JOB SCHEDULING IN GRID COMPUTING USING GENETIC ALGORITHM

A PROJECT REPORT

Submitted by

S.S.DEEPIKA (0710108009)

M.KASTHURI (0710108022)

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

KUMARAGURU COLLEGE OF TECHNOLOGY: COIMBATORE-641 049

BONAFIDE CERTIFICATE

Certified that this project report entitled "**Job Scheduling in Grid Computing using Genetic Algorithm**" is the bonafide work of S.S.Deepika and M.Kasthuri 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.

SIGNATURE	**SIGNATURE**
Mrs.P.Devaki.,M.E.(Ph.D)	**Mrs.P.Devaki.,M.E.(Ph.D)**
HEAD OF THE DEPARTMENT	**PROJECT GUIDE**
Department of Computer	Department of Computer
Science and Engineering	Science and Engineering
Kumaraguru College of Technology	Kumaraguru College of Technology
Coimbatore-641049	Coimbatore-641049

The candidate with University Register Nos. 0710108009, 0710108022 were examined by us in the project viva-voce examination held on 20.04.11.

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION

We hereby declare that the project entitled **”Job Scheduling in Grid Computing using Genetic Algorithm”** 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, Coimbatore.

Place: Coimbatore

Date: 19.04.11

(S.S.Deepika)

(M.Kasthuri)

ACKNOWLEDGEMENT

We are intend to express our heartiest thanks to our chairman **PadmaBhushan Arutselvar N.Mahalingam,B.sc., F.I.E** and the Co-Chairman **Dr.B.K.Krishnaraj Vanavarayar** for given us this opportunity to embark on this project.

We would like to make a special acknowledgement and thanks to our correspondent **M.Balasubramaniam, M.com., M.B.A.**, and our Director **Dr.J.Shanmugam** for his support and encouragement throughout the project.

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 , Research and Development for his support and encouragement throughout the project.

We express deep gratitude and gratefulness to **our Guide Mrs.P.Devaki M.E.,(Ph.D).,** Project coordinator and Head of the 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

The grid computing system is a new, powerful and innovative system for a group of heterogeneous distributed computing systems. Scheduling is a key problem in emergent computational systems, such as Grid in order to benefit from the large computing capacity of such systems.

When all the jobs need the same resource, overloading or stagnation of that particular resource may occur in the Grid environment .To overcome this problem, our project aims to use Genetic Algorithm(GA) for efficiently allocating jobs(homogeneous) to resources in a Grid system. We present the usefulness of GA for designing efficient Grid schedulers when makespan is minimized.

We have assumed that the tasks property is submitted statically i.e. (offline or in a predictive manner). It is also assumed that each machine executes a single task at a time in the order in which the tasks are assigned. The number of tasks and machines, the time taken to complete the tasks in each machine is known in advance. Given all these details, the goal of task scheduling is to allocate the tasks at machines in a way that the makespan (overall completion time of all tasks) and the waiting time of each task is reduced.

We have taken up existing strategies namely min-min, max-min, minimum completion time and minimum execution time and the results of these algorithms are compared with the results obtained from the proposed genetic algorithm. Genetic algorithm is chosen because it is an evolutionary process and the solution obtained from it will be optimal.

TABLE OF CONTENTS

CHAPTER I

1. OVERVIEW OF GRID ENVIRONMENT

CHAPTER II

2. PROBLEM OVERVIEW

CHAPTER V

5. COMPARISON GRAPHS

CHAPTER VI

LIST OF ABBREVIATIONS

ETC	- Expected Time to Compute
MET	- Minimum Execution Time
MCT	- Minimum Completion Time
QoS	- Quality of Service
GA	- Genetic Algorithm

LIST OF FIGURES

3.4. Low Task High Machine - Consistent

3.5. Low Task High Machine - Inconsistent

3.6. Low Task High Machine – Partially Consistent

3.7. High Task High Machine - Consistent

3.8. High Task High Machine - Inconsistent

3.9. High Task High Machine – Partially Consistent

3.10. High Task Low Machine - Consistent

3.11. High Task Low Machine - Inconsistent

3.12. High Task Low Machine – Partially 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 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 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 computer 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 nonvoluntary 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, nonvoluntary 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 partners 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.

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. 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 chance of failure is 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* 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.

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

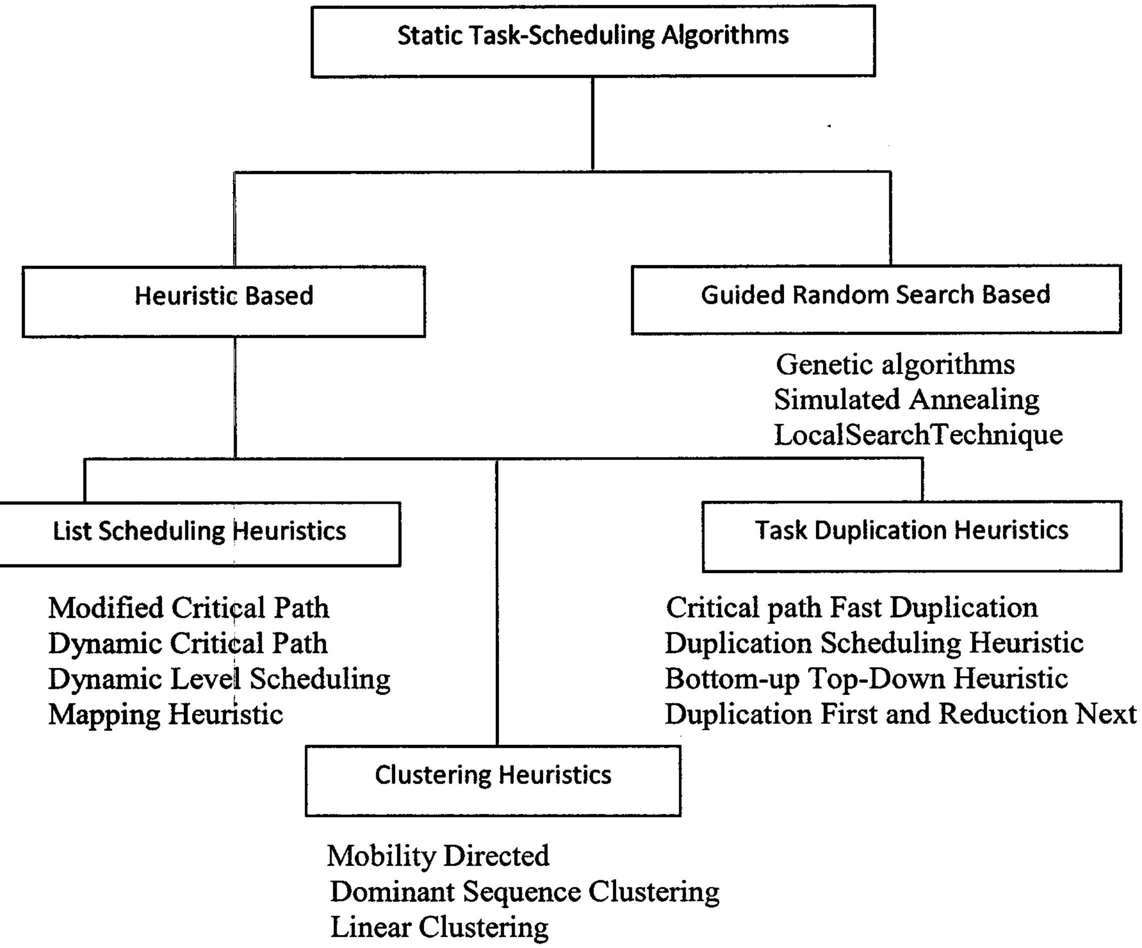

Static Task-Scheduling Algorithms
Heuristic Based
Guided Random Search Based
Genetic algorithms
Simulated Annealing
LocalSearchTechnique
List Scheduling Heuristics
Task Duplication Heuristics
Modified Critical Path
Dynamic Critical Path
Dynamic Level Scheduling
Mapping Heuristic
Critical path Fast Duplication
Duplication Scheduling Heuristic
Bottom-up Top-Down Heuristic
Duplication First and Reduction Next
Clustering Heuristics
Mobility Directed
Dominant Sequence Clustering
Linear Clustering

Fig 1

CHAPTER II

PROBLEM OVERVIEW

2.1 PROBLEM DEFINITION

Given a set of tasks (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 tasks at nodes so that the total makespan is minimized and the resource utilization is maximized. Genetic algorithm is used in order to obtain an optimal solution.

2.2 ASSUMPTIONS

- Tasks are all independent and the task property is submitted statically – offline or batch mode.
- Each machine executes a single task at a time in the order in which the tasks are assigned.
- The number of tasks and machines, the time taken to complete the tasks in each machine is known in advance.

2.3 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 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 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,...} 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).[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

2.4 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

GENETIC ALGORITHM

A genetic algorithm (GA) [6] is an iterative search procedure widely used in solving optimization problems, motivated by biological models of evolution. In each iteration, a population of candidate solutions is maintained. Genetic operators such as mutation and crossover are applied to evolve the solutions and to find the good solutions that have a high probability to survive for the next iteration.

BASIC DESCRIPTION

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

CHAPTER III

OVERVIEW OF GENETIC ALGORITHM

3.1 INTRODUCTION

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

3.2 STEPS IN GENETIC ALGORITHM

OUTLINE OF THE GENETIC ALGORITHM

1. Generate a random population of n chromosomes which are suitable solutions.
2. Establish a method to evaluate the fitness f(x) of each chromosome x in the population .
3. Create a new population by repeating the following steps until the new population is complete.

4.

- Selection - Select from the population according to some fitness scheme.It involves QoS Satisfaction – To ensure that all chromosomes in the mutated population satisfies QoS parameters.
- Crossover- New offspring formed by a crossover with the parents.
- Mutation - With a mutation probability mutate new offspring atEach locus (position in chromosome).

5. Use the newly generated population for further run of algorithm.

3.3 GENETIC ALGORITHM FOR TASK SCHEDULING

3.3.1 ENCODING

To solve a problem via GAs, it is necessary to find a mapping of a potential candidate for a solution onto a sequence of binary digits, the so called chromosome. In our case, however, it is more efficient to represent chromosomes as strings of integers. The length of the chromosomes is given by the number of tasks that should be allocated. Every gene in the chromosome represents the processor where the task is running on.

Solutions of a given problem obtained from existing algorithms are encoded in the form of chromosomes. These chromosomes form the initial population. The chromosomes are the task allocation vectors and order vectors of the obtained solutions. Task allocation vector is of length equal to the number of given tasks and each value represents the processor to which the corresponding task is allotted. For instance, the value p_k in the vector indicates the processor to which the j^{th} task is allotted.

Similarly, order vector of dimension equal to the number of tasks contains the order of execution of the corresponding task in the assigned processor.

Eg: task vector [8]

2	1	3	1	2

Order vector [8]

1	2	1	1	2

Here, task1 gets executed in processor2 first, task5 the second in processor2. Tasks 4 and 2 get executed in processor1 in the respective order. Task3 in processor3.

Solutions obtained from max-min and min-min scheduling algorithms are encoded as above and these form the initial population, the two parents to crossover. The schedule produced by these algorithms is located at an approximate area in the search space around the optimal schedule. Genetic algorithm searches that area to improve the schedule.

To reduce the complexity of the Genetic algorithm, the number of chromosomes in the population is fixed throughout its operation. [6]

3.3.2 SELECTION

For the selection process, we introduce a criterion called QoS factors satisfaction. 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.

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 MachineCapability$_{ik}$ values are calculated.

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

$$\frac{QoS^{\text{MachineCapability}_{ik}}}{QoS^{\text{TaskRequirements}_{ik}}} > 1$$

Hence for each QoS factor the above equation is applied. Hence for a particular job if a resource satisfies all the QoS factors, then the QoS matrix is generated with each QoS factor having a guided probability value associated with it.

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

$$w =< w_1..w_k > 0 \leq w_k \leq 1 \sum_{i=1}^{k} w_i = 1$$

The QoS Satisfaction matrix will be generated based on the formula

QoSSatisfaction$_{ij}$ = $\sum_{l=1}^{k}\left(\frac{QoS^{TaskRequirements_i}}{QoS^{MachineCapability_j}} \times w_l\right)$ if equation is satisfied for all the QoS factors.

Else QoSSatisfaction$_{ij}$ = 0

The QoSSatisfaction$_{ij}$ indicates the QoS satisfaction value for resource$_j$ executing job$_i$. From the QoS matrix, for each job$_i$, we find the resource$_j$ that has the highest QoS satisfaction value and allocate the particular job$_i$ to that resource$_j$. If all the resources didn't satisfy QoS for a particular job, then we assign that job to the resource that has the lowest ETC for that job.

Each of the chromosomes after mutation process undergoes selection process based on the QoS Satisfaction matrix. In each chromosome, if a gene or job which doesn't satisfy QoS is found, then we replace the value of gene with resource that has the highest QoS satisfaction.

At the end of the process we will get 4 chromosomes with QoS satisfaction and we calculate the makespan for each of them. Fitness function is given by

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

Hence we get f(x) value for each chromosome and the top 2 chromosomes that has the highest f(x) will be given to the next step in the evolutionary process as initial seed until the makespan of parents and offfsprings converges.

3.3.3 CROSSOVER

[6]A common and a single random crossover point is chosen in both the parents. The values after the crossover point in both the parents are swapped to produce the two new offspring. The offspring generated contains the qualities of both the parents.

Eg:

Parent1

2	4	1	5	3

Parent2

2	5	4	3	1

Crossover point: 3

After crossover:

Offspring1

2	4	1	3	1

Offspring2

2	5	4	5	3

3.3.4 MUTATION

[6]Mutation is performed by exchanging any two values in the newly generated offspring. Mutation is to prevent all solutions from falling into a local optimum of solved problem that is to preserve the diversity of the population. The points for mutation are also selected at random.

Eg:

3	1	4	2	4	1	3	2

Mutation points: 3 and 8

After Mutation:

3	1	2	2	4	1	3	4

3.3.5 ALGORITHM

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

Constants: cross over probability μ_c=0.9, mutation probability μ_m=0.05

2. Method:

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

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

Step3: For the selection criterion, generate the QoS satisfaction matrix . From the QoS satisfaction matrix for each job_i identify a $resource_j$ that has the highest QoS satisfaction value and allocate the particular job_i to that $resource_j$. If all the resources didn't satisfy QoS for a particular job, then we assign that job to the resource that has the lowest ETC for that job. For each seed the makespan is calculated.

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

Step5: Calculate the makespan.

Step6: End

3.4 IMPLEMENTATION

GENETIC ALGORITHM

```
/*...............PROGRAM FOR GENERATING SIMULATION MODEL..............*/

import java.io.*;

import java.util.*;

import java.io.IOException;

public class ga

{

        public static void main(String args[])throws Exception
```

```
    {       }

    public static void gacalc(double trmp[][][],double task1[][],double ll[][],int
no_tasks,int no_machines,int iteration,String gafile,String gautil)throws Exception

    {

        max1 ip1=new max1();

        min1 ip2=new min1();

        int no_iterations=20;

        double maccap[]=new double[no_machines];

        int resordervector[]=new int[no_tasks];

        for(int f=0;f<no_machines;f++)

        maccap[f]=0;

        int index=0;

        double maxx,minn;

        double minmkspan[]=new double[no_iterations];

        crossovermutation cm=new crossovermutation();

        int ordervector1[]=new int[no_tasks];

        int ordervector2[]=new int[no_tasks];

        ordervector1=ip1.maxmin(ll,no_tasks);

        ordervector2=ip2.minmin(ll,no_tasks);
```

```
            int resarr[][]=new int[no_iterations][no_tasks];
      resarr=cm.crossovercalc(ordervector1,ordervector2,no_tasks,no_machines,tr
mp,task1,ll,no_iterations);
            System.out.println("resarr");
            for(int w=0;w<no_iterations;w++)
            {
            for(int a=0;a<no_tasks;a++)
            System.out.print(resarr[w][a]+"\t");
            System.out.println();
            }
            for(int i=0;i<no_iterations;i++)
                 {
                       for(int j=0;j<no_machines;j++)
                       {
                             for(int t=0;t<no_tasks;t++)
                             {
                                   if(resarr[i][t]==j)
                                         maccap[j]=maccap[j]+ll[t][j];
                             }
                 System.out.println();

                System.out.print(maccap[j]+"\t");

                }

                maxx=maccap[0];

                for(int m=0;m<no_machines;m++)

                {

                        if(maxx<maccap[m])

                        {

                                maxx=maccap[m];

                        }

                }

                        minmkspan[i]=maxx;

                        System.out.println("maxmakespan"+maxx+"\t");

                 for(int x=0;x<no_machines;x++)

                        maccap[x]=0;

        }

        for(int c=0;c<no_iterations;c++)

        System.out.println("minmakespan"+minmkspan[c]+"\t");

        minn=minmkspan[0];
```

```
            for(int s=0;s<no_iterations;s++)
            {
                if(minn>minmkspan[s])
                {
                    minn=minmkspan[s];
                    index=s;
                }
            }
            displaymakespan(minn,gafile);
            System.out.println("resordervector");
            for(int r=0;r<no_tasks;r++)
            {
                resordervector[r]=resarr[index][r];
            }
            makespancalc(resordervector,no_tasks,no_machines,ll,gautil);

    }
    public static double makespancalc(int ov[],int no_tasks,int
no_machines,double etc[][],String gautil)throws Exception
```

```
                {

                        int count=0;

                        double max;

                        double machinecap[]=new double[no_machines];

                        for(int j=0;j<no_machines;j++)

                        {

                                machinecap[j]=0;

                        }

                        for(int k=0;k<no_machines;k++)

                        {

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

                                {

                                                if(ov[i]==count)


machinecap[count]=round((machinecap[count]+etc[i][count]),4);

                                }

                                count++;

                        }

                        max=machinecap[0];
```

```
                    for(int l=0;l<no_machines;l++)

                    {

                            if(max<machinecap[l])

                                    max=machinecap[l];

                    }

                    for(int j1=0;j1<no_machines;j1++)

                    {

                            System.out.println(machinecap[j1]);

                    }


    MachineUtilization(max,machinecap,no_machines,gautil);

                    return max;

            }

/****************RESOURCE UTILIZATION********************/

public  static  void  MachineUtilization(double  makespan,double  machinecap[],int
no_machines,String gautil)throws Exception

    {

            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];
        }
        System.out.println("Machine Utilization");
        for(int i=0;i<no_machines;i++)
        System.out.println(machineUtil[i]+"\t");
        totalResourceUtil=round(totalResourceUtil/no_machines * 100,4);
    System.out.println("Total Resource Utilization: "+totalResourceUtil+ " %");
        displaymakespan(totalResourceUtil,gautil);  }
/*************TO DISPLAY MAKESPAN***************/
        public static void displaymakespan(double makespan,String
gafile)throws Exception
    {
    BufferedWriter out = new BufferedWriter(new FileWriter(gafile,true));
                try
                    {
                    Double mkspn=new Double(makespan);
```

```
                        String makespan1=mkspn.toString();

                        out.write(makespan1);

                        out.write("\r\n");

                        }

                    out.close();

        }

        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;

    }

}

/************TO DO CROSSOVER MUTATION**************/

public class crossovermutation

{

public static void main(String args[])throws Exception

    {
```

```
	DataInputStream in=new DataInputStream (System.in);

	}

	public static int[][] crossovercalc(int ordervector1[],int ordervector2[],int
no_tasks,int no_machines,double mc[][][],double tr[][],double etc[][],int
no_iterations)throws Exception

	{

		int total=no_tasks+no_tasks;

		double QOS[][]=new double [no_tasks][no_machines];

		int resultarr[]=new int[total];

		int resarr[][]=new int[no_iterations][no_tasks];

		QOSmatrix(mc,tr,no_tasks,no_machines,QOS);

	for(int w=0;w<no_iterations;w++)

		{System.out.println("iteration value"+w);

resultarr=crossover(ordervector1,ordervector2,no_tasks,QOS,no_machines,etc);

		for(int a=0;a<no_tasks;a++)

		{

			ordervector1[a]=resultarr[a];

			System.out.print(ordervector1[a]+"\t");

		}
```

```
        System.out.println();
         for(int b=no_tasks;b<total;b++)
         {
               ordervector2[b-no_tasks]=resultarr[b];
               System.out.print(ordervector2[b-no_tasks]+"\t");
         }
         for(int d=0;d<no_tasks;d++)
         {
               resarr[w][d]=ordervector1[d];
         }
         }
         return resarr;
         }
/***************CROSSOVER************/
         public static int[] crossover(int parent1[],int parent2[],int
no_tasks,double QOS[][],int no_machines,double etc[][])throws Exception
          {
               int total=no_tasks+no_tasks;
               System.out.println("etc matrix");
```

```
for(int i=0;i<no_tasks;i++)

{

        for(int j=0;j<no_machines;j++)

        System.out.print(etc[i][j]+"\t");

        System.out.println();

}

int child1[]=new int[no_tasks];

int child2[]=new int[no_tasks];

int res[]=new int[total];

int lastpoint=no_tasks-1;

double crossoverprobability=0.9;

double Pc=round(Math.random(),1);

System.out.println("probcross"+Pc);

if(Pc<crossoverprobability)

{

        int crosspoint=(int)(1+Math.random()*(lastpoint-1));

        System.out.println("crosspoint"+crosspoint);

        for(int c=0;c<crosspoint;c++)

        {
```

```
        child1[c]=parent1[c];

        child2[c]=parent2[c];

}

for(int c=crosspoint;c<no_tasks;c++)

{

        child1[c]=parent2[c];

        child2[c]=parent1[c];

}

System.out.println("parent1");

for(int l=0;l<no_tasks;l++)

{

        System.out.print(parent1[l]+"\t");

}

System.out.println();

System.out.println("parent2");

        for(int m=0;m<no_tasks;m++)

{

        System.out.print(parent2[m]+"\t");

}
```

```
        System.out.println();

        System.out.println("child1");

        for(int p=0;p<no_tasks;p++)

{

        System.out.print(child1[p]+"\t");

}

        System.out.println();

        System.out.println("child2");

        for(int q=0;q<no_tasks;q++)

{

        System.out.print(child2[q]+"\t");

}

System.out.println();

System.out.println("parent1");

mutate(parent1,etc,no_tasks);

for(int p1=0;p1<no_tasks;p1++)

System.out.print(parent1[p1]+"\t");

System.out.println("parent2");

mutate(parent2,etc,no_tasks);
```

```
                for(int r1=0;r1<no_tasks;r1++)

                        System.out.print(parent2[r1]);

                        System.out.println();

                System.out.println("child1");

                mutate(child1,etc,no_tasks);

                for(int j1=0;j1<no_tasks;j1++)

                        System.out.print(child1[j1]);

                        System.out.println();

                System.out.println("child2");

                mutate(child2,etc,no_tasks);

                for(int g1=0;g1<no_tasks;g1++)

                        System.out.print(child2[g1]);

                        System.out.println();

QOSsatisfaction(QOS,etc,no_tasks,no_machines,parent1);

QOSsatisfaction(QOS,etc,no_tasks,no_machines,parent2);

QOSsatisfaction(QOS,etc,no_tasks,no_machines,child1);

QOSsatisfaction(QOS,etc,no_tasks,no_machines,child2);
res=selection(parent1,parent2,child1,child2,etc,no_tasks,no_machines);

                return res;
```

```
           }

           else

           {

                 System.out.println("no crossover");

QOSsatisfaction(QOS,etc,no_tasks,no_machines,parent1);

QOSsatisfaction(QOS,etc,no_tasks,no_machines,parent2);

double make1,make2;
make1=makespancalc(parent1,no_tasks,no_machines,etc);
make2=makespancalc(parent2,no_tasks,no_machines,etc);

if(make1<make2)

      {

                            for(int t=0;t<no_tasks;t++)

                              {

                                  res[t]=parent1[t];

                            }

                            for(int y=no_tasks;y<total;y++)

                            {

                                  res[y]=parent2[y-no_tasks];

                            }
```

```
                                        }
                                        else
                                        {
                                        for(int t1=0;t1<no_tasks;t1++)
                                         {
                                                res[t1]=parent2[t1];
                                        }
                                        for(int y1=no_tasks;y1<total;y1++)
                                        {
                                                res[y1]=parent1[y1-no_tasks];
                                        }
                                        }
                                  return res;
                            }
                      }
/*************MUTATION************/
            public static void mutate(int arrmutate[],double etc[][],int no_tasks)
            {
                  double mutationprobability=0.05;
double Pm=round(Math.random(),2);

System.out.println("probmutate"+Pm);

if(Pm<mutationprobability)

{

        double min=etc[0][arrmutate[0]];

        int mutatepoint1=0;

        int mutatepoint2=0;

        for(int x=0;x<no_tasks;x++)

        {

                if(min>etc[x][arrmutate[x]])

                {

                        min=etc[x][arrmutate[x]];

                        mutatepoint1=x;

                }

        }

        System.out.println("min"+min);

        double max=etc[0][arrmutate[0]];

        for(int y=0;y<no_tasks;y++)

        {
                if(max<etc[y][arrmutate[y]])
                {
                        max=etc[y][arrmutate[y]];
                        mutatepoint2=y;
                }
        }
        System.out.println("max"+max);
        int temp=arrmutate[mutatepoint1];
        arrmutate[mutatepoint1]=arrmutate[mutatepoint2];
        arrmutate[mutatepoint2]=temp;
        System.out.println("mutatepoint1"+mutatepoint1);
        System.out.println("mutatepoint2"+mutatepoint2);
        System.out.println("mutated array");
        for(int z=0;z<no_tasks;z++)
        System.out.print(arrmutate[z]);
        System.out.println();
}
else
        System.out.println("no mutation");
```

```
}
/************SELECTION************/
public static int[] selection(int p1[],int p2[],int child1[],int child2[],double etc[][],int no_tasks,int no_machines)throws Exception
{
double makespan1=0,makespan2=0,makespan3=0,makespan4=0;
        double fitness1=0,fitness2=0,fitness3=0,fitness4=0;
        double max1=0,max2=0;
        int index=0;
        int iteration=5;
        double minmakespan=0;
        int total=no_tasks+no_tasks;
        int result[]=new int[total];
        double fitness[]=new double[4];
        int result1[]=new int[no_tasks];
        int result2[]=new int[no_tasks];
        makespan1=makespancalc(p1,no_tasks,no_machines,etc);
        makespan2=makespancalc(p2,no_tasks,no_machines,etc);
        makespan3=makespancalc(child1,no_tasks,no_machines,etc);
```

```
                makespan4=makespancalc(child2,no_tasks,no_machines,etc);

     System.out.println("mk1,mk2,mk3,mk4"+makespan1+"\t"+makespan2+"\t"
+makespan3+"\t"+makespan4);

                fitness1=1/makespan1;

                fitness2=1/makespan2;

                fitness3=1/makespan3;

                fitness4=1/makespan4;

System.out.println("fitness"+fitness1+"\t"+fitness2+"\t"+fitness3+"\t"+fitness4);

                fitness[0]=fitness1;

                fitness[1]=fitness2;

                fitness[2]=fitness3;

                fitness[3]=fitness4;

                max1=fitness[0];

                for(int k=0;k<4;k++)

                {

                     if(max1<fitness[k])

                     {

                          max1=fitness[k];
```

```
        index=k;
    }
}
if(index==0)
    minmakespan=makespan1;
    else if(index==1)
        minmakespan=makespan2;
        else if(index==2)
            minmakespan=makespan3;
            else if(index==3)
                minmakespan=makespan4;
if(max1==fitness1)
{
    System.out.println("result1");
    for(int e=0;e<no_tasks;e++)
    {
        result1[e]=p1[e];
    }
    fitness[0]=0;
```

```
}
else if(max1==fitness2)
{
        System.out.println("result1");
        for(int f=0;f<no_tasks;f++)
        {
                result1[f]=p2[f];
        }
        fitness[1]=0;
}
else if(max1==fitness3)
{
        System.out.println("result1");
        for(int g=0;g<no_tasks;g++)
        {
                result1[g]=child1[g];
        }
        fitness[2]=0;
}
```

```
else if(max1==fitness4)
{
        System.out.println("result1");
        for(int h=0;h<no_tasks;h++)
        {
                result1[h]=child2[h];
        }
        fitness[3]=0;
}
max2=fitness[0];
for(int g=0;g<4;g++)
{
        if(max2<fitness[g])
        {
                max2=fitness[g];
        }
}
if(max2==fitness1)
{
        System.out.println("result2");
        for(int e1=0;e1<no_tasks;e1++)
        {
                result2[e1]=p1[e1];
        }
        System.out.println();
        fitness[0]=0;
}
else if(max2==fitness2)
{
        System.out.println("result2");
        for(int f1=0;f1<no_tasks;f1++)
        {
                result2[f1]=p2[f1];
        }
        System.out.println();
        fitness[1]=0;
}
else if(max2==fitness3)
```

```
{
        System.out.println("result2");
        for(int g1=0;g1<no_tasks;g1++)
        {
                result2[g1]=child1[g1];
        }
        fitness[2]=0;
}
else if(max2==fitness4)
{
        System.out.println("result2");
        for(int h1=0;h1<no_tasks;h1++)
        {
                result2[h1]=child2[h1];
        }
        fitness[3]=0;
}
System.out.print("result1");
        for(int i=0;i<no_tasks;i++)
```

```
System.out.print(result1[i]+"\t");

System.out.println();

System.out.print("result2");

for(int j=0;j<no_tasks;j++)

System.out.print(result2[j]+"\t");

System.out.println();

for(int t=0;t<no_tasks;t++)

{

        result[t]=result1[t];

}



for(int y=no_tasks;y<total;y++)

{

        result[y]=result2[y-no_tasks];

}

System.out.print("merged array");

for(int p=0;p<total;p++)

        System.out.print(result[p]+"\t");

        return result;
        }
/*************MAKESPAN CALCULATION**************/
    public    static    double    makespancalc(int    ov[],int    no_tasks,int
no_machines,double etc[][])
                {
                        int count=0;
                        double max;
                        double machinecap[]=new double[no_machines];
                        for(int j=0;j<no_machines;j++)
                        {
                                machinecap[j]=0;
                        }
                        for(int k=0;k<no_machines;k++)
                        {
                                for(int i=0;i<no_tasks;i++)
                                {
                                                if(ov[i]==count)
    machinecap[count]=round((machinecap[count]+etc[i][count]),4);
```

```
                }
                count++;
            }
            max=machinecap[0];
            for(int l=0;l<no_machines;l++)
            {
                if(max<machinecap[l])
                    max=machinecap[l];
            }
            for(int j1=0;j1<no_machines;j1++)
            {
                System.out.println(machinecap[j1]);
            }
            return max;
        }
      /*********CALCULATION OF QoS MATRIX*********/
    public static void QOSmatrix(double mc[][][],double tr[][],int no_tasks,int
no_machines,double QOS[][])
    {
```

```
int c=0,r=0,d=0;

double cost=0,ram=0,deadline=0,w=0;

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

{

        for(int j=0;j<no_machines;j++)

        {

                cost=(tr[i][0]/mc[j][i][0]);

                if(cost>=1.0)

                {

                        c=1;

                        cost=cost*0.5;

                }

                else

                        cost=0.0;

                ram=(mc[j][i][1]/tr[i][1]);

                if(ram>=1.0)

                {

                        r=1;

                        ram=ram*0.25;
}
else
        ram=0.0;
deadline=(tr[i][2]/mc[j][i][2]);
if(deadline>=1.0)
{
        d=1;
        deadline=deadline*0.25;
}
else
        deadline=0.0;
if((c==1)&&(r==1)&&(d==1))
{
        w=round((cost+ram+deadline),4);
        QOS[i][j]=w;
        c=0;
        d=0;
        r=0;
}
```

```
                    else

                    {

                            c=0;

                            d=0;

                            r=0;

                            QOS[i][j]=0;

                    }

                    System.out.print(QOS[i][j]+"\t");

               }

               System.out.println();

          }}
/************CHECKING FOR QoS SATISFACTION*************/

     public static void QOSsatisfaction(double QOS[][],double etc[][],int
no_tasks,int no_machines,int ordervector[])

     {

          double max,min;

          int machine=0,machine1=0;

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

          {
```

```
if(QOS[i][ordervector[i]]==0)
{
        max=QOS[i][0];
        for(int j=0;j<no_machines;j++)
        {
                if(max<QOS[i][j])
                {
                        max=QOS[i][j];
                        machine=j;
                }
        }
        if(max!=0)
                ordervector[i]=machine;
                else
                {       min=etc[i][0];
                        for(int k=0;k<no_machines;k++)
                        {
                                if(min>etc[i][k])
                                {
```

```
                                min=etc[i][k];

                                machine1=k;

                            }

                        }ordervector[i]=machine1;

                    }

            }

        }

            System.out.println("qos satisfied ordervector");

        for(int o=0;o<no_tasks;o++)

                System.out.print(ordervector[o]+"\t");

    }

}
```

CHAPTER IV

EXPERIMENTAL RESULTS AND DISCUSSION

4.1 COMPARISON METRICS

4.1.1 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]

4.1.2 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

5.1MAKESPAN COMPARISON

Fig 2.1

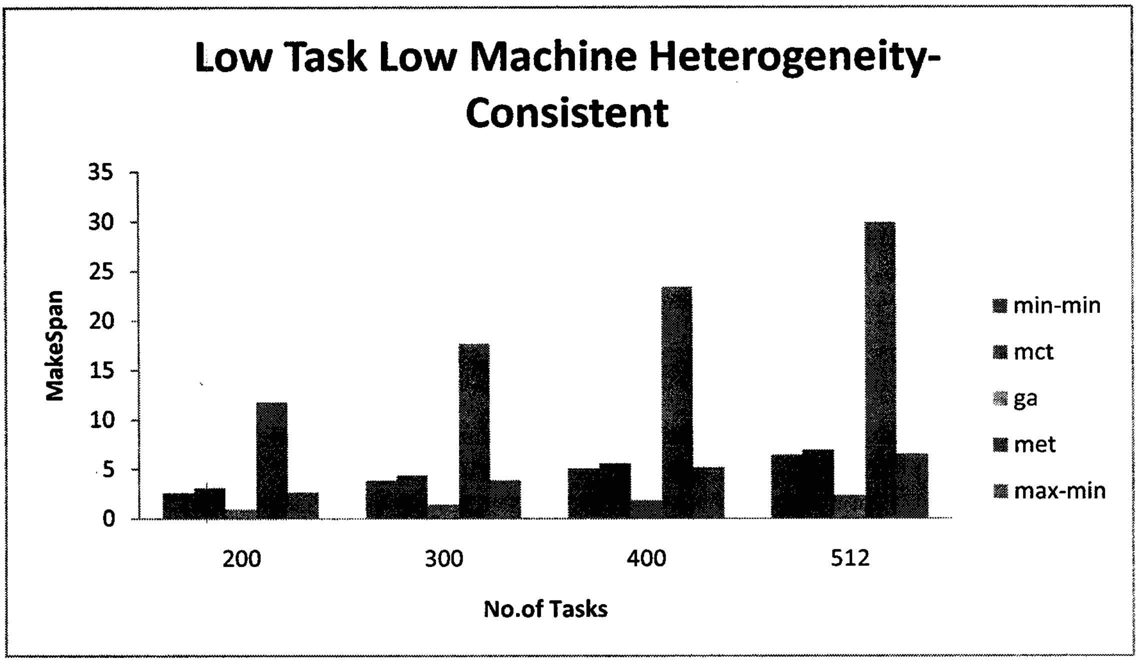


Fig 2.2

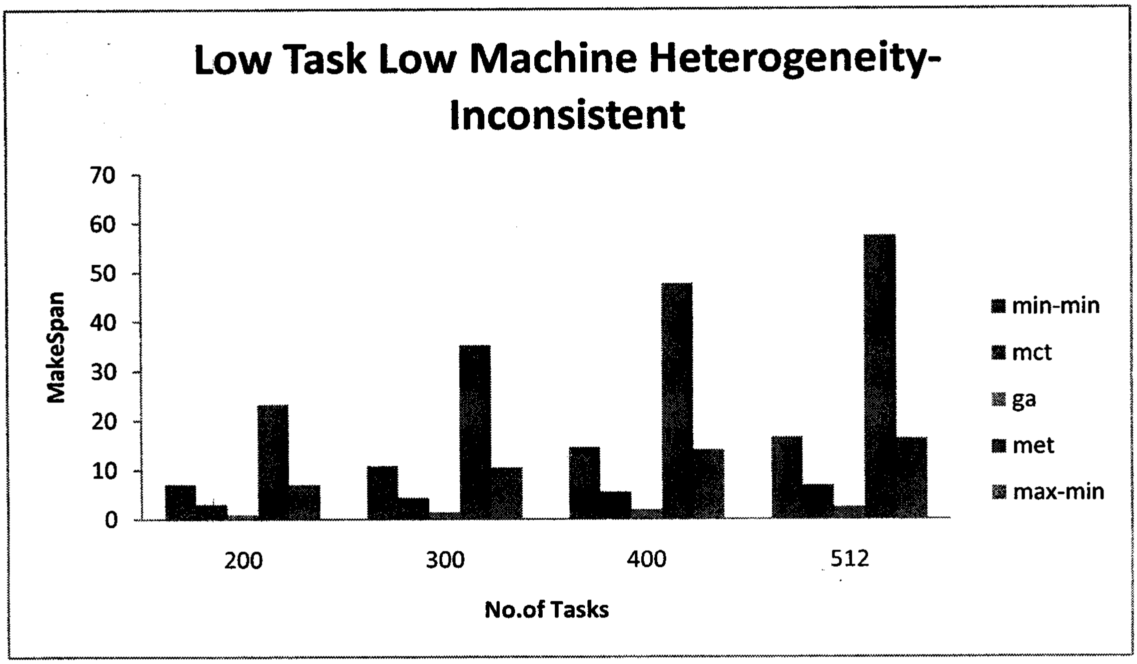

Low Task Low Machine Heterogeneity-
Inconsistent
MakeSpan
70
60
50
40
30
20
10
0
200
300
400
512
No.of Tasks
min-min
mct
ga
met
max-min

Fig 2.3

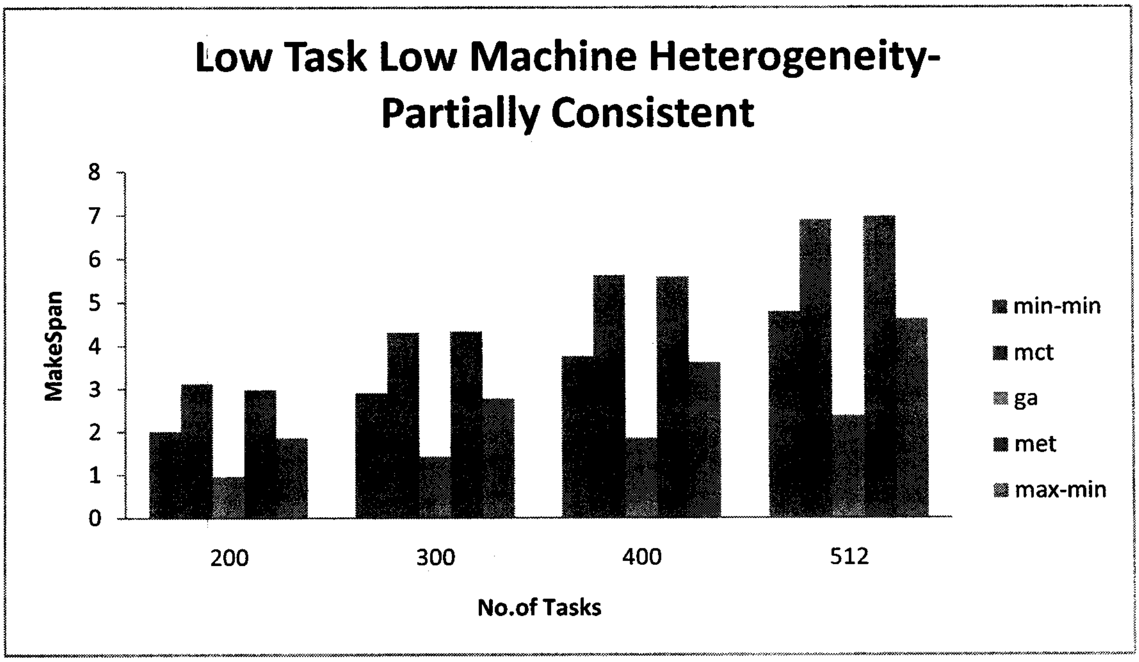

Low Task Low Machine Heterogeneity-
Partially Consistent
MakeSpan
8
7
6
5
4
3
2
1
0
200
300
400
512
No.of Tasks
min-min
mct
ga
met
max-min

Fig 2.4

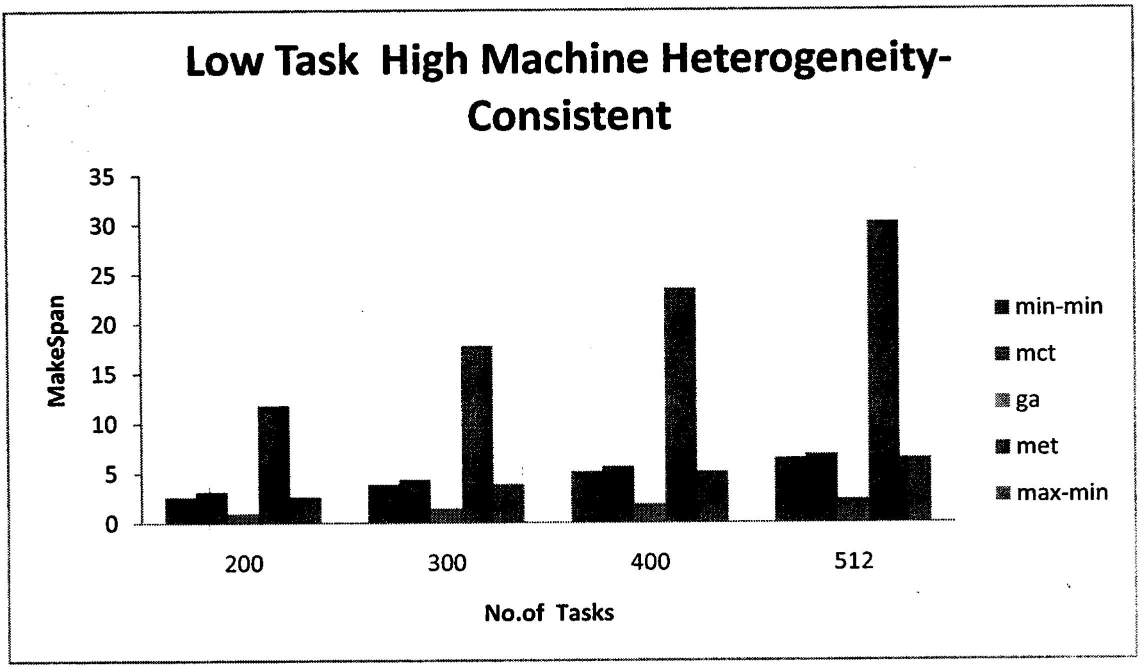

Low Task High Machine Heterogeneity-Consistent
MakeSpan
35
30
25
20
15
10
5
0
200
300
400
512
No.of Tasks
min-min
mct
ga
met
max-min

Fig 2.5

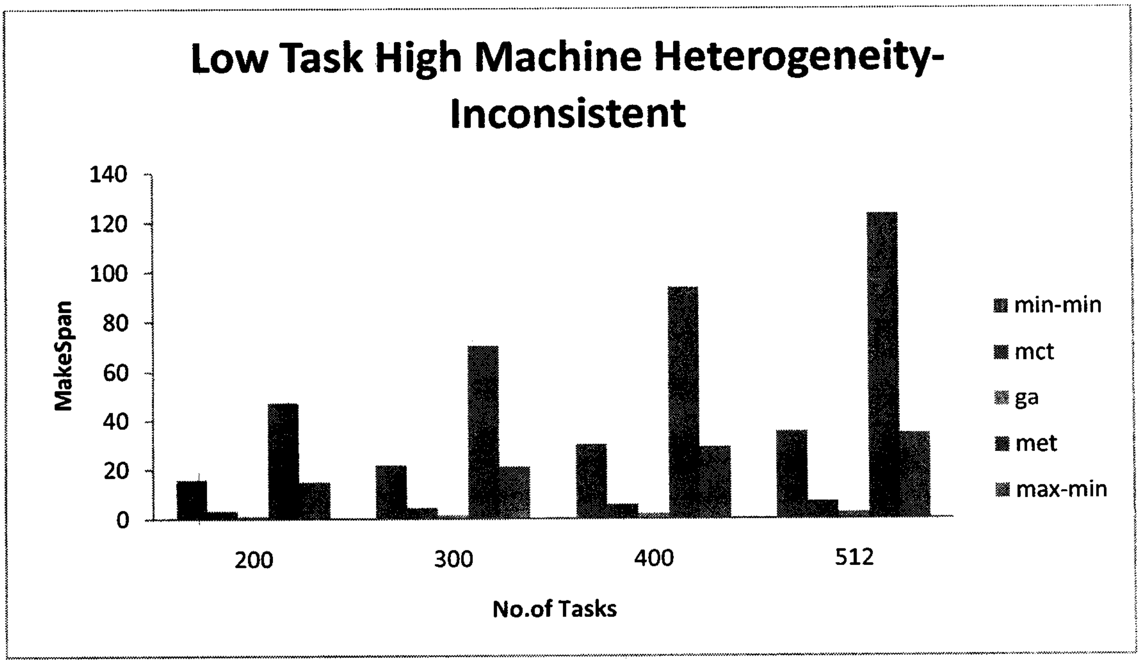

Low Task High Machine Heterogeneity-Inconsistent
MakeSpan
140
120
100
80
60
40
20
0
200
300
400
512
No.of Tasks
min-min
mct
ga
met
max-min

Fig 2.6

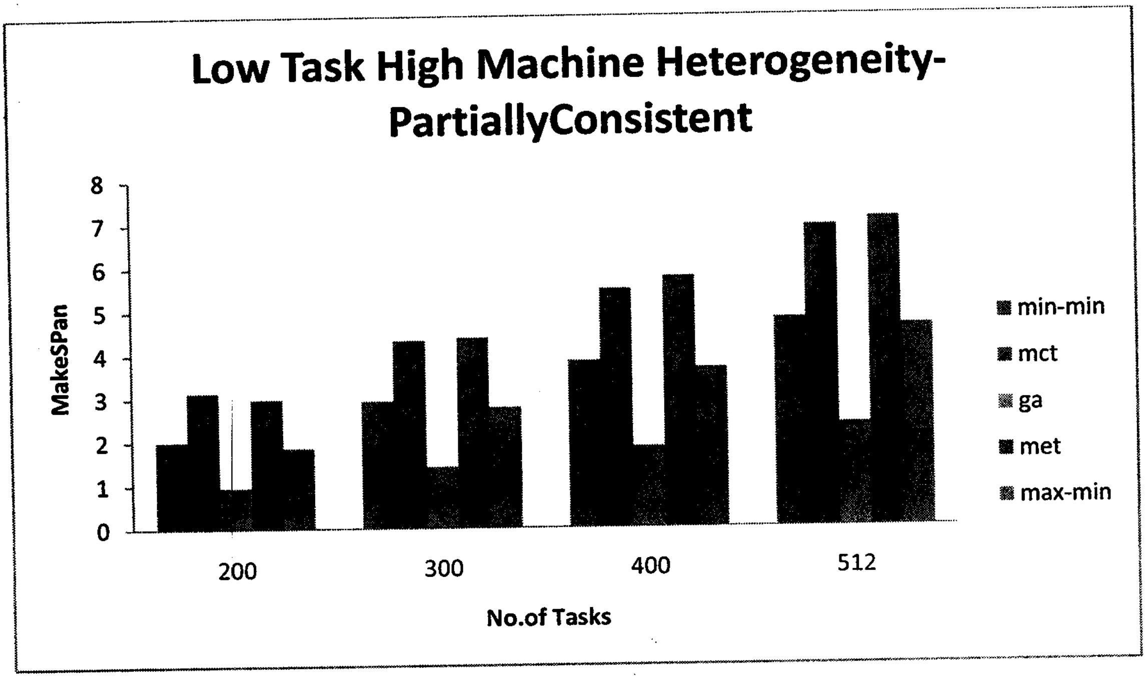

Low Task High Machine Heterogeneity-PartiallyConsistent
MakeSPan
0
1
2
3
4
5
6
7
8
200
300
400
512
No.of Tasks
min-min
mct
ga
met
max-min

Fig 2.7

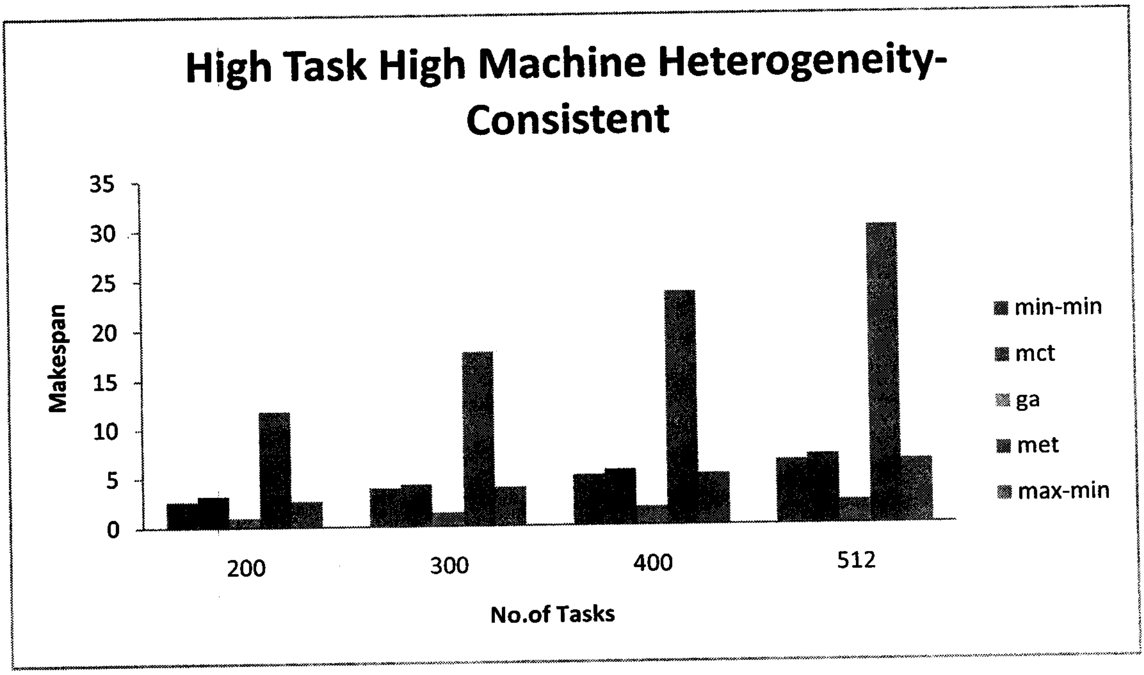

High Task High Machine Heterogeneity-Consistent
Makespan
0
5
10
15
20
25
30
35
200
300
400
512
No.of Tasks
min-min
mct
ga
met
max-min

Fig 2.8

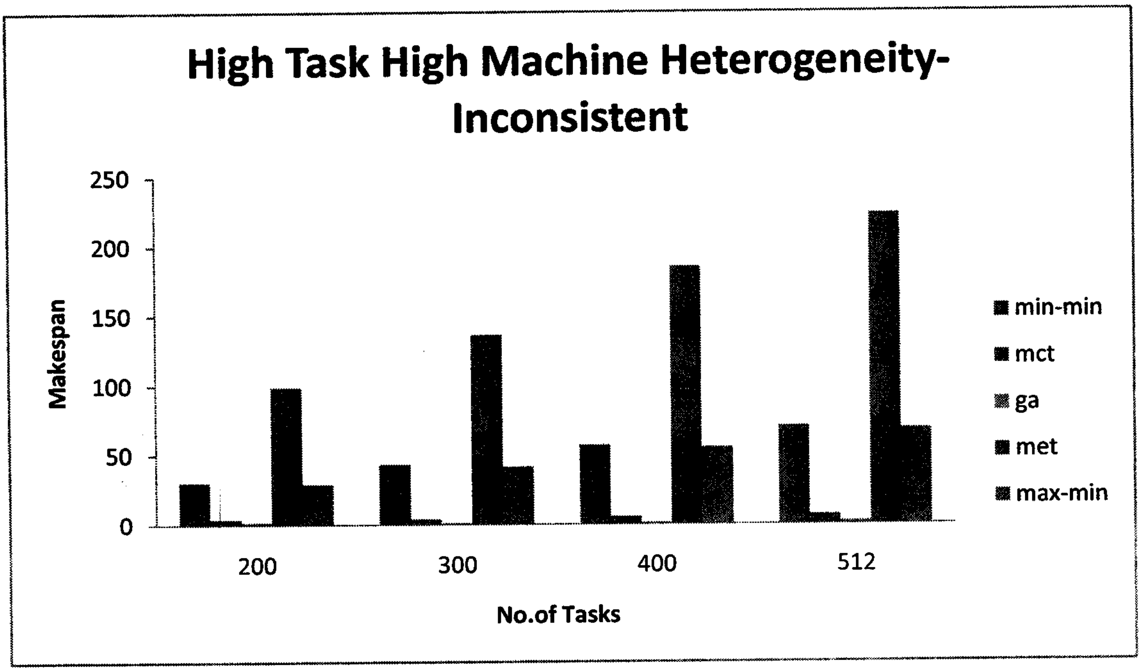

High Task High Machine Heterogeneity-Inconsistent
Makespan
250
200
150
100
50
0
200
300
400
512
No.of Tasks
min-min
mct
ga
met
max-min

Fig 2.9

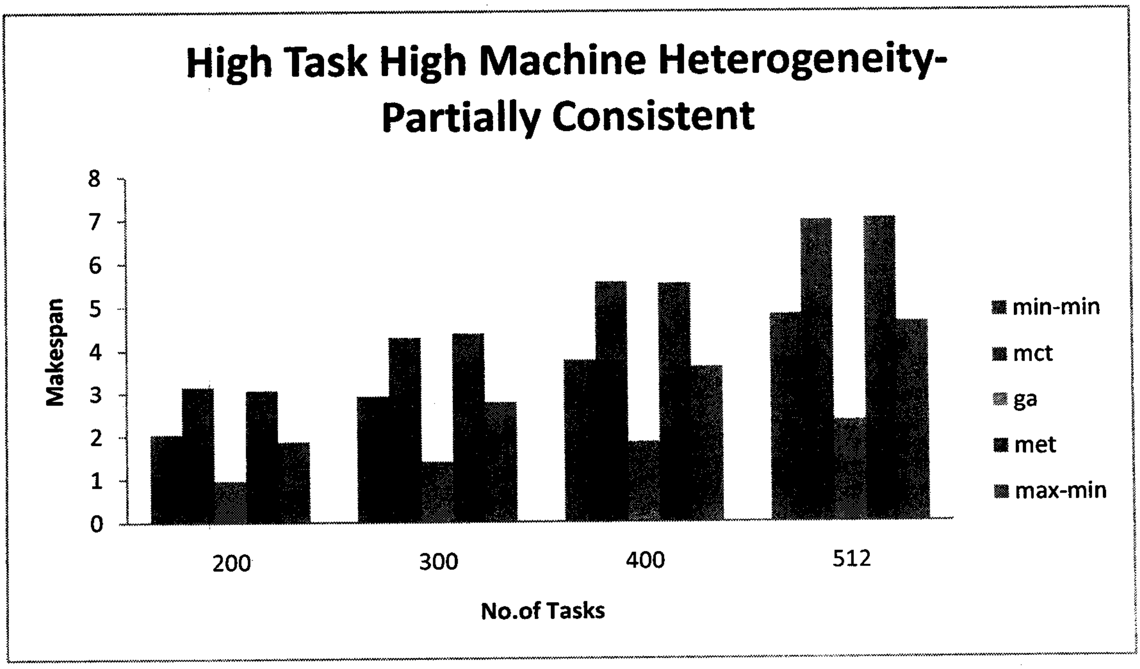

High Task High Machine Heterogeneity-Partially Consistent
Makespan
8
7
6
5
4
3
2
1
0
200
300
400
512
No.of Tasks
min-min
mct
ga
met
max-min

Fig 2.10

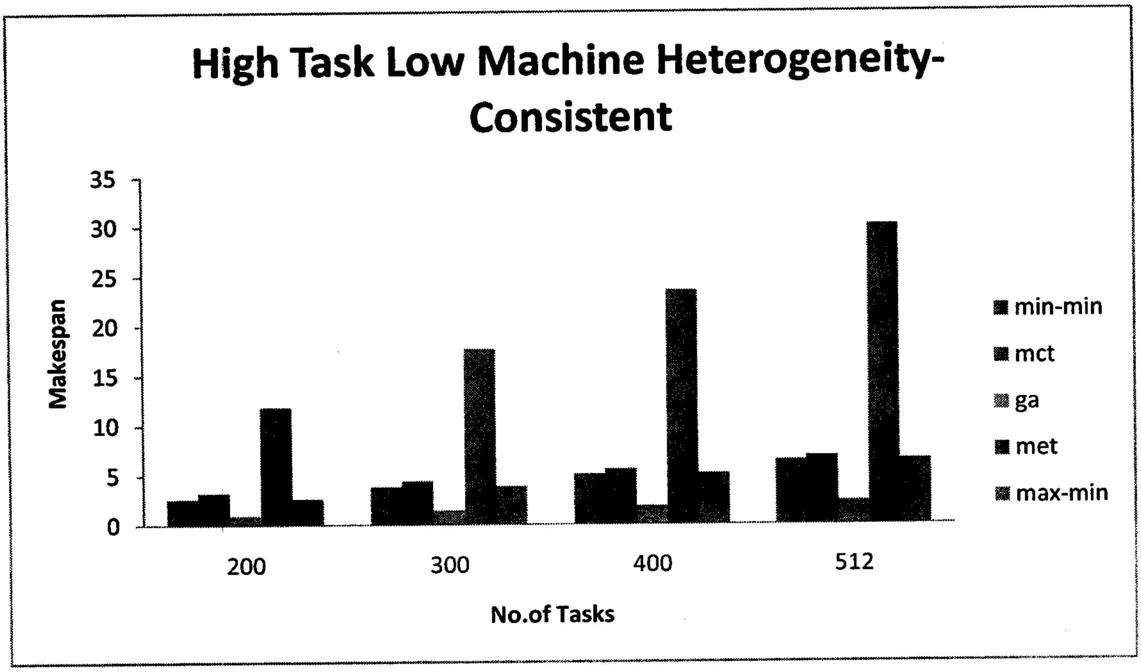


Fig 2.11

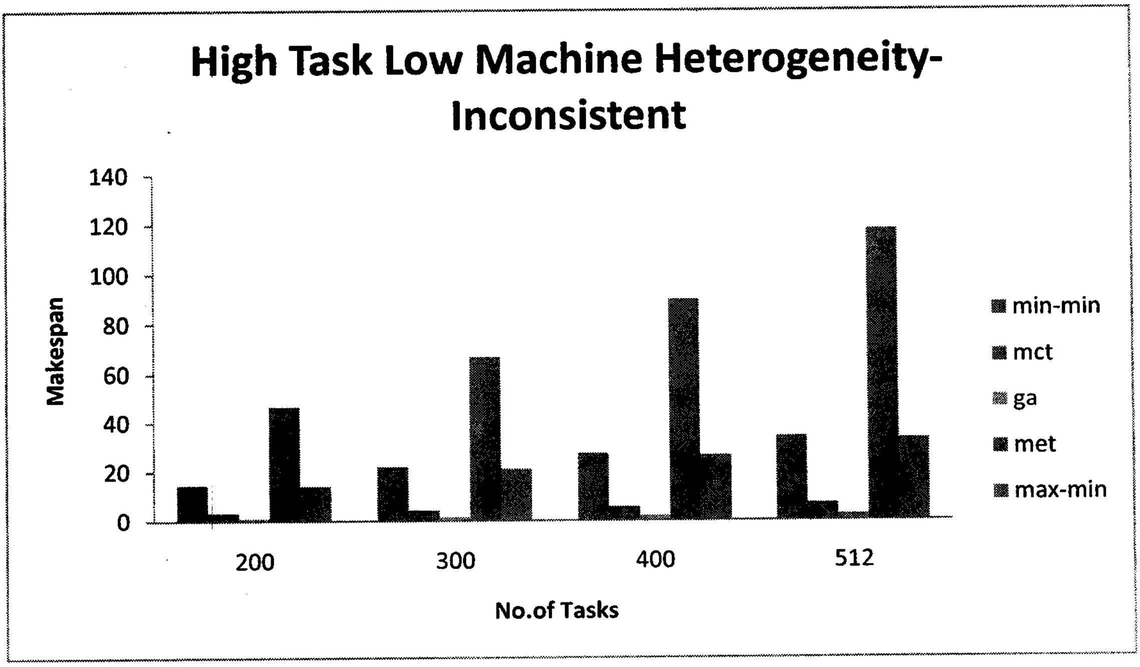
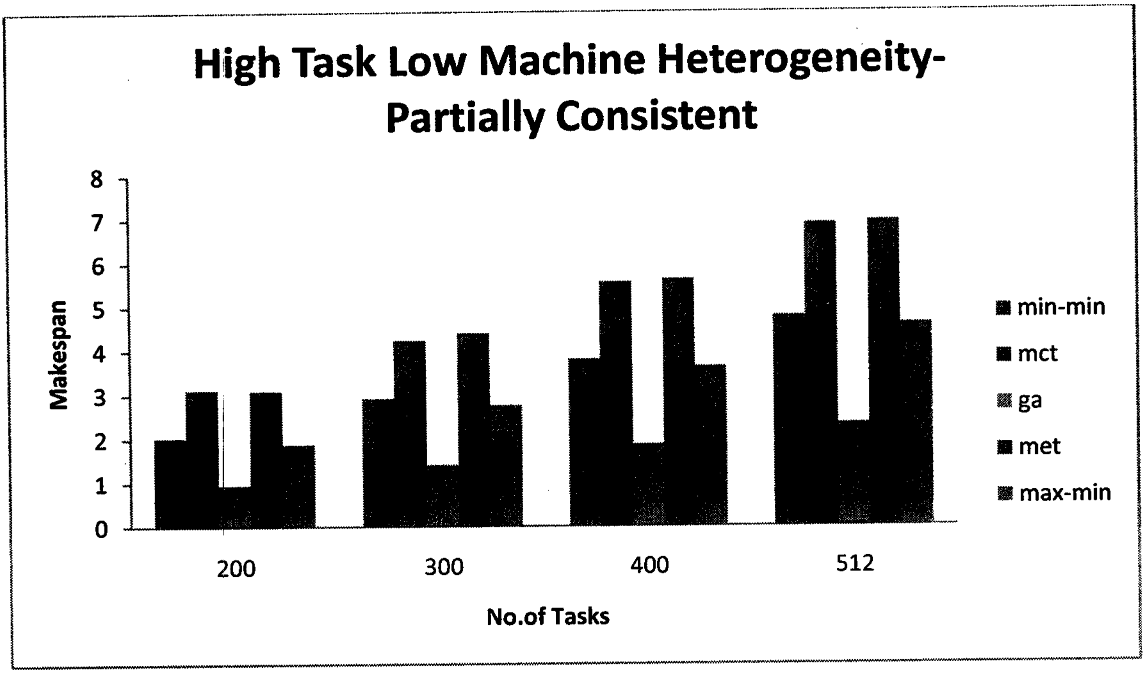


Fig 2.12



5.2 AVERAGE RESOURCE UTILIZATION COMPARISON

Fig 3.1

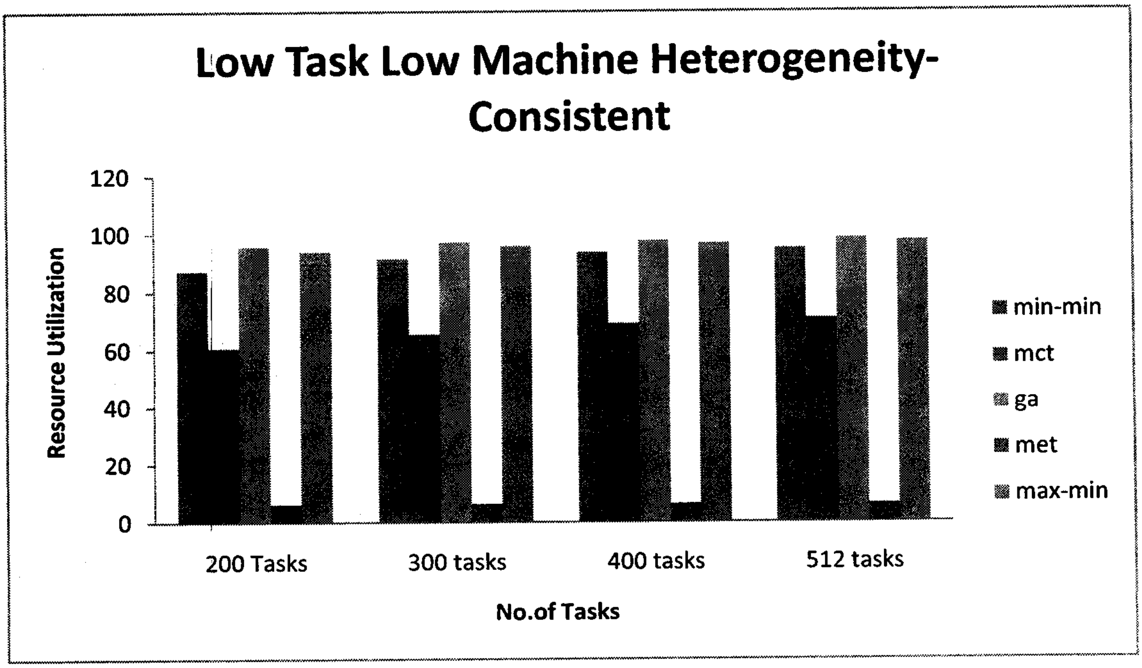


Fig 3.2

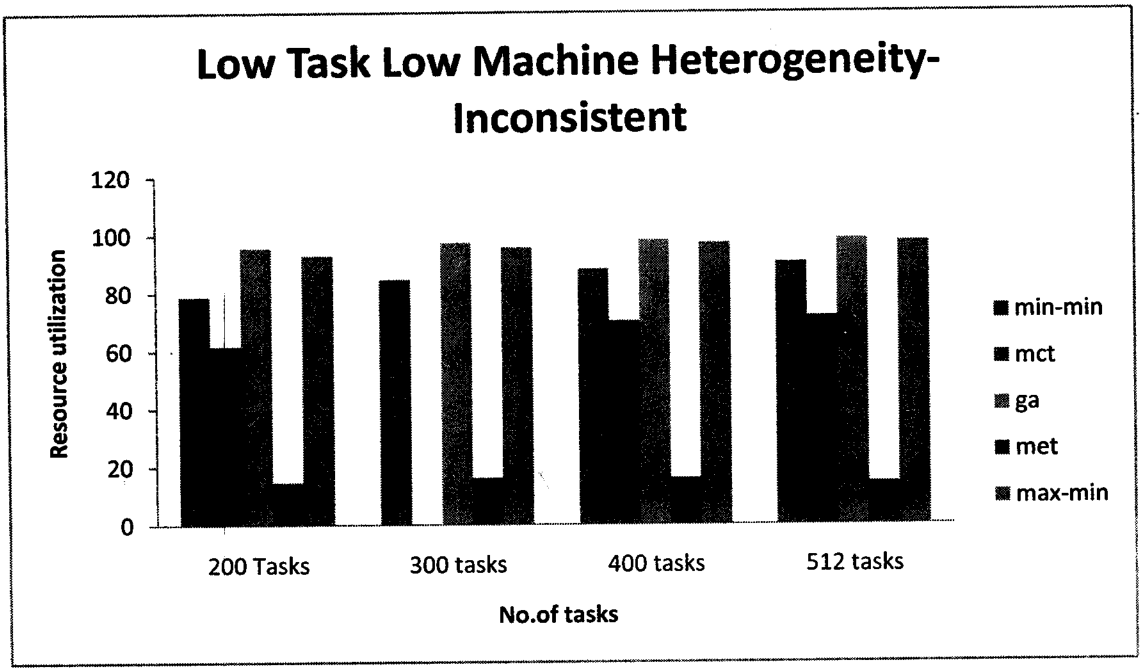

Low Task Low Machine Heterogeneity-Inconsistent
Resource utilization
120
100
80
60
40
20
0
200 Tasks
300 tasks
400 tasks
512 tasks
No.of tasks
min-min
mct
ga
met
max-min

Fig 3.3

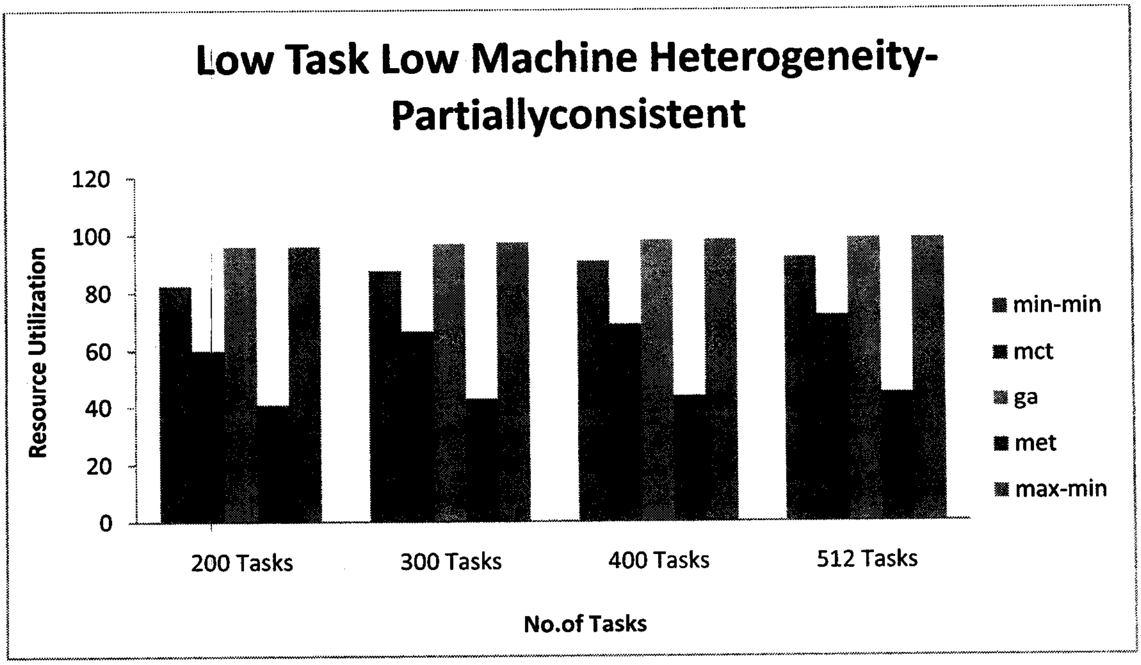

Low Task Low Machine Heterogeneity-Partiallyconsistent
Resource Utilization
120
100
80
60
40
20
0
200 Tasks
300 Tasks
400 Tasks
512 Tasks
No.of Tasks
min-min
mct
ga
met
max-min

Fig 3.4

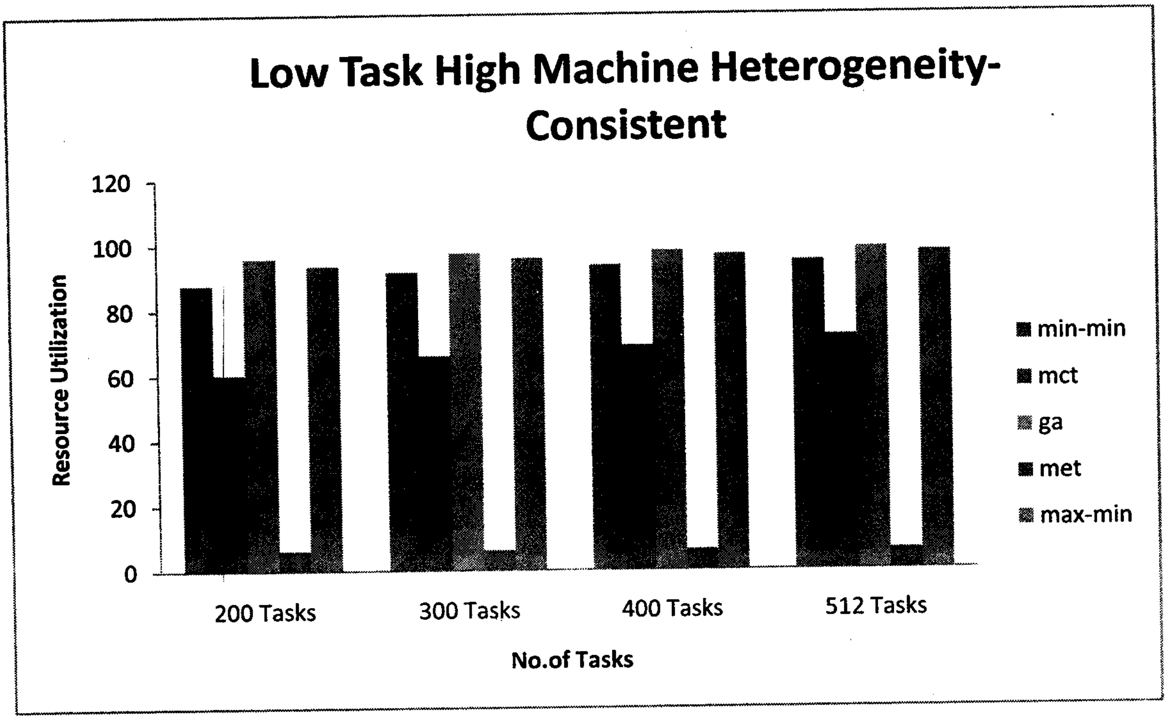

Low Task High Machine Heterogeneity-Consistent
120
100
80
60
40
20
0
Resource Utilization
200 Tasks
300 Tasks
400 Tasks
512 Tasks
No.of Tasks
min-min
mct
ga
met
max-min

Fig 3.5

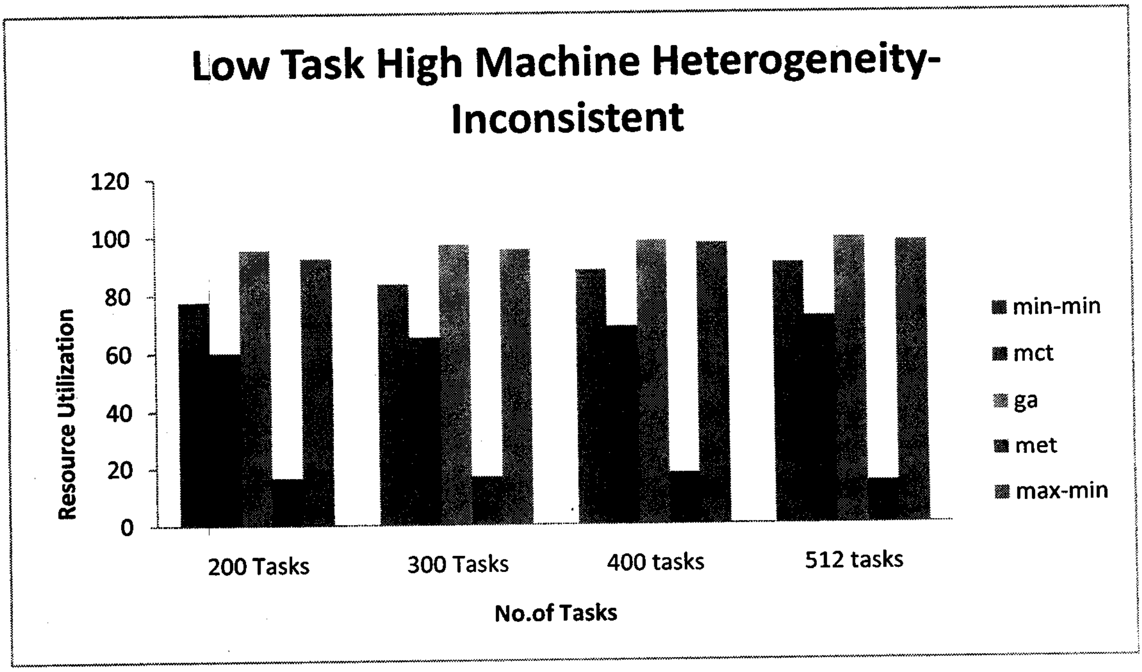

Low Task High Machine Heterogeneity-Inconsistent
120
100
80
60
40
20
0
Resource Utilization
200 Tasks
300 Tasks
400 tasks
512 tasks
No.of Tasks
min-min
mct
ga
met
max-min

Fig 3.6

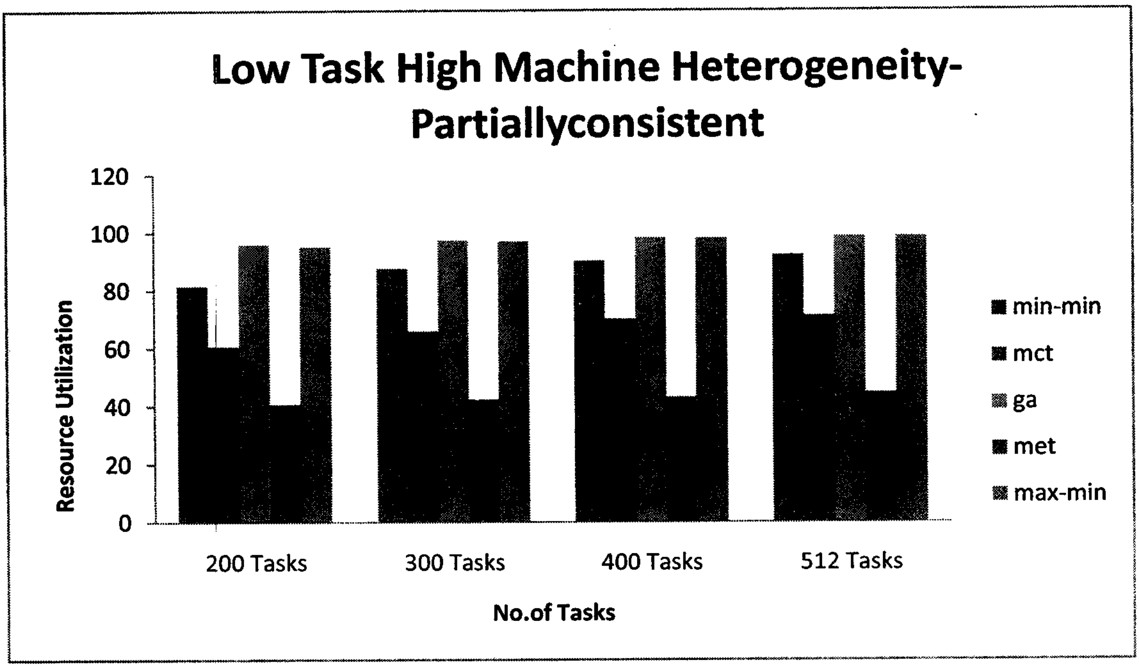

Low Task High Machine Heterogeneity-
Partiallyconsistent
120
100
80
60
40
20
0
Resource Utilization
200 Tasks
300 Tasks
400 Tasks
512 Tasks
No.of Tasks
min-min
mct
ga
met
max-min

Fig 3.7

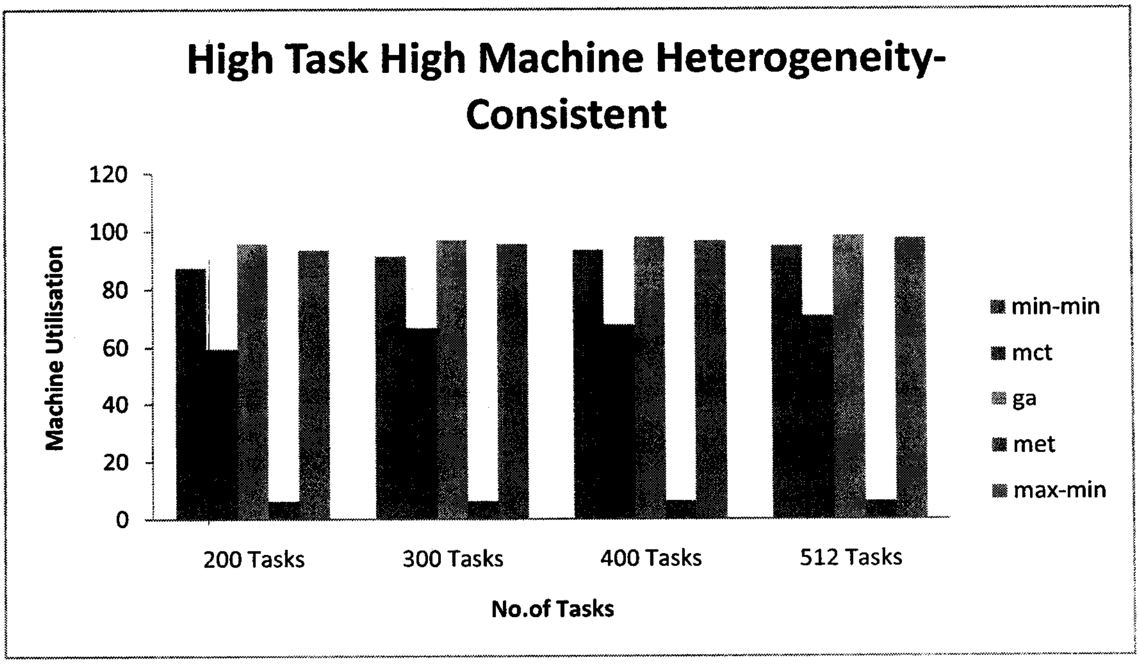

High Task High Machine Heterogeneity-
Consistent
120
100
80
60
40
20
0
Machine Utilisation
200 Tasks
300 Tasks
400 Tasks
512 Tasks
No.of Tasks
min-min
mct
ga
met
max-min

Fig 3.8

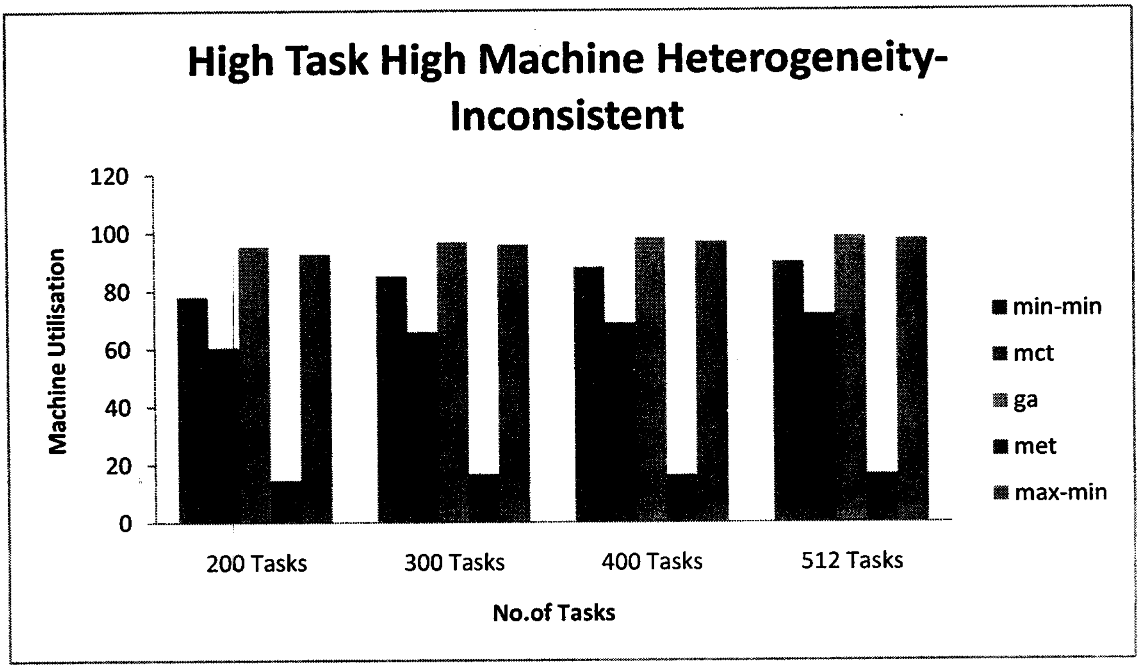

High Task High Machine Heterogeneity-
Inconsistent
Machine Utilisation
120
100
80
60
40
20
0
200 Tasks
300 Tasks
400 Tasks
512 Tasks
No.of Tasks
min-min
mct
ga
met
max-min

Fig 3.9

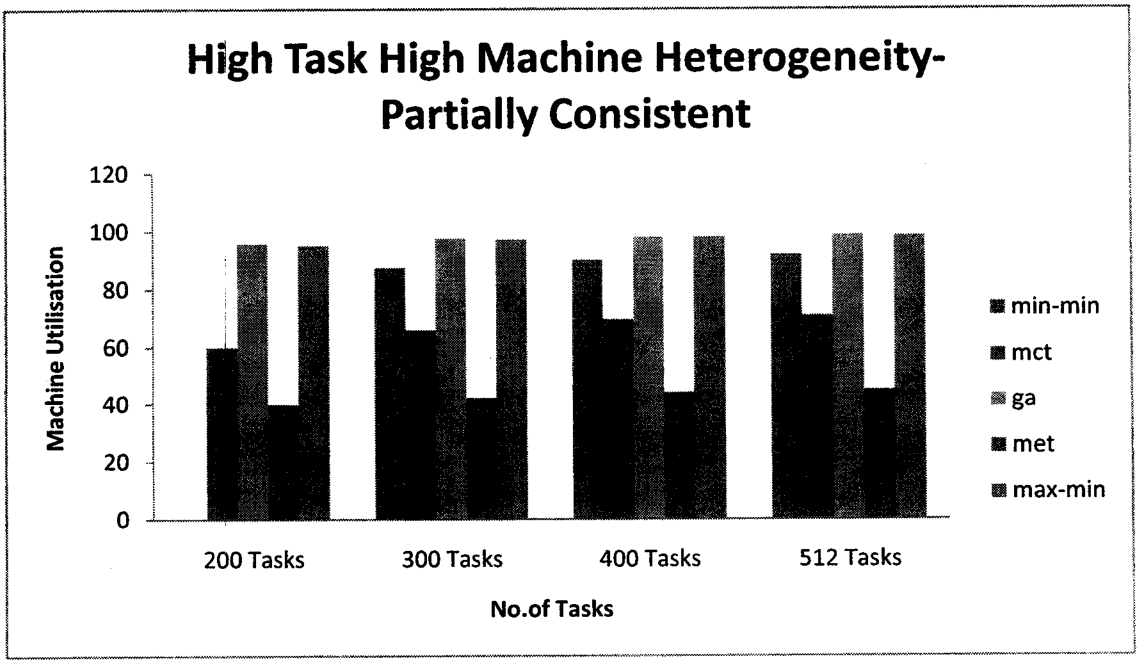

High Task High Machine Heterogeneity-
Partially Consistent
Machine Utilisation
120
100
80
60
40
20
0
200 Tasks
300 Tasks
400 Tasks
512 Tasks
No.of Tasks
min-min
mct
ga
met
max-min

Fig 3.10

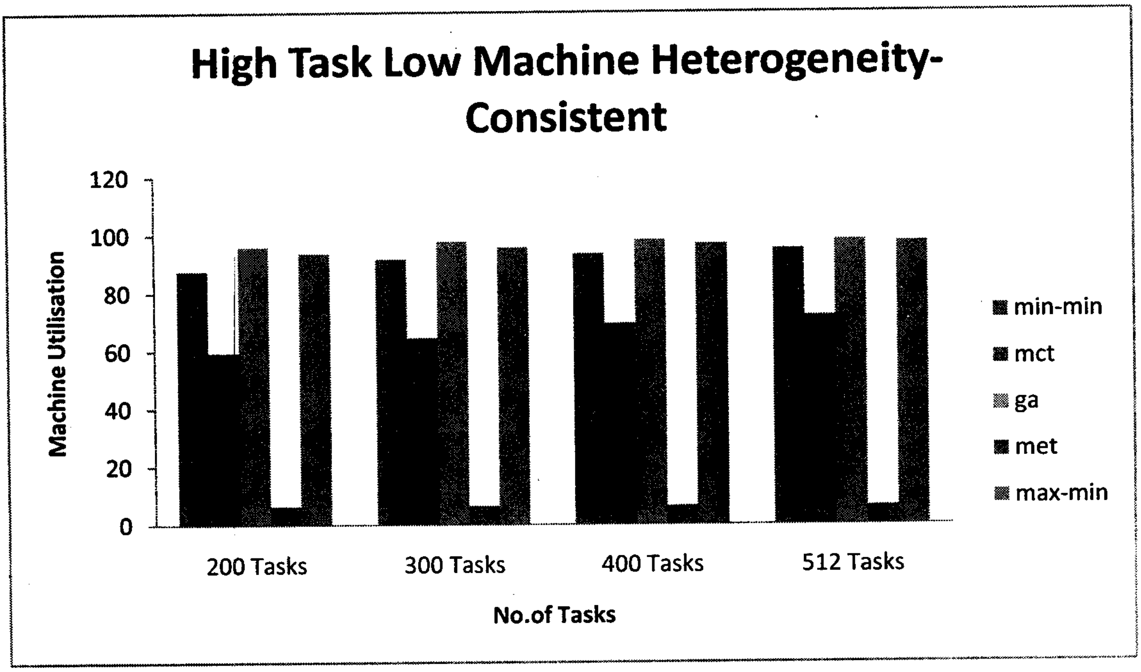


Fig 3.11

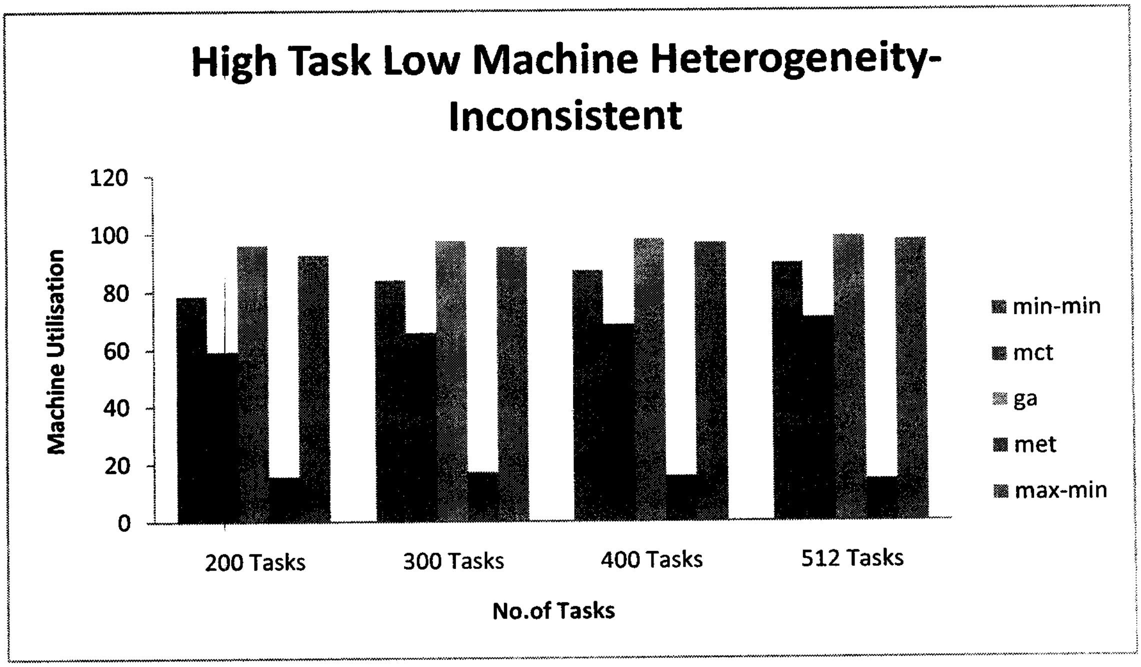
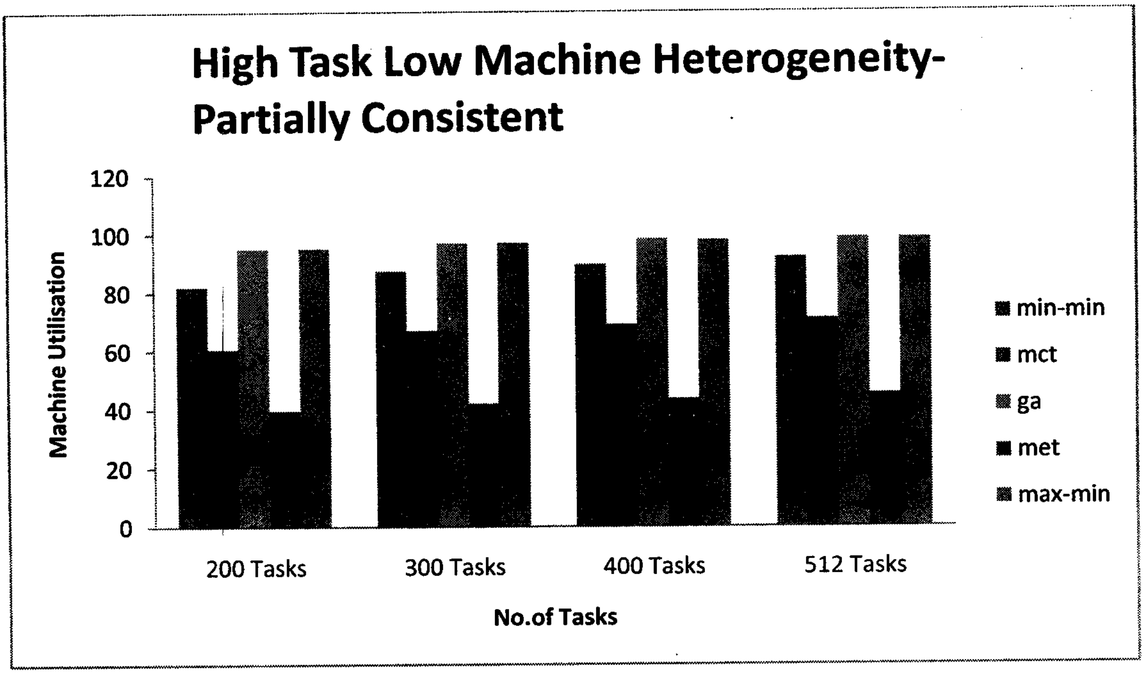


Fig 3.12


High Task Low Machine Heterogeneity-Partially Consistent
Machine Utilisation
120
100
80
60
40
20
0
200 Tasks
300 Tasks
400 Tasks
512 Tasks
No.of Tasks
min-min
mct
ga
met
max-min

CHAPTER VI

6.1 CONCLUSION

In this project, the algorithm first checks the Qos Satisfying criteria and then schedules the task to the relevant machine. The genetic algorithm implementation with QoS Satisfaction shows that it yields better performance than the already existing strategies taken up for comparison i.e. genetic algorithm has minimized the makespan and has given better resource utilization for almost all cases.

6.2 REFERENCES

[1] Manish Parashar, Senior Member , IEEE and Craig A.Lee, Member, IEEE, "Grid Computing: Introduction and Overview".

[2] 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 onSmart-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] D. E. Goldberg, “Genetic Algorithms in Search, Optimization and Machine Learning”, Addison-Wesley, New York, NY, 1989.

[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.

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