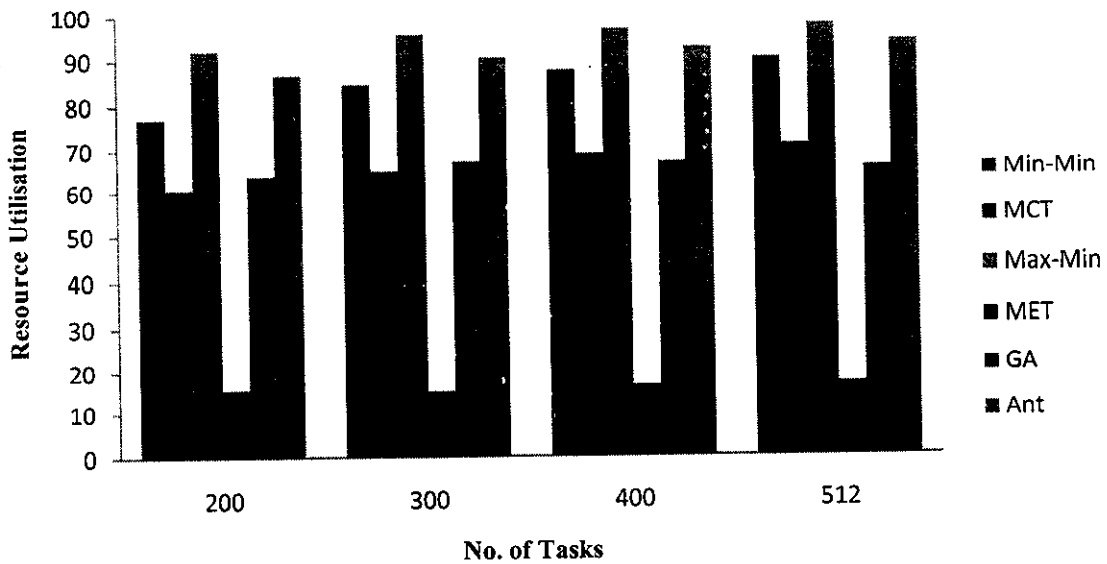


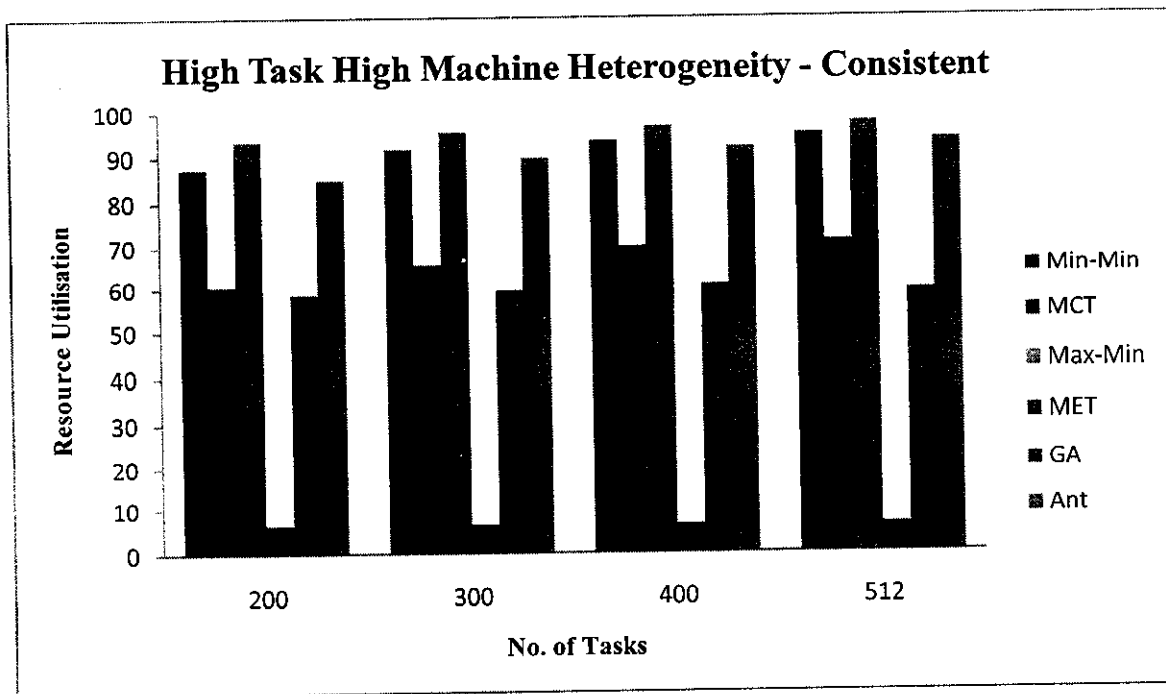
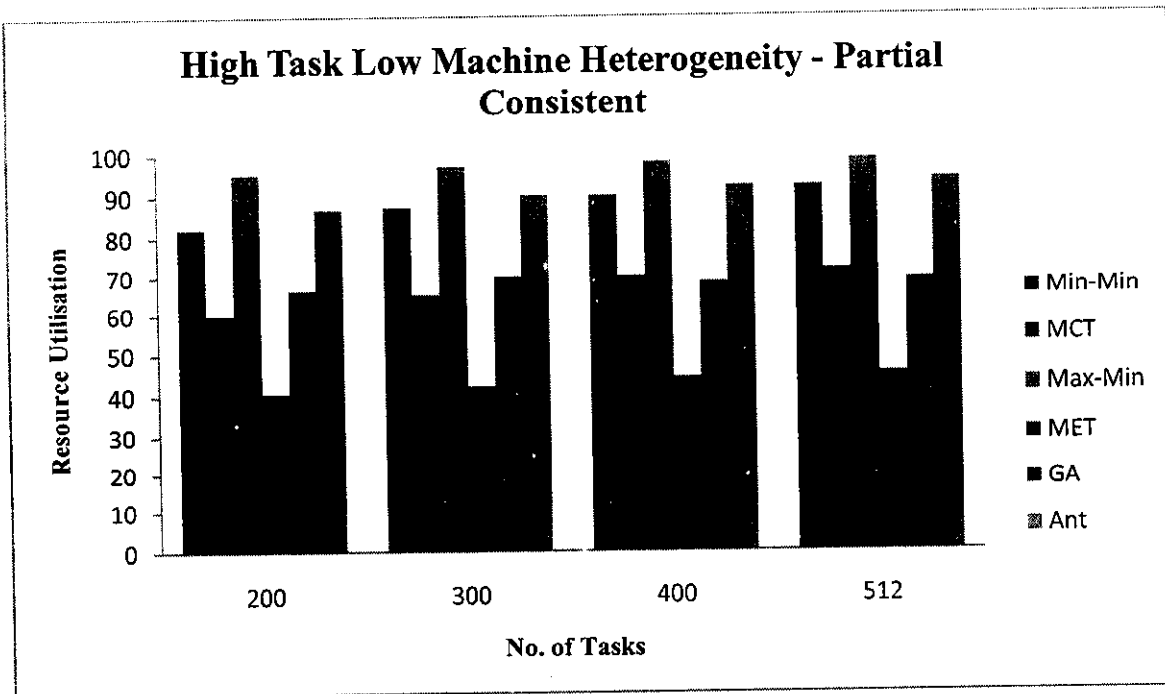


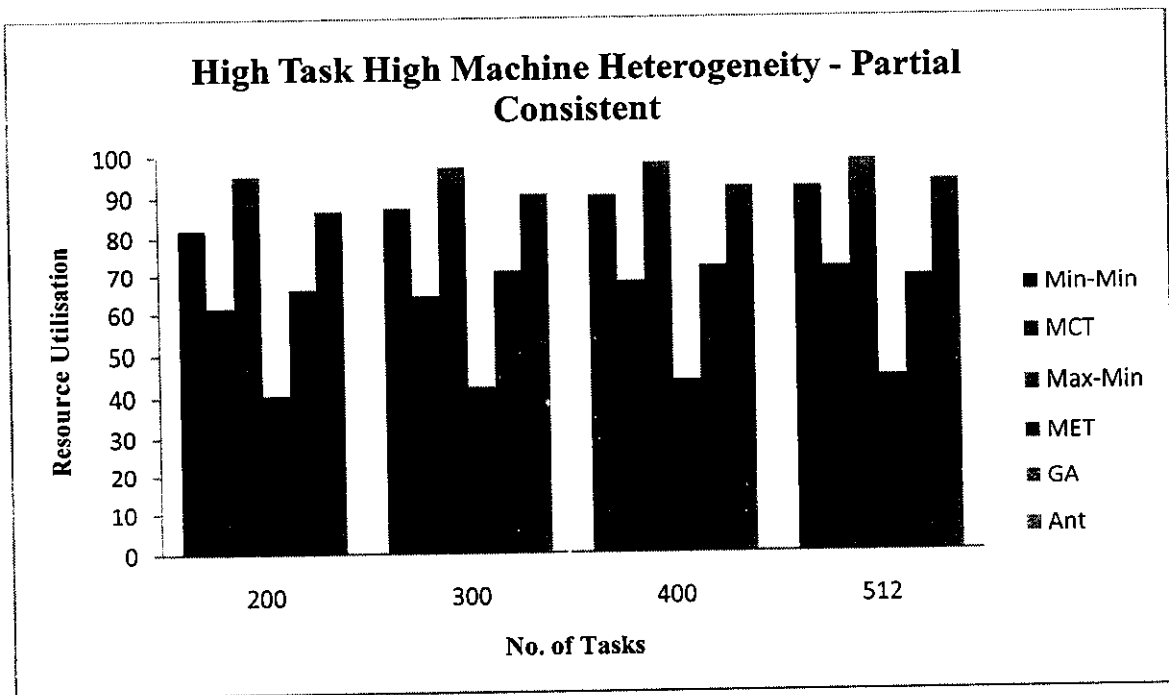
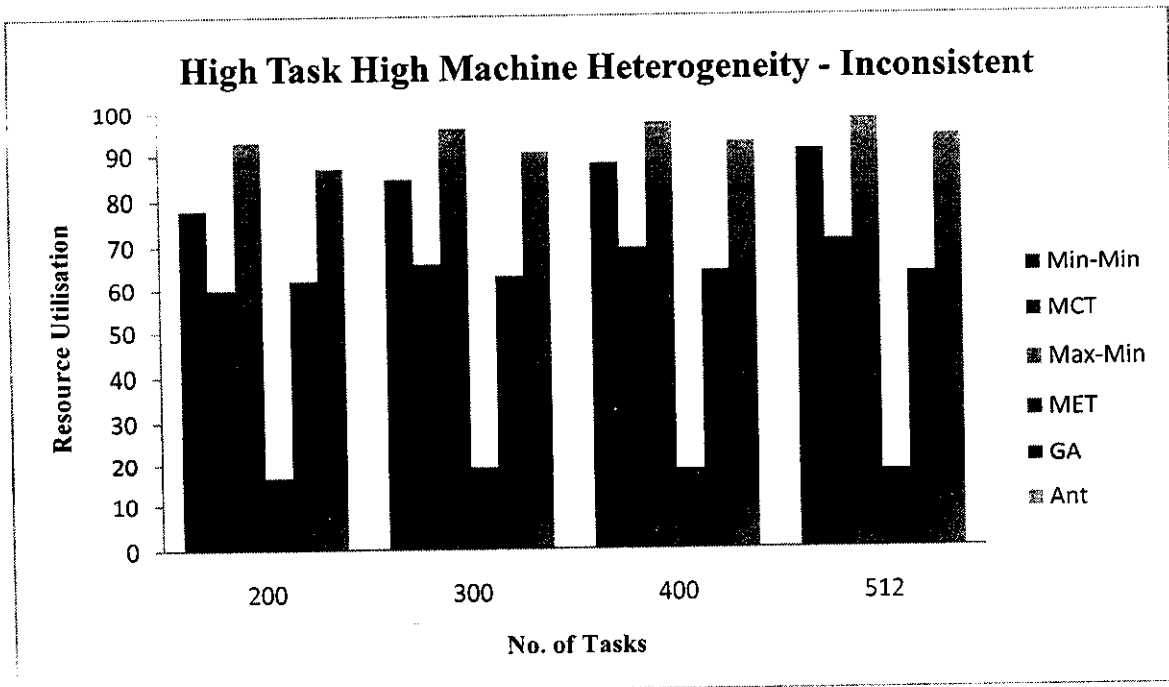
**High Task Low Machine Heterogeneity - Consistent**



**High Task Low Machine Heterogeneity - Inconsistent**







## CHAPTER 5

### CONCLUSION AND FUTURE WORK

#### 5.1. Conclusion

Experimental studies show that in all combinations of heterogeneity of task and machine, our proposed WQACO algorithm gives better results in terms of makespan and resource utilization when compared to the existing algorithms. In this paper, the proposed algorithm chooses the best suited resource which satisfies the user requirement and also completes the task with minimum expected time to compute.

#### 5.2. Future Work

In future, we will change the QoS parameters and also the selection function with any other parameters which we don't take into account on our definitions. We will also improve upon the cross over operation to get better performance in makespan reduction. We will also experiment the genetic algorithm with different combinations of heuristics as initial seeds.

## CHAPTER 6

### APPENDIX

#### 6.1. SAMPLE SOURCE CODE

```
import min.*;
import max.*;
import ant.*;
import java.io.*;
import java.util.*;
import java.io.IOException;

public class input
{
    public static void main(String args[])throws Exception
    {
        int options;
        DataInputStream in=new DataInputStream (System.in);
        System.out.print("ENTER NO. OF no_tasks:");
        int no_no_tasks=Integer.parseInt(in.readLine());
        calc c=new calc(no_no_tasks);
        int i=0;
        do
        {
            System.out.println("1-->low low 2-->low high 3-->high low 4-->high
high 5-->Exit");
            System.out.println("Enter the option:");
            options=Integer.parseInt(in.readLine());
            for(i=0;i<100;i++)
            {
                input.options(c,options,no_no_tasks,i);
            }
        }while(true);
    }
}
```



static void options(calc c,int options,int no\_tasks,int i) throws Exception

{

```
DataInputStream in=new DataInputStream (System.in);
double task[][]=new double[512][3];
double task1[][]=new double[512][3];
double task2[][]=new double[512][3];
double temp[][]=new double[512][17];
double trmp[][][]=new double[16][512][3];
double trmp1[][][]=new double[512][17][3];
double trmp2[][][]=new double[512][17][3];
double mach1[][][]=new double[16][512][3];
double ll[][]=new double[512][17];
double llsort[][]=new double[512][17];
double llpart[][]=new double[512][17];
double lh[][]=new double[512][17];
double lhsort[][]=new double[512][17];
double lhpart[][]=new double[512][17];
double hl[][]=new double[512][17];
double hlsort[][]=new double[512][17];
double hlpart[][]=new double[512][17];
double hh[][]=new double[512][17];
double hhsort[][]=new double[512][17];
double hhpart[][]=new double[512][17];
double min=0.0,avg=0.0;
double temp4[]=new double[819200];
double sblow=5.00,sbhigh=10.00,srlow=5.00,srhigh=10.00;
int no_machines=16;

int antordervector[]=new int[512];
int antordervector1[]=new int[512];
int antordervector2[]=new int[512];

ga ga1=new ga();
```

```

min min1=new min();
max max1=new max();
MET met1=new MET();
mct mct1=new mct();
antpi ant1=new antpi();
int num;
char type;
String mac=null,tas=null,llinc=null,minms=null,maxms=null;
String mac1=null,tas1=null,llc=null,minms1=null,maxms1=null;
String mac2=null,tas2=null,llp=null,minms2=null,maxms2=null;
String gafile=null,gafile1=null,gafile2=null;
String metms=null,metms1=null,metms2=null;
String mctms=null,mctms1=null,mctms2=null;
String metutil=null,metutil1=null,metutil2=null;
String mctutil=null,mctutil1=null,mctutil2=null;
String minutil=null,minutil1=null,minutil2=null;
String maxutil=null,maxutil1=null,maxutil2=null;
String gautil=null,gautil1=null,gautil2=null;

String ip=null,pi=null,assign=null,antuti=null,antmakespan=null,pifinal=null;
String ip1=null,pi1=null,assign1=null,antuti1=null,antmakespan1=null,pifinal1=null;
String ip2=null,pi2=null,assign2=null,antuti2=null,antmakespan2=null,pifinal2=null;

switch(options)
{
    case 1:
        /*...LOW TASK AND LOW MACHINE HETEROGENITY...*/

        //System.out.println("INCONSISTENT MATRIX");

        mac="d:\\hybrid\\case1\\inconsistent\\machinecapability.txt";
        tas="d:\\hybrid\\case1\\inconsistent\\taskrequirement.txt";
        llinc="d:\\hybrid\\case1\\inconsistent\\inconsistentetc.txt";

```

```

minms="d:\\hybrid\\case1\\inconsistent\\minmakespan.txt";
minutil="d:\\hybrid\\case1\\inconsistent\\minutilisation.txt";
maxutil="d:\\hybrid\\case1\\inconsistent\\maxutilisation.txt";
maxms="d:\\hybrid\\case1\\inconsistent\\maxmakespan.txt";
metms="d:\\hybrid\\case1\\inconsistent\\metmakespan.txt";
metutil="d:\\hybrid\\case1\\inconsistent\\metutilisation.txt";
mctms="d:\\hybrid\\case1\\inconsistent\\mctmakespan.txt";
mctutil="d:\\hybrid\\case1\\inconsistent\\mctutilisation.txt";
gafile="d:\\hybrid\\case1\\inconsistent\\gamakespan.txt";
gautil="d:\\hybrid\\case1\\inconsistent\\gautilization.txt";

```

```

antmakespan="d:\\hybrid\\case1\\inconsistent\\antmakespan.txt";
antuti="d:\\hybrid\\case1\\inconsistent\\antuti.txt";
ip="d:\\hybrid\\case1\\inconsistent\\initpi.txt";
pi="d:\\hybrid\\case1\\inconsistent\\pi.txt";
assign="d:\\hybrid\\case1\\inconsistent\\assign.txt";
pifinal="d:\\hybrid\\case1\\inconsistent\\pifinal.txt";

```

```

ll=c.matrix(min,sblow,srflow);
c.display(ll,llinc,i);
task=c.initialise(temp);
c.display1(task,tas,i);
trmp=c.machinecap(mach1,ll);
c.display2(trmp,mac,i);
min1.minmin(ll,no_tasks,minms,minutil);
max1.maxmin(ll,no_tasks,maxms,maxutil);
met1.metcalc(ll,no_tasks,no_machines,metms,metutil);
mct1.mctcalc(ll,no_tasks,no_machines,mctms,mctutil);

```

```

antordervector=ant1.piinit(trmp,task,ll,no_tasks,ip,pi,assign,antmakespan,pifinal,i,ant
uti);

```

```
gal.gacalc(trmp,task,ll,no_tasks,no_machines,i,gafile,gautil,antordervector);
```

```
//System.out.println("Consistent matrix: "+i);
mac1="d:\\hybrid\\case1\\consistent\\machinecapability.txt";
tas1="d:\\hybrid\\case1\\consistent\\taskrequirement.txt";
llc="d:\\hybrid\\case1\\consistent\\consistentetc.txt";
minms1="d:\\hybrid\\case1\\consistent\\minmakespan.txt";
maxms1="d:\\hybrid\\case1\\consistent\\maxmakespan.txt";
minutil1="d:\\hybrid\\case1\\consistent\\minutilisation.txt";
maxutil1="d:\\hybrid\\case1\\consistent\\maxutilisation.txt";
metms1="d:\\hybrid\\case1\\consistent\\metmakespan.txt";
metutil1="d:\\hybrid\\case1\\consistent\\metutilisation.txt";
mctms1="d:\\hybrid\\case1\\consistent\\mctmakespan.txt";
mctutil1="d:\\hybrid\\case1\\consistent\\mctutilisation.txt";
gafile1="d:\\hybrid\\case1\\consistent\\gamakespan.txt";
gautil1="d:\\hybrid\\case1\\consistent\\gautilization.txt";

antmakespan1="d:\\hybrid\\case1\\consistent\\antmakespan.txt";
antuti1="d:\\hybrid\\case1\\consistent\\antuti.txt";
ip1="d:\\hybrid\\case1\\consistent\\initpi.txt";
pi1="d:\\hybrid\\case1\\consistent\\pi.txt";
assign1="d:\\hybrid\\case1\\consistent\\assign.txt";
pifinal1="d:\\hybrid\\case1\\consistent\\pifinal.txt";

llsort=c.sort(ll);
c.display(llsort,llc,i);
task1=c.initialise(temp);
c.display1(task1,tas1,i);
trmp1=c.machinecap(mach1,ll);
c.display2(trmp1,mac1,i);
min1.minmin(llsort,no_tasks,minms1,minutil1);
max1.maxmin(llsort,no_tasks,maxms1,maxutil1);
```

```
met1.metcalc(llsort,no_tasks,no_machines,metms1,metutil1);
mct1.mctcalc(llsort,no_tasks,no_machines,mctms1,mctutil1);
```

```
antordervector1=ant1.piinit(trmp1,task1,llsort,no_tasks,ip1,pi1,assign1,antmakespan1
,pifinal1,i,antutil1);
gal.gacalc(trmp1,task1,llsort,no_tasks,no_machines,i,gafile1,gautil1,antordervector1);
```

```
//System.out.println("Partially consistent: "+i);
```

```
mac2="d:\\hybrid\\case1\\partialconsistent\\machinecapability.txt";
tas2="d:\\hybrid\\case1\\partialconsistent\\taskrequirement.txt";
llp="d:\\hybrid\\case1\\partialconsistent\\partialconsistentetc.txt";
minms2="d:\\hybrid\\case1\\partialconsistent\\minmakespan.txt";
maxms2="d:\\hybrid\\case1\\partialconsistent\\maxmakespan.txt";
minutil2="d:\\hybrid\\case1\\partialconsistent\\minutilisation.txt";
maxutil2="d:\\hybrid\\case1\\partialconsistent\\maxutilisation.txt";
metms2="d:\\hybrid\\case1\\partialconsistent\\metmakespan.txt";
metutil2="d:\\hybrid\\case1\\partialconsistent\\metutilisation.txt";
mctms2="d:\\hybrid\\case1\\partialconsistent\\mctmakespan.txt";
mctutil2="d:\\hybrid\\case1\\partialconsistent\\mctutilisation.txt";
gafile2="d:\\hybrid\\case1\\partialconsistent\\gamakespan.txt";
gautil2="d:\\hybrid\\case1\\partialconsistent\\gautilization.txt";
```

```
antmakespan2="d:\\hybrid\\case1\\partialconsistent\\antmakespan.txt";
antuti2="d:\\hybrid\\case1\\partialconsistent\\antuti.txt";
ip2="d:\\hybrid\\case1\\partialconsistent\\initpi.txt";
pi2="d:\\hybrid\\case1\\partialconsistent\\pi.txt";
assign2="d:\\hybrid\\case1\\partialconsistent\\assign.txt";
pifinal2="d:\\hybrid\\case1\\partialconsistent\\pifinal.txt";
```

```
llpart=c.partsort(ll);
c.display(llpart,llp,i);
```

```

task2=c.initialise(temp);
c.display1(task2,tas2,i);
trmp2=c.machineecap(mach1,ll);
c.display2(trmp2,mac2,i);
min1.minmin(llpart,no_tasks,minms2,minutil2);
max1.maxmin(llpart,no_tasks,maxms2,maxutil2);
met1.metcalc(llpart,no_tasks,no_machines,metms2,metutil2);
mct1.mctcalc(llpart,no_tasks,no_machines,mctms2,mctutil2);

```

```

antordervector2=ant1.piinit(trmp2,task2,llpart,no_tasks,ip2,pi2,assign2,antmakespan2
,pifinal2,i,antuti2);
ga1.gacalc(trmp2,task2,llpart,no_tasks,no_machines,i,gafile2,gautil2,antordervector2);

```

```
break;
```

```
case 2:
```

```

/*...LOW TASK AND HIGH MACHINE HETEROGENITY...*/
mac="d:\\hybrid\\case2\\inconsistent\\machinecapability.txt";
tas="d:\\hybrid\\case2\\inconsistent\\taskrequirement.txt";
llinc="d:\\hybrid\\case2\\inconsistent\\inconsistentetc.txt";
minms="d:\\hybrid\\case2\\inconsistent\\minmakespan.txt";
maxms="d:\\hybrid\\case2\\inconsistent\\maxmakespan.txt";
minutil="d:\\hybrid\\case2\\inconsistent\\minutilisation.txt";
maxutil="d:\\hybrid\\case2\\inconsistent\\maxutilisation.txt";
metms="d:\\hybrid\\case2\\inconsistent\\metmakespan.txt";
metutil="d:\\hybrid\\case2\\inconsistent\\metutilisation.txt";
mctms="d:\\hybrid\\case2\\inconsistent\\mctmakespan.txt";
mctutil="d:\\hybrid\\case2\\inconsistent\\mctutilisation.txt";
gafile="d:\\hybrid\\case2\\inconsistent\\gamakespan.txt";
gautil="d:\\hybrid\\case2\\inconsistent\\gautilization.txt";

antmakespan="d:\\hybrid\\case2\\inconsistent\\antmakespan.txt";

```

```
antuti="d:\\hybrid\\case2\\inconsistent\\antuti.txt";
ip="d:\\hybrid\\case2\\inconsistent\\initpi.txt";
pi="d:\\hybrid\\case2\\inconsistent\\pi.txt";
assign="d:\\hybrid\\case2\\inconsistent\\assign.txt";
pifinal="d:\\hybrid\\case2\\inconsistent\\pifinal.txt";
```

```
lh=c.matrix(min,sblow,srhigh);
c.display(lh,llinc,i);
task=c.initialise(temp);
c.display1(task,tas,i);
trmp=c.machinecap(mach1,lh);
c.display2(trmp,mac,i);
min1.minmin(lh,no_tasks,minms,minutil);
max1.maxmin(lh,no_tasks,maxms,maxutil);
met1.metcalc(lh,no_tasks,no_machines,metms,metutil);
mct1.mctcalc(lh,no_tasks,no_machines,mctms,mctutil);
```

```
antordervector=ant1.piinit(trmp,task,lh,no_tasks,ip,pi,assign,antmakespan,pifinal,i,ant
uti);
```

```
gal.gacalc(trmp,task,lh,no_tasks,no_machines,i,gafile,gautil,antordervector);
```

```
//System.out.println("Consistent matrix: "+i);
```

```
mac1="d:\\hybrid\\case2\\consistent\\machinecapability.txt";
tas1="d:\\hybrid\\case2\\consistent\\taskrequirement.txt";
llc="d:\\hybrid\\case2\\consistent\\consistentetc.txt";
minms1="d:\\hybrid\\case2\\consistent\\minmakespan.txt";
maxms1="d:\\hybrid\\case2\\consistent\\maxmakespan.txt";
minutil1="d:\\hybrid\\case2\\consistent\\minutilisation.txt";
maxutil1="d:\\hybrid\\case2\\consistent\\maxutilisation.txt";
metms1="d:\\hybrid\\case2\\consistent\\metmakespan.txt";
```

```
metutil1="d:\\hybrid\\case2\\consistent\\metutilisation.txt";
mctms1="d:\\hybrid\\case2\\consistent\\mctmakespan.txt";
mctutil1="d:\\hybrid\\case2\\consistent\\mctutilisation.txt";
gafile1="d:\\hybrid\\case2\\consistent\\gamakespan.txt";
gautill1="d:\\hybrid\\case2\\consistent\\gautilization.txt";
```

```
antmakespan1="d:\\hybrid\\case2\\consistent\\antmakespan.txt";
antuti1="d:\\hybrid\\case2\\consistent\\antuti.txt";
ip1="d:\\hybrid\\case2\\consistent\\initpi.txt";
pi1="d:\\hybrid\\case2\\consistent\\pi.txt";
assign1="d:\\hybrid\\case2\\consistent\\assign.txt";
pifinal1="d:\\hybrid\\case2\\consistent\\pifinal.txt";
```

```
lhsort=c.sort(lh);
c.display(lhsort,llc,i);
task1=c.initialise(temp);
c.display1(task1,tas1,i);
trmp1=c.machineecap(mach1,lh);
c.display2(trmp1,mac1,i);
min1.minmin(lhsort,no_tasks,minms1,minutil1);
max1.maxmin(lhsort,no_tasks,maxms1,maxutil1);
met1.metcalc(lhsort,no_tasks,no_machines,metms1,metutil1);
mct1.mctcalc(lhsort,no_tasks,no_machines,mctms1,mctutil1);
```

```
antordervector1=ant1.piinit(trmp1,task1,lhsort,no_tasks,ip1,pi1,assign1,antmakespan
1,pifinal1,i,antuti1);
ga1.gacalc(trmp1,task1,lhsort,no_tasks,no_machines,i,gafile1,gautill1,antordervector1);
```

```
//System.out.println("Partially consistent: "+i);
```

```
mac2="d:\\hybrid\\case2\\partialconsistent\\machinecapability.txt";
```



```

tas2="d:\\hybrid\\case2\\partialconsistent\\taskrequirement.txt";
llp="d:\\hybrid\\case2\\partialconsistent\\partialconsistentetc.txt";
minms2="d:\\hybrid\\case2\\partialconsistent\\minmakespan.txt";
maxms2="d:\\hybrid\\case2\\partialconsistent\\maxmakespan.txt";
minutil2="d:\\hybrid\\case2\\partialconsistent\\minutilisation.txt";
maxutil2="d:\\hybrid\\case2\\partialconsistent\\maxutilisation.txt";
metms2="d:\\hybrid\\case2\\partialconsistent\\metmakespan.txt";
metutil2="d:\\hybrid\\case2\\partialconsistent\\metutilisation.txt";
mctms2="d:\\hybrid\\case2\\partialconsistent\\mctmakespan.txt";
mctutil2="d:\\hybrid\\case2\\partialconsistent\\mctutilisation.txt";
gafile2="d:\\hybrid\\case2\\partialconsistent\\gamakespan.txt";
gautil2="d:\\hybrid\\case2\\partialconsistent\\gautilization.txt";

antmakespan2="d:\\hybrid\\case2\\partialconsistent\\antmakespan.txt";
antuti2="d:\\hybrid\\case2\\partialconsistent\\antuti.txt";
ip2="d:\\hybrid\\case2\\partialconsistent\\initpi.txt";
pi2="d:\\hybrid\\case2\\partialconsistent\\pi.txt";
assign2="d:\\hybrid\\case2\\partialconsistent\\assign.txt";
pifinal2="d:\\hybrid\\case2\\partialconsistent\\pifinal.txt";

lhpart=c.partsort(lh);
c.display(lhpart,llp,i);
task2=c.initialise(temp);
c.display1(task2,tas2,i);
trmp2=c.machinecap(mach1,lh);
c.display2(trmp2,mac2,i);
min1.minmin(lhpart,no_tasks,minms2,minutil2);
max1.maxmin(lhpart,no_tasks,maxms2,maxutil2);
met1.metcalc(lhpart,no_tasks,no_machines,metms2,metutil2);
mct1.mctcalc(lhpart,no_tasks,no_machines,mctms2,mctutil2);

```

```
antordervector2=ant1.piinit(trmp2,task2,lhpart,no_tasks,ip2,pi2,assign2,antmakespan
2,pifinal2,i,antuti2);
ga1.gacalc(trmp2,task2,lhpart,no_tasks,no_machines,i,gafile2,gautil2,antordervector2);
```

```
break;
```

```
case 3:
```

```
/*...HIGH TASK AND LOW MACHINE HETEROGENITY...*/
```

```
mac="d:\\hybrid\\case3\\inconsistent\\machinecapability.txt";
```

```
tas="d:\\hybrid\\case3\\inconsistent\\taskrequirement.txt";
```

```
llinc="d:\\hybrid\\case3\\inconsistent\\inconsistentetc.txt";
```

```
minms="d:\\hybrid\\case3\\inconsistent\\minmakespan.txt";
```

```
maxms="d:\\hybrid\\case3\\inconsistent\\maxmakespan.txt";
```

```
minutil="d:\\hybrid\\case3\\inconsistent\\minutilisation.txt";
```

```
maxutil="d:\\hybrid\\case3\\inconsistent\\maxutilisation.txt";
```

```
metms="d:\\hybrid\\case3\\inconsistent\\metmakespan.txt";
```

```
metutil="d:\\hybrid\\case3\\inconsistent\\metutilisation.txt";
```

```
mctms="d:\\hybrid\\case3\\inconsistent\\mctmakespan.txt";
```

```
mctutil="d:\\hybrid\\case3\\inconsistent\\mctutilisation.txt";
```

```
gafile="d:\\hybrid\\case3\\inconsistent\\gamakespan.txt";
```

```
gautil="d:\\hybrid\\case3\\inconsistent\\gautilization.txt";
```

```
antmakespan="d:\\hybrid\\case3\\inconsistent\\antmakespan.txt";
```

```
antuti="d:\\hybrid\\case3\\inconsistent\\antuti.txt";
```

```
ip="d:\\hybrid\\case3\\inconsistent\\initpi.txt";
```

```
pi="d:\\hybrid\\case3\\inconsistent\\pi.txt";
```

```
assign="d:\\hybrid\\case3\\inconsistent\\assign.txt";
```

```
pifinal="d:\\hybrid\\case3\\inconsistent\\pifinal.txt";
```

```
hl=c.matrix(min,sbhigh,srlow);
```

```
c.display(hl,llinc,i);
```

```

task=c.initialise(temp);
c.display1(task,tas,i);
trmp=c.machinecap(mach1,hl);
c.display2(trmp,mac,i);
min1.minmin(hl,no_tasks,minms,minutil);
max1.maxmin(hl,no_tasks,maxms,maxutil);
met1.metcalc(hl,no_tasks,no_machines,metms,metutil);
mct1.metcalc(hl,no_tasks,no_machines,mctms,mctutil);

```

```

antordervector=ant1.piinit(trmp,task,hl,no_tasks,ip,pi,assign,antmakespan,pifinal,i,ant
uti);

```

```

gal.gacalc(trmp,task,hl,no_tasks,no_machines,i,gafile,gautil,antordervector);

```

```

// System.out.println("Consistent matrix: "+i);

```

```

mac1="d:\\hybrid\\case3\\consistent\\machinecapability.txt";
tas1="d:\\hybrid\\case3\\consistent\\taskrequirement.txt";
llc="d:\\hybrid\\case3\\consistent\\consistentetc.txt";
minms1="d:\\hybrid\\case3\\consistent\\minmakespan.txt";
maxms1="d:\\hybrid\\case3\\consistent\\maxmakespan.txt";
minutil1="d:\\hybrid\\case3\\consistent\\minutilisation.txt";
maxutil1="d:\\hybrid\\case3\\consistent\\maxutilisation.txt";
metms1="d:\\hybrid\\case3\\consistent\\metmakespan.txt";
metutil1="d:\\hybrid\\case3\\consistent\\metutilisation.txt";
mctms1="d:\\hybrid\\case3\\consistent\\mctmakespan.txt";
mctutil1="d:\\hybrid\\case3\\consistent\\mctutilisation.txt";
gafile1="d:\\hybrid\\case3\\consistent\\gamakespan.txt";
gautil1="d:\\hybrid\\case3\\consistent\\gautilization.txt";

```

```

antmakespan1="d:\\hybrid\\case3\\consistent\\antmakespan.txt";
antuti1="d:\\hybrid\\case3\\consistent\\antuti.txt";

```

```

ip1="d:\\hybrid\\case3\\consistent\\initpi.txt";
pi1="d:\\hybrid\\case3\\consistent\\pi.txt";
assign1="d:\\hybrid\\case3\\consistent\\assign.txt";
pifinal1="d:\\hybrid\\case3\\consistent\\pifinal.txt";

```

```

hlsort=c.sort(hl);
c.display(hlsort,llc,i);
task1=c.initialise(temp);
c.display1(task1,tas1,i);
trmp1=c.machineecap(mach1,hl);
c.display2(trmp1,mac1,i);
min1.minmin(hlsort,no_tasks,minms1,minutil1);
max1.maxmin(hlsort,no_tasks,maxms1,maxutil1);
met1.metcalc(hlsort,no_tasks,no_machines,metms1,metutil1);
mct1.mctcalc(hlsort,no_tasks,no_machines,mctms1,mctutil1);

```

```

antordervector1=ant1.piinit(trmp1,task1,hlsort,no_tasks,ip1,pi1,assign1,antmakespan
1,pifinal1,i,antutil1);
ga1.gacalc(trmp1,task1,hlsort,no_tasks,no_machines,i,gafile1,gautil1,antordervector1);

```

```

//System.out.println("Partially consistent: "+i);

```

```

mac2="d:\\hybrid\\case3\\partialconsistent\\machinecapability.txt";
tas2="d:\\hybrid\\case3\\partialconsistent\\taskrequirement.txt";
llp="d:\\hybrid\\case3\\partialconsistent\\partialconsistentetc.txt";
minms2="d:\\hybrid\\case3\\partialconsistent\\minmakespan.txt";
maxms2="d:\\hybrid\\case3\\partialconsistent\\maxmakespan.txt";
minutil2="d:\\hybrid\\case3\\partialconsistent\\minutilisation.txt";
maxutil2="d:\\hybrid\\case3\\partialconsistent\\maxutilisation.txt";
metms2="d:\\hybrid\\case3\\partialconsistent\\metmakespan.txt";
metutil2="d:\\hybrid\\case3\\partialconsistent\\metutilisation.txt";
mctms2="d:\\hybrid\\case3\\partialconsistent\\mctmakespan.txt";

```

```

mctutil2="d:\\hybrid\\case3\\partialconsistent\\mctutilisation.txt";
gafile2="d:\\hybrid\\case3\\partialconsistent\\gamakespan.txt";
gautil2="d:\\hybrid\\case3\\partialconsistent\\gautilization.txt";

antmakespan2="d:\\hybrid\\case3\\partialconsistent\\antmakespan.txt";
antuti2="d:\\hybrid\\case3\\partialconsistent\\antuti.txt";
ip2="d:\\hybrid\\case3\\partialconsistent\\initpi.txt";
pi2="d:\\hybrid\\case3\\partialconsistent\\pi.txt";
assign2="d:\\hybrid\\case3\\partialconsistent\\assign.txt";
pifinal2="d:\\hybrid\\case3\\partialconsistent\\pifinal.txt";

```

```

hlp2=c.partsort(hl);
c.display(hlp2,lp,i);
task2=c.initialise(temp);
c.display1(task2,tas2,i);
trmp2=c.machineicap(mach1,hl);
c.display2(trmp2,mac2,i);
min1.minmin(hlp2,no_tasks,minms2,minutil2);
max1.maxmin(hlp2,no_tasks,maxms2,maxutil2);
met1.metcalc(hlp2,no_tasks,no_machines,metms2,metutil2);
mct1.mctcalc(hlp2,no_tasks,no_machines,mctms2,mctutil2);

```

```

antordervector2=ant1.piinit(trmp2,task2,hlp2,no_tasks,ip2,pi2,assign2,antmakespan
2,pifinal2,i,antuti2);
ga1.gacalc(trmp2,task2,hlp2,no_tasks,no_machines,i,gafile2,gautil2,antordervector2);

```

```

break;

```

```

case 4:

```

```

/*...HIGH TASK AND HIGH MACHINE HETEROGENITY...*/
mac="d:\\hybrid\\case4\\inconsistent\\machinecapability.txt";
tas="d:\\hybrid\\case4\\inconsistent\\taskrequirement.txt";

```

```

llinc="d:\\hybrid\\case4\\inconsistent\\inconsistentetc.txt";
minms="d:\\hybrid\\case4\\inconsistent\\minmakespan.txt";
maxms="d:\\hybrid\\case4\\inconsistent\\maxmakespan.txt";
minutil="d:\\hybrid\\case4\\inconsistent\\minutilisation.txt";
maxutil="d:\\hybrid\\case4\\inconsistent\\maxutilisation.txt";
metms="d:\\hybrid\\case4\\inconsistent\\metmakespan.txt";
metutil="d:\\hybrid\\case4\\inconsistent\\metutilisation.txt";
mctms="d:\\hybrid\\case4\\inconsistent\\mctmakespan.txt";
mctutil="d:\\hybrid\\case4\\inconsistent\\mctutilisation.txt";
gafile="d:\\hybrid\\case4\\inconsistent\\gamakespan.txt";
gautil="d:\\hybrid\\case4\\inconsistent\\gautilization.txt";

antmakespan="d:\\hybrid\\case4\\inconsistent\\antmakespan.txt";
antuti="d:\\hybrid\\case4\\inconsistent\\antuti.txt";
ip="d:\\hybrid\\case4\\inconsistent\\initpi.txt";
pi="d:\\hybrid\\case4\\inconsistent\\pi.txt";
assign="d:\\hybrid\\case4\\inconsistent\\assign.txt";
pifinal="d:\\hybrid\\case4\\inconsistent\\pifinal.txt";

..

hh=c.matrix(min,sbhigh,srhigh);
c.display(hh,llinc,i);
task=c.initialise(temp);
c.display1(task,tas,i);
trmp=c.machinecap(mach1,hh);
c.display2(trmp,mac,i);
min1.minmin(hh,no_tasks,minms,minutil);
max1.maxmin(hh,no_tasks,maxms,maxutil);
met1.metcalc(hh,no_tasks,no_machines,metms,metutil);
mct1.mctcalc(hh,no_tasks,no_machines,mctms,mctutil);

```

```

antordervector=ant1.piinit(trmp,task,hh,no_tasks,ip,pi,assign,antmakespan,pifinal,i,an
tuti);

```

```
ga1.gacalc(trmp,task,hh,no_tasks,no_machines,i,gafile,gautil,antordervector);
```

```
//System.out.println("Consistent matrix: "+i);
```

```
mac1="d:\\hybrid\\case4\\consistent\\machinecapability.txt";
```

```
tas1="d:\\hybrid\\case4\\consistent\\taskrequirement.txt";
```

```
llc="d:\\hybrid\\case4\\consistent\\consistentetc.txt";
```

```
minms1="d:\\hybrid\\case4\\consistent\\minmakespan.txt";
```

```
maxms1="d:\\hybrid\\case4\\consistent\\maxmakespan.txt";
```

```
minutil1="d:\\hybrid\\case4\\consistent\\minutilisation.txt";
```

```
maxutil1="d:\\hybrid\\case4\\consistent\\maxutilisation.txt";
```

```
metms1="d:\\hybrid\\case4\\consistent\\metmakespan.txt";
```

```
metutil1="d:\\hybrid\\case4\\consistent\\metutilisation.txt";
```

```
mctms1="d:\\hybrid\\case4\\consistent\\mctmakespan.txt";
```

```
mctutil1="d:\\hybrid\\case4\\consistent\\mctutilisation.txt";
```

```
gafile1="d:\\hybrid\\case4\\consistent\\gamakespan.txt";
```

```
gautil1="d:\\hybrid\\case4\\consistent\\gautilization.txt";
```

```
antmakespan1="d:\\hybrid\\case4\\consistent\\antmakespan.txt";
```

```
antuti1="d:\\hybrid\\case4\\consistent\\antuti.txt";
```

```
ip1="d:\\hybrid\\case4\\consistent\\initpi.txt";
```

```
pi1="d:\\hybrid\\case4\\consistent\\pi.txt";
```

```
assign1="d:\\hybrid\\case4\\consistent\\assign.txt";
```

```
pifinal1="d:\\hybrid\\case4\\consistent\\pifinal.txt";
```

```
hhsort=c.sort(hh);
```

```
c.display(hhsort,llc,i);
```

```
task1=c.initialise(temp);
```

```
c.display1(task1,tas1,i);
```

```
trmp1=c.machinecap(mach1,hh);
```

```
c.display2(trmp1,mac1,i);
```

```
min1.minmin(hhsort,no_tasks,minms1,minutil1);
```

```
max1.maxmin(hhsort,no_tasks,maxms1,maxutil1);
met1.metcalc(hhsort,no_tasks,no_machines,metms1,metutil1);
mct1.mctcalc(hhsort,no_tasks,no_machines,mctms1,mctutil1);
```

```
antordervector1=ant1.piinit(trmp1,task1,hhsort,no_tasks,ip1,pi1,assign1,antmakespan
1,pifinal1,i,antutil1);
ga1.gacalc(trmp1,task1,hhsort,no_tasks,no_machines,i,gafile1,gautil1,antordervector1);
```

```
//System.out.println("Partially consistent: "+i);
```

```
mac2="d:\\hybrid\\case4\\partialconsistent\\machinecapability.txt";
tas2="d:\\hybrid\\case4\\partialconsistent\\taskrequirement.txt";
llp="d:\\hybrid\\case4\\partialconsistent\\partialconsistentetc.txt";
minms2="d:\\hybrid\\case4\\partialconsistent\\minmakespan.txt";
maxms2="d:\\hybrid\\case4\\partialconsistent\\maxmakespan.txt";
minutil2="d:\\hybrid\\case4\\partialconsistent\\minutilisation.txt";
maxutil2="d:\\hybrid\\case4\\partialconsistent\\maxutilisation.txt";
metms2="d:\\hybrid\\case4\\partialconsistent\\metmakespan.txt";
metutil2="d:\\hybrid\\case4\\partialconsistent\\metutilisation.txt";
mctms2="d:\\hybrid\\case4\\partialconsistent\\mctmakespan.txt";
mctutil2="d:\\hybrid\\case4\\partialconsistent\\mctutilisation.txt";
gafile2="d:\\hybrid\\case4\\partialconsistent\\gamakespan.txt";
gautil2="d:\\hybrid\\case4\\partialconsistent\\gautilization.txt";
```

```
antmakespan2="d:\\hybrid\\case4\\partialconsistent\\antmakespan.txt";
antuti2="d:\\hybrid\\case4\\partialconsistent\\antuti.txt";
ip2="d:\\hybrid\\case4\\partialconsistent\\initpi.txt";
pi2="d:\\hybrid\\case4\\partialconsistent\\pi.txt";
assign2="d:\\hybrid\\case4\\partialconsistent\\assign.txt";
pifinal2="d:\\hybrid\\case4\\partialconsistent\\pifinal.txt";
```

```
hhpart=c.partsort(hh);
```



```

c.display(hhpart,llp,i);
task2=c.initialise(temp);
c.display1(task2,tas2,i);
trmp2=c.machineap(mach1,hh);
c.display2(trmp2,mac2,i);
min1.minmin(hhpart,no_tasks,minms2,minutil2);
max1.maxmin(hhpart,no_tasks,maxms2,maxutil2);
met1.metcalc(hhpart,no_tasks,no_machines,metms2,metutil2);
mct1.mctcalc(hhpart,no_tasks,no_machines,mctms2,mctutil2);

```

```

antordervector2=ant1.piinit(trmp2,task2,hhpart,no_tasks,ip2,pi2,assign2,antmakespan
2,pifinal2,i,antuti2);

```

```

gal.gacalc(trmp2,task2,hhpart,no_tasks,no_machines,i,gafile2,gautil2,antordervector2);

```

```

break;

```

```

case 5:

```

```

    System.exit(0);

```

```

    break;

```

```

}

```

```

}

```

```

}

```

```

class calc

```

```

{

```

```

    int no_tasks,no_machines,i,j,k;

```

```

    double w[]=new double[512];

```

```

    double x[]=new double[512];

```

```

    double y[][]=new double[512][17];

```

```

    double z[][]=new double[512][17];

```

```

    double temp1[][]=new double[512][3];

```

```

    double c=0.5,ra=0.25,d=0.25,a=0;

```

```

    public calc(int nno_tasks)

```

```

    {

```

```

        no_tasks=nno_tasks;

```

```

    }
    public double[][] matrix(double min,double sb,double sr)
    {
        for(i=0;i<no_tasks;i++)
        {
            for(j=0;j<16;j++)
            {
                y[i][j]=round((min+Math.random()*sb),4);
            }
        }
        for(i=0;i<16;i++)
        {
            x[i]=round((min+Math.random()*sr),4);
        }
        for(i=0;i<no_tasks;i++)
        {
            for(j=0;j<16;j++)
            {
                z[i][j]=round((y[i][j]*x[j]),4);
            }
        }
        return z;
    }
}
/*****MACHINE CAPABILITY*****/
public double[][][] machinecap(double mac[][][],double et[][])throws Exception
{
    double min1=1.0,max1=10.0;
    double min2=5.0,max2=10.0;
    for(i=0;i<16;i++)
    {
        double cost=round((min1+Math.random()*(max1-min1)),4);
        double ram=round((min2+Math.random()*(max2-min2)),4);
    }
}

```

```

        for(j=0;j<no_tasks;j++)
        {
            for(k=0;k<3;k++)
            {
                if(k==0)
                    mac[i][j][k]=round(((et[j][i])*cost),4);//COST
                IS MULTIPLIED BY A CONSTANT
                else if(k==1)//RAM=2
                    mac[i][j][k]=ram;
                else
                    mac[i][j][k]=round((et[j][i]),4);//DEADLINE IS
                RETRIEVED FROM ETC MATRIX
            }
        }
    }
    return mac;
}

```

/\*\*\*/\*\*\*\*\*USER INPUTS\*\*\*\*\*/

```

public double[][] initialise(double arr[][] throws Exception
{
    double min1=800.00,max1=1200.00;
    double min2=80.0,max2=120.00;
    double min3=1.00,max3=6.00;
    for(i=0;i<no_tasks;i++)
    {
        for(j=0;j<1;j++)
        {
            double cost=round((min1+Math.random()*(max1-min1)),4);
            arr[i][j]=cost;
            double ram=round((min3+Math.random()*(max3-min3)),4);
            arr[i][j+1]=ram;
            double deadline=round((min2+Math.random()*(max2-min2)),4);

```

```

        arr[i][j+2]=deadline;
    }
}
for(i=0;i<no_tasks;i++)
{
    for(j=0;j<1;j++)
    {
        temp1[i][j]=round((arr[i][j]),4);
        temp1[i][j+1]=round((arr[i][j+1]),4);
        temp1[i][j+2]=round((arr[i][j+2]),4);
    }
}
return temp1;
}
/***** DISPLAYS WEIGHT MATRIX *****/
public void display1(double disp[][] ,String s,int count)throws Exception
{
    String sn1="\r\n";
    OutputStream out=new FileOutputStream(s,true);
    BufferedOutputStream bfo=new BufferedOutputStream(out);
    String ETCCCount = "\nIteration";
    String disp1= ETCCCount.concat(Integer.toString(count));
    byte b[]=disp1.getBytes();
    bfo.write(b);
    bfo.write(sn1.getBytes());
    for(i=0;i<no_tasks;i++)
    {
        String s2="null";
        for(j=0;j<3;j++)
        {
            Double fObj = new Double(disp[i][j]);
            String s1 = fObj.toString();
            s2=s1.concat("\t");
        }
    }
}

```

```

        byte by[]=s2.getBytes();
        bfo.write(by);
    }
    String sn="\r\n";
    bfo.write(sn.getBytes());

}
bfo.write(sn1.getBytes());
bfo.close();
}
/*****Display machine capability*****/
public void display2(double trmp[][][],String s,int count)throws Exception
{
    OutputStream out=new FileOutputStream(s,true);
    BufferedOutputStream bfo=new BufferedOutputStream(out);
    String sn="\r\n";
    String ETCCCount = "\nIteration ";
    String disp1= ETCCCount.concat(Integer.toString(count));
    byte b[]=disp1.getBytes();
    bfo.write(b);
    bfo.write(sn.getBytes());
    for(i=0;i<16;i++)
    {
        String b1 = Integer.toString(i);
        String b2="Machine";
        String b3=b2.concat(b1);
        byte by1[]=b3.getBytes();
        bfo.write(by1);
        bfo.write(sn.getBytes());
        for(j=0;j<no_tasks;j++)
        {
            String a1 = Integer.toString(j);
            String a2="Task";

```

```

        String a3=a2.concat(a1);
        String a4=a3.concat("\t");
        byte by2[]=a4.getBytes();
        bfo.write(by2);
        String s2="null";
        for(k=0;k<3;k++)
        {
            Double fObj = new Double(trmp[i][j][k]);
            String s1 = fObj.toString();
            s2=s1.concat("\t");
            byte by[]=s2.getBytes();
            bfo.write(by);
        }
        bfo.write(sn.getBytes());
    }
    bfo.write(sn.getBytes());
}
bfo.close();
}
/*****MATRIX DISPLAY*****/
public void display(double[][] arr,String s,int count)throws Exception
{
    OutputStream out=new FileOutputStream(s,true);
    BufferedOutputStream bfo=new BufferedOutputStream(out);
    String ETCCount = "\nETC";
    String disp= ETCCount.concat(Integer.toString(count));
    byte b[]=disp.getBytes();
    bfo.write(b);
    String s3="\r\n";
    String s4="\t";
    bfo.write(s3.getBytes());
    bfo.write(s4.getBytes());
    for(int k=0;k<16;k++)

```

```

    {
        String b1 = Integer.toString(k);
        String b2="M";
        String b3=b2.concat(b1);
        String b4=b3.concat("\t\t");
        byte by2[]=b4.getBytes();
        bfo.write(by2);
    }
    bfo.write(s3.getBytes());
    for(int i=0;i<no_tasks;i++)
    {
        String c1 = Integer.toString(i);
        String c2="Task";
        String c3=c2.concat(c1);
        String c4=c3.concat("\t");
        byte by3[]=c4.getBytes();
        bfo.write(by3);
        String s2 = null;
        for(int j=0;j<16;j++)
        {
            Double fObj = new Double(arr[i][j]);
            String s1 = fObj.toString();
            s2=s1.concat("\t");
            byte by[]=s2.getBytes();
            bfo.write(by);//WRITES THE MATRIX TO THE FILE
        }
        bfo.write(s3.getBytes());
    }
    bfo.write(s3.getBytes());
    bfo.close();
}
/*.....MATRIX SORTING-CONSISTENCY.....*/
public double[][] sort(double arr[][])
```

```

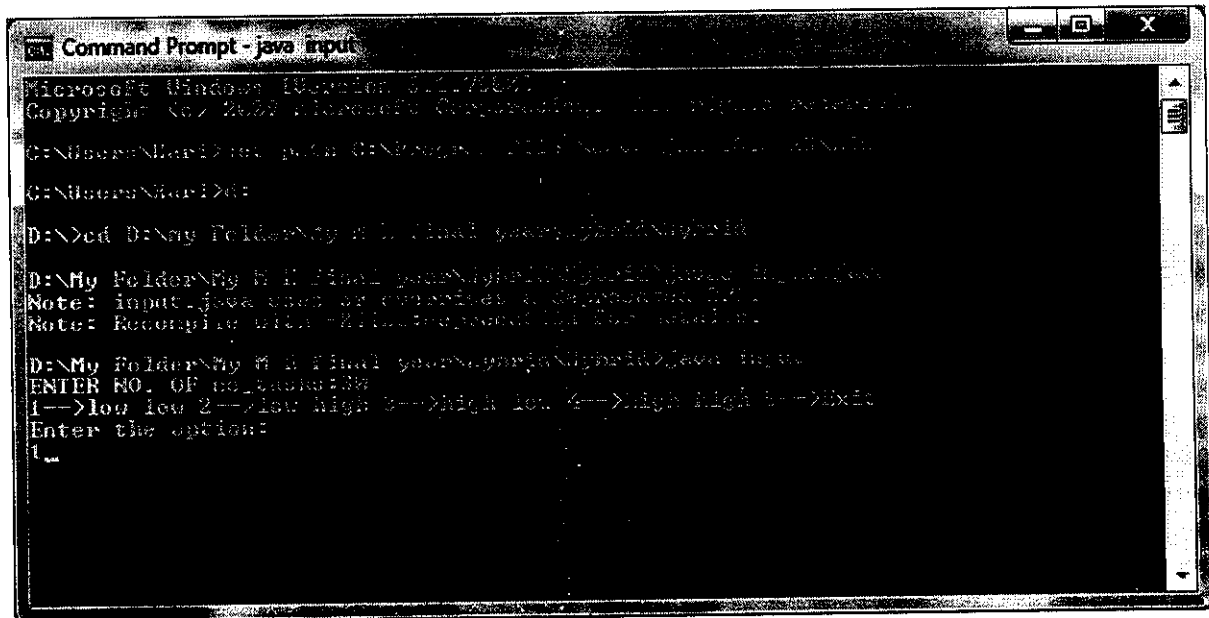
{
    double tmp;
    for(int i=0;i<no_tasks;i++)
    {
        for(int j=0;j<16;j++)
        {
            for(int k=j+1;k<16;k++)
                if(arr[i][j]>arr[i][k])
                {
                    tmp=arr[i][j];
                    arr[i][j]=arr[i][k];
                    arr[i][k]=tmp;
                }
        }
    }
    return arr;
}
/*.....MATRIX SORTING-PARTIALLY CONSISTENCY.....*/
public double[][] partsort(double arr[][])
{
    double tmp;
    for(int i=0;i<no_tasks;i++)
    {
        for(int j=0;j<16;j++)
        {
            if(j%2!=0)
                continue;
            for(int k=i+1;k<no_tasks;k++)
            {
                if(arr[i][j]>arr[k][j])
                {
                    tmp=arr[i][j];
                    arr[i][j]=arr[k][j];

```



```
        arr[k][j]=tmp;
    }
}
}
return arr;
}
/***** ROUND OFF FUNCTION *****/
public static double round(double val, int places)
{
    long factor = (long)Math.pow(10,places);
    val = val * factor;
    long tmp = Math.round(val);
    return (double)tmp / factor;
}
}
```

## 6.2. SNAPSHOTS



```
Command Prompt - java input
Machine Availability
0.5985 0.3427 0.6873 2.1397 0.4214 0.7512 0.2218 0.2214 0.1944
0.3822 0.117 0.885 0.2658 0.1208 0.2208
Makespan: 0.2583
Machine Utilization
0.7977
0.4567
0.6897
0.1865
0.6237
1.0
0.2765
0.515
0.6265
0.2591
0.5094
0.1586
0.4672
0.6804
0.4175
0.3816
Total Resource Utilization: 38.57% x
ANI MAKESPAN:0.6483
Total Resource Utilization: 38.2894 %
```

```
minmakespan - Notepad
File Edit Format View Help
0.4285
0.5849
0.636
0.4818
0.5473
0.6191
0.5306
0.7584
0.5793
0.5665
0.5731
0.6456
0.5283
0.5345
0.6256
0.6605
0.5574
0.5039
0.5786
0.6463
0.4507
0.542
```

gamakespan - Notepad

File Edit Format View Help

```
0. 23078023131349956
0. 23430512629471945
0. 25547897440558265
0. 20054875069519762
0. 7073021868846484
0. 23389889250589702
0. 5865173883770349
0. 3028208117819444
0. 24939022778807973
0. 3817697180042512
0. 43423174969984346
0. 20311685214726383
0. 28332759935204077
0. 23860972784491163
0. 6292098438247636
0. 6202478472662061
0. 6156211356832471
0. 21458412460120524
0. 269654781006227
0. 23846466775647346
0. 5549420261024572
0. 20670967430858334
```

antmakespan - Notepad

File Edit Format View Help

```
0. 3936
0. 6137
0. 3349
0. 2005
0. 5957
0. 2339
0. 7942
0. 6091
0. 4483
0. 7046
0. 2509
0. 2088
0. 5255
0. 3166
0. 835
0. 2949
0. 6156
0. 2126
0. 5915
0. 5298
0. 2002
0. 2623
```

## REFERENCES

- [1] Emir Imamagic, Branimir Radic, Dobrisa Dobrenic, An approach to grid scheduling by using Condor-G matchmaking mechanism, in : Journal of computing and information technology –CIT 14,2006,4,329-336 (2006).
- [2] Ruay-Shiung Chang, Jih- Sheng Chang, Po-Sheng Lin, An ant algorithm for balanced job scheduling in grids, in: future generation computer systems 25(2009) 20-27, Elsevier, science direct. . .
- [3] Dr.K.Vivekanandan, D.Ramyachitra, B.Anbu, Optimization techniques for grid scheduling- a comparative study in: proceedings of the international conference, “computational systems and communication technology” May, 2010.
- [4] Li Weidong, Zhao Gaishan, Wei Hailiang, Xie Xianghui, Wang Yujing and Zhou Huiqing, Application of Grid Computing in Petroleum Exploration in: proceedings of the fifth international conference on grid and cooperative computing workshops, IEEE 2006.
- [5] Albert Zomaya, Opportunities for Grid Computing in Bio- and Health-Informatics, IEEE 2004.
- [6] Vishnu Kant Soni, Raksha Sharma, Manoj Kumar Mishra, An analysis of various job scheduling strategies in grid computing, in: proceedings 2010 second international conference on signal processing systems (ICSPPS), IEEE 2010.
- [7] Joshy Joseph, Craig Fellenstein, Grid computing , pearson education 2004.
- [8] Mr. Rakesh Kumar, Navjot Kaur, Job Scheduling in Grid Computers.
- [9] Asef AL-Khateeb, Rosni Abdullah and Nur'Aini Abdul Rashid, Job Type Approach for Deciding Job Scheduling in Grid Computing Systems, Journal of Computer Science 5 (10): 745-750, 2009.
- [10] <http://www.gridcafe.org/>

- [11] J.G.Webster, "Heterogeneous Distributed Computing," Encyclopaedia of Electrical and Electronics Engineering, Vol. 8, pp. 679-690, 1999.
- [12] Bing Tang, Yingying Yin, Quan Liu and Zude Zhou "Research on the Application of Ant Colony Algorithm in Grid Resource Scheduling" Journal on Wireless Communications, Networking and Mobile Computing, 2008, Page(s):1 - 4.
- [13] D. Kondo, D.P. Anderson, J. McLeod, Performance evaluation of scheduling policies for volunteer computing, in: Proc. IEEE International Conference on e-Science and Grid computing, Dec. 2007, pp. 415-422.
- [14] BOINC website,<http://boinc.berkeley.edu/>.
- [15] D.Feitelson, L.Rudolph, U.Schwiegelshohm, K.Sevcik and P.Wong, "Theory and Practice in Parallel Job Scheduling", JSSPP, pp. 1-34, 1997.
- [16] R. F. Freund, M. Gherrity, S. Ambrosius, M. Camp-bell, M. Halderman,D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet," 7th IEEE Heterogeneous Computing Workshop (HCW'98), pp. 184-199, March 1998
- [17] Kobra Etminani, M. Naghibzadeh, "A Min-Min Max-Min Selective Algorithm for Grid Task Scheduling", Internet 2007,ICI 2007, 3rd IEEE/IFIP International Conference in Central Asia on, Sept. 2007.
- [18] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," 8th IEEE Heterogeneous Computing Workshop (HCW '99), pp. 30-44, San Juan, Puerto Rico, April 1999.
- [19] D. E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley, New York, NY, 1989.
- [20] T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B.Yao, D. Hensgen, and R. Freund. "A comparison study of static mapping heuristics for a class of meta tasks on heterogeneous computing systems", 8th IEEE Heterogeneous Computing Workshop (HCW'99), pages 15-29, April 1999.

[21] T. Loukopoulos, P. Lampsas, P. Sigalas, “ Improved Genetic Algorithms and List Scheduling Techniques for Independent Task Scheduling in Distributed Systems”, 8th International Conference on Parallel and Distributed Computing, Application and Technologies, 2007.

[22] Jamshid Bagherzadeh, Mojtaba Madadyar Adeg “An Improved Ant Algorithm for Grid Scheduling Problem”, International CSI Computer Conference CSICC'09 pp. 323 – 328, 2009

[23] Li Li, Wang Keqi, Zhou Chunnan “An Improved Ant Colony Algorithm Combined with Particle Swarm Optimization Algorithm for Multi-objective Flexible Job Shop scheduling Problem”, International Conference on Machine Vision and Human-machine Interface 2010