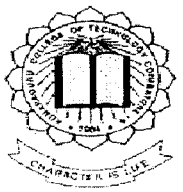
P-3126



TASK SCHEDULING IN GRID COMPUTING ENVIRONMENT USING ANT COLONY ALGORITHM

A PROJECT REPORT

Submitted by

KARTHIKA.K

KAVYA.D

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY, CHENNAI-600 025

MAY 2010

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report entitled "**Task scheduling in Grid Computing Environment using Ant Colony Algorithm**" is the bonafide work of Karthika.K and Kavya.D, who carried out the research under my supervision. Certified also, 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.

SIGNATURE

Dr.S.Thangasamy, Phd

HEAD OF THE DEPARTMENT

Department of Computer

Science and Engineering

Kumaraguru College of Technology

Coimbatore-641006

SIGNATURE

Mrs.P.Devaki

SUPERVISOR

Asst.Professor

Department of Computer

Science and Engineering

Kumaraguru College of Technology

Coimbatore-641006

The candidate with University Register Nos. 71206104022 and 71206104024 were examined by us in the project viva-voce examination held on

16.4.2010

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION

We hereby declare that the project entitled "**Task scheduling using Ant Colony Algorithm in Grid Computing Environment**" 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, Chennai.

K. Karthika

(Karthika.K)

Place: Coimbatore

Date: 16.4.2010

(Kavya.D)

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, Ph.D.,** Dean, Professor and Head of Department of Computer Science &Engineering, for his support and encouragement throughout the project.

We express deep gratitude and gratefulness to **our Guide Mrs.P.Devaki M.E.,(Ph.D).,** 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 the clients in a distributed environment; an efficient scheduling algorithm is needed.

A good schedule 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 Max-min and the Min-min algorithms. The Ant colony algorithm simulates the natural behavior of an ant in search of food to find an optimal path.

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

TABLE OF CONTENTS

CHAPTER III

CHAPTER IV

CHAPTER V

LIST OF ABBREVIATIONS:

ETC-Expected Time to Complete

ACO-Ant Colony Optimization

PI- Pheromone Indicator

LIST OF FIGURES

6. Average Waiting time

6.1. Low Low- Inconsistent

6.2. Low Low- Partially Consistent

6.3. Low Low- Consistent

6.4. Low High- Inconsistent

6.5. Low High- Partially Consistent

6.6. Low High- Consistent

6.7. High Low- Inconsistent

6.8. High Low- Partially Consistent

6.9. High Low- Consistent

6.10. High High- Inconsistent

6.11. High High- Partially Consistent

6.12. High High- Consistent

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 is 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. [4]

1.2 GRID COMPUTING

Grid computing can be viewed as a means to apply the resources from a collection of computers in a network and to harness all the compute power into a single project. 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** is the combination of computer

resources from multiple administrative domains for a common goal. Grids are usually used for solving scientific, technical or business problems that require a great number of computers processing cycles for processing of large amounts of data.

Grid computing concerns the application of the resources of many computers in a network to a single problem at the same time - usually to a scientific or technical problem that requires a great number of computer processing cycles or access to large amounts of data .

Grid computing 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). 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.

Knowledge grids:

A knowledge grid is an extension to the current grid in which data, 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 under utilized 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. [3]

1.5 OVERVIEW OF TASK SCHEDULING:

Scheduling is defined as the problem of allocation of machines over time to competing jobs [1]. The *m* x *n* task 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 [25]. 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.

1.6 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.

1.7 CLASSIFICATION OF STATIC TASK-SCHEDULING ALGORITHMS

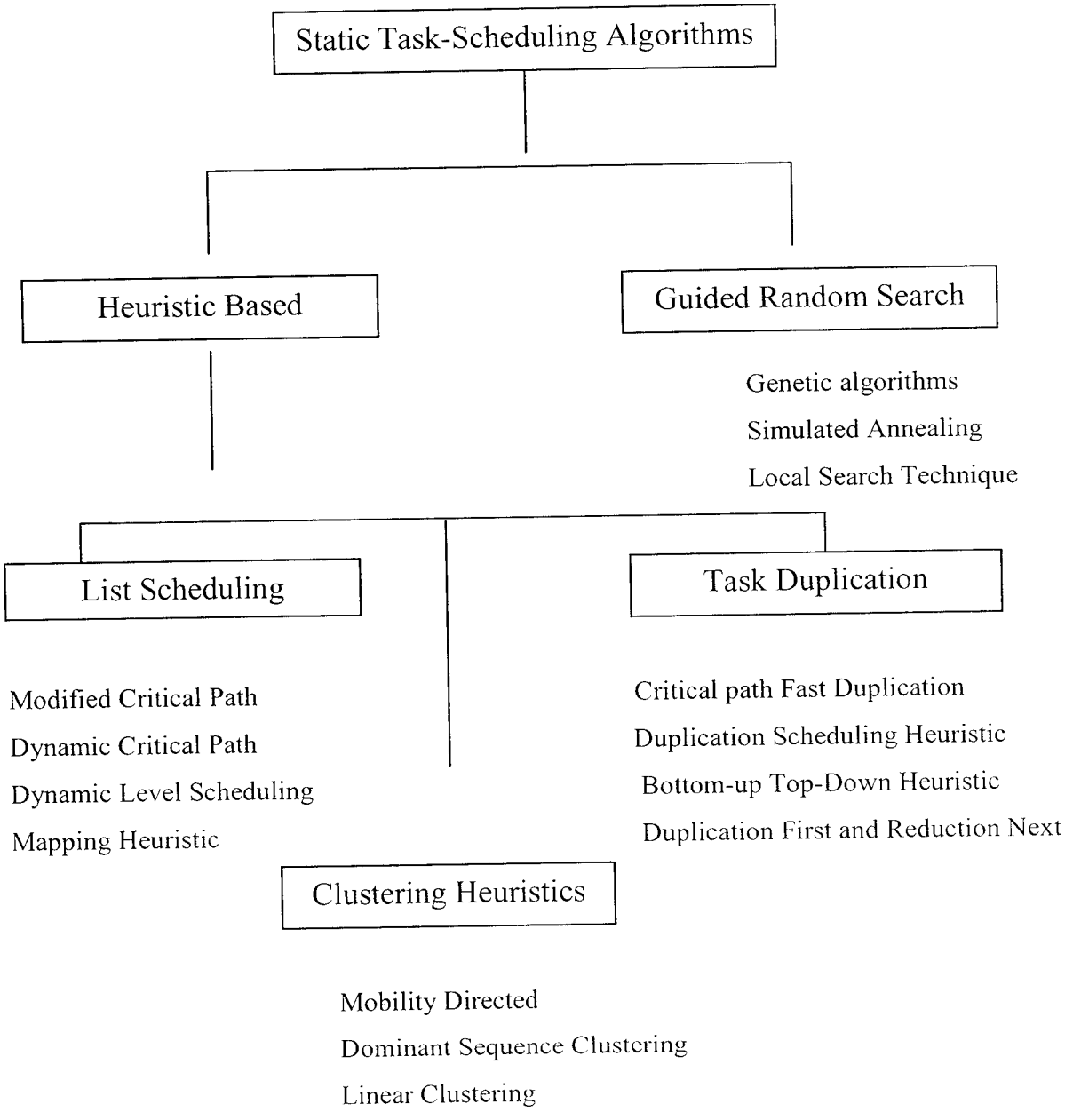


Fig 1: Classification of Static task-Scheduling algorithms

CHAPTER II

PROBLEM OVERVIEW

2.1 PROBLEM DEFINITION:

Given a set of tasks with certain characteristics, e.g., estimated execution time and a set of processing nodes with their own parameters, the goal of task scheduling is to allocate tasks at nodes so that the total make span and average waiting time is minimized. Genetic 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 machine 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, mj) 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 mj (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 x 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 x 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 x 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, mj) = B (i) x 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 x m ETC matrix is full.

The amount of variance among the execution times of tasks in the meta-task for a given machine is defined as task heterogeneity. Task heterogeneity was varied by changing the value of constant 'a' used to multiply the elements of vector W (i). Machine heterogeneity represents the variation that is possible among the execution times for a given task across all the machines. 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,...} of row I are extracted sorted, and replaced in order , while the row elements in column positions {1,3,5...} remain unordered (i.e., the even columns are consistent and odd columns are in general inconsistent).[1]

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 EXISTING STRATEGIES TAKEN UP FOR COMPARISON

2.3.1 MIN MIN STRATEGY:

The Min Min heuristic begins with the set U of all unmapped tasks. Then, the set of minimum completion times, M= [min0_j<+ (ct (ti, mj)), for each ti # U], is found. Next, the task with the overall minimum completion time from M is selected and assigned to the corresponding machine (hence the name Min_min). Last, the newly mapped task is removed from U, and the process repeats until all tasks are mapped (i.e., U is empty).

Min_min maps the tasks in the order that changes the machine availability status by the least amount that any assignment could. Let ti be the first task mapped by Min_min onto an empty system. The machine that finishes ti the earliest, say mj, is also the machine that executes ti the fastest. For every task that Min_min maps after ti, the Min_min heuristic changes the availability status of mj by the least possible amount for every assignment. [1][5][9]

MIN-MIN ALGORITHM: [8]

Step1: Select the minimum execution time in each row.

Step2: From the set of selected minimums, select the minimum time.

Step3: Assign the task which has that selected execution time to the corresponding processor.

Step4: If in case, that processor has been allocated for any other task, then select the next minimum time for that task and assign it to the corresponding processor.

Step5: Repeat the above steps till every task is assigned.

2.3.2. MAX MIN STRATEGY:

Max Min:

The Max Min heuristic is very similar to Min Min. The Max Min heuristic also begins with the set U of all unmapped tasks. Then, the set of minimum completion times, M, is found. Next, the task with the overall maximum completion time from M is selected and assigned to the corresponding machine (hence the name Max Min). Last, the newly mapped task is removed from U, and the process repeats until all tasks are mapped (i.e., U is empty)

Intuitively, Max Min attempts to minimize the penalties incurred from performing tasks with longer execution times. Assume, for example, that the metatask being mapped has many tasks with very short execution times and one task with a very long execution time. Mapping the task with the longer execution time to its best machine first allows this task to be executed concurrently with the remaining tasks (with shorter execution times). For this case, this would be a better mapping than a Min Min mapping, where all of the shorter tasks would execute first, and then the longer running task would execute while several machines sit idle. Thus, in cases similar to this example, the Max Min heuristic may give a mapping with a more balanced load across machines and a better makespan. [1][5][9]

MAX MIN ALGORITHM:

Step1: Select the minimum execution time in each row.

Step2: From the set of selected minimums, select the maximum time.

Step3: Assign the task which has that selected execution time to the corresponding processor.

Step4: If in case, that processor has been allocated for any other task, then select the next minimum time for that task and assign it to the corresponding processor.

Step5: Repeat the above steps till every task is assigned.

2.4 PROPOSED ALGORITHM:

Ant Colony Algorithm:

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

Basic Description

Ant algorithm has been successfully used to solve many ***NP problem.*** The algorithm has inherent parallelism, and we can validate its scalability. So it's obvious that ant 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.

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.

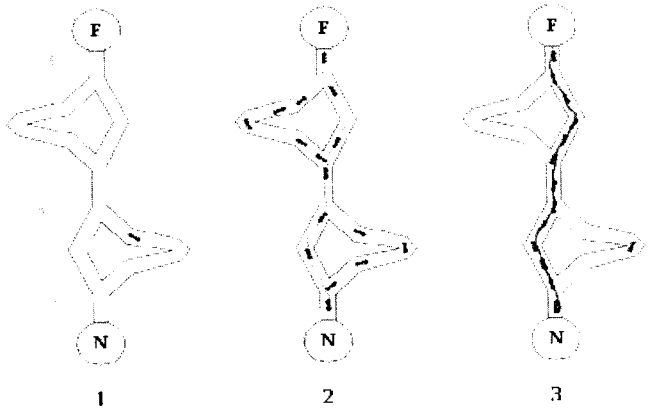


Fig 2: General ant behavior

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.

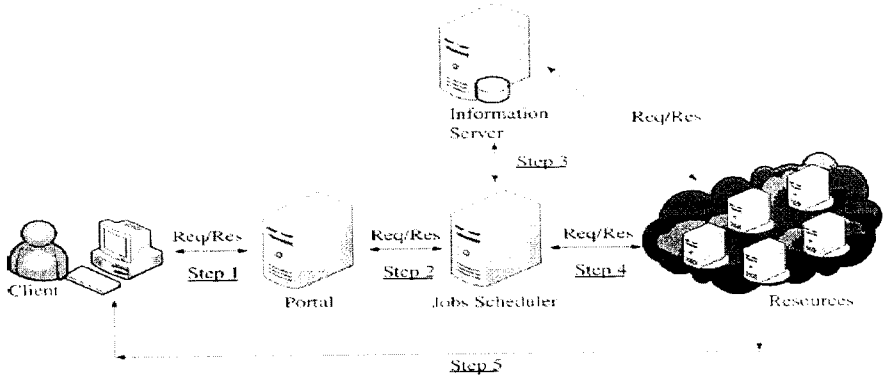


Fig 3: System Architecture

3.3 The proposed Ant Colony Algorithm:

In order to map the ant system to the grid system

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.

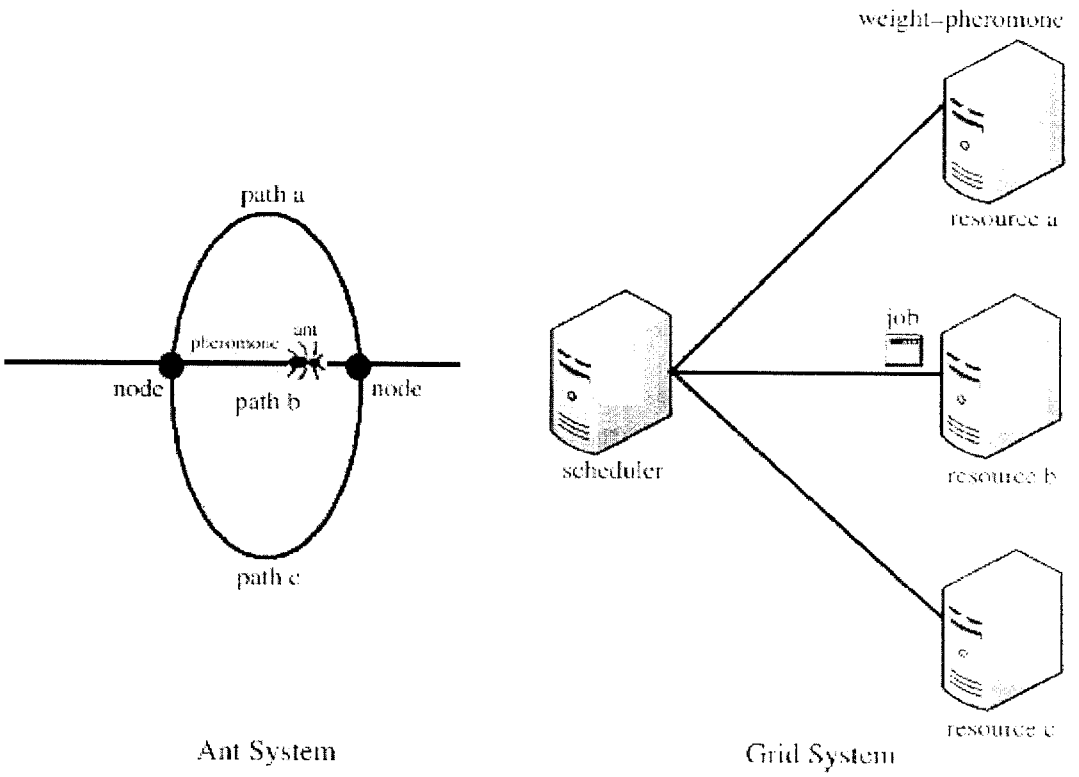


Fig 4: Mapping between the ant system and the grid system.

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. 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 $M_i/Bandwidth_j$ where Mi is the size of a given job_i and $Bandwidth_j$ 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 matrix(ETC), E[I,j] With the pheromone indicator is defined by

   $$\mathbf{PI_{ij} = [M_i/Bandwidth_j + T_i/CPU\_speed_i]^{-1}}$$

   Where PIij 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_i and $bandwidth_j$ are the status of resource.

4. The pheromone indicator tells that when a job is assigned to a resource, we consider the resource status, the size of 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 there are n resources and m jobs. We have the PI matrix as follows

$$PI= \begin{pmatrix} PI_{11} & PI_{12} & \cdots & PI_{in} \\ . & . & \cdots & . \\ . & . & \cdots & . \\ . & . & \cdots & . \\ PI_{m1} & PI_{m2} & \cdots & PI_{mn} \end{pmatrix}$$

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 $(1-\rho_j)$ further, where $1>\rho_j \geq 0$.

3.3 Ant Colony Algorithm For Task Scheduling

Example:

Assume there are three jobs j1,j2,j3 and three resources r1,r2,r3 in a grid. Let the initial states of each resource be assumed. The size of each job is 2MB, 3MB, 5MB. CPU cycles needed for each job are 3M,2M and 1M respectively.

The initial pheromone indicator of each entry in the PI matrix is

$$PI = \begin{pmatrix} PI11=3.88 & PI12=8.33 & PI13=5.28 \\ PI21=5.82 & PI22=12.49 & PI23=7.91 \\ PI31=11.64 & PI32=24.99 & PI33=15.83 \end{pmatrix}$$

Here PI32 has the highest pheromone value so the job j3 is assigned to the resource r2 and the row is removed.

$$PI = \begin{pmatrix} PI11=3.88 & PI12=8.33 & PI13=5.28 \\ PI21=5.82 & PI22=12.49 & PI23=7.91 \end{pmatrix}$$

After r2 finishes j3 the scheduler dispatches next job. The entries of the PI matrix must be updated in order to get newest pheromone for the next job submission. Now the assumptions made at the initial stage are changed.

The new values of the pheromone in the row corresponding to the resource will have to be multiplied by $(1-\rho j)$ where ρj is the overhead incurred in the resource j

after completing the job i, but this overhead does not affect the other two resources.

CODING:

```
package fpack;
/*.....PROGRAM FOR GENERATING SIMULATION MODEL.....*/
import java.io.*;
import java.util.*;
import java.io.IOException;
public class mar8
{
public static void main(String args[])throws Exception
{
        DataInputStream in=new DataInputStream (System.in);
        System.out.print("ENTER NO. OF TASKS:");
        int no_tasks=Integer.parseInt(in.readLine());
        calc c=new calc(no_tasks);
        /*.....GENERATING LOW AND HIGH HETEROGENETIC MATRICES.....*/
        Random r=new Random();
```

```
        do
        {
                System.out.println("1-->MAX-MIN \n 2-->MIN-MIN\n3-->EXIT\n");
                System.out.println("Enter which algorithm to perform:");
                int maxormin=Integer.parseInt(in.readLine());
                mar8.options(r, c, maxormin);
        }while(true);
    }
}
}
////PI MATRIX
package fpack;
import java.io.BufferedOutputStream;
import java.io.DataInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
```

```
import java.util.Random;
public class Pimatrix
{   static int o=0,q=0,z=0;
        static double size[]=new double[600];
        static Integer cpuspeed[]=new Integer[50];
        static Integer band[]=new Integer[50];
        static float t[][]=new float[600][20];
        static float load[]=new float[20];
         static int tmat[]=new int[600];
         static int pmat[]=new int[600];
         static int P[]=new int[20];
         static float a,b,c;
         static int l;
         static float Pisort[][]=new float[512][20];
         static float Pinew[][]=new float[512][20];
         static float Pi[][]=new float[512][20];
         static int task[][]=new int[512][20];
         static int processor[][]=new int[512][20];
```

```
static float Pi1d[]=new float[600];

static int task1d[]=new int[600];

static int processor1d[]=new int[600];

public static void main(String args[]) throws NumberFormatException,
IOException

{

o=0;

int r;

int min = 10;

int max = 20;

DataInputStream in=new DataInputStream (System.in);

for(int i=0;i<16;i++)

{

if(i==0)

cpuspeed[0]=3200;

else

cpuspeed[i]=cpuspeed[i-1]-50;

r = (int) (Math.random() * (max - min + 1) ) + min;

load[i]=r;
```

```
            band[i]=r/10;
            }
        System.out.println("Enter the no of tasks");
        int no=Integer.parseInt(in.readLine());
        initialize(no);
        readfile(no);
}
    public static void initialize(int no)
    {
            double r;
            double min = 10;
            double max = 20;
    for(int n=0;n<no;n++)
        {
            r = (double) (Math.random() * (max - min + 1) ) + min;
            size[n]=r;
        }
 }
```

```
public static float random(int no)
{
    float r = 0;
    float min = 0;
    float max = 1;
   for(int n=0;n<no;n++)
      {
            r = (float) (Math.random() * (max - min + 1) ) + min;
      }
     return  r;
}
public static void readfile(int no) throws IOException
{
            String filename[]={"i1c1","i1c2","i1c3","i1c4"};
            for(int h=0;h<4;h++)
            //int h=0;
            {
                  z=0;
```

```
                    o=0;
                    q=0;
                    formmatrix(no,filename[h]);
                }
            }
public static void formmatrix(int no,String filename) throws IOException
{
 int row=0;
String s5 = null;
t=readexcel1.readfile(no,filename);
String s="e:\\\\Pi1\\\\",s3=null;
s3=s.concat(filename);
s5=s3.concat("1.xls");
  OutputStream out=new FileOutputStream(s5);
    BufferedOutputStream bfo=new BufferedOutputStream(out);
    System.out.println("Inside formmatrix");
  //      formula substitution
    for(int i=0;i<no;i++)
```

```
{
        for(int j=0;j<16;j++)
        {
            a=(float) size[i]/band[j];
            c=cpuspeed[j];
            b=t[i][j]*100/c;
            float d=a+b;
            if(d==0)
                Pi[i][j]=0;
            else
            Pi[i][j]=1/d;
            task[i][j]=i;
            processor[i][j]=j;
        }
}
for(int i=0;i<no;i++)
{
        for(int j=0;j<16;j++)
```

```
                {
                    System.out.print(Pi[i][j]+"\t");
                }
                }
            if(z!=0)
                for(int i=0;i<no;i++)
                {
                    for(int j=0;j<16;j++)
                    {
                        if(processor[i][j]==pmat[q-1])
                            Pi[i][j]=Pi[i][j]*(1-random(no));
                    }
                }
            System.out.println("End");
            //converting to 1 dimension
int k=0;
            for(int i=0;i<no;i++)
            {
```

```
            for(int j=0;j<16;j++)
            {
          Pi1d[k]=Pi[i][j];
          task1d[k]=task[i][j];
          processor1d[k]= processor[i][j];
          k++;
      }     }
//creating new pi
      for(int i=0;i<no;i++)
      {
            for(int j=0;j<16;j++)
            {
                  if(task1d[row]==i)
                        Pinew[i][j]=0;
                  else
                              Pinew[i][j]=Pi[i][j];
            }
      }
```

```
//writing to file
for(int i=0;i<no;i++)
    {
     String s2 = null;
          for(int j=0;j<16;j++)
          {
               Float fObj = new Float(Pinew[i][j]);
               String s1 = fObj.toString();
              s2=s1.concat("\t");
           byte by[]=s2.getBytes();
        bfo.write(by);
          }
          String s4="\n";
         byte by1[]=s4.getBytes();
bfo.write(by1);
 }
 bfo.close();
System.out.print("entered file");
```

```
while(z<no)
{
        z++;
        formmatrix(no,filename);
}
 s="e:\\\\";s3=null;
 s3=s.concat("taskvec.txt");
 OutputStream out1=new FileOutputStream(s3);
 BufferedOutputStream bfo1=new BufferedOutputStream(out1);
for(int i=0;i<no;i++)
{
     String s2 = null;
     String tv=  new Integer(pmat[i]).toString();
    s2=tv.concat("\t");
    System.out.print(s2);
   byte byp[]=s2.getBytes();
   bfo1.write(byp);
}
```

```
            bfo1.close();
    }
public static int function(int no)
{
        int t=0;
        boolean flag=true,fl=true;
        while(fl)
        {
            for(int y=0;y<q;y++)
            {
                if(processor1d[t]==pmat[y])
                    flag=false;
        }
        if(flag)
        {
            pmat[q++]=processor1d[t];
            System.out.print("assigned"+pmat[q-1]);
            fl=false;
```

```
            }
          else
           {            System.out.print("sickikichu ma");
            while(l<no&&&task1d[t]!=task1d[l++]);

                t=l-1;flag=true;
            }
          }
        return t;
    }
  }
  //// PI MATRIX GENERATION
  package fpack;
  import java.io.*;
 import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
```

```
import java.sql.Statement;
public class pimat
{
    public static void main(String args[]) throws Exception
    {
        int itrn=0;
        DataInputStream in=new DataInputStream (System.in);
        do
        {
        System.out.print("ENTER NO. OF TASKS:");
        int no_tasks=Integer.parseInt(in.readLine());
        antcolony.calculation(no_tasks);
        }while(itrn<10);
    }
}
class antcolony
{
    File f,f1;
```

```
        static File fomax;
        static File fomin, ftmax, ftmin;
    static int len;
    int
i0=0,i1=0,i2=0,i3=0,i4=0,i5=0,i6=0,i7=0,i8=0,i9=0,i10=0,i11=0,i12=0,i13=0,i14=
0,i15=0;
    static String[] temp=null;
    static String str=null,fullstr="";
    static Connection con=null;
    static Statement st=null;
    static int[] ordervector1=new int[512];
    static int[] taskvector1=new int[512];
    static int[] temparr=new int[512];
    int[] pr0=new int[32];
    int[] pr1=new int[32];
    int[] pr2=new int[32];
    int[] pr3=new int[32];
    int[] pr4=new int[32];
    int[] pr5=new int[32];
```

```
    int[] pr6=new int[32];
    int[] pr7=new int[32];
    int[] pr8=new int[32];
    int[] pr9=new int[32];
    int[] pr10=new int[32];
    int[] pr11=new int[32];
    int[] pr12=new int[32];
    int[] pr13=new int[32];
    int[] pr14=new int[32];
    int[] pr15=new int[32];
  static float[] prtime=new float[16];
   static
files12={"c1","c2","c3","c4","i1","i2","i3","i4","p1","p2","p3","p4"};
                                                                    String[]
    int rawRandomNumber;
    int min = 1;
      static int max;
   float max1=0,min1=0;

    static void calculation(int no_tasks) throws Exception
```

```
{
        max=no_tasks-1;

for(int it10=1;it10<=10;it10++)//10 iterations loop
{
 System.out.println("\n\nITERATION:"+it10);
  int f12=0;
  float mspan;
  FileOutputStream ou=new
FileOutputStream("E:\\\\"+no_tasks+"\\iteration"+it10+"\\ant\\makespan1.txt");
        BufferedOutputStream bf=new BufferedOutputStream(ou);
        FileOutputStream out=new
FileOutputStream("E:\\\\"+no_tasks+"\\iteration"+it10+"\\ant\\wtime1.txt");
        BufferedOutputStream bf1=new BufferedOutputStream(out);
        for(f12=0;f12<12;f12++)                //12 files iteration
         {
              int itcount=0;
              float[] avgwtime=new float[500];
```

```
fomax=new
File("E:\\\\"+no_tasks+"\\iteration"+it10+"\\\\order\\"+files12[f12]+".txt");
        ftmax=new
File("E:\\\\"+no_tasks+"\\iteration"+it10+"\\\\task\\"+files12[f12]+".txt");
        ordervector1=makeint(fomax);
        taskvector1=makeint(ftmax);
        int itercount=0,iter=0,continuous=0;
        mspan=makespan(taskvector1,files12[f12],it10);
      System.out.print(mspan+"\t\t");
avgwtime[itcount++]=wtc1.avgwt(ordervector1,taskvector1,files12[f12],it10,no_ta
sks);
        System.out.println("avgerage waiting time:"+avgwtime[itcount-1]);
        String files=files12[f12];
        String mkspan=new Float(mspan).toString();
        String wittab=files.concat(":\t");
        String witval=wittab.concat(mkspan);
        String towrite=witval.concat("\t");
        String wittab1=files.concat(":\t");
       Float fObj = new Float(avgwtime[itcount-1]);
      String s1 = fObj.toString();
```

```
            String wttime=wittab1.concat(s1);
            String towrite1=wttime.concat("\t");
            byte writebyte[]=towrite.getBytes();
            bf.write(writebyte);
            byte writebyte1[]=towrite1.getBytes();
            bf1.write(writebyte1);
          }                              //12 files iteration
            bf.close();
            bf1.close();
}//it10 loop ends
}
 static int[] makeint(File f) throws Exception
          {
           int[] intarr1=new int[512];
           int[] tmparr=new int[512];
           if(!f.exists()&& f.length()<0)
                System.out.println("The specified file is not exist");
                else
```

```
 {
   FileInputStream finp=new FileInputStream(f);
   BufferedReader in = new BufferedReader(new FileReader(f));
   str=null;
   fullstr="";
while ((str = (in.readLine())) != null)
{
  fullstr+=str;
}
len=0;
len=fullstr.length();
temp=null;
temp=fullstr.split("       ");
try
{
 for(int i=0;i<temp.length;i++)
        {
       intarr1[i]=Integer.parseInt(temp[i]);
```

```
                }
            }
            finally
                {
             finp.close();
                }
              }
            temparr=intarr1;
            return temparr;
        }
  static float makespan(int[] taskvector1,String filename,int it10) throws Exception
  {
            try{
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                con = DriverManager.getConnection( "jdbc:odbc:etc32");
              st = con.createStatement();
                ResultSet rs = st.executeQuery( "Select * from
[it"+it10+filename+"$]" );
                int tasknum=0;
```

```
while(rs.next())
{
        if(taskvector1[tasknum]==0)
                prtime[0]+=rs.getFloat(1);
        else if(taskvector1[tasknum]==1)
                prtime[1]+=rs.getFloat(2);
        else if(taskvector1[tasknum]==2)
                prtime[2]+=rs.getFloat(3);
        else if(taskvector1[tasknum]==3)
                prtime[3]+=rs.getFloat(4);
        else if(taskvector1[tasknum]==4)
                prtime[4]+=rs.getFloat(5);
        else if(taskvector1[tasknum]==5)
                prtime[5]+=rs.getFloat(6);
        else if(taskvector1[tasknum]==6)
                prtime[6]+=rs.getFloat(7);
        else if(taskvector1[tasknum]==7)
                prtime[7]+=rs.getFloat(8);
```

```
else if(taskvector1[tasknum]==8)
        prtime[8]+=rs.getFloat(9);
else if(taskvector1[tasknum]==9)
        prtime[9]+=rs.getFloat(10);
else if(taskvector1[tasknum]==10)
        prtime[10]+=rs.getFloat(11);
else if(taskvector1[tasknum]==11)
        prtime[11]+=rs.getFloat(12);
else if(taskvector1[tasknum]==12)
        prtime[12]+=rs.getFloat(13);
else if(taskvector1[tasknum]==13)
        prtime[13]+=rs.getFloat(14);
else if(taskvector1[tasknum]==14)
        prtime[14]+=rs.getFloat(15);
else if(taskvector1[tasknum]==15)
        prtime[15]+=rs.getFloat(16);
tasknum++;
}
```

```
 rs.close();
        st.close();
        con.close();
 }
 catch(Exception ex) {
        System.err.print("Exception: ");
        System.err.println(ex.getMessage());
 }
 float max=prtime[0];
 for(int k=1;k<16;k++)
 {
       if(prtime[k]>max)

             max=prtime[k];
 }
 for(int qq=0;qq<16;qq++)
       prtime[qq]=0;
return max;
```

```
  }
  }
  ////WAITING TIME CALCULATION
  package fpack;
  import java.io.BufferedReader;
  import java.io.File;
  import java.io.FileInputStream;
  import java.io.FileReader;
  import java.sql.Connection;
  import java.sql.DriverManager;
  import java.sql.ResultSet;
  import java.sql.ResultSetMetaData;
  import java.sql.Statement;
public class wtc1 {
        public static void main(String args[]) throws Exception
        {
        }
        public static float avgwt(int[] ordervector,int[] taskvector,String filename,int
it10,int no_tasks)
```

```
{
        float[] etimep0=new float[512];
        float[][] timearr1=new float[512][16];
        float[] timearr=new float[512];
    float[] taskwt=new float[512];
    Connection con=null;
    Statement st=null;
    int
len,i0=0,i1=0,i2=0,i3=0,i4=0,i5=0,i6=0,i7=0,i8=0,i9=0,i10=0,i11=0,i12=0,i13=0,i
14=0,i15=0;
     String[] temp=null;
    int[] prino=new int[512];
    float[][] time=new float[512][17];
     int[][] pr=new int[16][512];
     String columnValue[][]=new String[512][16];
    for(int r=0;r<512;r++)
        for(int c=0;c<16;c++)
            time[r][c]=0;
     int rc=0;
```

```
int i=0;
try{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    con = DriverManager.getConnection( "jdbc:odbc:etc48");
st = con.createStatement();
int l=0;
    ResultSet rs = st.executeQuery( "Select * from [it"+it10+filename+"$]" );
    ResultSetMetaData rsmd = rs.getMetaData();
    int numberOfColumns = rsmd.getColumnCount();
    while (rs.next())
    {
        for (int j= 1; j <= numberOfColumns; j++)
        {
            time[i][j]=rs.getFloat(j);
    }
        i++;  }
st.close();
con.close();
```

```
    }
    catch(Exception ex)
    {
                System.err.print("Exception: ");
                System.err.println(ex.getMessage());
        }
  for(int j=0;j<32;j++)
 {
 if(taskvector[j]==0)
         pr[i0++][0]=j;
  else if(taskvector[j]==1)
         pr[i1++][1]=j;
  else if(taskvector[j]==2)
         pr[i2++][2]=j;
 else if(taskvector[j]==3)
         pr[i3++][3]=j;
else if(taskvector[j]==4)
         pr[i4++][4]=j;
```

```
else if(taskvector[j]==5)
pr[i5++][5]=j;
else if(taskvector[j]==6)
pr[i6++][6]=j;
else if(taskvector[j]==7)
pr[i7++][7]=j;
else if(taskvector[j]==8)
pr[i8++][8]=j;
else if(taskvector[j]==9)
pr[i9++][9]=j;
else if(taskvector[j]==10)
pr[i10++][10]=j;
else if(taskvector[j]==11)
pr[i11++][11]=j;
else if(taskvector[j]==12)
pr[i12++][12]=j;
else if(taskvector[j]==13)
pr[i13++][13]=j;
```

```
        else if(taskvector[j]==14)
                pr[i14++][14]=j;
        else if(taskvector[j]==15)
                pr[i15++][15]=j;
}
        int k=0,pcount=0;
        for(pcount=0;pcount<16;pcount++)
        for(int l=0;l<i1;l++)
                {
                        int tno=pr[l][pcount];
                        int ono=ordervector[tno];
                        prino[ono]=tno;
                        etimep0[k]=time[tno][1];
                        k++;
                }
                float wt[]=new float[512];
                int k1;
                for(k1=0;k1<k;k1++)
```

```
            {
            if(k1==0);
            else
        wt[k1]=wt[k1-1]+etimep0[k1-1];
            }
            int total=0;
            float avgwtime=0;
            for(int k2=0;k2<k;k2++)
                    total+=wt[k2];
          avgwtime=total/no_tasks;
             return avgwtime;
}
}
```

CHAPTER IV

Experimental Results and Discussion

Comparison Metrics:

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:

Makespan=MAX(CT_i)

The less the makespan of a scheduling algorithm, the better it works. [2]

4.2 Average Waiting Time:

The tasks allotted to the processor have to be executed in a sequence. The tasks in the sequence can be executed only when the preceding tasks in the sequence finish their execution. The time lapse between the task allocation and the start of execution of the task gives the waiting time of the task. The average of the waiting time computed for all the tasks should be a minimum for the algorithm to be efficient.

CHAPTER V

5. Comparison Graphs

5.1Makespan Comparison

LOW LOW:

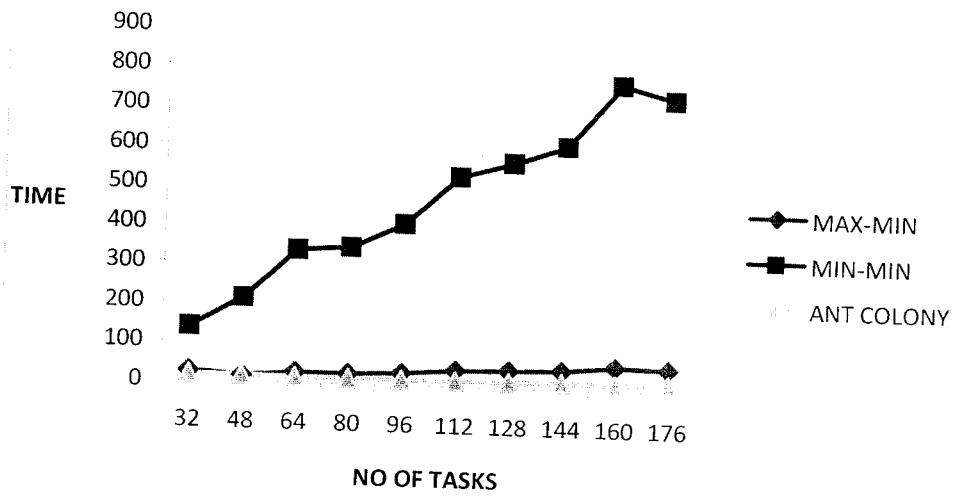


Fig 5.1: INCONSISTENT

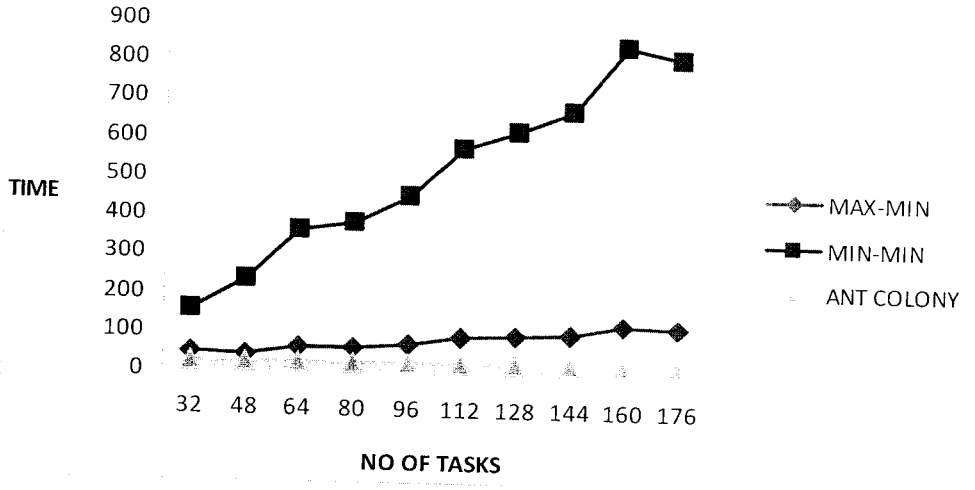


Fig 5.2: PARTIALLY CONSISTENT

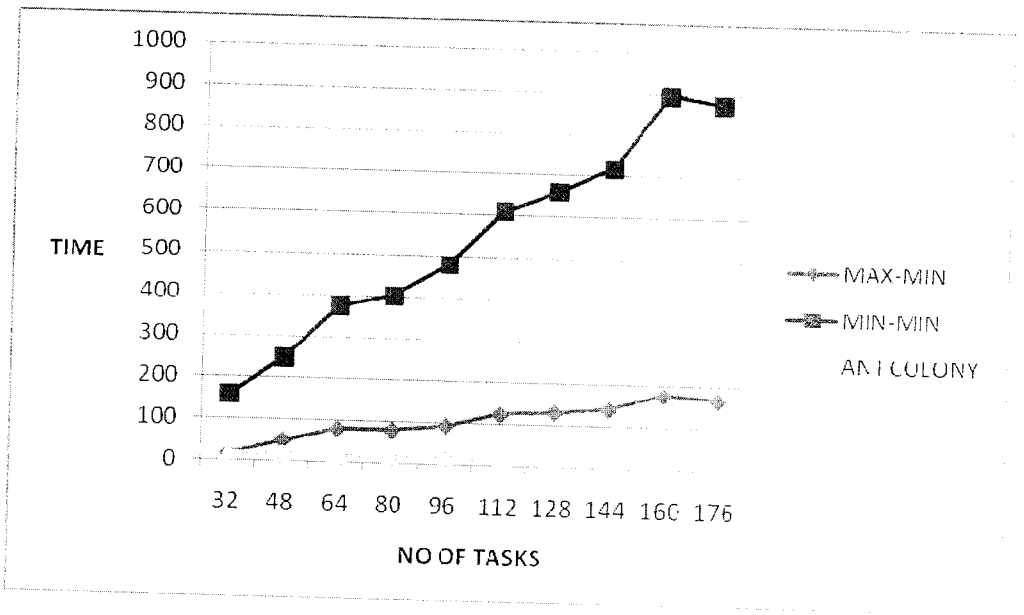


Fig 5.3: CONSISTENT

LOW HIGH:

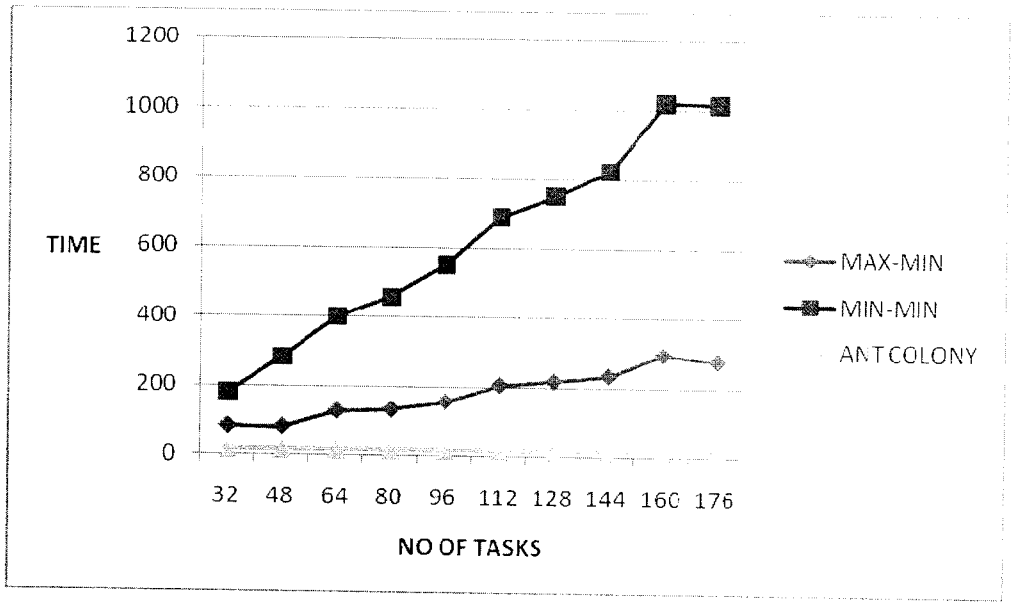


Fig 5.4: INCONSISTENT

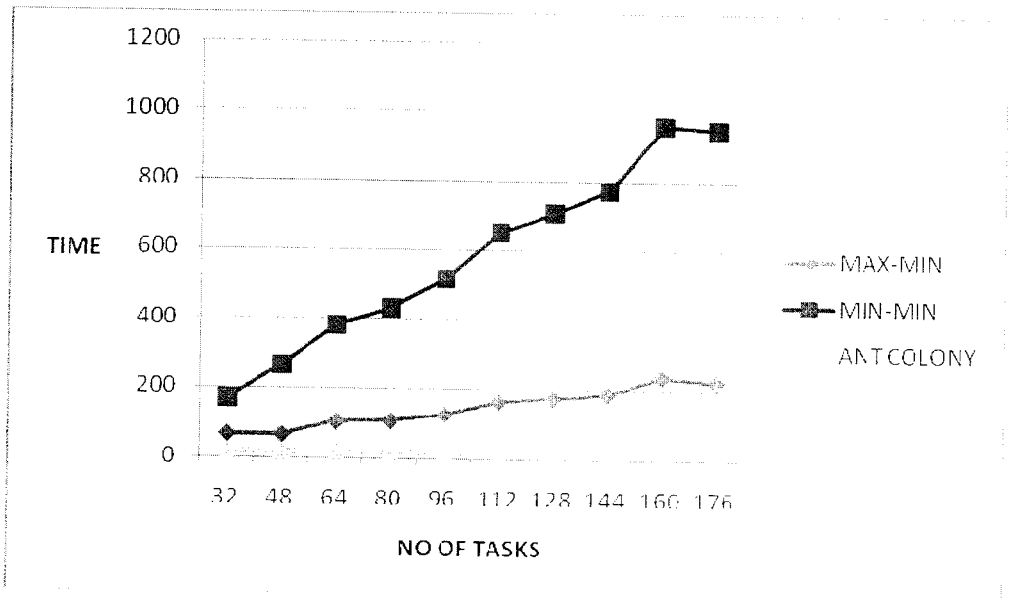


Fig 5.5: PARTIALLY CONSISTENT

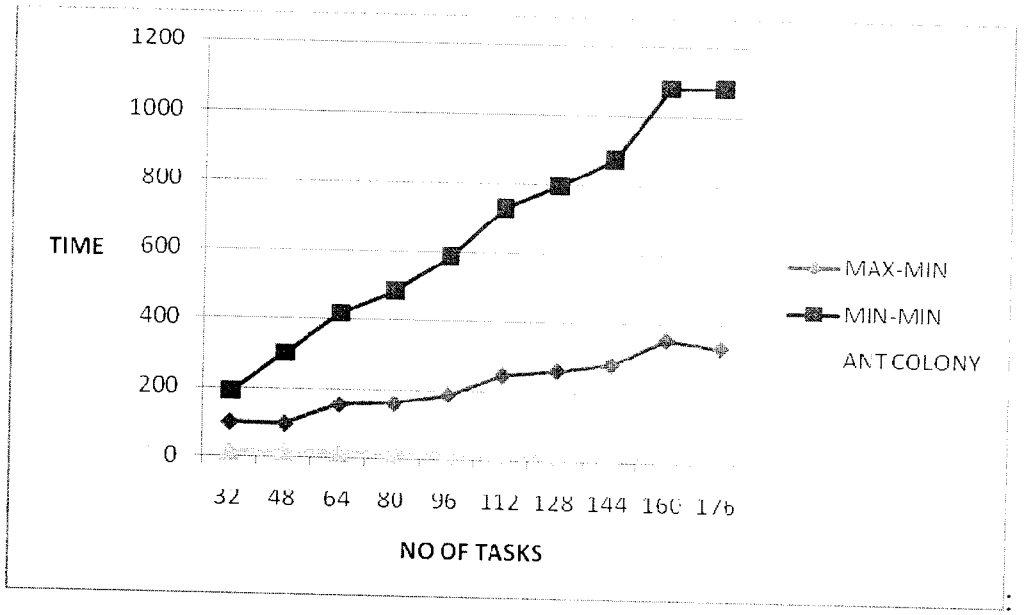


Fig 5.6: CONSISTENT

HIGH LOW:

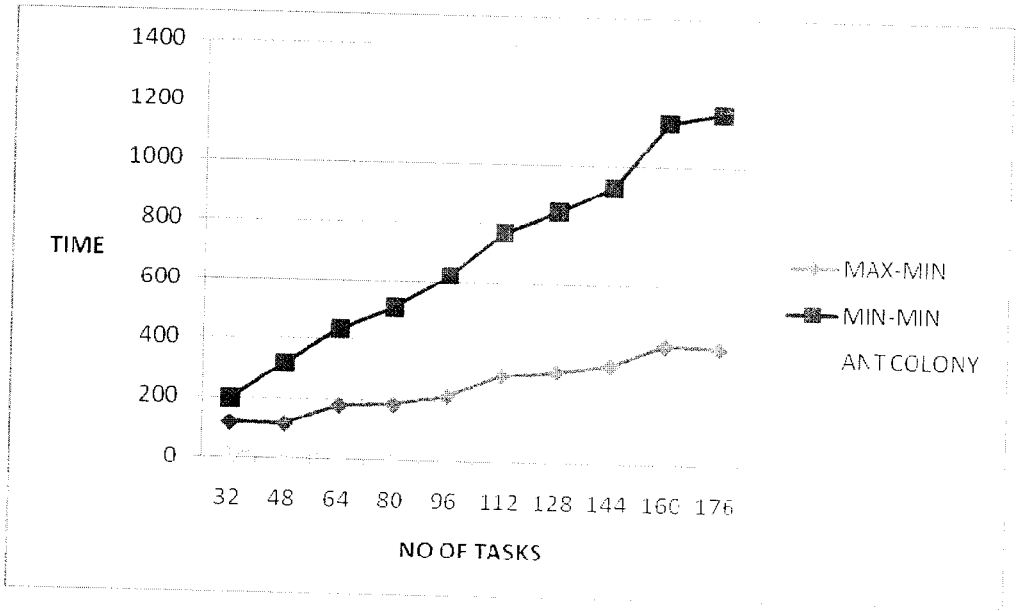


Fig 5.7: INCONSISTENT

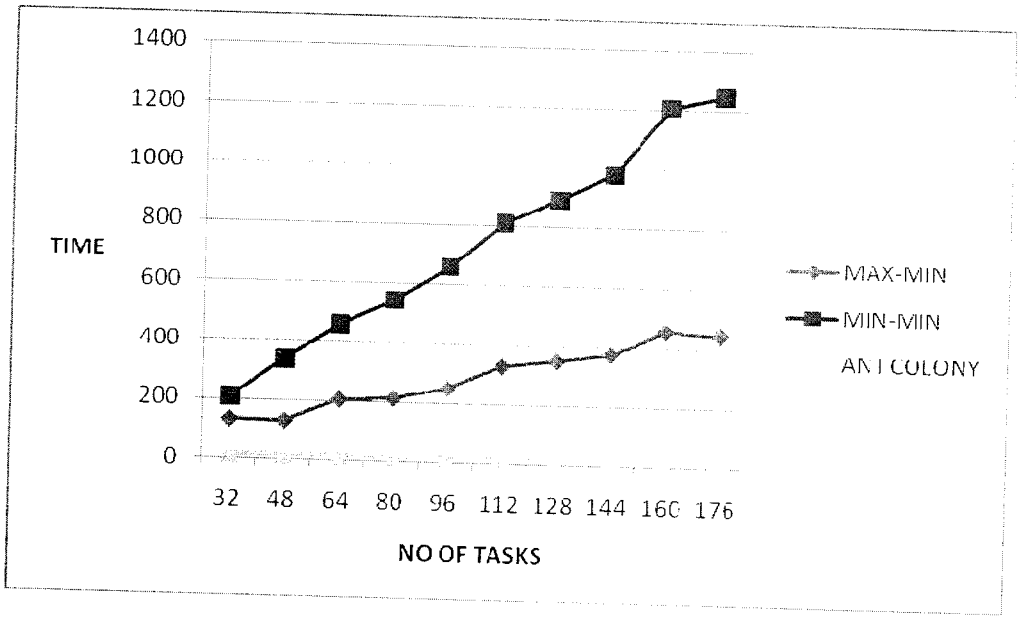


Fig 5.8: PARTIALLY CONSISTENT

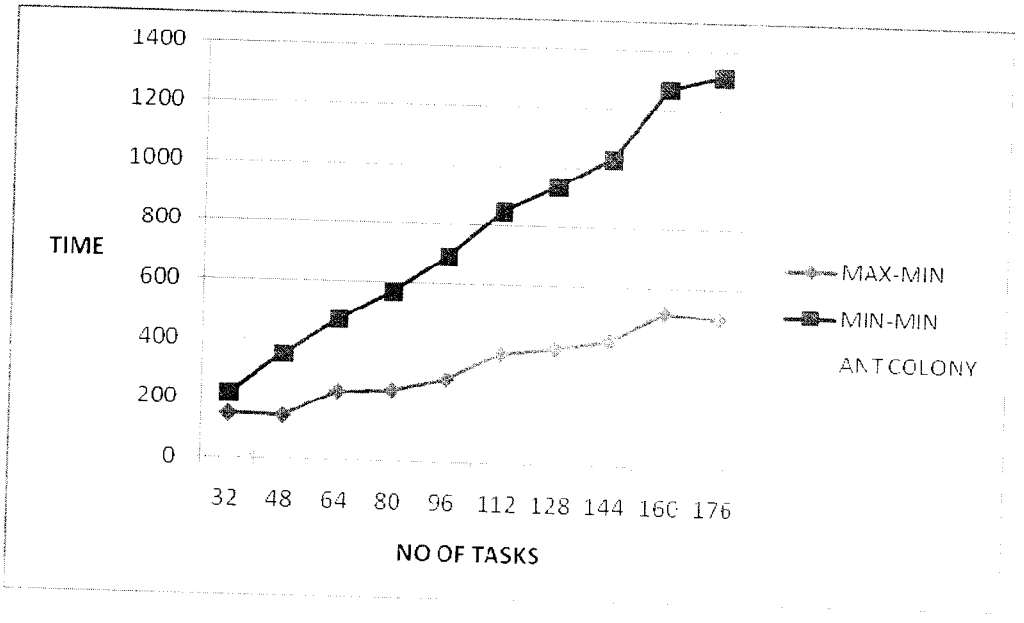


Fig 5.9: CONSISTENT

HIGH HIGH

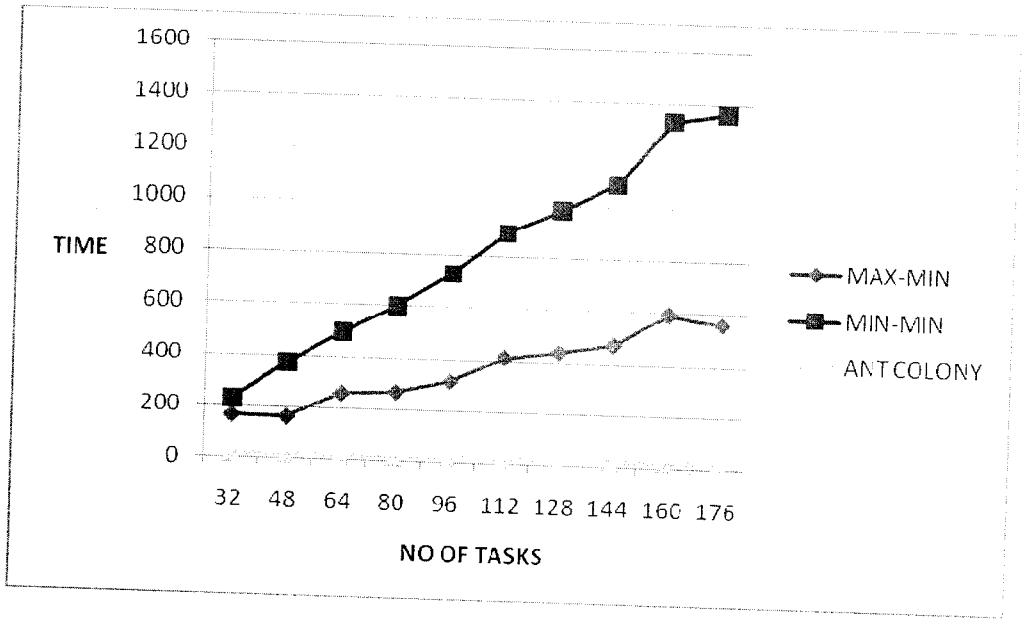


Fig 5.10: INCONSISTENT

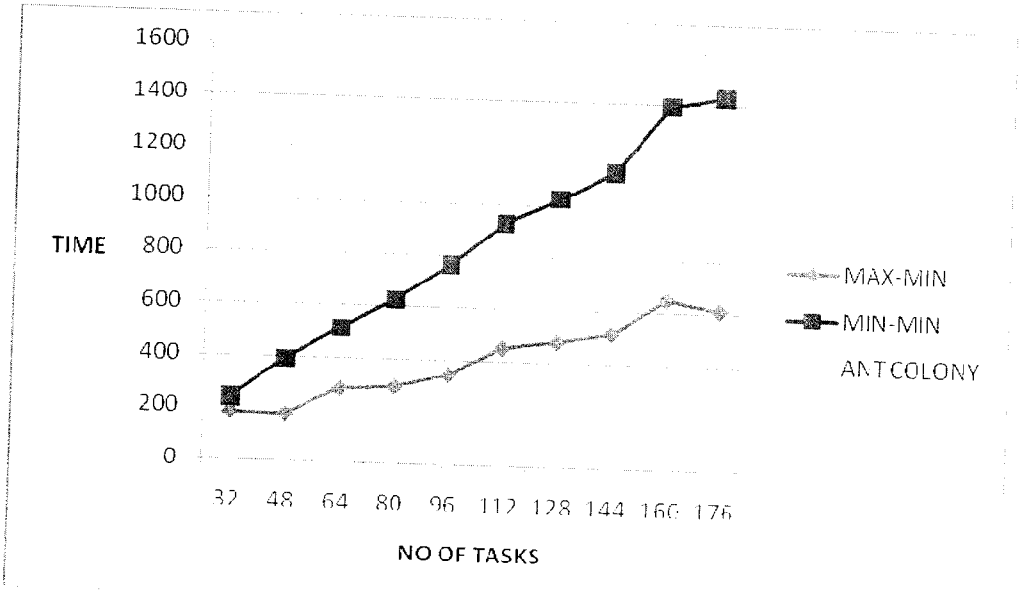


Fig 5.11: PARTIALLY CONSISTENT

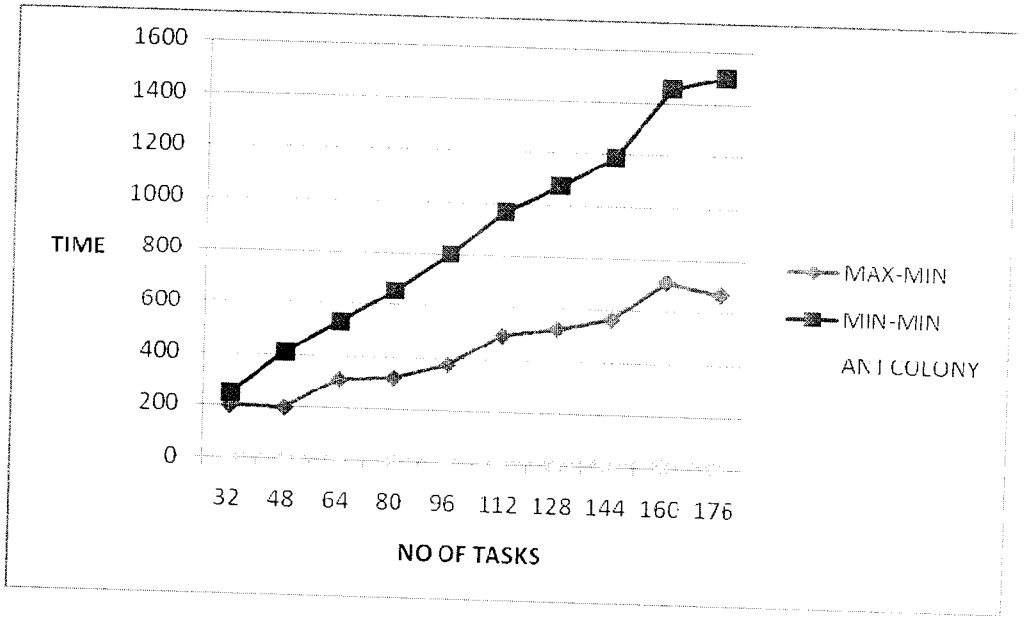


Fig 5.12: CONSISTENT

5.2 Average Waiting Time Comparison

LOW LOW:

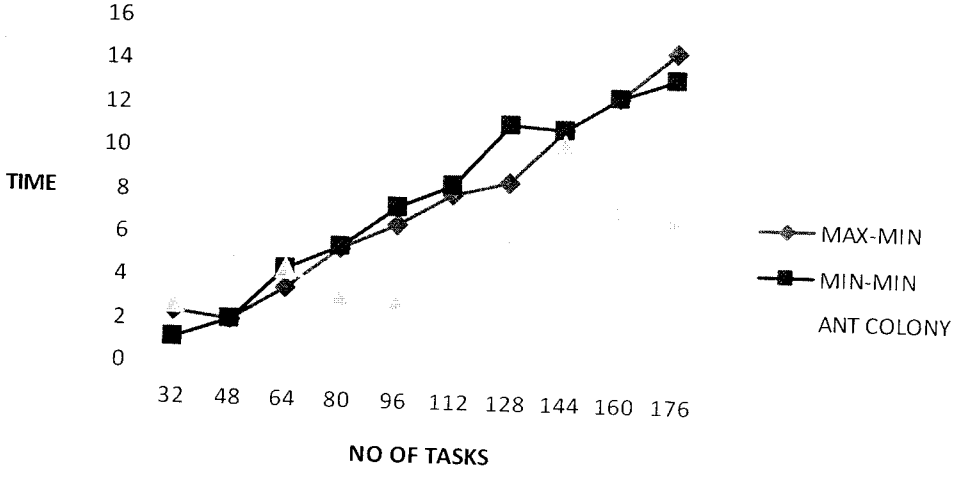


Fig 6.1: INCONSISTENT:

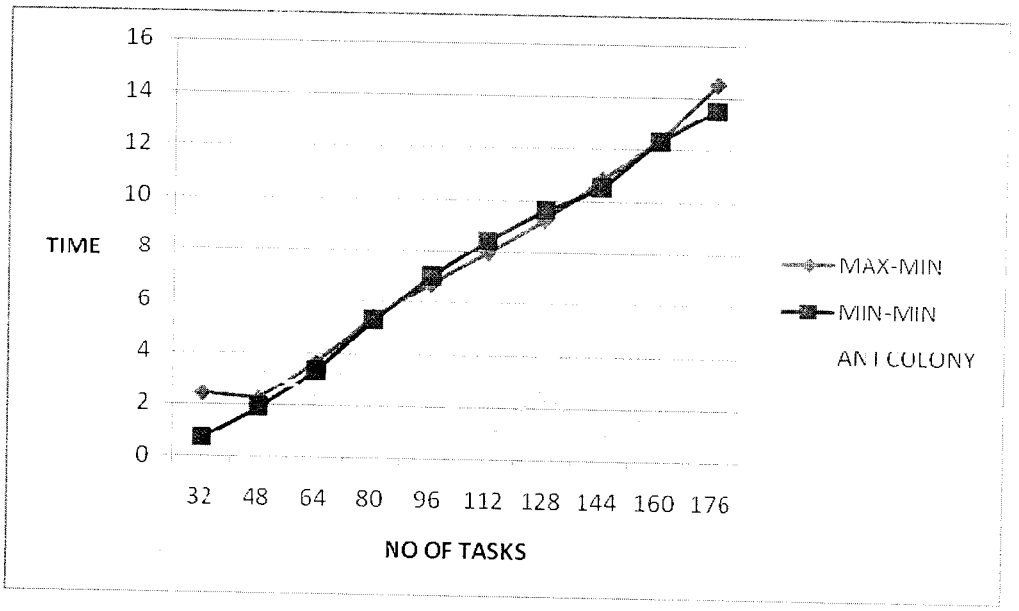


Fig 6.2: PARTIALLY CONSISTENT:

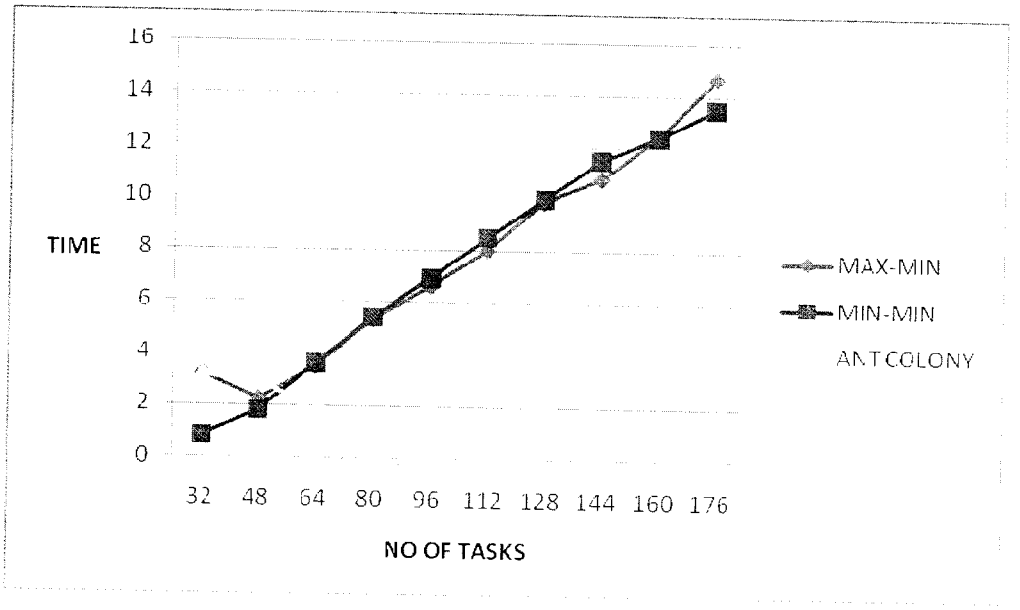


Fig 6.3: CONSISTENT:

LOW HIGH:

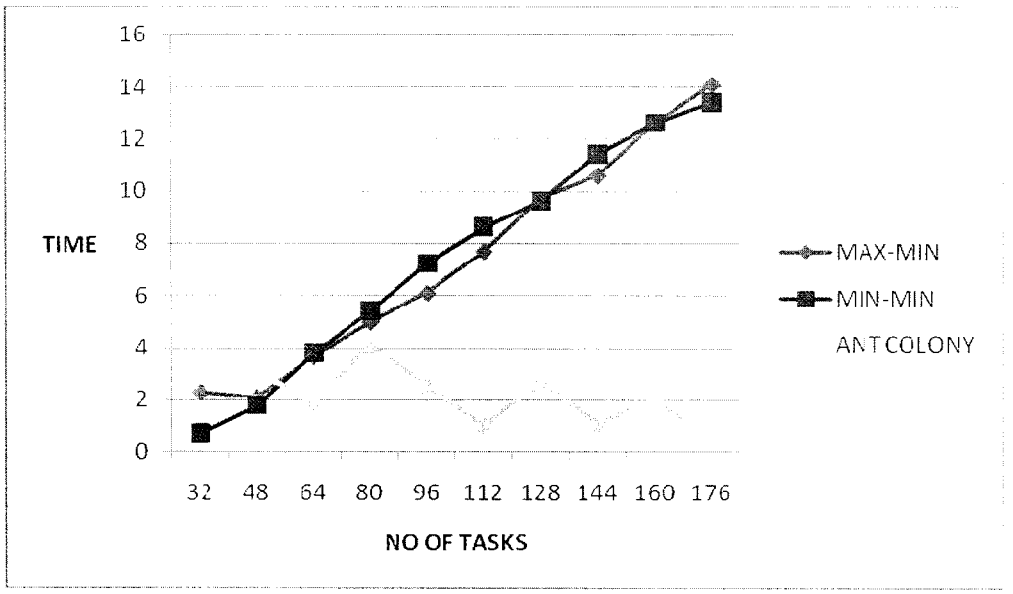


Fig 6.4: INCONSISTENT:

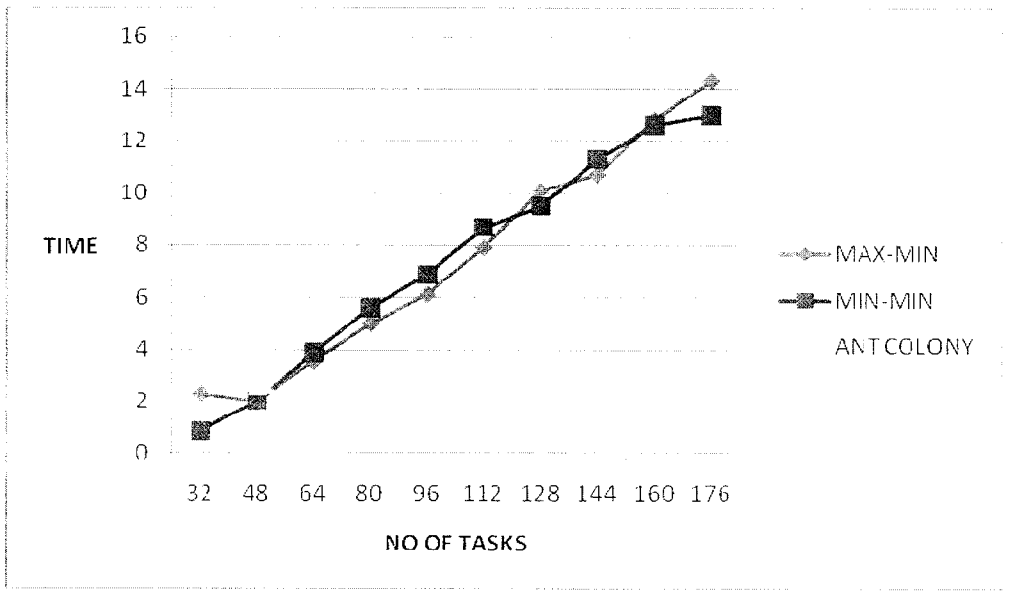


Fig 6.5: PARTIALLY CONSISTENT:

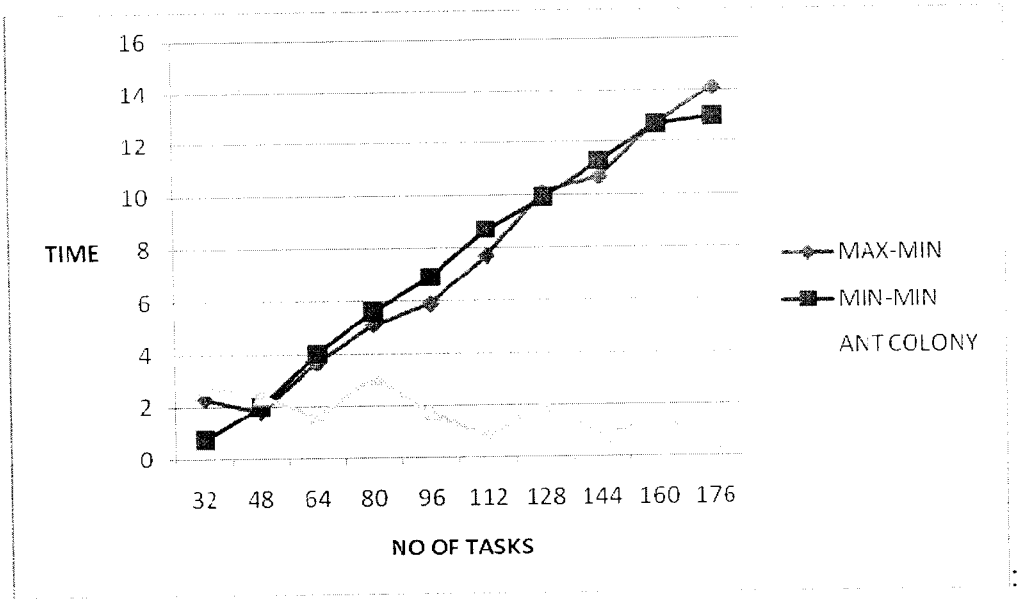


Fig 6.6: CONSISTENT

HIGH LOW:

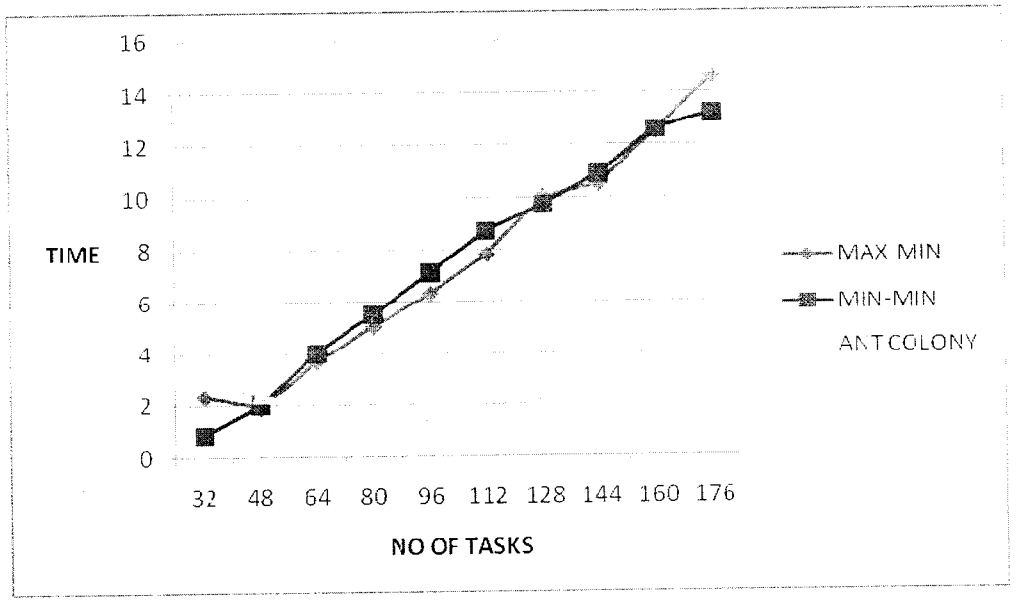


Fig 6.7: INCONSISTENT

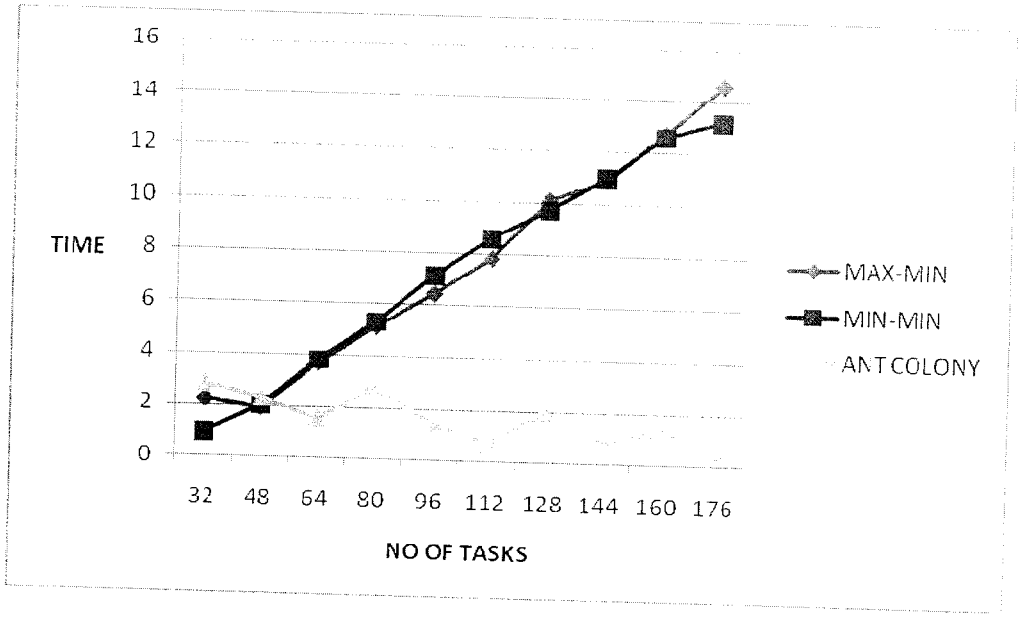


Fig 6.8: PARTIALLY CONSISTENT

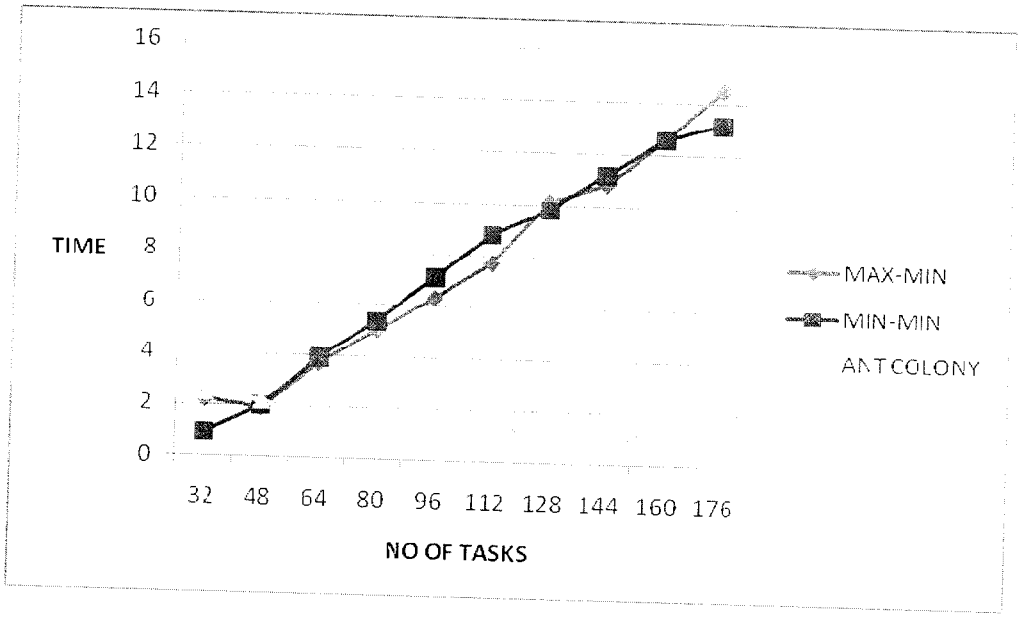


Fig 6.9: CONSISTENT

HIGH HIGH

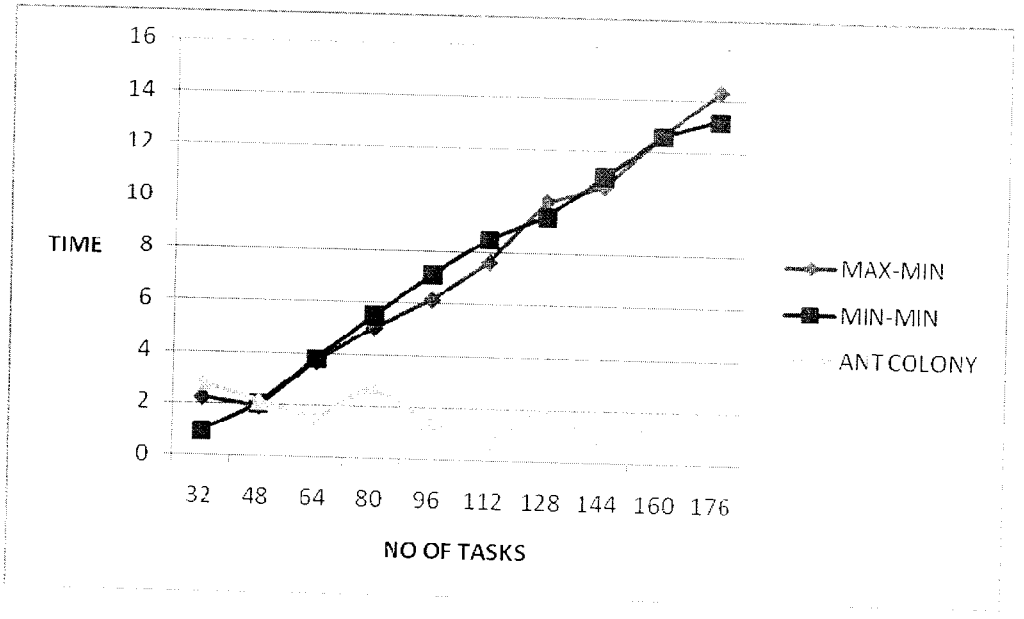


Fig 6.10: INCONSISTENT

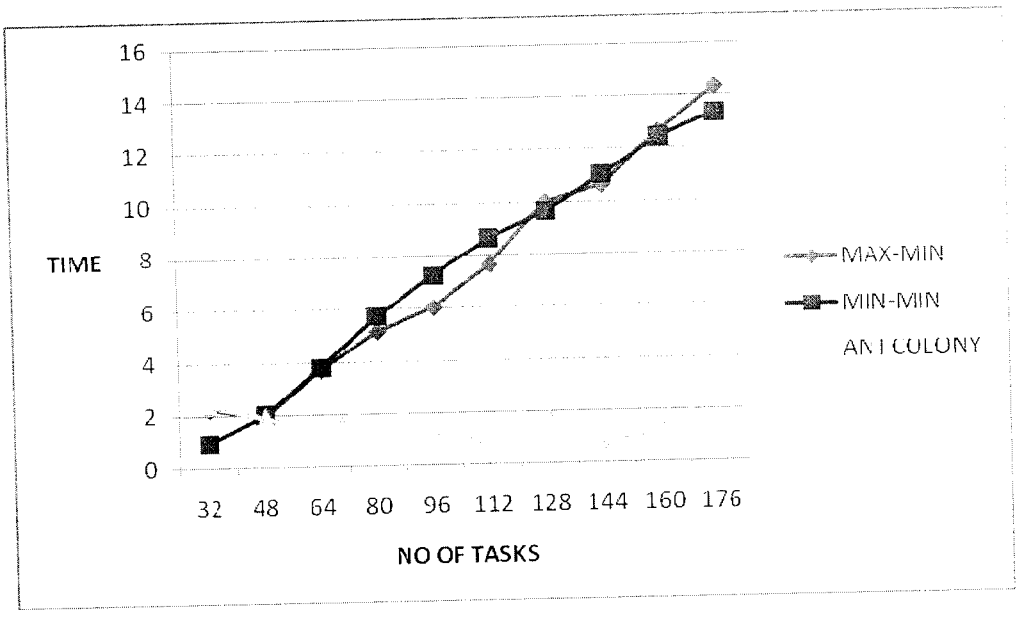


Fig 6.11: PARTIALLY CONSISTENT

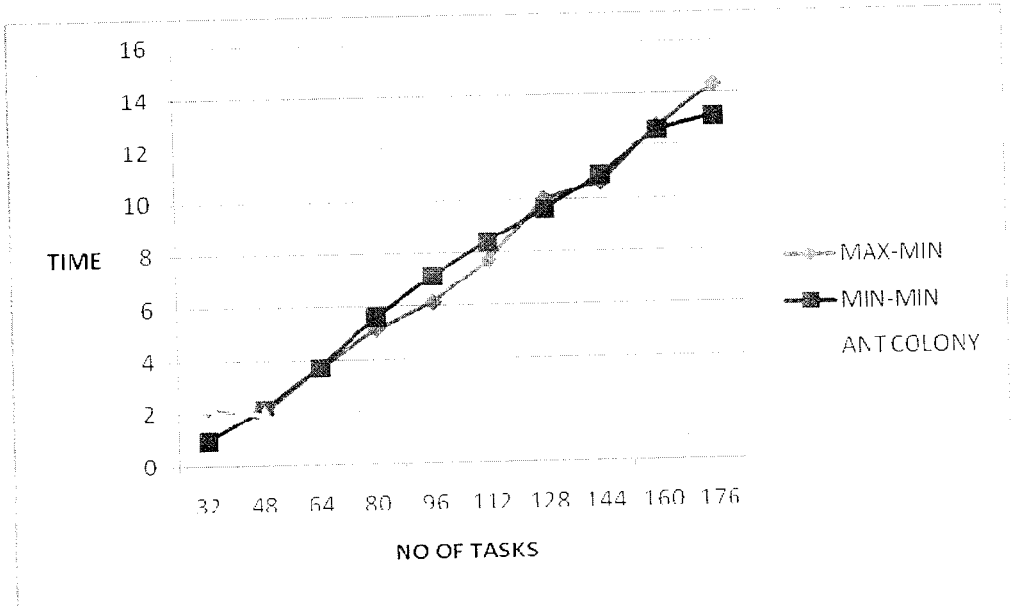


Fig 6.12: CONSISTENT

5.3Conclusion

We have taken up Min-Min strategy and Max Min strategy in order to compare the performance of ant algorithm. The ant colony algorithm implementation shows that it yields better performance than the already existing strategies taken up for comparison (i.e.) ant colony algorithm has given lesser make span and lesser waiting time for almost all cases when compared to Min-Min and Max Min strategies. The pheromone update functions do balance the system load. ACO is compared with two different algorithms and it is found to the best. The makespan is much reduced which is the aim of this project.

5.4 References

1.A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems
Tracy D. Braun, Howard Jay Siegel,2 and Noah Beck
School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana 47907-1285
2. A Min-Min Max-Min Selective Algorithm for Grid Task Scheduling
Kobra Etminani .Prof. M. Naghibzadeh
3.www.redbooks.ibm.com
4. Grid Computing: Introduction and Overview, Manish Parashar, Senior Member ,IEEE and Craig A.Lee, Member, IEEE
5. An enhanced ant algorithm for grid scheduling problem, Kousalya.K and Balasubramanie.P
6.A Min-Min Max-Min Selective Algorihtm for Grid Task Scheduling ,Kobra Etminani

Dept. of Computer Engineering Ferdowsi University of Mashad Mashad, Iran ,Prof. M. Naghibzadeh Dept. of Computer Engineering Ferdowsi University of Mashad ,Mashad, Iran

7. IEEE paper "An ant algorithm for balanced job scheduling in grids" – Ruay-Shiung Chang. Jih-Sheng Chang, Po-Sheng Lin, June 2008.

8."The research of Ant Colony and Genetic Algorithm in Grid Task Scheduling" - Jing Liu Chen, Yuqing Dun, Lingmin Liu, Ganggang Dong, 2008.