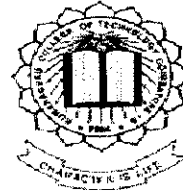




P-3601



# **JOB SCHEDULING IN GRID COMPUTING USING ANT COLONY OPTIMIZATION**

**A PROJECT REPORT**

*Submitted by*

**DIVYA N**

**KAVIYA S**

**NATHIYA P**

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

**BACHELOR OF ENGINEERING**

*in*

**COMPUTER SCIENCE AND ENGINEERING**

**KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE**

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

**APRIL 2011**

**ANNA UNIVERSITY OF TECHNOLOGY: COIMBATORE**

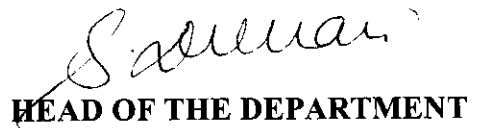
**BONAFIDE CERTIFICATE**

Certified that this project report entitled “**Job Scheduling in Grid Computing using Ant Colony Optimization**” is the bonafide work of Divya N, Kaviya S and Nathiya P who carried out the research under my supervision. Certified also, that to the best of my knowledge the work reported here in 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))**



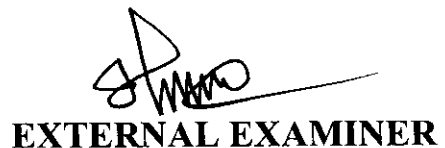
**HEAD OF THE DEPARTMENT**

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

The candidate with University Register Nos. 0710108013, 0710108024 and 0710108029 were examined by us in the project viva-voce examination held on 20.04.2011



**INTERNAL EXAMINER**



**EXTERNAL EXAMINER**

## DECLARATION

We hereby declare that the project entitled "**Job Scheduling in Grid Computing using Ant Colony Optimization**" is a record of original work done by us and to the best of our knowledge, a similar work has not been submitted to Anna University or any Institutions, for fulfillment of the requirement of the course study.

The report is submitted in partial fulfillment of the requirement for the award of the Degree of Bachelor of Computer Science and Engineering of Anna University of Technology, Coimbatore.

Place: Coimbatore

Date: 19.04.2011

*N. Divya*  
(Divya N)

*S. Kaviya*  
(Kaviya S)

*P. Nathiya*  
(Nathiya P)

## ACKNOWLEDGEMENT

We extend our sincere thanks to our Principal, **Dr. S. Ramachandran.**, Kumaraguru College of Technology, Coimbatore, for being a constant source of inspiration and providing us with the necessary facility to work on this project.

We would like to make a special acknowledgement and thanks to **Dr. S. Thangasamy**, Dean R&D, for his support and encouragement throughout the project.

We express our gratitude and gratefulness to our Guide **Ms. P. Devaki, M.E., (Ph.D )**, Head of the Department, Department of Computer Science & Engineering, for her supervision, enduring patience, active involvement and guidance.

We would like to convey our honest thanks to **all Faculty members** of the Department for their enthusiasm and wealth of experience from which we have greatly benefited.

We also thank our **friends and family** who helped us to complete this project fruitfully.

## ABSTRACT

Currently, the users of internet have increased geometrically. Grid computing utilizes the distributed heterogeneous resources in order to support complicated computing problems in a computational grid. The problem of optimally mapping the tasks onto the machines is shown to be NP-complete. Certain assumptions are made for this matching. To increase the efficiency of task distribution to resources in a distributed environment; an efficient scheduling algorithm is needed.

A good scheduler would adjust its scheduling strategy according to changing status of the entire environment and types of jobs. Therefore dynamic algorithm in job scheduling such as Ant Colony Optimization is appropriate for grids.

Ant Colony Optimization (ACO) is an outperforming algorithm and it is compared and analyzed with other existing scheduling algorithms. The Ant colony algorithm imitates the behavior of real ant colonies in nature to search for food and to connect to each other by pheromone laid on the paths travelled. An ant in the ant system is similar to a job in the grid system. The pheromone indicator of the ant system represents the weight of resource in the grid system i.e. the capability of the resource and the value represents the QoS satisfaction of the resource.

The main aim is to reduce the makespan of a given set of jobs and also to reduce the waiting time of the jobs in a distributed environment.

## **LIST OF ABBREVIATIONS**

ETC	-	Expected Time to Compute
MET	-	Minimum Execution Time
MCT	-	Minimum Computation Time
QoS	-	Quality of Service
ACO	-	Ant Colony Optimization
PI	-	Pheromone Indicator

## LIST OF FIGURES

1. Classification of Static task-Scheduling algorithms
2. General ant behavior
3. System Architecture
4. Mapping between ant system and grid system
5. Makespan
  - 5.1 Low Task Low Machine- Inconsistent
  - 5.2. Low Task Low Machine - Partially Consistent
  - 5.3. Low Task Low Machine - Consistent
  - 5.4. Low Task High Machine- Inconsistent
  - 5.5. Low Task High Machine - Partially Consistent
  - 5.6. Low Task High Machine - Consistent
  - 5.7. High Task Low Machine- Inconsistent
  - 5.8. High Task Low Machine - Partially Consistent
  - 5.9. High Task Low Machine - Consistent
  - 5.10. High Task High Machine- Inconsistent
  - 5.11. High Task High Machine - Partially Consistent
  - 5.12 High Task High Machine – Consistent
6. Average Resource Utilization
  - 6.1. Low Task Low Machine- Inconsistent
  - 6.2.Low Task Low Machine Partially Consistent

- 6.3. Low Task Low Machine - Consistent
- 6.4. Low Task High Machine- Inconsistent
- 6.5. Low Task High Machine - Partially Consistent
- 6.6. Low Task High Machine - Consistent
- 6.7. High Task Low Machine- Inconsistent
- 6.8. High Task Low Machine - Partially Consistent
- 6.9. High Task Low Machine - Consistent
- 6.10. High Task High Machine- Inconsistent
- 6.11. High Task High Machine - Partially Consistent
- 6.12. High Task High Machine - Consistent



# TABLE OF CONTENTS

## CHAPTER I

### 1. Overview of Grid Environment

1.1 Introduction	1
1.2 Grid Computing	1
1.3 Classification of Emerging Grids	2
1.4 Benefits of Grid Computing	12
1.5 Issues in Grid Computing	13
1.6 Overview of Task Scheduling	14
1.7 Classification of Static Task Scheduling Algorithms	17

## CHAPTER II

### 2. Problem Overview

2.1 Problem Definition	18
2.2 ETC Matrix Generation	18
2.3 Assumption	21
2.4 Existing Strategies taken Up for Comparison	22
2.4.1 MAX MIN STRATEGY	
2.4.2 MIN MIN STRATEGY	
2.4.3 MCT STRATEGY	
2.4.4 MET STRATEGY	
2.5 Proposed Algorithm	22

## **CHAPTER III**

### **3. Overview of Ant Colony Algorithm**

3.1 General ant behavior	23
3.2 Architecture of the System	23
3.3 The proposed Ant Colony Algorithm	24
3.4 Steps in Ant Colony Algorithm	25
3.5 Ant Colony Algorithm for Task Scheduling	26
3.6 Coding	30

## **CHAPTER IV**

### **4. Experimental Results and Discussion**

4.1 Makespan	46
4.2 Average Resource Utilization	46

## **CHAPTER V**

### **5. Comparison Graphs**

5.1 Makespan Comparison	47
5.2 Average Resource Utilization Comparison	53
5.3 Conclusion	59
5.4 References	60

# **CHAPTER I**

## **1. OVERVIEW OF GRID ENVIRONMENT**

### **1.1 INTRODUCTION**

The growth of internet along with the availability of powerful computers and high speed networks as low cost commodity components is changing the way the scientists and engineers do computing and also is changing how society in general manages information. These new technologies have enabled the clustering of a wide variety of geographically distributed resources, such as supercomputers, storage systems, data sources, instruments. A grid is a collection of resources owned by multiple organizations that are coordinated to allow them to solve a common problem. The grid vision has been described as a world in which computational power (resources, services, data) is as readily available to users with differing levels of expertise in diverse areas and in which these services can interact to perform specified tasks efficiently and securely with minimal human intervention. [1]

### **1.2 GRID COMPUTING**

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 computationally intensive problems. The resources may either be homogeneous or heterogeneous in nature and they are shared from multiple administrative domains. Grid computing can be a cost effective way to resolve IT issues in the areas of data, computing and collaboration; especially if they require enormous amounts of compute power, complex computer processing cycles or access to large data sources. Grid computing needs to be a secure, coordinated

sharing of heterogeneous computing resources across a networked environment that allows users to get their answers faster.

Grid computing requires the use of software that can divide and farm out pieces of a program to as many as several thousand computers. Grid computing can be thought of as distributed and large-scale cluster computing and as a form of network-distributed parallel processing. It can be confined to the network of computer workstations within a corporation or it can be a public collaboration (in which case it is also sometimes known as a form of peer-to-peer computing).

Grids are a form of distributed computing whereby a “super virtual computer” is composed of many networked loosely coupled computers acting in concert to perform very large tasks. This 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 of e-commerce and Web services.

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. It is also true that while a Grid may be dedicated to a specialized application, a single Grid may be used for many different purposes.

### **1.3 A CLASSIFICATION OF EMERGING GRIDS**

In the literature, two characteristics categorize traditional grids: the type of solutions they provide and the scope or size of the underlying organization(s). We propose four additional nomenclatures to facilitate the classification of emerging grids: accessibility, interactivity, user-centricity, and manageability.

## **Grids classified by solution**

The main solution that computational grids offer is CPU cycles. These grids have a highly aggregated computational capacity. Depending on the hardware deployed, computational grids are further classified as desktop, server, or equipment grids. In desktop grids, scattered, idle desktop computer resources constitute a considerable amount of grid resources, whereas in server grids resources are usually limited to those available in servers. An equipment or instrument grid includes a key piece of equipment, such as a telescope. The surrounding grid—a group of electronic devices connected to the equipment—controls the equipment remotely and analyzes the resulting data.

In data grids, the main solutions are storage devices. They provide an infrastructure for accessing, storing, and synchronizing data from distributed data repositories such as digital libraries or data warehouses.

Service or utility grids provide commercial computer services such as CPU cycles and disk storage, which people in the research and enterprise domains can purchase on demand.

Access grids consist of distributed input and output devices, such as speakers, microphones, video cameras, printers, and projectors connected to a grid. These devices provide multiple access points to the grid from which clients can issue requests and receive results in large-scale distributed meetings and training sessions. If clients use wireless or mobile devices to access the grid, it's considered a wireless access grid or a mobile access grid.

## **Grids classified by size**

Global grids are established over the Internet to provide individuals or organizations with grid power anywhere in the world. This is also referred to as Internet computing. Some literature further classifies global grids into voluntary and non-voluntary grids. Voluntary grids offer an efficient solution

for distributed computing. They let Internet users contribute their unused computer resources to collectively accomplish nonprofit, complex scientific computer-based tasks. Resource consumption is strictly limited to the controlling organization or application. On the other hand, non-voluntary grids contain dedicated machines only.

National grids are restricted to the computer resources available within a country's borders. They're available only to organizations of national importance and are usually government funded.

Project grids are also known as enterprise grids or partner's grids. They're structurally similar to national grids, but rather than aggregating resources for a country, they span multiple geographical and administrative domains. They're available only to members and collaborating organizations through a special administrative authority.

Intra-grids or campus grids, in which resources are restricted to those available within a single organization, are only for the host organization's members to use.

Departmental grids are even more restricted than enterprise grids. They're only available to people within the department boundary.

Personal grids have the most limited scope of underlying organization. They're available at a personal level for the owners and other trusted users. Personal grids are still at a very early stage.

### **Accessible grids**

In this context, accessibility means making grid resources available regardless of the access devices' physical capabilities and geographical locations. The highly structured networks of supercomputers and high-performance workstations that dominate grids today typically don't provide such accessibility. In traditional, restricted-access grids, grid nodes are stationary with a predefined wired infrastructure and entry points.

Wireless, mobile, and ad hoc grids have emerged to support grid accessibility. An accessible grid consists of a group of mobile or fixed devices with wired or wireless connectivity and predefined or ad hoc infrastructures.

One of the most critical issues in understanding accessible grids is having an accurate definition, or at least determination, of each grid type (ad hoc, wireless, and mobile). Yet, researchers offer no consistent definition of any of these three terms. Ad hoc grids stress the ad hoc nature of virtual organizations, wireless grids emphasize the wireless connectivity, and mobile grids focus on mobility-related issues such as job migration and data replication.

An accessible grid's main characteristic is its highly dynamic nature, which results from the frequently changing structure of underlying networks and VOs due to nodes switching on and off, nodes entering and leaving, node mobility, and so on. This is why traditional service discovery, management, and security mechanisms might not be optimal for accessible grids.

Accessible grids are accessible from more geographical locations and social settings than traditional grids. This opens the door for new applications in emergency communication, disaster and battlefield management, e-learning, and e-healthcare, among other fields.

### **Ad hoc grids:**

Grids' ad hoc, sporadic nature was observed within the first documented Globus Grid application (see [www.globus.org](http://www.globus.org)). However, traditional grids fail to support certain aspects of ad hoc environments, such as constantly changing membership with a lack of structured communications infrastructure. As a result, ad hoc grids have emerged.

An ad hoc grid is a spontaneous formation of cooperating heterogeneous computing nodes into a logical community without a preconfigured fixed infrastructure and with minimal administrative requirements. Thus, the traditional static grid infrastructure is extended to encompass dynamic additions with no requirements of formal, well-defined, or agreed-upon grid entry points. Instead, nodes can join as long as they can discover other members.

Some researchers strictly define ad hoc grids as grid environments without fixed infrastructures: all their components are mobile. This grid is referred to as a mobile ad hoc grid. However, ad hoc grids focus on the grid's ad hoc nature rather than the nodes' mobility.

Ad hoc grids' main challenge is their dynamic topology, due to the rebooting of workstations and the movement or replacement of computational nodes. Technical details concerning ad hoc grid challenges and implementations are available elsewhere.

### **Wireless grids:**

The wireless grid extends grid resources to wireless devices of varying sizes and capabilities such as sensors, mobile phones, laptops, special instruments, and edge devices. These devices might be statically located, mobile, or nomadic, shifting across institutional boundaries and connected to the grid via nearby devices such as desktops.

Many technical concerns arise when integrating wireless devices into a grid. These include low bandwidth and high security risks, power consumption, and latency. So, several communities, including the Interdisciplinary Wireless Grid Team are exploring these new issues to ensure that future grid peers can be wireless devices.



## **Mobile grids:**

Mobile grids make grid services accessible through mobile devices such as PDAs and smart phones. Researchers usually consider these devices to be at best marginally relevant to grid computing because they're typically resource limited in terms of processing power, persistent storage, runtime heap, battery lifetime, screen size, connectivity, and bandwidth. In contrast, recent studies suggest a very different picture. The millions of mobile devices sold annually shouldn't be ignored, and some mobile devices' raw processing power is not insignificant given their mobility. Furthermore, in emergency situations, such as during natural disasters and on battlefields, wireless mobile devices might be the only available communication and computation services. The most important argument is that it's difficult to materialize the SOKU and AmI visions without using such devices.

As in the case of wireless devices, there are already two approaches to integrating mobile devices into grid systems. In the first approach, the grid includes at least one mobile node that actively participates by providing computational or data services. In the second approach, mobile devices serve as an interface to a stationary grid for sending requests and receiving results. Sometimes this approach is labeled mobile access to grid infrastructure, or simply mobile access grids.

Recently, researchers have made numerous efforts toward establishing mobile grids. You can find details concerning mobile grid requirements and challenges elsewhere. Researchers have proposed various techniques for implementing the mobile grid vision, including centralized and P2P structure, intelligent mobile agents, mobile grid middleware, and many more. Existing mobile grid projects include Akogrimo, ISAM, and MADAM.

## **Interactive grids**

Some potential NGG application areas, such as real-time embedded control systems and video gaming, require rapid response times and online interactivity. The classic request/response communication paradigm of traditional grid systems (such as batch grids) can't accommodate this, so interactive grids are emerging to support real-time interaction.

Interactivity in grid environments can be implemented at two layers: the Web portal layer and the grid middleware layer. In the former, a Web-based grid portal is used to submit interactive jobs to a secure shell process rather than directly to the grid middleware. ScGrid portal falls into this category. In the latter, grid middleware is extended to support interactivity. Examples of this category include Cross Grid and `edutain@grid`.

These examples mainly highlight explicit interactions between a grid and its users, so they're labeled explicit interactive grids. However, this is only one possible form of interaction in grid environments. Another is between a grid and its surroundings to implement a context-aware grid, which uses sensors to interactively build the context and actuators to adapt grid behaviors accordingly. The research agendas of many emerging grid projects in the areas of embedded and pervasive systems, such as RUNES, SENSE, Hydra, and MORE emphasize context awareness.

## **User-centric grids**

Traditional grids are designed specifically for people involved in research and large industry domains. Hence, they lack user centricity and personalization features. Consequently, it's difficult for personal users—that is, individuals outside these domains—to construct or use traditional grids. Most traditional grid systems are non-personalized grids.

Personalized grids are emerging grid systems with highly customizable Web portals that make them adaptable to users' needs. User

centricity is a design philosophy that focuses on the needs of a system's users. *Personalization* is a more restrictive philosophy that aims to adapt the whole system's design to a specific user. In grid computing, user centricity could begin by displaying the user's name on a Web portal, and might end with the personalization of all information, resources, and networks underpinning grids. Research to support user centricity in grid computing is in its infancy.

We use the term *user-centric grids* to refer to two types of emerging grids: personalized and personal. Personalized grids have highly customizable Web portals to provide user-friendly access points to grid resources for people in different domains. For instance, the myGrid project lets scientists establish multiple views that provide access to a user-defined subset of the registered services. These views can be specific to individual scientists or to more specialized discovery services. The Akogrimo project saves all learners' profiles and needs, such as his or her context information, and automatically loads them whenever they sign on, providing a customized, user-friendly environment for each learner. A personal grid is a personalized grid with an underlying VO of limited scope and size. It's used and/or owned by individuals. You can find a framework for a personal grid that consists of a set of networked personal desktop computers elsewhere.

### **Manageable grids**

A grid is highly complex and dynamic, making its management extremely challenging. Traditional grid-management approaches require centralized servers, extensive knowledge of the underlying systems, and a large group of experienced staff. So, grids are emerging with manageability as a main focus.

Centralized grids are traditional grid systems that use a central management scheme. In distributed grids, such as P2P grids, management is distributed.

Manageability is the capacity to manage, organize, heal, and control a system; hence, a manageable grid is a sophisticated grid that automatically manages, adapts, monitors, diagnoses and fixes itself. A manageable system has intelligent control embedded into its infrastructure to automate its management procedure. A variety of technologies are available to support grid manageability at both the hardware and software levels. At the software level, a wide range of techniques, from traditional log files to recent technologies such as Java Management Extensions and knowledge technologies, can support manageability. At the hardware level, technologies from simple embedded sensors to standalone intelligent robots can achieve this. Additionally, changing the underlying grid architecture—for example, from centralized client/server to P2P structures—can support manageability.

Manageable grids offer a simplified installation and greatly reduce configuration and administration, which, in turn, reduces management costs and dramatically enhances scalability. Existing research in this area includes autonomic grids, knowledge grids, and organic grids. Hybrid grids use different combinations of management schemes. For instance, a grid environment might implement a distributed P2P management scheme at the cluster level and a centralized management structure at the higher grid level.

### **Autonomic grids:**

Autonomic computing, initiated by IBM in 2001, is named after the human body's autonomic nervous system. The autonomic nervous system regulates body systems without any external help; likewise, an autonomic computing system controls the functioning of computer systems without user

intervention. The main goal of autonomic computing is to make managing large computing systems (such as grids) less complex.

An autonomic grid can configure, reconfigure, protect, and heal itself under varying and unpredictable conditions and optimize its work to maximize resource use. You can find applications, challenges, and various methods that have been proposed to work toward autonomic grids elsewhere. Examples of autonomic grid projects include the IBM OptimalGrid and AutoMAGI.

P-3601



### **Knowledge grids:**

A knowledge grid is an extension to the current grid in which resources, and services have well-defined meanings that are annotated with semantic metadata so both machines and humans can understand them. The aim is to move the grid from an infrastructure for computation and data management to a pervasive knowledge-management infrastructure. Examples of knowledge grid projects include Onto Grid, InteliGrid, and K-Wf Grid. Several communities are working to realize knowledge grids, including the Semantic Grid Group from the Open Grid Forum. Reviews of the status and future vision of knowledge grids, including applications, challenges, and critical issues, are detailed elsewhere.

### **Organic grids:**

Traditionally, “organic” means forming an integral element of a whole, having systematic coordination of parts, and/or having the characteristics of an organism and developing in the manner of a living plant or animal. In grid computing, the organic grid refers to a new design for desktop grids that relies on a decentralized P2P approach, a distributed scheduling scheme, and mobile agents. The basic idea comes from the manner in which complex patterns can emerge from the interplay of many

agents in an ant colony. However, work on organic grids is at a very early stage.

#### **1.4 BENEFITS OF GRID COMPUTING**

Grid computing appears to be a promising trend for three reasons: (1) its ability to make more cost-effective use of a given amount of computer resources, (2) as a way to solve problems that can't be approached without an enormous amount of computing power, and (3) because it suggests that the resources of many computers can be cooperatively and perhaps synergistically harnessed and managed as a collaboration toward a common objective. In some grid computing systems, the computers may collaborate rather than being directed by one managing computer. One likely area for the use of grid computing will be pervasive computing applications - those in which computers pervade our environment without our necessary awareness.

Moreover the following can be summarized as the merits of Grid Computing

1. Exploiting underutilized resources
2. Parallel CPU capacity
3. Virtual resources and virtual resources for collaboration
4. Access to other resources
5. Resource Balancing
6. Reliability
7. Management

Grid computing enables organizations (real and virtual) to take advantage of various computing resources in ways not previously possible. They can take advantage of underutilized resources to meet business requirements while minimizing additional costs. The nature of a computing grid allows organizations to take advantage of parallel processing, making many applications financially feasible as well as allowing them to complete

sooner. Grid computing makes more resources available to more people and organizations while allowing those responsible for the IT infrastructure to enhance resource balancing, reliability, and manageability. [2]

## 1.5 ISSUES IN GRID COMPUTING

A grid is a **distributed and heterogeneous** environment that 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.

## **1.6 OVERVIEW OF TASK SCHEDULING:**

Scheduling is defined as the problem of allocation of machines over time to competing jobs [3]. The  $m \times n$  job scheduling problem denotes a problem where a set of  $n$  jobs has to be processed on a set of  $m$  machines. Each job consists of a chain of operations, each of which requires a specified processing time on a specific machine. The allocation of system resources to various tasks, known as task scheduling, is a major assignment of the operating system. The system maintains prioritized queues of jobs waiting for CPU time and must decide which job to take from which queue and how much time to allocate to it, so that all jobs are completed in a fair and timely manner.

The task 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.

**Task scheduling in Grids:** In a Grid system

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



## **Job scheduling in grids**

Job scheduling is well studied within the computer operating systems. Most of them can be applied to the grid environment with suitable modifications. In the following we introduce several methods for grids. The FPLTF (Fastest Processor to Largest Task First) algorithm schedules tasks to resources according to the workload of tasks in the grid system. The algorithm needs two main parameters such as the CPU speed of resources and workload of tasks. The scheduler sorts the tasks and resources by their workload and CPU speed then assigns the largest task to the fastest available resource. If there are many tasks with heavy workload, its performance may be very bad. Dynamic FPLTF (DPLTF) is based on the static FPLTF, it gives the highest priority to the largest task.

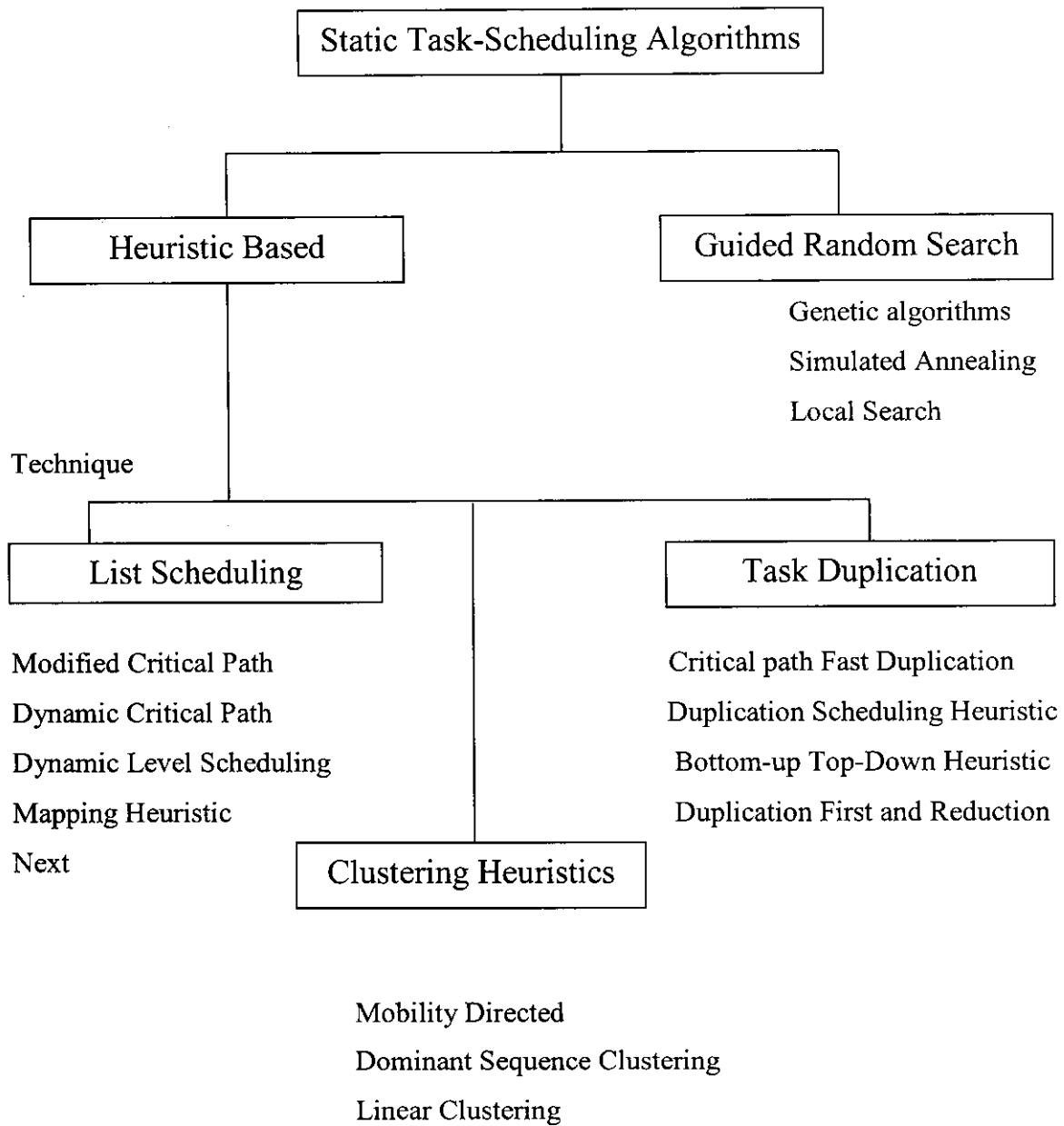
DPLTF needs prediction information on processor speeds and task workload. The WQR (Work Queue with Replication) is based on the work queue (WQ) algorithm. The WQR sets a faster processor with more tasks than a slower processor and it applies FCFS and random transfer to assign resources. WQR replicates tasks in order to transfer to available resources. The amount of replications is defined by the user. When one of the replication tasks is finished, the scheduler will cancel the remaining replication tasks. The WQR's shortcoming is that it takes too much time to execute and transfer replication tasks to resource for execution.

Min-min set the tasks which can be completed earliest with the highest priority. The main idea of Min-min is that it assigns tasks to resources which can execute tasks the fastest. Max-min set the tasks which has the maximum earliest completion time with the highest priority. The main idea of Max-min is that it overlaps the tasks with long running time with the tasks with short

running time. For instance, if there is only one long task, Min-min will execute short tasks in parallel and then execute long task. Max-min will execute short tasks and long task in parallel. The RR (Round Robin) algorithm focuses on the fairness problem. RR uses the ring as its queue to store jobs. Each job in queue has the same execution time and it will be executed in turn. If a job can't be completed during its turn, it will store back to the queue waiting for the next turn. The advantage of RR algorithm is that each job will be executed in turn and they don't have to wait for the previous one to complete. But if the load is heavy, RR will take long time to complete all jobs. Priority scheduling algorithm gives each job a priority value and uses it to dispatch jobs. The priority value of each job depends on the job status such as the requirement of memory sizes, CPU time and so on. The main problem of this algorithm is that it may cause indefinite blocking or starvation if the requirement of a job is never being satisfied.

The FCFS (First Come First Serve) algorithm is a simple job scheduling algorithm. A job which makes the first requirement will be executed first. The main problem of FCFS is its convoy effect. If all jobs are waiting for a big job to finish, the convoy effect occurs. The convoy effect may lead to longer average waiting time and lower resource utilization.

# 1.7 CLASSIFICATION OF STATIC TASK-SCHEDULING ALGORITHMS



**Fig 1: Classification of Static task-Scheduling algorithms**

## CHAPTER II

### PROBLEM OVERVIEW

#### 2.1 PROBLEM DEFINITION:

Given a set of jobs ( $n$ ) with QoS parameters ( Cost, Ram and Deadline) and a set of heterogeneous machines( $m$ ) with their own QoS parameters( Cost, Ram and Deadline) such that(  $m < n$ ), the aim of the job scheduling algorithm is to allocate jobs at nodes so that the total makespan is minimized and the resource utilization is maximized. ACO algorithm is used in order to obtain an optimal solution.

#### 2.2 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  $0 \leq i < t$ . 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  $ETC(t_i, m_j) = W(i) \times X(j)$  for  $0 \leq j < m$ . (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 [4]. 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  $m_{(m-1)j}$  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).[3]

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 [4]: (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 LOW 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

**2.3 ASSUMPTION**

- All jobs/tasks 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.

## **2.4 EXISTING STRATEGIES TAKEN UP FOR COMPARISON**

**2.4.1 MIN MIN STRATEGY: [5][6]**

**2.4.2. MAX MIN STRATEGY: [5][6]**

**2.4.3. MCT STRATEGY: [5][6]**

**2.4.4. MET STRATEGY: [5][6]**

## **2.5 PROPOSED ALGORITHM**

### **ACO Algorithm**

ACO is a heuristic algorithm [7] with efficient local search for combinatorial problems. ACO imitates the behavior of real ant colonies in search for its food and connect to each other by pheromone laid on paths travelled. Many researchers solve NP-hard problems such as Travelling salesman problem, graph coloring problem using ACO approach.

### **Basic Description**

It is assumed that each task is an ant and the algorithm sends the ant to search for resources. The pheromone update is based on the QoS factors and the availability of the resources.



## CHAPTER III

### 3. Overview of Ant Colony Algorithm

#### 3.1 General ant behavior:

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 paths gets evaporated.

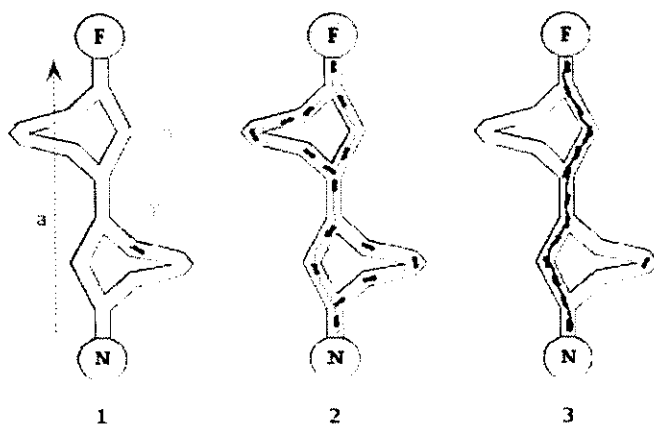


Fig 2: General ant behavior

#### 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.

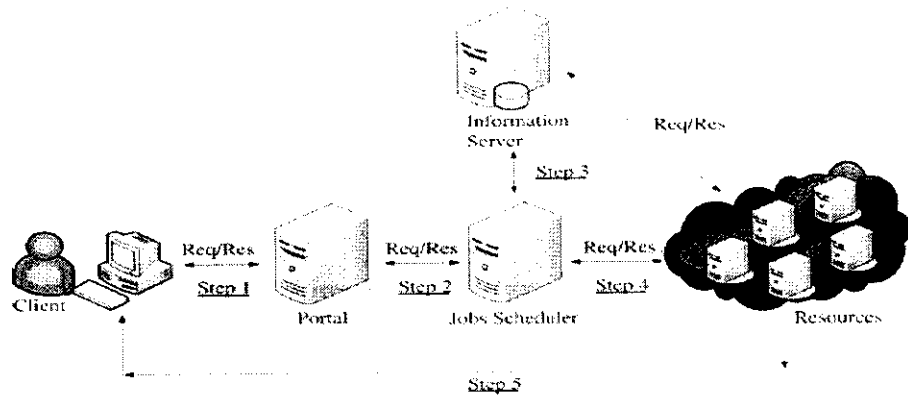


Fig 3: System Architecture

### 3.3 The proposed Ant Colony Algorithm:

The relationship between the ant system and the grid system is mapped as follows.

- a) An ant -An ant in the ant system is a job in the grid system
- b) 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 at the resource has a better computing power.

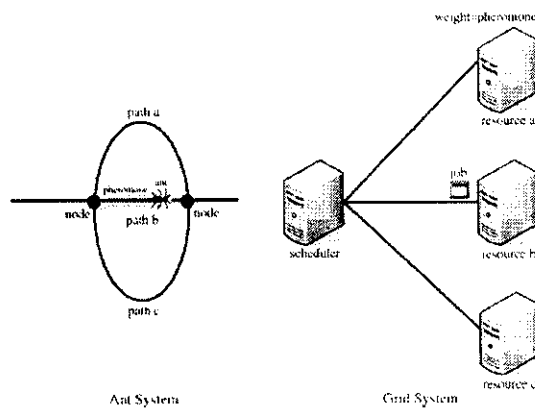


Fig 4: Mapping between the ant system and the grid system.

The scheduler collects data from the information server and uses the data to calculate a weight value of resource. The pheromone (weight) of each resource is stored in the scheduler and the scheduler uses it as a parameter for ACO algorithm. At last the scheduler selects a resource by a scheduling algorithm and sends the job to the selected resource.

### 3.4 Steps in Ant Colony Algorithm:

1. **Input** : ETC matrix of size  $n \times m$ , Task Requirement matrix, Machine capability matrix for QoS(Cost, RAM and Deadline) parameters.
2. Task requirements matrix represents the user requirements of QoS(Cost, RAM and Deadline) factors for executing the particular task.

$$\text{TaskRequirements}_{ik} = \begin{pmatrix} \text{Cost}_1 & \text{RAM}_1 & \text{Deadline}_1 \\ \vdots & \vdots & \vdots \\ \text{Cost}_n & \text{RAM}_n & \text{Deadline}_n \end{pmatrix}$$

Where  $i$  denotes the task and  $k$  denotes the number of QoS parameters.

3. Machine Capability matrix for each machine $_j$ , indicates the QoS factors (Cost, RAM and deadline) associated with machine $_j$  for each task $_i$ . The machine capability matrix for a machine $_j$  is given by,

$$(\text{MachineCapability}_{ik})_j = \begin{pmatrix} \text{Cost}_1 * \text{Deadline}_1 & \text{RAM}_1 & \text{Deadline}_1 \\ \vdots & \vdots & \vdots \\ \text{Cost}_n * \text{Deadline}_n & \text{RAM}_n & \text{Deadline}_n \end{pmatrix}$$

For each machine the  $\text{MachineCapability}_{ik}$  values are calculated.

4. The initial pheromone value of each job across each resource is equal to the pheromone indicator. The initial pheromone indicator value is calculated based on the QoS factor(Cost, RAM and Deadline) values in machine capability and task matrix and guided probability value as,

$$PI_{ij} = \left[ \frac{\text{TaskRequirements}_{i1}}{(\text{MachineCapability}_{i1})_j} \times 0.5 + \frac{(\text{MachineCapability}_{i2})_j}{\text{TaskRequirements}_{i2}} \times 0.25 + \frac{\text{TaskRequirements}_{i3}}{(\text{MachineCapability}_{i3})_j} \times 0.25 \right] * \left[ \frac{1}{\text{ETC}_{ij}} \right]$$

Where  $PI_{ij}$  indicates the pheromone indicator value for task<sub>i</sub> assigned to machine<sub>j</sub>. TaskRequirements<sub>i1</sub> represents the value of user expectation of QoS factor<sub>1</sub> (Cost, RAM and Deadline) for each task<sub>i</sub>. (MachineCapability<sub>i1</sub>)<sub>j</sub> represents the value of machine capability matrix, for the particular machine j, the MachineCapability<sub>i1</sub> indicates the QoSfactor<sub>1</sub> of machine for executing task<sub>i</sub>.

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. Before assigning a task to the resource, the machine availability for each resource is calculated. Based on the minimum machine availability, the task is assigned to the resource.
6. Repeat the step 5 until all tasks are assigned.
7. Calculate the makespan
8. End

### 3.5 Ant Colony Algorithm for Task Scheduling

#### Example:

**Step 1:** Assume there are five jobs  $T_1, T_2, T_3, T_4, T_5$  and three resources  $R_1, R_2, R_3$  in a grid. Sample low low inconsistent matrix is generated. The matrix is

	$R_0$	$R_1$	$R_2$
$T_0$	1.70	0.23	1.13
$T_1$	2.52	0.30	1.74
$T_2$	3.01	0.17	2.26
$T_3$	2.09	0.73	1.59
$T_4$	1.63	0.06	0.65

#### Step 2: User Requirements

Task Requirement matrix for the above sample is:

	Cost	RAM	Deadline
T <sub>0</sub>	1198.09	117.76	2.66
T <sub>1</sub>	840.22	119.80	4.74
T <sub>2</sub>	1057.20	104.44	4.78
T <sub>3</sub>	1119.71	82.37	2.99
T <sub>4</sub>	1171.39	115.52	5.73

### Step 3: Machine Capability

	R <sub>0</sub>			R <sub>1</sub>			R <sub>2</sub>	
	Cost	RAM	Deadline	Cost	RAM	Deadline	Cost	RAM
Deadline								
T <sub>0</sub>	3.47	5.29	1.70	1.02	9.64	0.23	8.82	5.01
			1.13					
T <sub>1</sub>	5.14	5.29	2.52	1.29	9.64	0.30	13.58	5.01
			0.30					
T <sub>2</sub>	6.14	5.29	3.01	0.74	9.64	0.17	17.65	5.01
			2.26					
T <sub>3</sub>	4.25	5.29	2.09	3.14	9.64	0.73	12.43	5.01
			1.59					
T <sub>4</sub>	3.33	5.29	1.63	0.29	9.64	0.06	5.07	5.01
			0.65					

### Step 4: Initial PI Calculation

	R <sub>0</sub>	R <sub>1</sub>	R <sub>2</sub>
T <sub>0</sub>	172.76	595.90	68.42
T <sub>1</sub>	81.83	336.45	31.33
T <sub>2</sub>	86.23	750.84	30.19
T <sub>3</sub>	131.61	179.51	45.32
T <sub>4</sub>	176.36	2304.61	118.82

**Step 5:** The maximum PI value in the matrix above matrix is 2304.61. So task4 is assigned to resource1. The next maximum value PI value is 750.84. Now the availability matrix becomes,

	R <sub>0</sub>	R <sub>1</sub>	R <sub>2</sub>
T <sub>4</sub>	0.00	0.00	0.00
T <sub>2</sub>	3.01	0.24	2.26
Avail	3.01	0.24	2.26

The minimum availability for the above matrix is 0.24. Hence task2 is assigned to resource1. The availability matrix now becomes,

	R <sub>0</sub>	R <sub>1</sub>	R <sub>2</sub>
Avail	0.00	0.24	0.00

The next maximum value is 595.90. Now the availability matrix for each machine becomes,

	R <sub>0</sub>	R <sub>1</sub>	R <sub>2</sub>
	0.00	0.24	0.00
T <sub>0</sub>	1.70	0.23	1.13
Avail	1.70	0.47	1.13

The minimum availability for the above matrix is 0.47. Hence task0 is assigned to resource1. The availability matrix now becomes,

	R <sub>0</sub>	R <sub>1</sub>	R <sub>2</sub>
Avail	0.00	0.47	0.00

The next maximum value is 336.45. Now the availability matrix for each machine becomes,

	R <sub>0</sub>	R <sub>1</sub>	R <sub>2</sub>
	0.00	0.47	0.00
T <sub>1</sub>	2.52	0.30	1.74
Avail	2.52	0.77	1.74

The minimum availability for the above matrix is 0.77. Hence task1 is assigned to resource1. The availability matrix now becomes,

	R <sub>0</sub>	R <sub>1</sub>	R <sub>2</sub>
Avail	0.00	0.77	0.00

The next maximum value is 179.51. Now the availability matrix for each machine becomes,

	R <sub>0</sub>	R <sub>1</sub>	R <sub>2</sub>
	0.00	0.77	0.00
T <sub>3</sub>	2.09	0.73	1.59
Avail	2.09	1.5	1.59

The minimum availability for the above matrix is 1.5. Hence task3 is assigned to resource1. Final availability for each machine is,

$$R_0 \rightarrow 0.00 \quad R_1 \rightarrow 1.5 \quad R_2 \rightarrow 0.00$$

Hence the makespan is **1.5**.

### **Resource Utilization:**

$$\text{Resource Utilization}_i = \text{Resource Availability}_i / \text{Makespan}$$

In this case resource1 is only used. So,

$$\begin{aligned} \text{Resource Utilization}_1 &= 1.5/1.5 \\ &= 1 \end{aligned}$$

Percentage of resource utilization is calculated based on the following formula:

$$\begin{aligned} \text{Resource Utilization (\%)} &= (\text{Resource Utilization} / \text{No. of resources}) * 100 \\ &= (1/3) * 100 \\ &= \mathbf{33.33\%} \end{aligned}$$

### 3.6 CODING:

```
/******ANT ALGORITHM******/  
package ant;  
import java.io.*;  
import java.util.*;  
import java.text.*;  
import java.io.IOException;  
public class antpi  
{  
public static void main(String args[])  
{  
}  
int no_machines=16;  
double temp5[]=new double[512];  
double temp6[]=new double[no_machines];  
int n1,n2;  
int i,j,k,c1,n;  
double no;  
int size;  
double temp1[][]=new double[512][no_machines];  
double ordervector[]=new double[512];  
double etcavail[][]=new double[512][no_machines];  
double temp2[][][]=new double[512][no_machines][3];  
double temp3[][][]=new double[512][no_machines][3];  
double temp8[][][]=new double[512][no_machines][3];  
double temp9[][][]=new double[512][no_machines][3];  
double pidx1[][][]=new double[512][no_machines][3];  
double pir,max1,avmin;  
int j1,s,z;
```



```

double c=0.5,ra=0.25,d=0.25,a=0.0;
int div1=0;
public void piinit(double mach[][][],double tsk1[][][],double et[][][],int
no_tasks,String ip,String pi,String assign,String makespan,String pifinal,int
count,String antuti)throws Exception
{
for(i=0;i<no_tasks;i++)
{
for(j=0;j<no_machines;j++)
{
for(k=0;k<1;k++)
{
if(mach[i][j][k]==0)
{
temp2[i][j][k]=0;
}
else
{
temp2[i][j][k]=(tsk1[i][k]/mach[i][j][k]);//BA
SED ON FORMULA COST=REQ/CAP
temp2[i][j][k]=round(temp2[i][j][k],4);
}
temp2[i][j][k+1]=(mach[i][j][k+1]/tsk1[i][k+1]);
if(mach[i][j][k+2]==0)
temp2[i][j][k+2]=0;
else
temp2[i][j][k+2]=(tsk1[i][k+2]/mach[i][j][k+2]);
}
}
}
}

```

```

    }
    for(n1=0;n1<temp5.length;n1++)
    temp5[n1]=0;
    for(n2=0;n2<temp6.length;n2++)
    temp6[n2]=0;
    temp7=0;
    cnt=0;
    String ss="Iteration ";
    String sn="\r\n";
    OutputStream out=new FileOutputStream(assign,true);
    BufferedOutputStream bfo=new BufferedOutputStream(out);
    String count1=Integer.toString(count);
    String ms=ss.concat(count1);
    byte by[]=ms.getBytes();
    bfo.write(by);
    bfo.write(sn.getBytes());
    bfo.close();
    display2(temp2,ip,no_tasks);
    pical(temp2,et,temp5,temp6,no_tasks,pi,assign,makespan,pifinal,count
,antuti);
    }
    /*****PI CALCULATION*****/
    public void pical(double temp2[][][],double et[][],double temp5[],double
temp6[],int no_tasks,String pi,String assign,String makespan,String
pifinal,int count,String antuti)throws Exception
    {
        for(i=0;i<no_tasks;i++)
        {
            for(j=0;j<no_machines;j++)

```

```

    {
    for(k=0;k<1;k++)
        {
            temp3[i][j][k]=((temp2[i][j][k]*0.5)+(temp2[i][j][k
            +1]*0.25)+(temp2[i][j][k+2]*0.25)*(1/et[i][j]));//
            MULTIPLICATION OF TASK
            temp3[i][j][k]=round(temp3[i][j][k],4);
        }
    }
}
double av[] = new double[no_machines];
double avail[] = new double[no_machines];
double m[] = new double[no_machines];
temp7=0;
for(i=0;i<no_tasks;i++)
    {
        for(j=0;j<no_machines;j++)
            {
                for(k=0;k<1;k++)
                    {
                        temp1[i][j]=temp3[i][j][k];
                        pidx1[i][j][k]=temp3[i][j][k];
                    }
            }
    }
display3(temp3,pi,no_tasks,count);

```

```

t2dconv(temp1,avail,et,temp5,temp6,cnt,av,assign,temp7,temp3,pidx1
,m,no_tasks,pi,makespan,pifinal,count,antuti);

```

```

    }
/*****WRITING 1D ARRAY IN FILE*****/
public void display3(double trmp[][][],String s,int no_tasks,int count)throws
Exception
{
    String sn="\r\n";
    OutputStream out=new FileOutputStream(s,true);
    BufferedOutputStream bfo=new BufferedOutputStream(out);
    String iter = "\nIteration";
    String disp1= iter.concat(Integer.toString(count));
    byte b[]=disp1.getBytes();
    bfo.write(b);
    bfo.write(sn.getBytes());
    for(i=0;i<no_tasks;i++)
        {
            String b1 = Integer.toString(i);
            String b2="task";
            String b3=b2.concat(b1);
            byte by1[]=b3.getBytes();
            bfo.write(by1);
            bfo.write(sn.getBytes());
            for(j=0;j<no_machines;j++)
                {
                    String a1 = Integer.toString(j);
                    String a2="Machine";
                    String a3=a2.concat(a1);
                    String a4=a3.concat("\t");
                    byte by2[]=a4.getBytes();
                    bfo.write(by2);
                }
        }
}

```

```

String s2="null";
for(k=0;k<1;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();
}
}

/***** 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;
}

/*****DISPLAY MACHINE CAPABILITY*****/
public void display2(double trmp[][][],String s,int no_tasks)throws
Exception
{
    OutputStream out=new FileOutputStream(s);

```

```

        BufferedOutputStream bfo=new
BufferedOutputStream(out);
        String sn="\r\n";
        for(i=0;i<no_tasks;i++)
            {
                String b1 = Integer.toString(i);
                String b2="Task";
                String b3=b2.concat(b1);
                byte by1[]=b3.getBytes();
                bfo.write(by1);
                bfo.write(sn.getBytes());
                for(j=0;j<no_machines;j++)
                    {
                        String a1 = Integer.toString(j);
                        String a2="Machine";
                        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();
    }
/*****CALCULATION OF MAXIMUM PI*****/
public void t2dconv(double temp1[][[]],double avail[],double et[][[]],double
temp5[],double temp6[],int cnt,double av[],String s1,int temp7,double[][][]
temp3,double[][][] pidx,double m[],int no_tasks,String pi,String
makespan,String pifinal,int count,String antuti)throws Exception
    {
        pir=maxpi(temp1,no_tasks);
        loop:for(i=0;i<no_tasks;i++)
        {
            for(j=0;j<no_machines;j++)
            {
                if(temp1[i][j]==pir)
                {
                    temp1[i][j]=-1;
                    break loop;
                }
            }
        }
        if((pir==-1)&&(cnt!=no_tasks))
            System.exit(0);
        pi3d(pir,avail,temp3,et,temp1,cnt,temp5,temp6,av,s1,temp7,pidx1,m,n
o_tasks,pi,makespan,pifinal,count,antuti);//SCHEDULING OF JOB
    }

```

```

/***** SCHEDULING *****/
public void pi3d(double pir,double avail[],double pidx[][][],double
et[],double temp1[],int cnt,double temp5[],double temp6[],double
av[],String s1,int temp7,double[][][] pidx1,double m[],int no_tasks,String
pi,String makespan,String pifinal,int count,String antuti)throws Exception
{
    String str="Task ";
    String str1=" Assigned to Machine ";
    int x;
    String sn="\r\n";

    loopb:for(i=0;i<no_tasks;i++)
    {
        loopc:for(j=0;j<no_machines;j++)
        {
            for(k=0;k<1;k++)
            {
                if((pidx[i][j][k]!=-1)&&(pir!=-1))
                {
                    if((pidx[i][j][k]==pir))
                    {
                        pidx[i][j][k]=-1;
                        n=check(temp5,i);
                        if((n==0))
                        {
                            break loopb;
                        }
                        c1=checkmac(temp6,j);
                        if(c1==0)

```



```

        {

for(s=0;s<no_machines;s++)

        {
        avail[s]+=et[i][s];
        }
        avmin=minpindex(avail);
        j1=search(avail,avmin);

loopf:for(s=0;s<no_machines;s++)

        {
        if(s==j1)
        continue loopf;
        else
        {
                avail[s]=avail[s]-et[i][s];
        }
        }

        ordervector[i]=j1;

        OutputStream out=new FileOutputStream(s1,true);
        BufferedOutputStream bfo=new
BufferedOutputStream(out);
        String t11=Integer.toString(i);
        String t21=Integer.toString(j1);
        String t31=str.concat(t11);
String t41=str1.concat(t21);
String t51=t31.concat(t41);
byte by[]=t51.getBytes();

```

```

bfo.write(by);
bfo.write(sn.getBytes());
cnt++;
temp7++;
temp1[i][j1]=-1;
temp5[i]=1;
temp6[j1]=1;
m[j1]+=et[i][j1];
bfo.close();
break loopb;
}
else
{
temp1[i][j]=-1;
ordervector[i]=j;
OutputStream out=new FileOutputStream(s1,true);
BufferedOutputStream bfo=new BufferedOutputStream(out);
String t1=Integer.toString(i);
String t2=Integer.toString(j);
String t3=str.concat(t1);
String t4=str1.concat(t2);
String t5=t3.concat(t4);
byte by[]=t5.getBytes();
bfo.write(by);
bfo.write(sn.getBytes());
cnt++;
temp7++;
m[j]+=et[i][j];
avail[j]=m[j];

```

```

        temp5[i]=1;
        temp6[j]=1;
        bfo.close();
        break loopb;
    }
}
}
}
}
}
}
if(temp7!=no_tasks)
t2dconv(temp1,avail,et,temp5,temp6,cnt,av,s1,temp7,temp3,pidx1,m,no_task
s,pi,
makespan,pifinal,count,antuti);
if(temp7==no_tasks)//IF ALL TASKS ARE ASSIGNED THEN MAKESPAN IS
CALCULATED
{
    makespan(m,makespan,s1,temp5,temp6,antuti,ordervector,no_tasks);
}
}
/***** SETS FLAGS FOR MACHINE AVAILABILITY *****/
public int checkmac(double temp6[],int i)throws Exception
{
    if(temp6[i]==0)
        return 1;//IF AVAILABLE FLAG IS SET
    return 0;//ELSE FLAG IS RESET
}
/***** SETS FLAGS FOR TASKS AVAILABILITY *****/
public int check(double temp5[],int j)throws Exception

```

```

    {
    if(temp5[j]==0)
    return 1;//IF AVAILABLE FLAG IS SET
    return 0;//ELSE FLAG IS RESET
    }

/*****FINDING THE MAXIMUM PI*****/
public double maxpi(double trmp[][[]],int no_tasks)throws Exception
    {
    double max=trmp[0][0];
    for(i=0;i<no_tasks;i++)
        {
        for(j=0;j<trmp[i].length;j++)
            {
            if(trmp[i][j]>=max)
            max=trmp[i][j];
            }
        }
    return max;
    }

/*****SEARCH FUNCTION*****/
public int search(double temp1[],double max)throws Exception
    {
    loop:for(j=0;j<no_machines;j++)
        {
        if(temp1[j]==max)
        break loop;
        }
    return j;
    }

```

```
/******CALCULATE MAXIMUM FOR A GIVEN INDEX******/
```

```
public double maxpindex(double temp1[][],int i)throws Exception
```

```
{
```

```
    double max=temp1[i][0];
```

```
    for(j=0;j<temp1[i].length;j++)
```

```
    {
```

```
        if(temp1[i][j]>=max)
```

```
        max=temp1[i][j];
```

```
    }
```

```
    return max;
```

```
}
```

```
/******CALCULATE MINIMUM FOR A GIVEN INDEX******/
```

```
public double minpindex(double t[])throws Exception
```

```
{
```

```
    double minimum = t[0];
```

```
    for(z=0;z<t.length;z++)
```

```
    {
```

```
        if (t[z] <minimum)
```

```
        {
```

```
            minimum = t[z]; // new maximum
```

```
        }
```

```
    }
```

```
    return minimum;}
```

```
/******MAKESPAN******/
```

```
public void makespan(double m[],String makespan,String ms,double
```

```
temp5[],double temp6[],String antuti,double ordervector[],int
```

```
no_tasks)throws Exception
```

```
{
```

```
    String ss="Makespan-->";
```

```

double max;
OutputStream out=new FileOutputStream(makespan,true);
BufferedOutputStream bfo=new BufferedOutputStream(out);
String sn1="\r\n";
max=round(maximum(m),4);
System.out.println("ANT MAKESPAN:" +max);
String m2=Double.toString(max);
byte by[]=m2.getBytes();
bfo.write(by);
bfo.write(sn1.getBytes());
bfo.close();

OutputStream out1=new FileOutputStream(ms,true);
BufferedOutputStream bfo1=new BufferedOutputStream(out1);
String ms1=Double.toString(max);
String max1=ss.concat(ms1);
byte by1[]=max1.getBytes();
bfo1.write(by1);
bfo1.write(sn1.getBytes());
bfo1.write(sn1.getBytes());
bfo1.close();

MachineUtilization(max,m,no_machines,antuti);
for(n1=0;n1<temp5.length;n1++)
temp5[n1]=0;
for(n2=0;n2<temp6.length;n2++)
temp6[n2]=0;
temp7=0;
cnt=0;
}
/*****MACHINE UTILIZATION*****/

```

```

public void MachineUtilization(double makespan,double machinecap[],int
no_machines,String uti)throws Exception
{
    OutputStream out=new FileOutputStream(uti,true);
    BufferedOutputStream bfo=new BufferedOutputStream(out);
    String sn1="\r\n";
    double machineUtil[]=new double[no_machines];
    double totalResourceUtil=0.0;
    for(int i=0;i<no_machines;i++)
    {
        machineUtil[i]=round(machinecap[i]/makespan,4);
        totalResourceUtil+=machineUtil[i];
    }
    totalResourceUtil=round(totalResourceUtil/no_machines *
100,4);

    System.out.println("Total Resource Utilization:
"+totalResourceUtil+ " %");
    String m2=Double.toString(totalResourceUtil);
    byte by[]=m2.getBytes();
    bfo.write(by);
    bfo.write(sn1.getBytes());
    bfo.close();
}}

```

## **CHAPTER IV**

### **Experimental Results and Discussion**

#### **4.1 Makespan**

Makespan is a measure of the throughput of the heterogeneous computing systems, such as grid. It can be calculated as the following relation:

$$\text{Makespan}=\text{MAX}(CT_i)$$

The less the makespan of a scheduling algorithm, the better it works. [5]

#### **4.2 Average Resource Utilization**

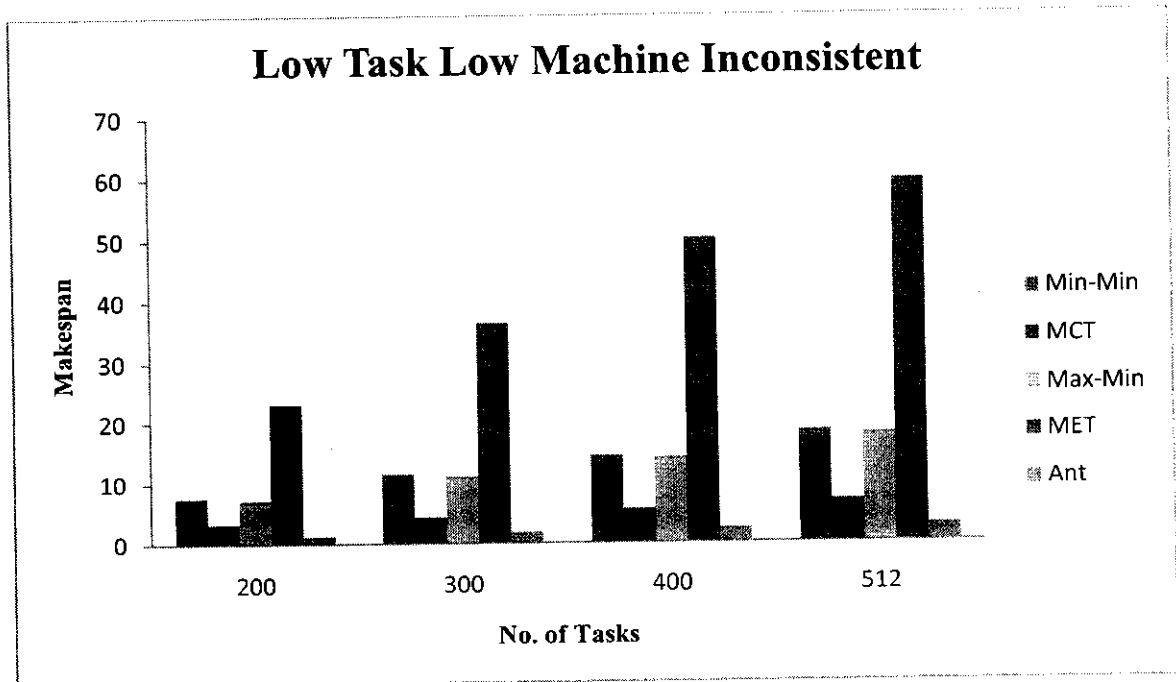
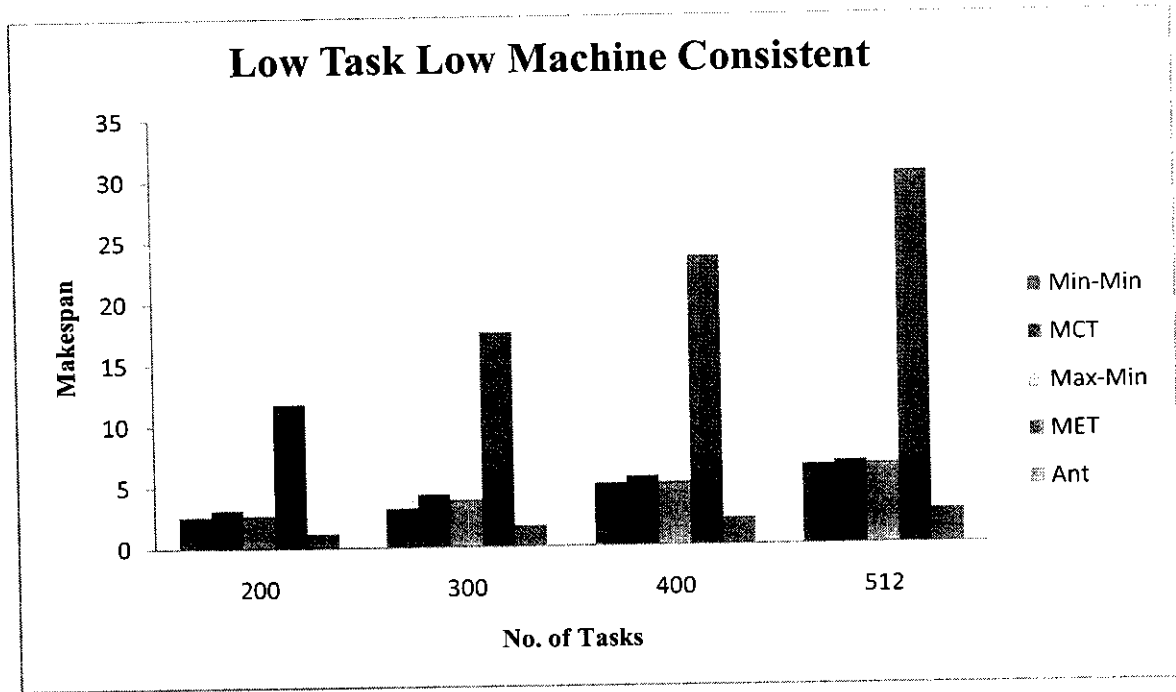
The capability of the software product to use appropriate amounts and types of resources, for example the amounts of main and secondary memory used by the program and the sizes of required temporary or overflow files, when the software performs its function under stated conditions.

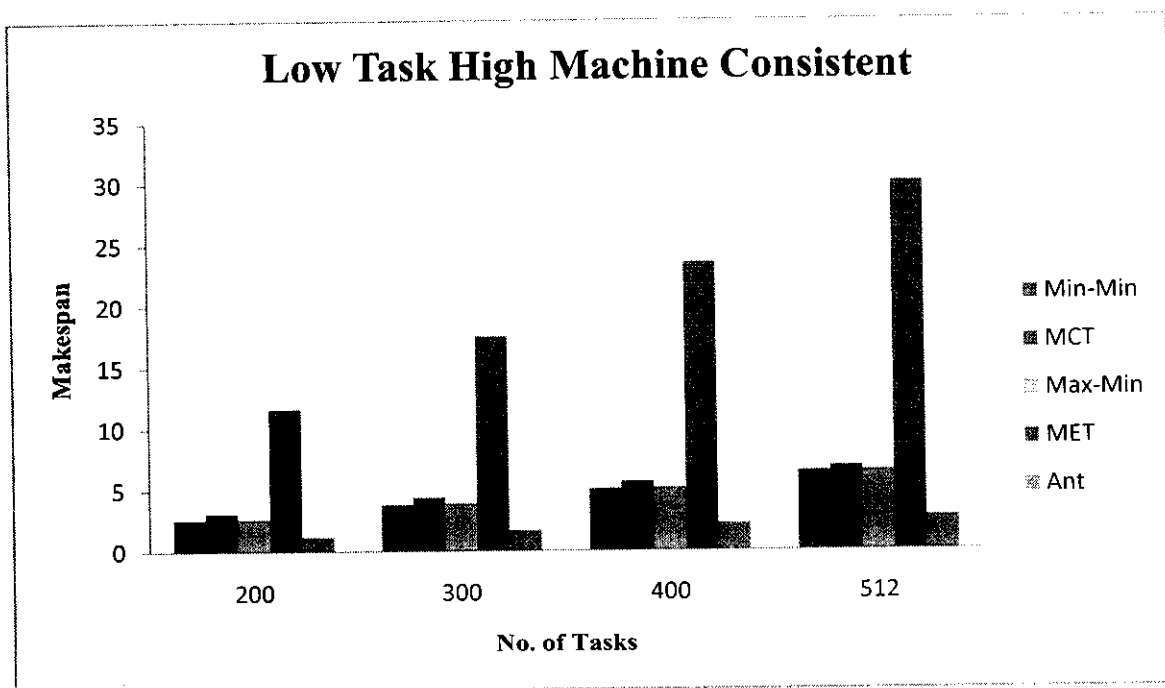
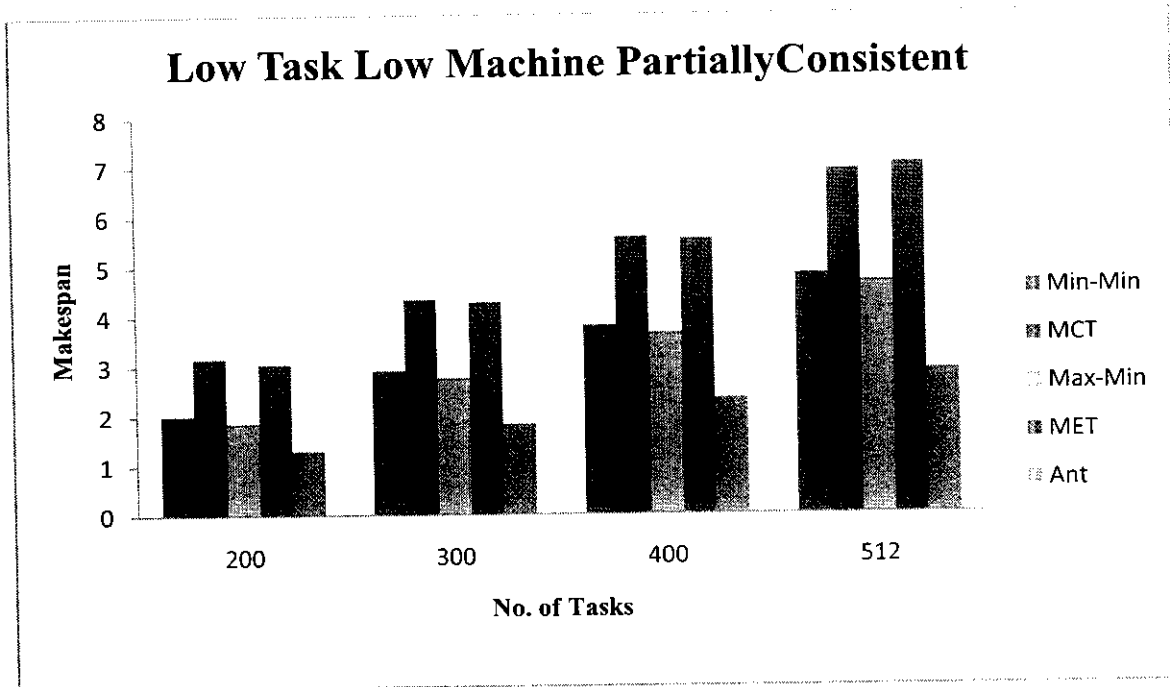


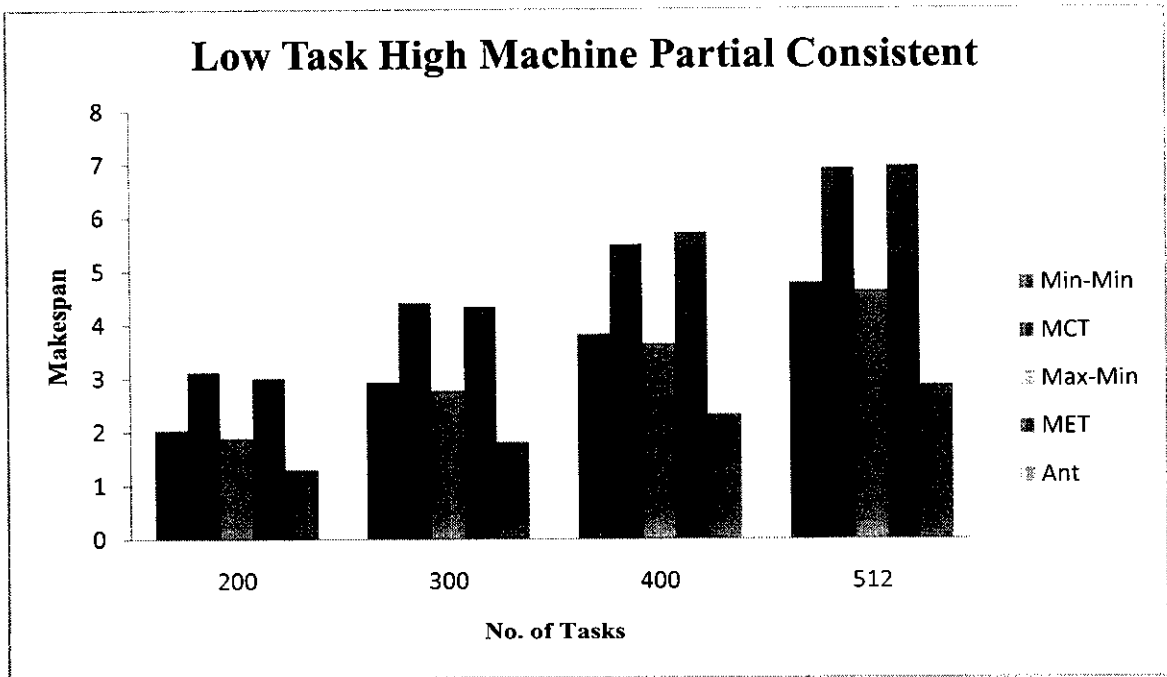
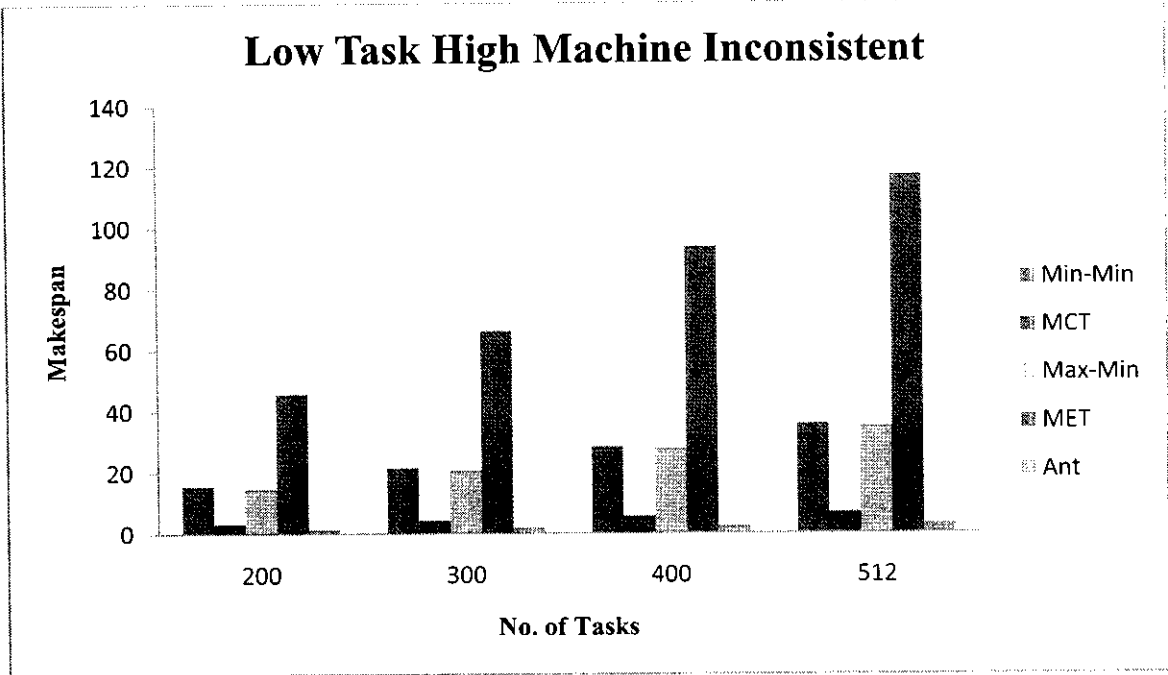
# CHAPTER V

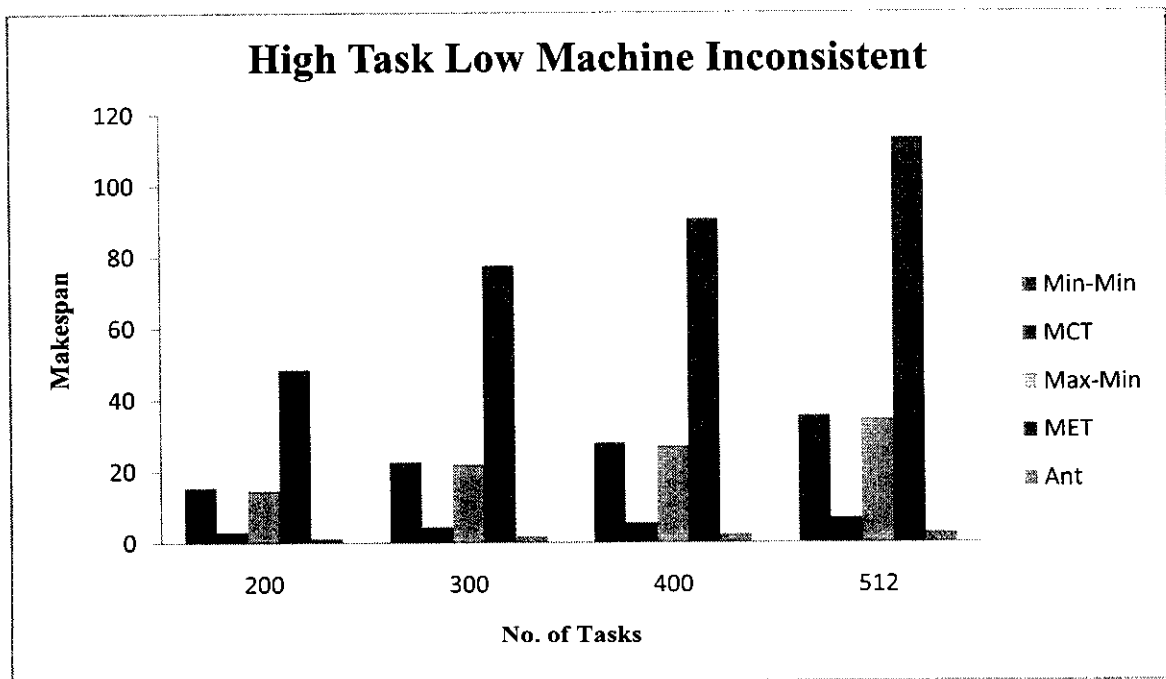
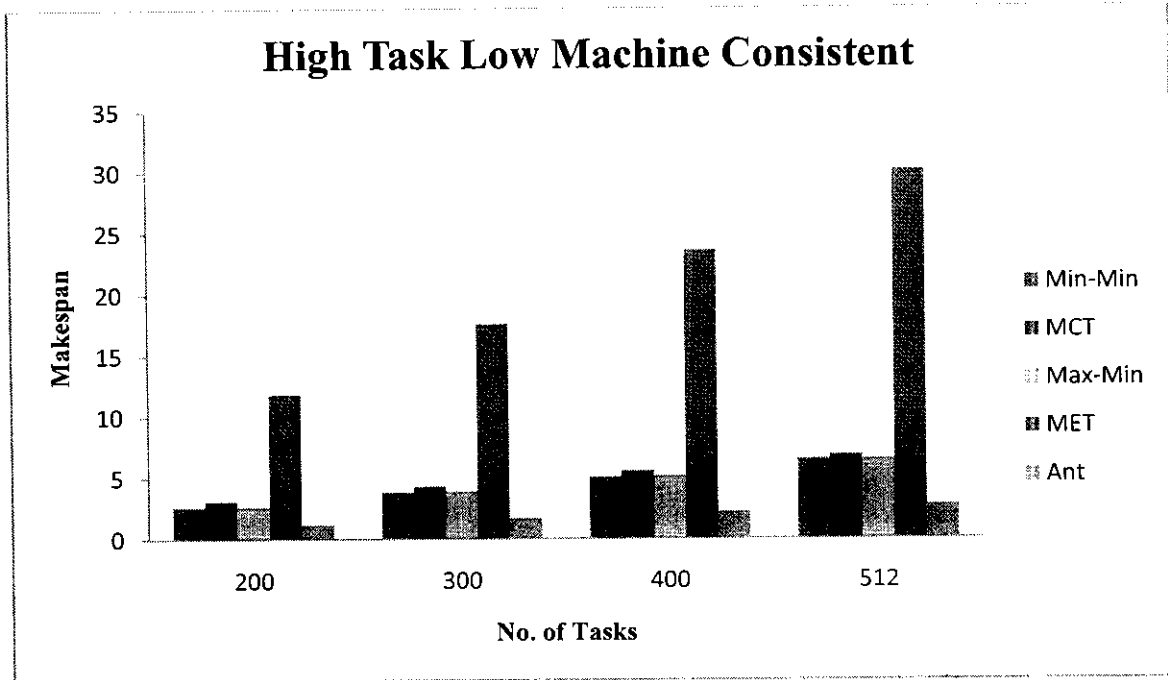
## 5. Comparison Graphs

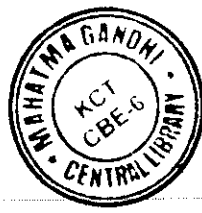
### 5.1 Makespan Comparison





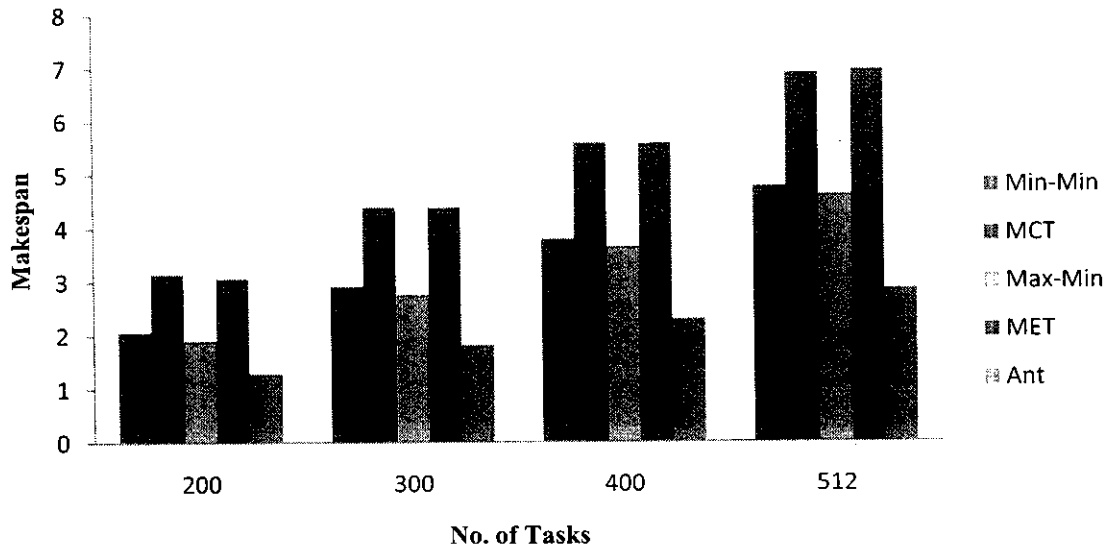




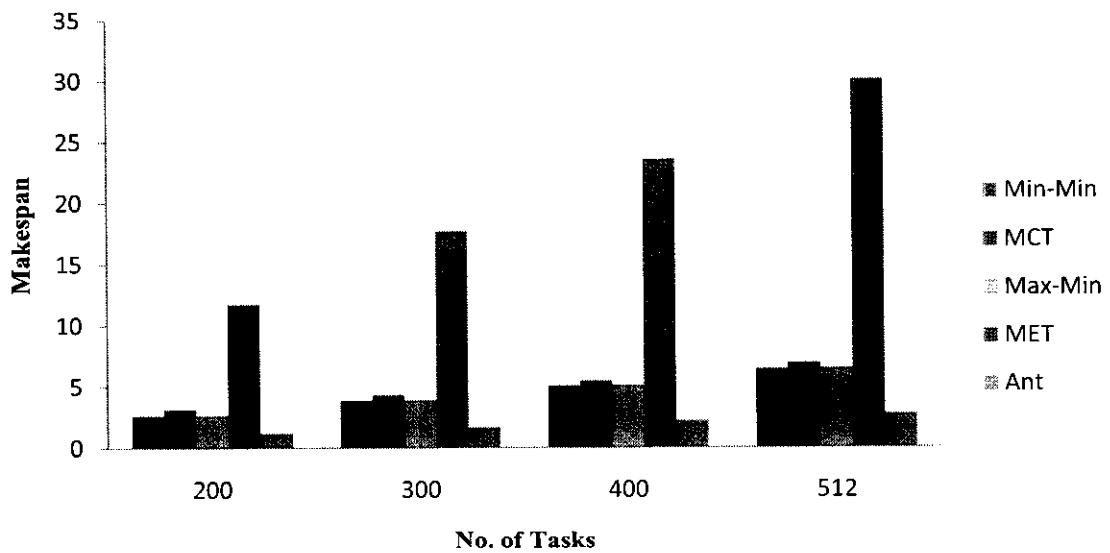


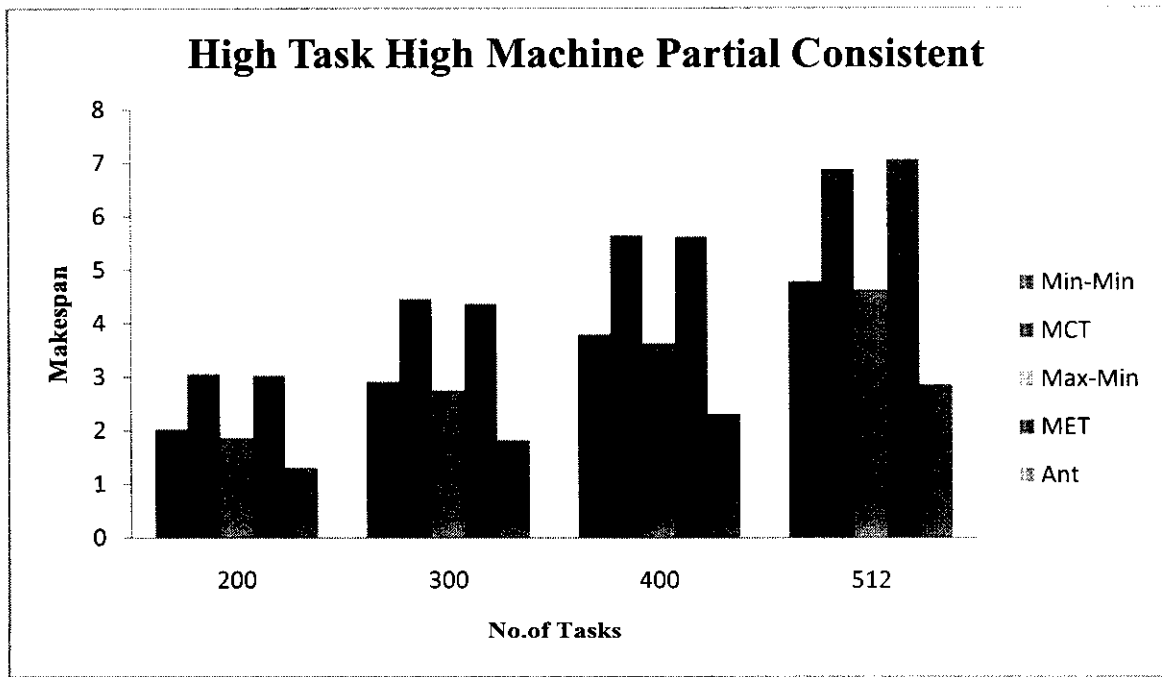
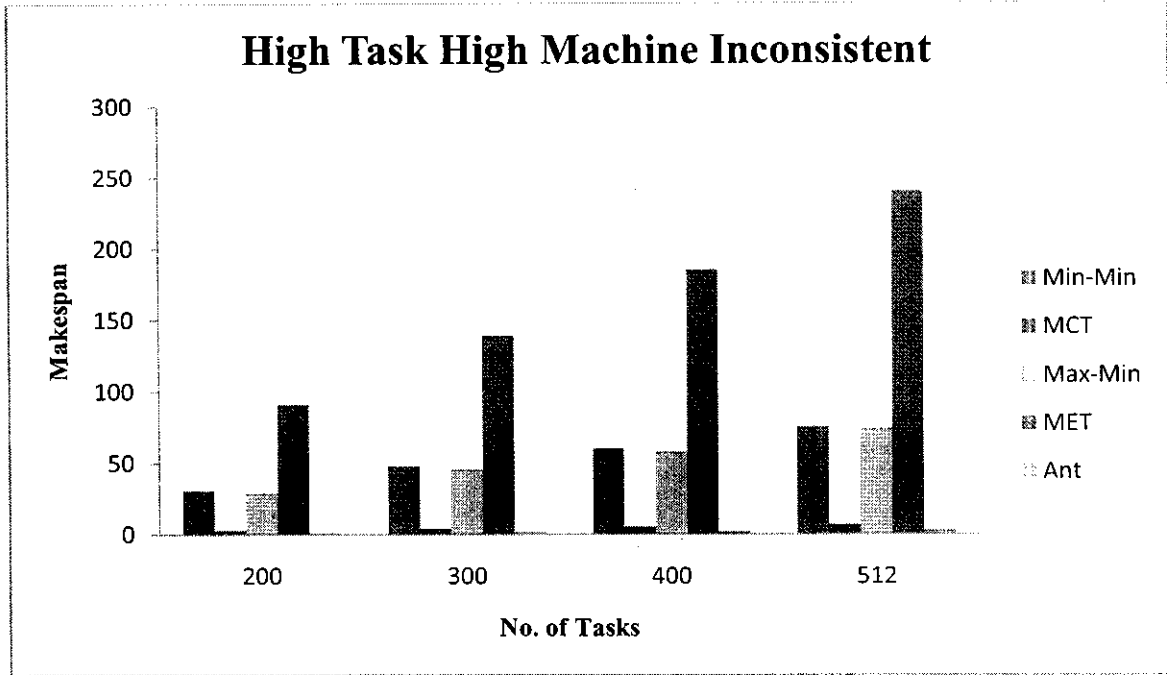
P. 3601

### High Task Low Machine Partial Consistent

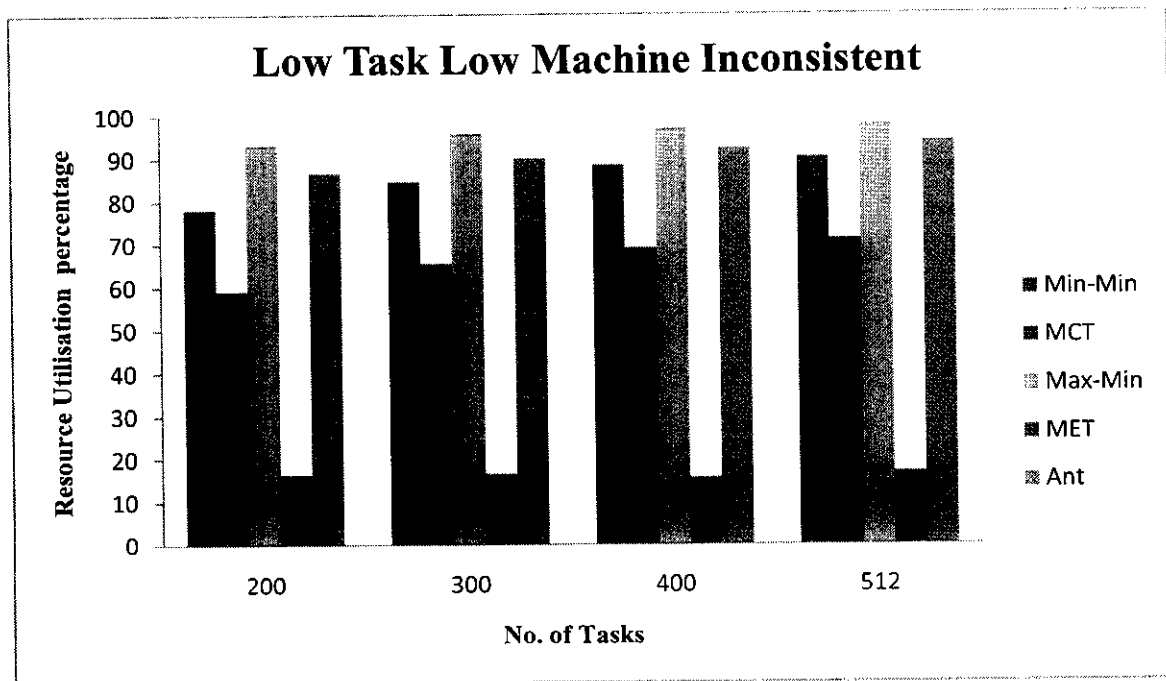
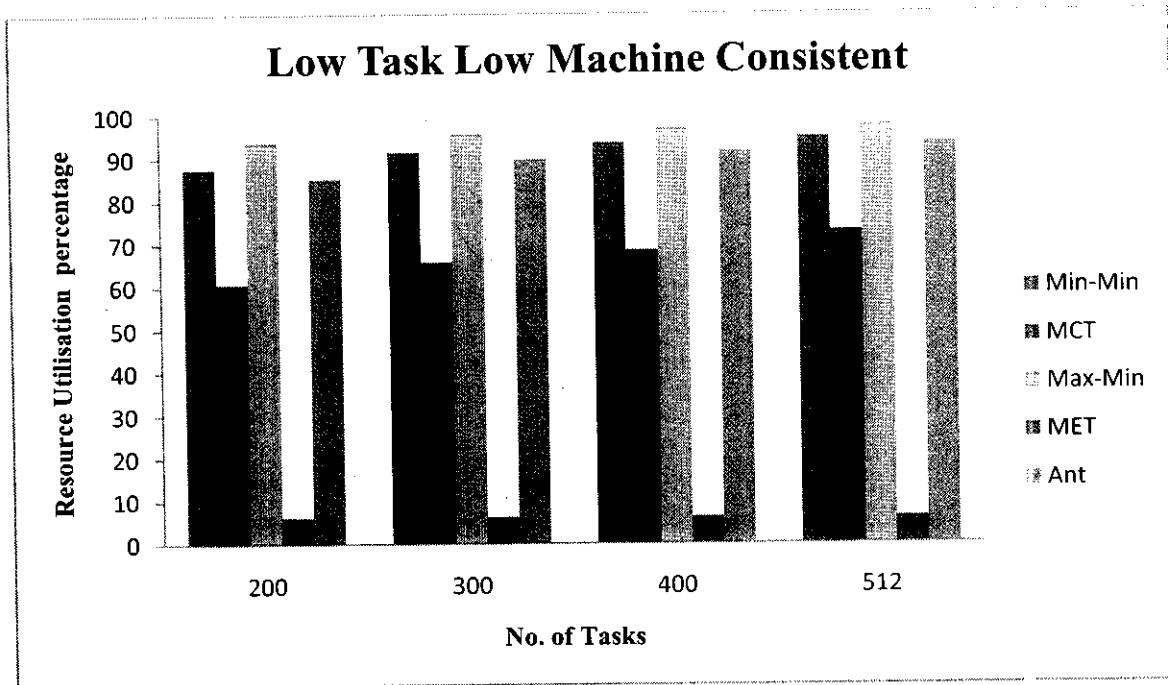


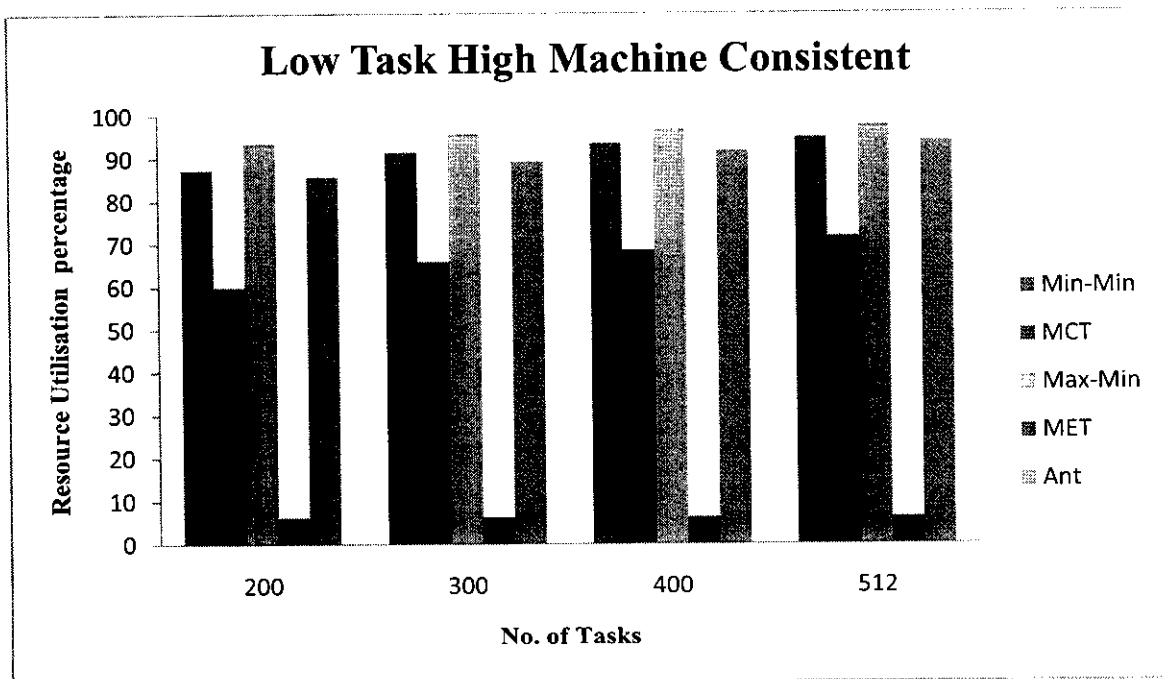
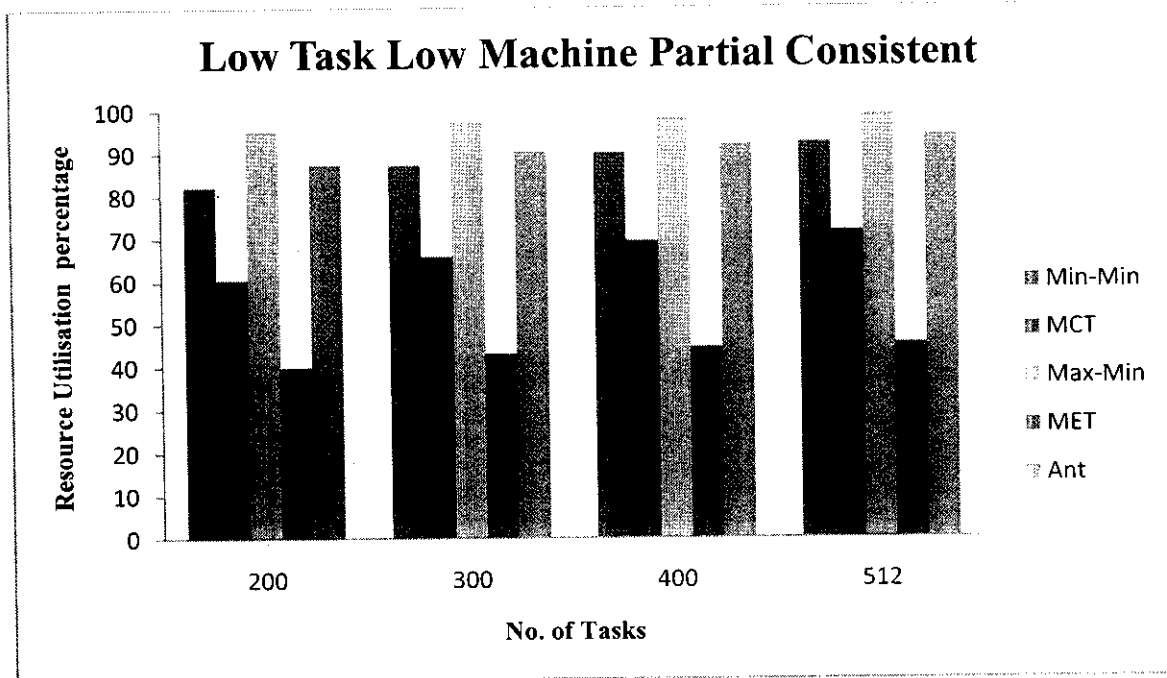
### High Task High Machine Consistent



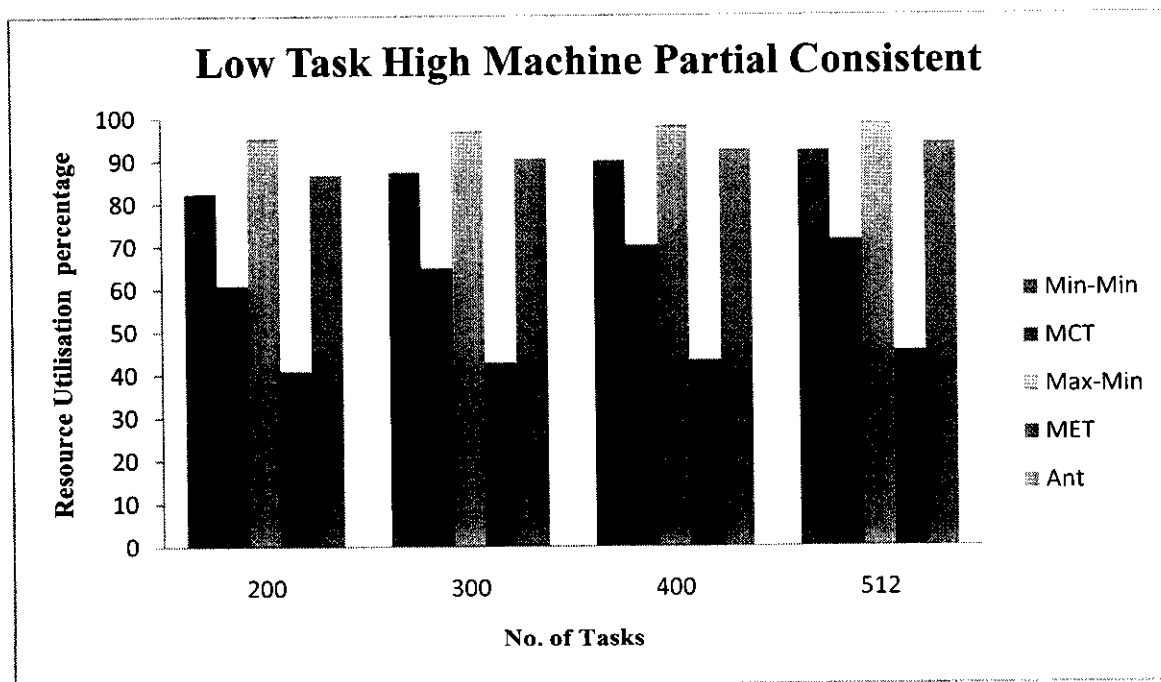
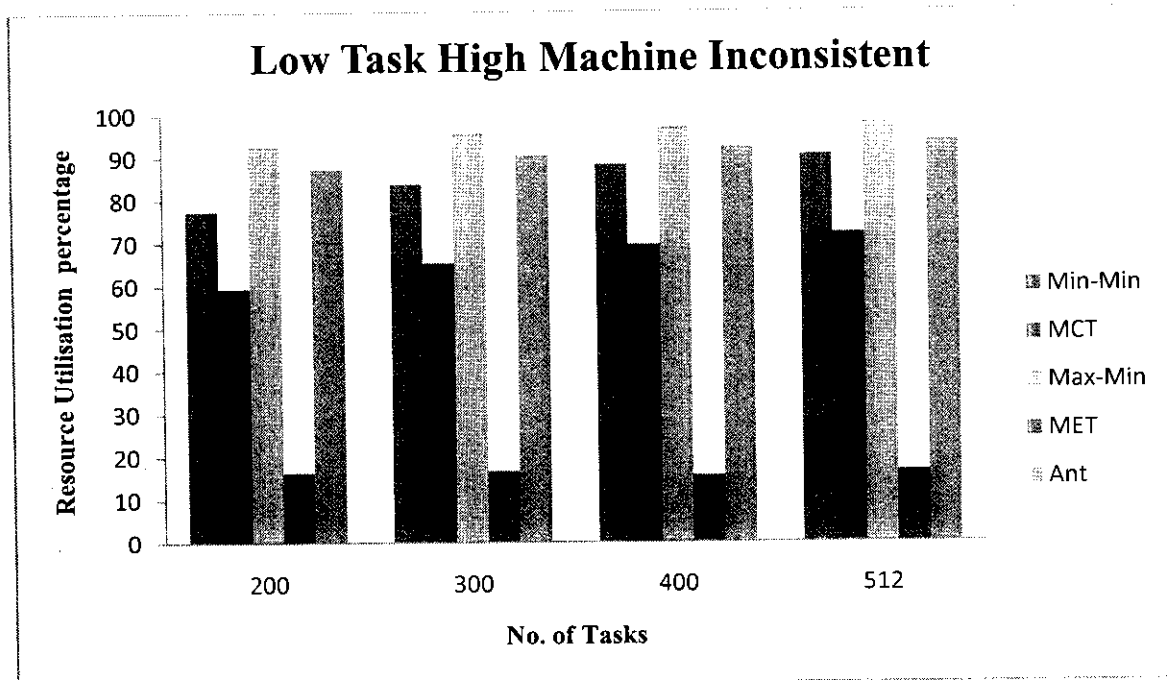


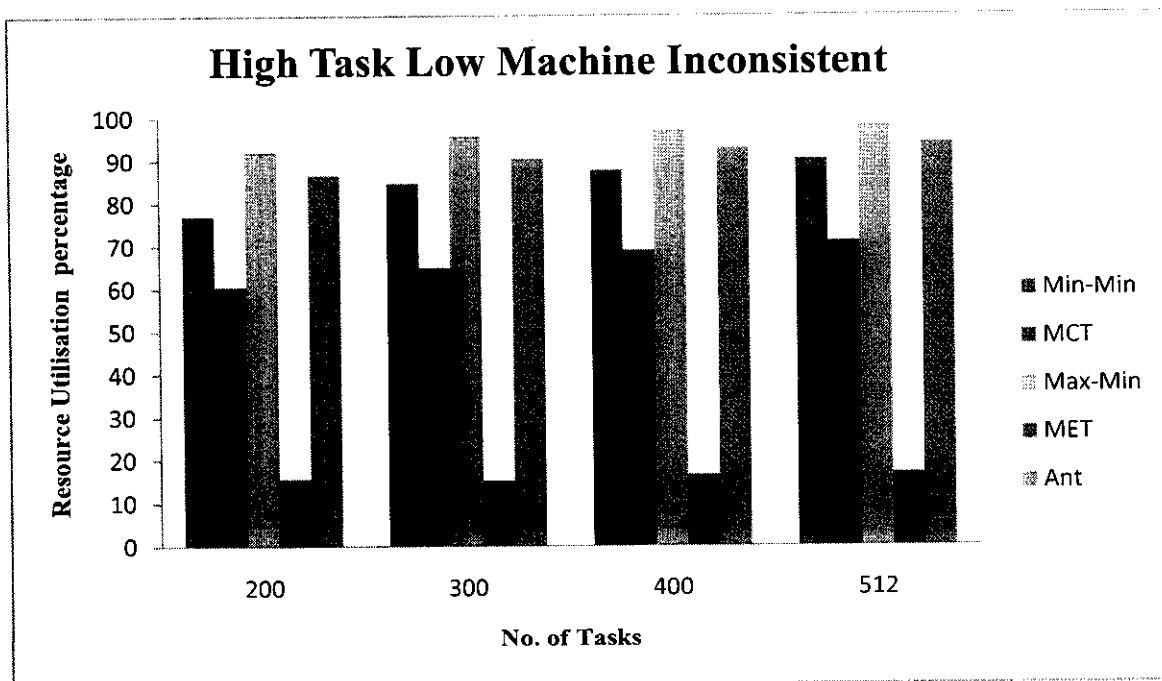
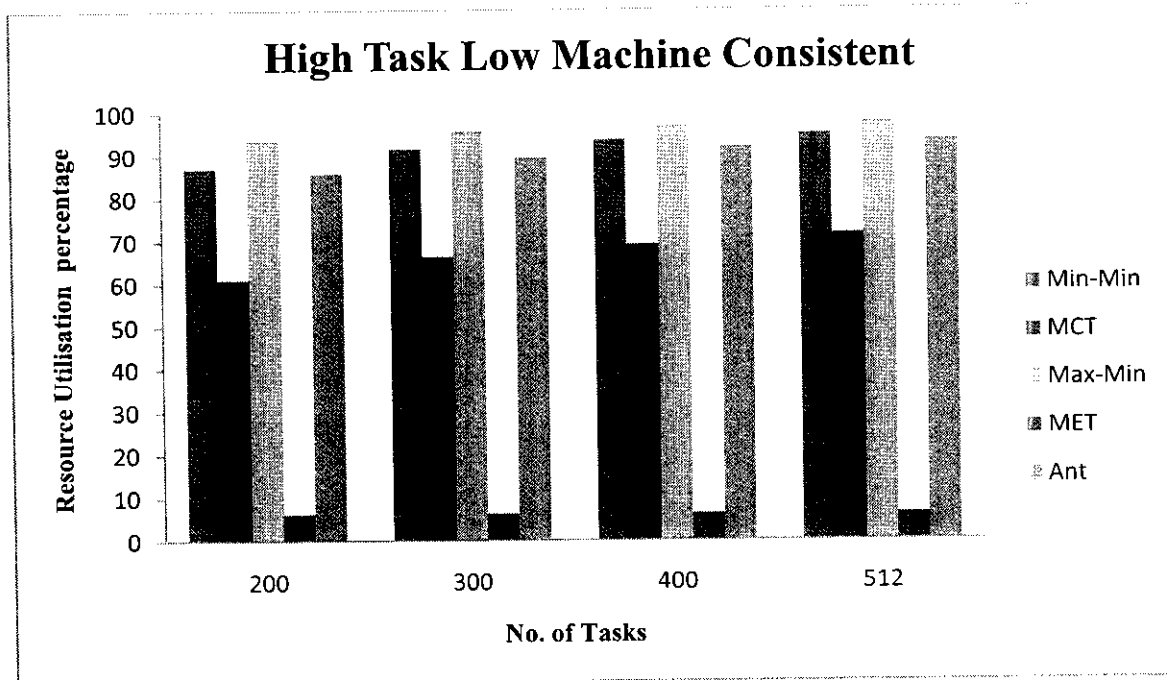
## 5.2 Average Resource Utilization Comparison

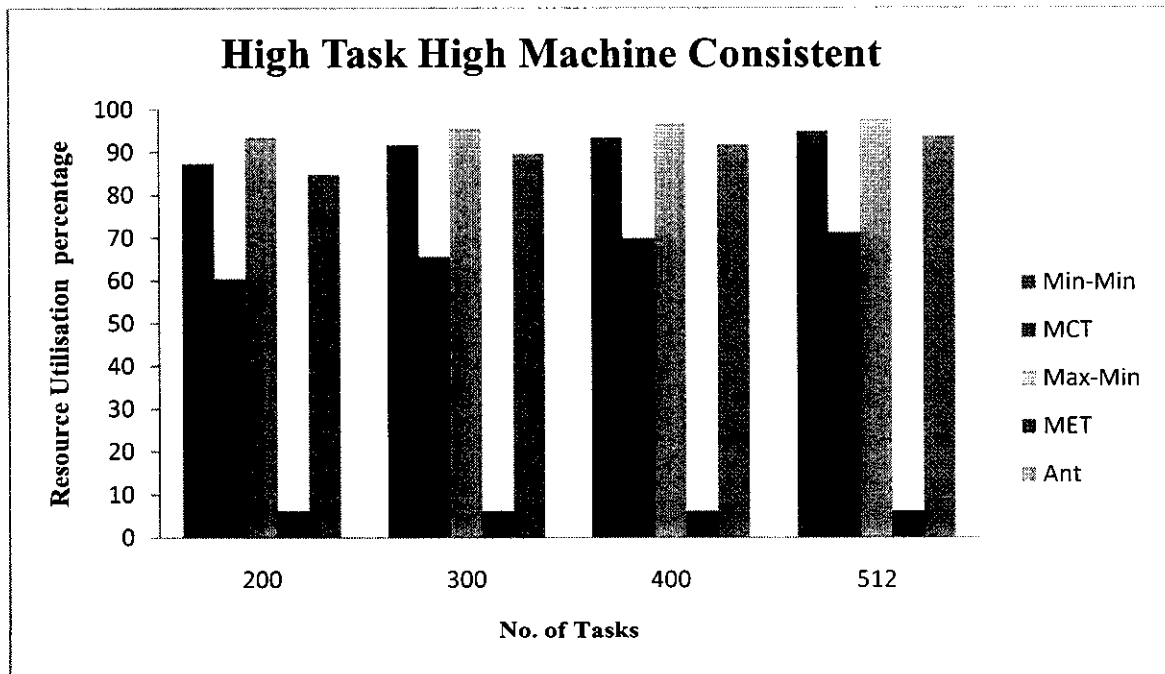
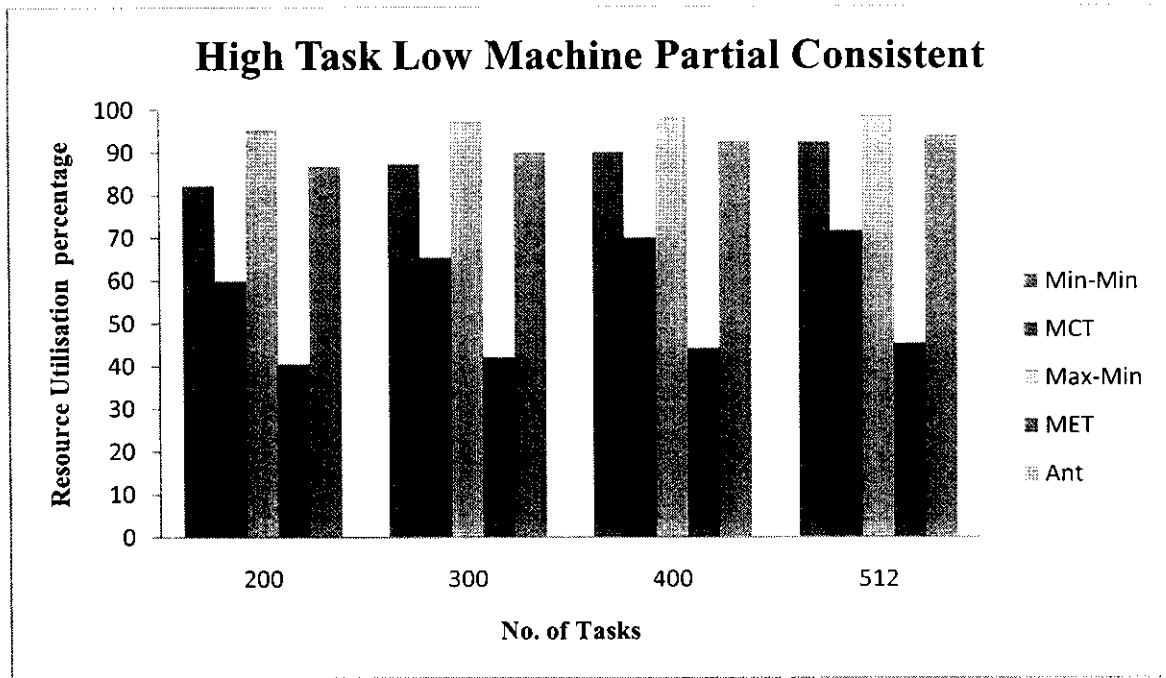


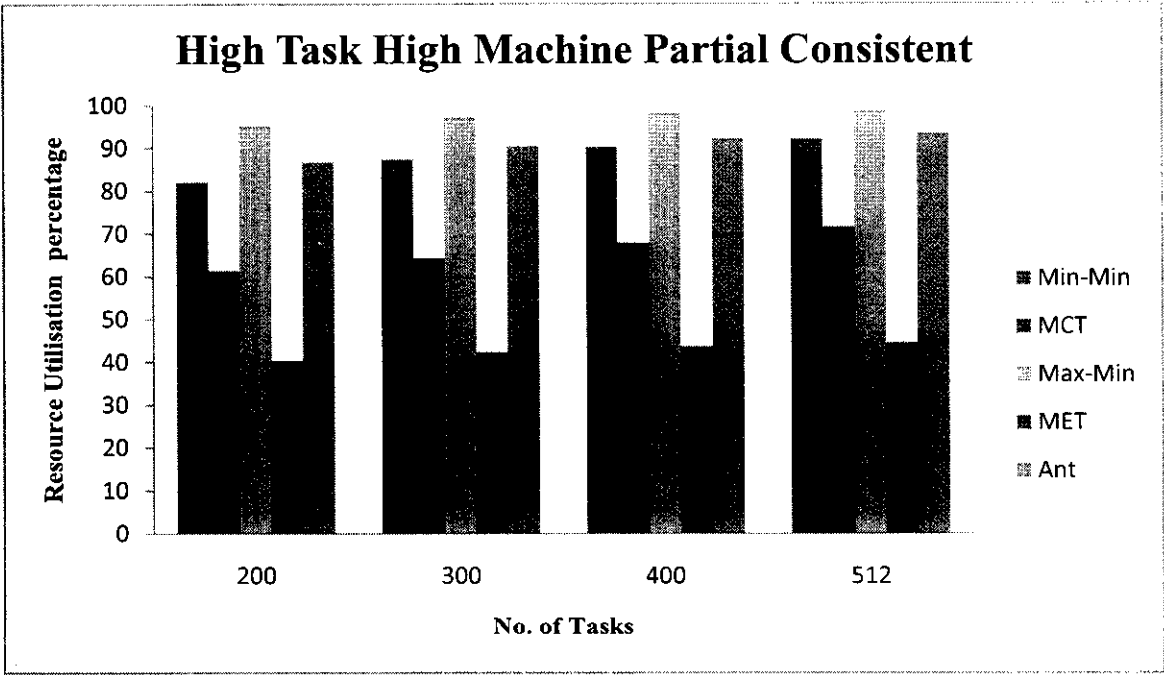
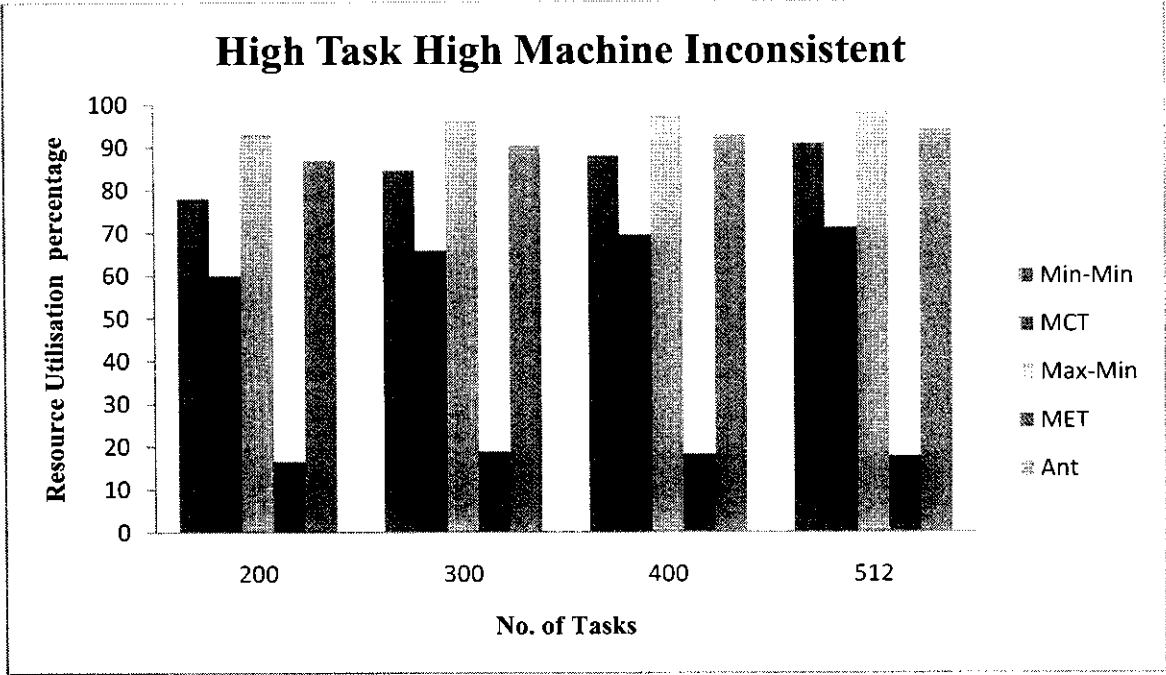












### **5.3 Conclusion**

In this project, we have presented weighted QoS Ant colony, which is an extension of existing Ant colony algorithms. Our algorithm yields significant improvements in performance over existing heuristic algorithms in most of the cases and minimizes the makespan and utilizes the resources effectively. This algorithm first checks the QoS Satisfying criteria and then schedules the task to the relevant machine. We have published our results in the International Conference on Emerging Trends in Computing held at Sri Ramakrishna Engineering College, Coimbatore.

## 5.4 References

- [1] Manish Parashar, Senior Member , IEEE and Craig A.Lee, Member, IEEE, “Grid Computing: Introduction and Overview”
- [2] [www.redbooks.ibm.com](http://www.redbooks.ibm.com)
- [3] Tracy D. Braun, Howard Jay Siegel , “A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems”, Noah Beck School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana 47907-1285
- [4] Armstrong. R, “Investigation of Effect of Different Run-Time Distributions on Smart-Net Performance”, (1997).
- [5] A Kobra Etminani, Prof. M. Naghibzadeh Dept. of Computer Engineering Ferdowsi University of Mashad Mashad, Iran ,Prof. M. Naghibzadeh Dept. of Computer Engineering Ferdowsi University of Mashad ,Mashad, Iran, “Min-Min Max-Min Selective Algorithm for Grid Task Scheduling”.
- [6] Kousalya.K and Balasubramanie.P, “An enhanced ant algorithm for grid scheduling problem”.
- [7] Ruay-Shiung Chang, Jih Sheng Chang, Po-Sheng Lin, “An algorithm for balanced job scheduling in grids”, Future Generation Computer Systems 25(2009) 20-27.