

**QoS Parameters Optimization in Computational Grid  
Using Genetic Algorithm**

Dissertation submitted to Jawaharlal Nehru University  
in partial fulfillment of the requirements  
for the award of degree of

**Master of Technology**  
in  
**Computer Science and Technology**

By  
Shiv Prakash  
Enrollment No.08/10MT/15

Under the Supervision of  
Dr. D.P. Vidyarthi



**School of Computer & Systems Sciences**  
**Jawaharlal Nehru University**  
**New Delhi India– 110067**  
**July 2010**

**Dedicated to Parents & Uncle**



**School of Computer & Systems Sciences**  
**जवाहरलाल नेहरू विश्वविद्यालय**  
**JAWAHARLAL NEHRU UNIVERSITY**  
**NEW DELHI-110067**

---

**Certificate**


This is certify that the dissertation entitled “QoS Parameters Optimization in Computational Grid using Genetic Algorithm” is being submitted by Mr. Shiv Prakash to School of Computer and Systems Sciences, Jawaharlal Nehru University New Delhi-110067, India in the partial fulfillment of the requirements for the award of the degree of Master of Technology in Computer Science and Technology. This work carried out by himself in the School of Computer and Systems Sciences under the supervision of Dr. D.P. Vidyarthi. The matter personified in the dissertation has not been submitted for the award of any other degree or diploma.

Supervisor

  
Dr. D.P. Vidyarthi

School of Computer & System Sciences  
Jawaharlal Nehru University  
New Delhi -110067

Dean

  
Prof. Sonajharia Minz

School of Computer & System Sciences  
Jawaharlal Nehru University  
New Delhi-110067

Prof. Sonajharia Minz  
Dean  
School of Computer & Systems Sciences  
Jawaharlal Nehru University  
New Delhi-110067



## Declaration

I hereby declare that the dissertation work entitled “QoS Parameters Optimization in Computational Grid using Genetic Algorithm” in partial fulfillment for the requirements for the award of degree of “Master of Technology in Computer Science and Technology” and submitted to School of Computer & Systems Sciences, Jawaharlal Nehru University, New Delhi-110067, India is the authentic record of my own work carried out during the time of Master of Technology in the supervision of Dr. Deo Prakash Vidyarthi. This dissertation comprises only my original work. This dissertation is less than 100,000 words in length, exclusive tables, figures and references.

The matter personified in the dissertation has not been submitted for the award of any other degree or diploma.

*Shiv Prakash*  
Shiv Prakash

Enrollment No. 08/10/MT/15

M.Tech (2008-2010)

SC&SS, JNU

New Delhi India -110067

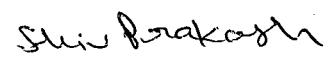
## Acknowledgement

I am very glad to express my sincere gratitude and thank to my supervisor **Dr. D.P. Vidyarthi** for his guidance in completing this dissertation. I would like to express special thank to Dr. D. P. Vidyarthi for many helpful discussions. His extensive and invaluable academic and research experience was very helpful in my dissertation. The mythology, philosophy and problem solving learn by him have been very beneficial in my dissertation work. What I have learned in my research work will definitely benefit in my career and life.

I would like to express my thanks to Dean SC&SS JNU, Prof. Sonajharia Minz in support to pursue my work in the School. Also my thanks go to School administration and librarian of software library and main library for supporting me, in whatever way they can, to make dissertation a success. Their support has been a real emphasize in completing this dissertation.

I would like to accord my sincere thanks to Mr. Dinesh Prasad Sahu, Mr. Md. Anbar, Mr. Pawan Kumar Tiwari and Mr. Sohan Kumar Yadav for their academic help for my dissertation work.

My parents & uncle not only provided me economical but emotional support also towards my endeavor for this project. I am grateful to them. Finally I would like to express thanks to each persons & thing which is directly or indirectly related to my dissertation work.

  
Shiv Prakash

Enrollment No. 08/10/MT/15

M.Tech (2008-2010)

SC&SS, JNU

New Delhi India -110067

# **QoS Parameters Optimization in Computational Grid using Genetic Algorithm**

## **Abstract**

Computational Grids have emerged as global cyber infrastructure for next generation computing. Because next generation scientific researches are being carried out by large collaboration of researchers distributed across the globe. Research communities are utilizing Computational Grid to share, manage and process the large set of tasks. This infrastructure supports the large, complex, compute intensive experiments that otherwise would have been very difficult to solve.

The dissertation discusses about the key concepts of Computational Grid. It focuses on the optimization of the QoS Parameters by scheduling the jobs in a computational grid. There are many constraints associated with the computational grid. Heterogeneity is one of the major constraints. The scheduling problem with such constraints is an NP class of problem. This problem adds further dimensionality by the grid platform which is highly heterogeneous in nature. Soft computing techniques are well suited to solve this type of problem. We, in our proposed work, have applied soft computing technique in particular Genetic Algorithm to solve this problem.

The dissertation, after brief discussion about the Grid and Genetic Algorithm, proposes a model for scheduling problem in Computational Grid. The proposed work also derives formula for the fitness function which is the turnaround time of the submitted job and speedup achieved by the grid. This function has been derived in two ways. One in which the job has not been considered as interactive and in the other the interactive modules of the jobs are considered. The model focuses on problem solving using Genetic Algorithms. We simulate the model to optimize QoS Parameters (Turnaround time and Speed up) using Genetic Algorithms by writing the program. The result has been displayed in the graph and has been analyzed. It is found that the solution converges after a finite number of generations by the experiments.

The dissertation also compares the results with one other soft computing technique (ACO). Though, it is found by the experiment of the time comparisons of results that the ACO based model is giving better results than Genetic Algorithm based result for this problem. Nonetheless, the study has been very useful in improving the model which is to be taken as the next move. It is to explore how the techniques of GA can be improved to suite the problem of interest and to tune it for performing better.

# Table of Contents

|   |           |
|---|-----------|
| Certificate.....                                  | i         |
| Declaration.....                                  | ii        |
| Acknowledgements.....                             | iii       |
| Abstracts.....                                    | iv        |
| Contents.....                                     | vi        |
| List of Figures.....                              | viii      |
| List of Tables.....                               | ix        |
| Abbreviations.....                                | x         |
| <br>  |           |
| <b>Chapter 1 Introduction.....</b>                | <b>1</b>  |
| 1.1 Computational Grid.....                       | 3         |
| 1.2 Genetic Algorithms.....                       | 6         |
| 1.3 GA as Soft Computing Tool.....                | 8         |
| 1.4 Organization of Dissertation.....             | 10        |
| <br>  |           |
| <b>Chapter 2 Computational Grid.....</b>          | <b>11</b> |
| 2.1 Multi-processor, Distributed and Cluster..... | 11        |
| 2.2 Grid computing.....                           | 14        |
| 2.3 Types of Grid.....                            | 16        |
| 2.3.1 Computational Grid.....                     | 16        |
| 2.3.2 Data Grid.....                              | 16        |
| 2.4 Issues in Computational Grid.....             | 17        |
| 2.5 The Problem.....                              | 18        |
| 2.5 Concluding Remarks.....                       | 25        |
| <br>  |           |
| <b>Chapter 3 Genetic Algorithm.....</b>           | <b>26</b> |
| 3.1 GA Operators.....                             | 27        |
| 3.1.1 Crossover.....                              | 27        |
| 3.1.2 Selection.....                              | 28        |
| 3.1.3 Mutation.....                               | 30        |
| 3.1.4 Inversion.....                              | 30        |
| 3.2 Why GA used.....                              | 31        |
| 3.3 Concluding Remarks.....                       | 31        |



|                  |  |           |
|------------------|--|-----------|
| <b>Chapter 4</b> | <b>Proposed Model.....</b>                                       | <b>32</b> |
| 4.1              | QoS Parameters.....  | 32        |
| 4.2              | Fitness Function.....  | 32        |
| 4.3              | The Model.....   | 34        |
| 4.3.1            | Genetic Algorithm for Computational Grid.....                    | 35        |
| 4.4              | Concluding Remarks.....  | 36        |
| <br>             |  |           |
| <b>Chapter 5</b> | <b>Experimental Evaluation.....</b>                              | <b>37</b> |
| 5.1              | Parallel Tasks Without Communication Cost.....                   | 38        |
| 5.1.1            | Observation with 50 Machines.....                                | 38        |
| 5.1.2            | Observation with 60 Machines.....                                | 43        |
| 5.2              | Parallel Tasks With Communication Cost.....                      | 47        |
| 5.2.1            | Observation with 50 Machines.....                                | 47        |
| 5.2.2            | Observation with 60 Machines.....                                | 47        |
| 5.2.3            | Observation with 50 Machines increase in Communication Cost..... | 54        |
| 5.2.4            | Observation with 50 Machines increase in Hamming distance.....   | 55        |
| 5.3              | Summary Result.....  | 56        |
| 5.4              | Comparison between GA and ACO .....                              | 58        |
| 5.3              | Concluding Remark.....   | 60        |
| <br>             |  |           |
| <b>Chapter 6</b> | <b>Conclusion and Future Scope.....</b>                          | <b>60</b> |
| 6.1              | Conclusion.....  | 60        |
| 6.2              | Future Scope.....  | 61        |

**References**

## List of Figures

|  |    |
|--|----|
| Figure 2.1 Architecture of a Multi-processor System.....       | 12 |
| Figure 2.2 Architecture of a Multi-computer System.....        | 12 |
| Figure 2.3 Architecture of a Distributed Computing System..... | 13 |
| Figure 2.4 Cluster Architecture.....                           | 14 |
| Figure 2.5 Grid Architecture.....                              | 15 |
| Figure 2.6 Grid Scheduling Architecture.....                   | 23 |
| Figure 2.7 Queuing Architecture of Grid.....                   | 24 |
| Figure 2.8 Workflow Management.....                            | 24 |
| Figure. 3.1 Roulette-wheel Selection Method.....               | 29 |
| Figure 5.1.1.1 Turnaround Time Observation with 50 Tasks.....  | 39 |
| Figure 5.1.1.2 Turnaround Time Observation with 60 Tasks.....  | 39 |
| Figure 5.1.1.3 Turnaround Time Observation with 70 Tasks.....  | 40 |
| Figure 5.1.1.4 Turnaround Time Observation with 80 Tasks.....  | 40 |
| Figure 5.1.1.5 Turnaround Time Observation with 90 Tasks.....  | 41 |
| Figure 5.1.1.6 Turnaround Time Observation with 100 Tasks..... | 41 |
| Figure 5.1.1.7 Turnaround Time Observation with 110 Tasks..... | 42 |
| Figure 5.1.1.8 Turnaround Time Observation with 120 Tasks..... | 42 |
| Figure 5.1.1.9 Turnaround Time Observation with 130 Tasks..... | 43 |
| Figure 5.1.2.1 Turnaround Time Observation with 50 Tasks.....  | 44 |
| Figure 5.1.2.2 Turnaround Time Observation with 60 Tasks.....  | 44 |
| Figure 5.1.2.3 Turnaround Time Observation with 70 Tasks.....  | 44 |
| Figure 5.1.2.4 Turnaround Time Observation with 80 Tasks.....  | 45 |
| Figure 5.1.2.5 Turnaround Time Observation with 90 Tasks.....  | 45 |
| Figure 5.2.1.1 Turnaround Time Observation with 50 Tasks.....  | 47 |
| Figure 5.2.1.2 Turnaround Time Observation with 60 Tasks.....  | 47 |
| Figure 5.2.1.3 Turnaround Time Observation with 70 Tasks.....  | 48 |
| Figure 5.2.1.4 Turnaround Time Observation with 80 Tasks.....  | 48 |
| Figure 5.2.1.5 Turnaround Time Observation with 90 Tasks.....  | 49 |
| Figure 5.2.1.6 Turnaround Time Observation with 100 Tasks..... | 49 |
| Figure 5.2.1.7 Turnaround Time Observation with 110 Tasks..... | 49 |
| Figure 5.2.1.7 Turnaround Time Observation with 120 Tasks..... | 50 |
| Figure 5.2.1.8 Turnaround Time Observation with 130 Tasks..... | 51 |
| Figure 5.2.2.1 Turnaround Time Observation with 50 Tasks.....  | 51 |
| Figure 5.2.2.2 Turnaround Time Observation with 60 Tasks.....  | 52 |
| Figure 5.2.2.3 Turnaround Time Observation with 70 Tasks.....  | 52 |
| Figure 5.2.2.4 Turnaround Time Observation with 80 Tasks.....  | 53 |
| Figure 5.2.2.5 Turnaround Time Observation with 90 Tasks.....  | 53 |
| Figure 5.2.3.1 Turnaround Time Observation with 100 Tasks..... | 54 |
| Figure 5.2.3.2 Turnaround Time Observation with 100 Tasks..... | 54 |
| Figure 5.2.4.1 Turnaround Time Observation with 100 Tasks..... | 55 |
| Figure 5.2.4.2 Turnaround Time Observation with 100 Tasks..... | 55 |
| Figure 5.4.1 Turnaround Time Observation with 12 Tasks.....    | 58 |
| Figure 5.4.2 Turnaround Time Observation with 18 Tasks.....    | 59 |

## List of Tables

|   |    |
|---|----|
| Table 5.1-Description of variables used in program..... | 37 |
| Table 5.2-Comparison Tasks without Communication .....  | 56 |
| Table 5.3-Comparison Tasks without Communication .....  | 56 |
| Table 5.4-Comparison Tasks with Communication .....     | 57 |
| Table 5.5-Comparison Tasks with Communication.....      | 57 |

# Abbreviations

| Abbreviations | : Meaning   |
|---------------|---|
| ACO           | : Ant Colony Optimization                             |
| ATA           | : Average Turnaround                                  |
| ATM           | : Automatic Tailor Machine                            |
| ATM           | : Asynchronous Transfer Mode                          |
| CPU           | : Central Processing Unit                             |
| CX            | : Cycle Crossover                                     |
| DB            | : Data Base   |
| DCS           | : Distributed Computing System                        |
| FCFS          | : First Come First Served                             |
| FIFO          | : First In First Out                                  |
| FT            | : Finishing Time                                      |
| FTP           | : File Transfer Protocol                              |
| GA            | : Genetic Algorithm                                   |
| IC            | : Insurance Company                                   |
| LHC           | : Large Hadron Collider                               |
| LIFO          | : Last In First Out                                   |
| LIGO          | : Laser Interferometer Gravitational Wave Observatory |
| MBPS          | : Million Bits per Second                             |
| MIPS          | : Million Instructions per Second                     |
| MPI           | : Message Passing Interface                           |
| M/M/1         | : Single Server Multiple Client                       |
| M/M/S         | : Multiple Server Multiple Client                     |
| MST           | : Minimum Spanning Tree                               |
| NUMA          | : Non Uniform Memory Access                           |
| OS            | : Operating System                                    |
| OSGA          | : Open Service Grid Architecture                      |
| PMX           | : Partially Mapped Crossover                          |
| QoS           | : Quality of Service                                  |
| RAM           | : Risk Assessment Model                               |
| RPC           | : Remote Procedure Call                               |
| SDSS          | : Sloan Digital Sky Survey                            |
| SJF           | : Shortest Job First                                  |
| SRTF          | : Shortest Remaining Time First                       |
| SSI           | : Single System Image                                 |
| TA            | : Turnaround Time                                     |
| TSP           | : Traveling Salesman Problem                          |
| UMA           | : Uniform Memory Access                               |
| VO            | : Virtual Organizations                               |

# Chapter 1

## Introduction

Parallel computing system [5, 6] is the one that aims to the minimization in the job/task execution time. Distributed systems [6] with multiple computing nodes also facilitate parallel execution apart from the other objectives as resource sharing and cooperative engineering. In the last decade the important evolutions in the design of distributed system have been accomplished by the internet revolution. Many a time it may not be possible to provide the best computing facilities e.g. processors, storage, data to a particular task or user as per their requirement. It may be because of various reasons. One reason may be affordability of the resources. Other may be availability. A possible solution for this may be to integrate the computing resources which are available at various places and enable authorized users to use it in a well-defined transparent manner. In *Grid* [1, 4, 5, 6, 8, 15, 16,17,18,29, 48, 49, 50, 51] the same concept has been adopted.

Scheduling over the grid is altogether different from scheduling [8, 12] of a uniprocessor system. In uniprocessor systems scheduling has the objective of maximum CPU utilization and thus tries to keep the CPU busy as much as possible. To achieve this concept of multiprogramming, multitasking and eventually multi-user have been developed. In Grid system, scheduling [9] exhibits cooperation from various resources towards problem solving by assigning various jobs/ sub-jobs demanding execution to the suitable grid resources. Normally the scheduler has the objective optimizing some characteristic parameter for the job or of the system while ensuring the proper load balancing of the grid. Job scheduling on a grid is known an NP-class problem [2, 3, 27]. Thus there is a possibility to apply soft computing techniques for solving this problem so that it is possible to optimize the characteristic parameter. In literature, many models have been proposed considering various optimization parameters for the scheduling problem. We, in our work, have considered the finishing time of a job to be optimized in the computational grid by adopting a GA based scheduling policy. We have designed a

fitness function for job completion time with two consideration; one in which we do not consider the network congestion [13, 14] and the other in which we do consider it. Quantification of the fitness function has been done both the ways. Contribution of the node in terms of its clock frequency, previous workload on the node in terms of the existing modules over the node, load on the link connecting the nodes and job's characteristics in terms of the cost of communication incurred due to the interactive modules have been accounted for in our quantification.

A scheduling model using simple Genetic Algorithm (GA) [2, 3] has been proposed. GA is a search based procedure which both explores and exploits the solution search space at the same time. It is based on the Darwin's theory of the "survival of the fittest" of the species. For the probabilistic optimization problems [31 35 45], GA is one of the best methods that offer a near-optimal solution.

We start by introducing basic concepts of Grid. Also, one of the soft computing techniques Genetic Algorithm has been introduced. A brief discussion over GAs and its applications has been done. Finally we have presented the organization of the dissertation.

Idea of the grid is derived from Electrical grid [49]. Computational Grid is analogous to a power grid that provides consistent, pervasive, dependable, transparent access to electricity irrespective of its source. Same is expected from a computational grid. The huge computational energy is expected to be available for the user. A user of the grid can request for any service of the grid. Grid middleware will look into the appropriate resources for the service; execute the services while meeting the expected Quality of Service (QoS). The whole process makes the availability of the computational power ubiquitous irrespective of the actual location of the resources similar to an electric grid which ensures power anywhere irrespective of location of the power generated.

A grid is classified based on its usage. For example, a data grid is expected to manage a huge amount of data (data warehouse) and to process the data (data mining). Computational grid is expected to serve the computational need of the application well and thus emphasize on the compute capability of its constituent nodes. Other types of Grid are bio grid, campus grid, collaboration grid, network grid, utility grid etc. As the name implies a bio grid specializes in effectively handling the biological applications, a collaboration grid may involve a number of people working on different parts of the same project without even disclosing their identities; network grid provides efficient communication links to the users with each node acting as a router between two communication points etc. A utility grid tries to incorporate both the features of the computational grid and the data grid by offering the computational and the data management services.

We, in our work, considered computational grid and the scheduling aspect of this.

## **1.1 Computational Grid**

A computational grid [5, 6, 8, 48, 49, 50, 51] deals, primarily, with the computational requirements of the job. It is defined as a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high end computational capabilities despite the geographical distribution of both resources and the users. The submitted a job for the execution over the grid, the grid middleware searches for the appropriate computational resources for the job execution from the pool of computational resources available with the grid. Depending on the execution policies and the job requirements, the job will eventually be scheduled on one of the suitable clusters which execute it and furnish the result. Grid users visualize the computational grid as an enormous source of computing Energy on which any job can be executed efficiently.

Computational grid couple geographically distributed resources such as PCs, workstations, clusters, and scientific instruments. These have appeared as a next generation computing platform for solving large scale complex problems in science,

engineering and commerce. However, application development, resource management, and scheduling in these environments continue to be a complex research intensive job.

## QoS Parameters in Computational Grid

Grid is used for dynamic sharing of resources for Virtual Organization (VO). Ian Foster Grid [18] check list which defines that a distributed architecture is a grid if it satisfies the following properties:

- Resources which are used for coordination are not to be in centralized control
- It uses standard and general purpose open protocols.
- It gives non-trivial Quality of Service.

The quality and performance of any system is evaluated using some parameters known as Quality of Service (QoS) [1, 7, 16, 17, 15, 20, 21, 26] parameters. Some QoS Parameters that are often useful in computational grid are Reliability, Make-Span, Cost, Security, Turnaround Time and Speed Up.

Turnaround time [12] for a job is estimated as the time taken by the job from its submission to the final execution. The prime objective of switching over from a uni-processor system to parallel/distributed system is to reduce the turnaround time for the job execution. Thus, it is always expected from a computational grid scheduler to allocate the job to those grid resources which results in the faster overall execution of the job i.e. with minimum turnaround time. Turnaround time [1, 7, 16, 17, 15, 20, 21, 26] for a job depends on various factors some of which are as follows.

- *Node speed:* The speed of the computing processor on which the job is to execute. Faster job execution is expected on faster machine.
- *Number of processors in the node:* A node may consist of multiple processors. More the number of processors in a node better will be its performance.



- *Existing workload:* This refers to the number of modules (jobs) already allocated on the node. More the pre-assigned workload, the higher will be the turnaround of the current job.
- *Local scheduling policy [6, 8, 12, 33] of the node:* Grid scheduler schedules the job on the appropriate nodes; however there is a local scheduler to all the nodes that affects the local scheduling decision. Grid scheduler does not alter the local scheduling policy of the node. It could be round robin, shortest job first or any such scheduling policy independently governing to the node in order to provide the required autonomy.
- *Degree of interaction in the job:* This refers to the amount of interaction, required by the job. If the job is interactive, the job modules require interaction for the data exchange. This affects an increase in the turnaround time.
- *Load on the network link:* As there are many jobs for execution and they may opt for making communication [13, 14] for interactive modules, load on the network link may vary from time to time. This eventually may affect the performance resulting in invariable turnaround time.

Apart from all the issues relating to maintaining the QoS and secured communication [13, 14, 24] it is always desired to have a reliable system that is able to digest system failures. Significantly, incorrect performance of the computers may lead to several devastating effects. A fault tolerant system [6] is one, which continues to perform even in the presence of hardware and software faults. A fault is a physical defect, imperfection, or flaw that occurs within some hardware or software component whereas an error is the manifestation of a fault and is a deviation from accuracy or incorrectness. Specifically, faults are the cause of errors and errors are the cause of failures.

Reliability [30, 31, 32, 34] is the ability of a system to perform and maintain its functions in routine circumstances, as well as hostile or unexpected circumstances. Thus, more the fault tolerance of the system more reliable it is. Reliability adds quality to the system so is an often desired parameter. Whenever a grid is designed, the hardware components used are specified with the failure rate so are the software components

during the software design. These failure rates constitute the reliability of the system, which is often desired to be high.

The dynamic nature of grid environments need secure communication. Therefore it requires challenging security issues. [12, 13, 14, 20, 21, 22, 23] Key points for secure communication are the following:

- a) Confidentiality: Confidentiality assures that data or message transmitted is handling by only authorized users. For implementing we use encryption and description algorithms.
- b) Integrity: Integrity is the intrinsic assurance that information is not tempered.
- c) Authentication: Authentication assures that principal (user or resource) is really operated. To implement this we ask username/password etc.
- d) Authorization: A remote shared resources are protected by only allowing authorized access.
- e) Non Duplication: It assures that action performed by non duplicated can not denied.

The turnaround time and the speed up are the most fundamental parameters. We have dealt with these fundamental parameters in our present work.

## **1.2 Genetic Algorithm**

GAs [2, 3, 25, 27, 28, 29, 45] is often used to solve the problem whose solution involves identifying the solution from a big search space. The problem being addressed in this work is the optimization of the QoS parameter in Computational grid that also involves a big search space. We are applying GAs to handle this problem.

GA works on the basis of natural selection and evolution. The GA operators are applied over a randomly generated population which comprised of a set of chromosomes (solutions), in each generation. These chromosomes are evaluated against a fitness function derived on the basis of the optimization objective. The chromosomes that offer

best fitnesses are selected for mating to reproduce offspring. This procedure is iterated over the generations resulting in good parents to reproduce better offspring. The process stops when the result converges. GA uses various operators to implement this operation which are as follows.

### ***Selection***

Selection operator [2, 3] selects those chromosomes which satisfy the requirements against the fitness function. It ensures that the chances of survival of the fittest individuals are more than the weaker ones. Since selection process decides the chromosomes selection for mating, it plays a vital role in exhibiting the performance of the GA. A good selection leads to the faster convergence of the results. A number of selection methods available are Roulette wheel selection, Tournament selection, Sigma selection, Rank based selection, Random selection, etc.

### ***Crossover***

Crossover [2, 3] is the operator responsible for mating the chromosomes. Given the chromosomes for mating, crossover site is selected and the strings from the selected site are exchanged. There are various types of crossover available such as single point crossover, multi-point crossover, uniform crossover etc. The characteristics of one parent are carried over to the offspring till the crossover point with characteristics from the other parent following the crossover point. This results in a generation of new offspring, inheriting characteristics of both the parents. This operator helps mating the parents with better fitness in the population leading to evolved children for the next generation. Some popular crossover operators in use are Arithmetic, Heuristic, Intermediate, Scattered, OR, PMX, CX etc.

### ***Mutation***

It prevents the population of chromosomes, generated over generations, from being too similar to each other. Thus it prevents the solution to be caught in local optima. In spite of climbing several peaks simultaneously by observing various chromosomes, the possibility of attaining false peak cannot be denied. Mutation [2, 3] works by randomly altering the gene pattern of the chromosomes thus helping in coming out of the local optima. The probability of mutation is often very low. The result of mutation can result in

either a weaker individual or a stronger one but it can be surely established that it subtly changes genes of the chromosomes thereby coming out of the local optima.

The pseudo-code of the simple GA is given below.

```
GA ()
{
  Generate initial population
  For each generation, do
  {
    Perform Selection
    Perform Crossover
    Perform Mutation
    Evaluate the fitness of the population
  } Until the stopping condition
}
```

### 1.3 GA as a Soft Computing Tool

To express the GA for the types of problems, we have explained the TSP problem here.

The origins of the TSP [2, 3, 25, 27, 28, 29] are not known. In literature of TSP was available from 1832. The example tours through Germany and Switzerland, but have no mathematical solution. Mathematical problems related to the TSP were given in the 1800s by the Irish mathematician W.R. Hamilton and British mathematician Thomas Kirkman. Hamilton's Icosian Game was a puzzle based on finding a Hamiltonian cycle [2, 3, 25, 28]. The general form of the TSP appears to have been first studied by mathematicians during the 1930s in Vienna and at Harvard. Karl Menger defines the problem. He tried to solve this problem using brute force method. He observes the non optimality of the nearest neighbor heuristic.

The TSP [2, 3, 25] can be expressed in many different ways. One way is that a salesman plans a trip that he takes to certain cities for the customers and then come back to the city in which he started. Can he plan the trip so that he begins and ends in the same city while visiting every other city only once and incur lowest cost? Another way a bank has many

ATM machines. Every day, a bank official goes from machine to machine gather computer information and service the machines. In what order should the machines be visited so that shortest possible route is obtained?

In TSP [2, 3, 25, 27, 28, 29, 45] the goal is to find the least cost route of 'n' different cities. All the cities are to be traversed exactly once and have to return to the start city. In doing so the cost of tour is the minimum. Testing every possibilities of tour of n cities mathematically is as total  $n!$  routes are possible. Thus the problem will be infeasible for large n. If for example 30 cities routes would be  $30! = 2.65 \times 10^{32}$  different tours, which is almost an impossible solution. To explore it, a method is applied that may not guarantee the best solution but a near-optimal solution in lesser time. We call it soft computing as the best result is not guaranteed.

Genetic Algorithm is one approach from the various soft-computing approaches which can be used to find a solution in less time although it might not be the best solution. First create a group of many random solutions called population. For this a greedy algorithm [57] may be used that gives preference looking for cities that are closed to each other.

Now pick two better (least cost) parents in the population and apply crossover to make two new child routes. A small percentage of time a child is mutated also. This is done to prevent all routes in the population looking identical. The new routes are inserted into the population replacing two longer routes. The size of the population remains same. New children routes are repeatedly created until the desired goal is reached.

The two complex issues using GAs to solve the TSP. One of them is encoding of tour another crossover algorithm. The crossover uses to combine two parent routes to make the child routes. For example, in 8 city problem where each city is visited in ascending order, the tour can be as follows: [1 2 3 4 5 6 7 8]. If cities are visited in descending order then the tour is represented as the following permutation [8 7 6 5 4 3 2 1].

## 1.4 Organization of Dissertation

The dissertation consists of five chapters which have been arranged as follows.

Chapter one discuss about Grid Computing in general. It also discusses a brief about GA and its Applications. Various QoS Parameters expected from a computational grid has also been explored in this chapter. It has been stated for what type of problems a Soft Computing can be used.

Chapter two discuss about Grid Computing in general. We started with Distributed Computing System and discuss Multi processor, Multi computer and Compute cluster. We elaborated about grid in general along-with the discussion about computational grid. Finally, we discuss about types of grid, issues in computation grid and concluding remarks.

Chapter three discuss about GA in general. This chapter discusses what is GA, how it solve the problems, what are GA operators, along-with its Applications. A discussion over the selection methods emphasizing one of the selection methods is done in this chapter.

Chapter four carries our proposed model .This chapter discusses various QoS Parameters, the one used in our model, quantification of the fitness function etc. An algorithm for the model is also presented before making the concluding remarks.

Chapter five contains the experimental evaluation. Various parameters affecting the objective have been tested through the experiments. Observation has also been drawn.

Conclusion of the work and Future possibility is given in chapter six.

### Computational Grid

Computational Grid is analogous to a power grid that provides consistent, pervasive, dependable, transparent access to electricity irrespective of its source. For any service requested by the user of the grid, grid middleware searches for the appropriate resources to serve the service, execute the services meeting the expected Quality of Service (QoS) and satisfying other objectives of the application. The whole process makes the availability of the computational power ubiquitous irrespective of the actual location of the resources just like an electric grid which ensures power anywhere irrespective of location of the power generated.

In this chapter, we have discussed about Multiprocessor System, Distributed Computing System, Cluster and Grid. A discussion about grid is in general before emphasizing about computational grid. We also have deliberated about types of grid, issues in computational grid before making concluding remarks.

#### 2.1 Multiprocessor, Distributed System and Cluster

**Multiprocessor system** [5, 6, 8, 12, 44] is essentially the tightly coupled systems that have more than one processor on its motherboard. It shares the memory and the clock. The architecture can be symmetric or asymmetric allowing all, some or only one CPU to execute. If the operating system is built to take advantage of multiprocessor system, it can run different processes (or different threads belonging to the same process) on different processors. . The architecture of the Multiprocessor System is shown in Figure 2.1.

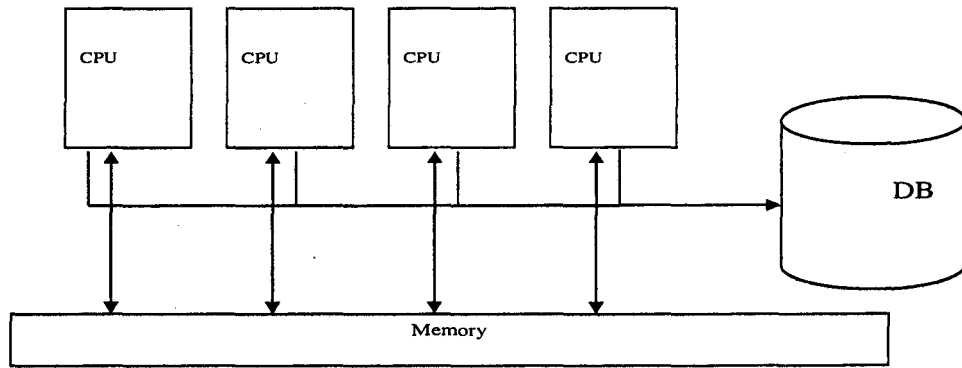


Figure 2.1 Architecture of a Multiprocessor System

**Multicomputer systems** [5, 6, 8, 12, 44] are a system made up of several independent computers interconnected by intercommunication network. Multicomputer systems can be homogeneous or heterogeneous: A homogeneous distributed system is one where all CPUs are similar and are connected by a single type of network. They are often used for parallel computing. A heterogeneous distributed system is made up of different kinds of computers, possibly with vastly differing memory sizes, processing power and even basic underlying architecture. They are in widespread use today, with many companies adopting this architecture due to the speed with which hardware goes obsolete and the cost of upgrading a whole system simultaneously. The architecture of the Multicomputer System is shown in Figure 2.2.

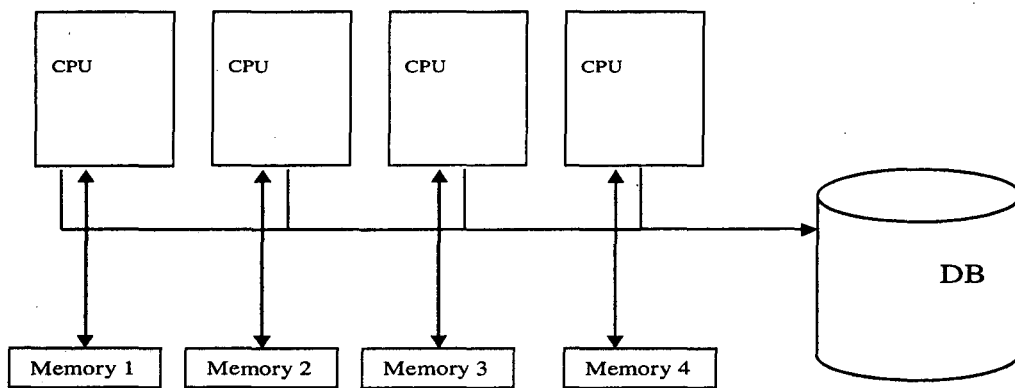


Figure 2.2 Architecture of a Multicomputer System

Distributed computing [6] is a method of computer processing in which different parts of a program run simultaneously on two or more computers that are connected with each



other making communication through the interconnection network. In distributed system multiple CPU's do not share memory and clock and work independently making communication as and when required. The memory here is distributed following the Non Uniform Memory Access (NUMA) model of memory classification. Each processor (CPU) is an autonomous computer with its own OS and communicates using Message Passing Interface (MPI) or Remote Procedure Call (RPC). The job can be submitted at any node and is distributed to various nodes as per their capabilities under a load distribution policy. These are used when strong computational power is required along with a certain degree of independence to the computational resources. The architecture of the Distributed Computing System (DCS) is shown in Figure 2.3.

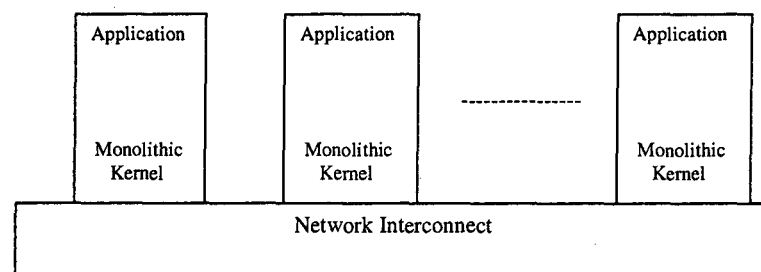


Figure 2.3 Architecture of a Distributed Computing System

Compute clusters [5, 6, 8, 12, 33, 44] are comprised of multiple stand-alone machines acting in parallel across a local high speed network. Distributed computing differs from cluster computing in that computers in a distributed computing environment are typically not exclusively running "group" tasks, whereas clustered computers are usually much more tightly coupled. Distributed computing also often consists of machines which are widely separated geographically.

In Cluster, multiple stand alone machines work in parallel connected by a high speed local area network with a Single System Image (SSI). Architecture of the Cluster Computing Network is shown in Figure 2.3.

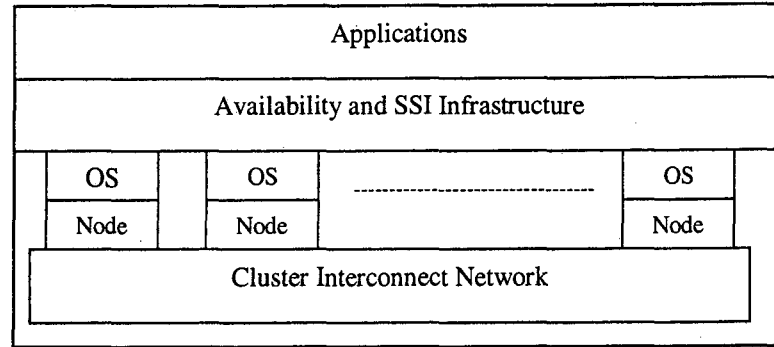


Figure 2.4 Cluster Architecture

## 2.2 Grid Computing

Grid computing has roots in parallel and distributed computing, dating back to the 80s and 90s. At the time, supercomputers dealt with problems including Grand Challenges (such as climate modeling) and computing intensive simulations in weather forecasting.

The origin of the idea about grid computing was developed by scientific community. The initial target was the processing power and storage intensive applications. The basic idea in grid computing [19] is to support resource sharing among the individuals and research organizations. It helps to utilize resources so as to increase the computational speed-up and reliability.

The term Grid computing, or simply Grid, was introduced in 1998. The driving force was the need to carry out compute intensive tasks. In response, the Grid was developed to provide computing power on demand to every application which needed it. Originally, Grid computing targeted supercomputing application users. In addition, the applications can be even more computationally intensive than conventional supercomputing applications i.e. if a single supercomputer or cluster can not manage the application workload, several supercomputers [5, 6] or clusters connected via a wide-area network can provide more CPU capacity.

Main aim of the supercomputers era was to deal with the large tasks that typically appear in computational science, engineering, and so on. A characteristic of such

supercomputers was their homogeneous, typically expensive hardware. Several centers worldwide provide computing power by means of supercomputers. Interconnectivity among processors within supercomputers was usually good, even within a computer center.

However, wide-area network connectivity among computer centers was often not very good. Even a homogeneous programming and security model [13, 14, 24] to connect these computing resources was not in place. Technologies such as MPI (Message Passing Interface) [5, 6] were mainly designed to run on a single machine where CPUs and networks have similar hardware and performance characteristics. Above all, super computing technology is too expensive.

Architecture of grid is multilayered in nature. There are five layers in the grid architecture [54] which is shown in the figure 2.5 which is given below. They have similarity with the internet protocol architecture.

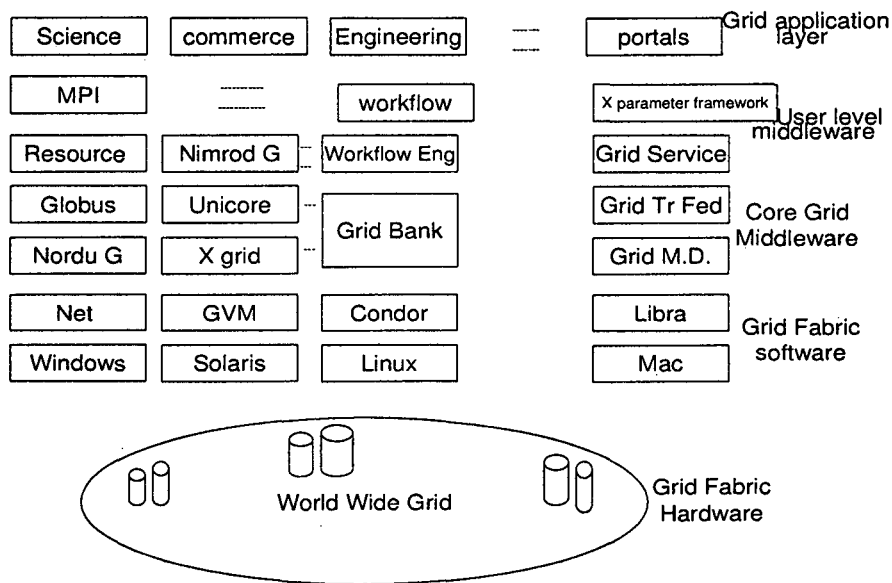


Figure 2.5 Grid Architecture

## **2.3 Types of Grid**

We are considering only grid which are used in Computer Science. The two important types of grid are the following.

### **2.3.1 Computational Grid**

Computational Grid is used for sharing computation (CPUs) which have high performance computability. The example of computational grid is Tera Grid which has more than 750 teraflops. This aggregates computing power of millions of home computers. The utility computing ventures such as Sun Grid computing Utility are also an example of computational grid.

A computational grid deals primarily with the computational requirements of the job. It is defined as a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high end computational capabilities despite the geographical distribution of both resources and the users. At one end of the grid are the end users or the participants of the grid. These users may demand the execution of a job using an intelligent interface. The grid middleware searches for the appropriate resources for the job from the pool of resources registered in the grid. Depending on the execution policies and the job requirements the job will eventually be scheduled on one of the suitable clusters which execute it and furnish the result back to the user. Thus the grid users visualize the grid as an enormous source of computational power in which any job can be executed efficiently on the suitable resources of the grid.

### **2.3.2 Data Grid**

The grid environment has enabled us to produce more efficient computational tools which in turn have resulted in more advanced research. These two factors have produced software tools and applications that can handle complex computations. Many a times these applications like high energy physics involving Large Hadron Collider (LHC) at CERN, The Laser Interferometer Gravitational Wave Observatory (LIGO), Sloan Digital Sky Survey (SDSS) requires generating and manipulating terabytes and petabytes of data.

Thus the amount of data involved the scale of the demand and the complexity of the infrastructure imposes enormous challenge for the effectiveness of the current methods and tools. The integrating architecture that addresses all these problems is the data grid. Data grid provides the excellent performance on petascale data intensive computing by the integration of various approaches adopted while encouraging the deployment of basic enabling techniques and revealing technology gaps that require further research and development. Thus data grid deals with data, data abstraction, data access, replica management, controlled sharing and management of large amounts of distributed data.

## **2.4 Issues in Computational Grid**

The grid is essentially a heterogeneous collection of computational and other resource. This heterogeneity introduces many related challenges. The challenges include diversity in terms of local resources, dynamic nature of the local resources, creation and management of services and maintaining the Quality of Service (QoS). Since grid is inherently a parallel and distributed system, the key issues regarding the design of the grid e.g. data locality and availability, implementation, scalability, anatomy, privacy, maintenance, fault tolerance, security, etc. need to be addressed well before the commercialization of the grid. Some of these issues demanding new technical approaches for the grid environment are discussed as follows.

### **Role of the End System**

End system plays a major role in the grid system as today's end systems are relatively small and they are connected to networks by interfaces and operating system mechanisms that are originally designed for reading and writing slow devices. Thus, these end systems needs to be developed to support high performance networking of grid architecture.

### **Job Preprocessing Requirements**

Grid involves number of virtual organizations (VO). Any query to the grid enters through the corresponding virtual organization only. Though, it is difficult to have common grid architecture as they are created to cater to different needs, at least a basic set of services e.g. Querying, Submitting and Monitoring need to be identified. Any process on a grid may proceed by

- *Obtaining the necessary authentication credentials (connectivity layer protocols):* The user should be authenticated for entering the grid. This can be done by the use of password or certificates. For this, a password may be given to the authorized users keeping in mind the security of the user keys.
- *Querying information (collective services):* Since a grid is a dynamic system it should be updated to keep track of the changes occurring inside it. Most of the information is dynamic and should be updated from time to time, whenever a job completes its execution or any resource participates or quits the grid. Mechanism of discovering resources on the grid is itself a challenging task for the designer. In the old schemes some centralized services, e.g. Condor matchmaker were used that contained all the information, but it had the problem of scalability and single point failure. Later decentralized schemes were introduced e.g., MDS – 2 where the grid information is stored and indexed by index servers that communicate via registered protocols. The best approach is to have the information available to all virtual organizations about the resources status so that load balancing can be achieved.
- *Submitting requests (resource protocols):* Keeping in mind the heterogeneity of the grid the request should be submitted for a appropriate machine, storage systems, and networks to initiate computations, move data and so on. High selectivity in the resources has an advantage of the best possible allocation but may lead to lower match probability. High regionalism may lead to non uniform distribution of the tasks but will reduce the communication cost. Selectivity depends on the nature of the job whereas regionalism is governed by how much communication overhead and network delays the job can tolerate. Both these factors depend on the existing load structure of the grid. In addition, access control should also be exercised as the users are working in the shared environment and there can be a number of resources over which the owner wants to have its complete authority.
- *Monitoring resources and computations (resource protocols):* The progress of the various computations and data transfers could be done by using means such as check pointing. It notifies the user when all the tasks are completed and detecting

and responding to failure conditions. This brings the security and fault tolerance features of the grid. Since large numbers of virtual organizations are part of the grid, large numbers of resources need to be monitored and tracked for various failures. These failures may range from submission failure to hardware/software failures. The grid should be able to meet out these failures gracefully.

The important issues [4, 5, 6, 7, 8, 9, 10] in Computational Grid are Resource Management, Scheduling, Latency, Throughput, Reliability, Availability and Security. Few of them are discussed below in brief.

**Resource Management** [12] is a process in which we manage the resources efficiently. Generally for the resource management we do the process management, memory management, file management, IO system management and secondary-storage management. Few of them as given in brief as follows.

**Scheduling** is a way in which processes assigned to run on suitable systems in such a way the scheduling parameters are optimized. The scheduling parameters are throughput, system utilization, and turnaround time, waiting time, response time and fairness. The scheduling parameters are discuss below

- ❖ **Throughput** [12] is measured as number of tasks executed per unit time. There are many possible throughput metrics depending on the definition of unit of work. Examples of throughput metrics at the resource layer include the effective transfer rate in Kbytes/sec under the Grid-FTP protocol used to transfer files from different computes nodes involved in running RAMs and the number of queries/sec that can be processed by the database server of a law enforcement agency needed by the RAM application.
- ❖ **System Utilization** [12] is to keep system as busy as possible.
- ❖ **Turnaround time** [12] is amount of time to execute particular process. In other words turnaround time for a job is estimated as the time taken by the job from its submission to the final execution. The prime objective of switching over from a uni-processor system to parallel/distributed system

is to reduce the turnaround time for the job execution. Thus, it is always expected from a computational grid scheduler to allocate the job to those grid resources which results in the faster overall execution of the job i.e. with minimum turnaround time.

- ❖ *Waiting time* [12] is amount of time spend to wait by a particular process in system for getting a resource. In other words waiting time for a job is estimated as the time taken by the job from its submission to the get system for execution. The waiting time depend on the parameters similar as turnaround time.
- ❖ *Response time* [12] is amount of time to get first response in time sharing system. The response time depend on the parameters similar as turnaround time.
- ❖ *Fairness* [10] of system is defined as the time taken by each system in grid environment is same.

Turnaround time for a job depends on various factors some of which are as follows.

- *Node speed*: The speed of the computing processor on which the job is to execute. Faster job execution is expected on faster machine.
- *Number of processors in the node*: A node may consist of multiple processors. More the number of processors in a node better will be its performance.
- *Existing workload*: This refers to the number of modules (jobs) already allocated on the node. More the pre-assigned workload, the higher will be the turnaround of the current job.
- *Local scheduling policy of the node*: Grid scheduler schedules the job on the appropriate nodes; however there is a local scheduler to all the nodes that affects the local scheduling decision. Grid scheduler does not alter the local scheduling policy of the node. It could be round robin, shortest job first or any such scheduling policy independently governing to the node in order to provide the required autonomy.



- *Degree of interaction in the job:* This refers to the amount of interaction, required by the job. If the job is interactive, the job modules require interaction for the data exchange. This affects an increase in the turnaround time.
- *Load on the network link:* As there are many jobs for execution and they may opt for making communication for interactive modules, load on the network link may vary from time to time. This eventually may affect the performance resulting in invariable turnaround time [4, 5, 6, 7, 8, 9, 10, 48, 49, 50, 51].

**Latency** [7] is related to the time it takes to execute a task and is measured in time units. Latency metrics can be defined at different granularities. For example, in the IC case, one may be interested in the average time it takes to complete immediate quote requests or in the 95-th percentile of the time to respond to non-immediate requests. Another example of latency metric is the elapsed time to return a delayed quote to the user. Sophisticated RAMs are typically complex parallel jobs that are decomposed into tasks allocated to different computing resources in the grid.

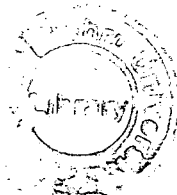
**Availability** [7] is defined as fraction of time that a resource/application is available for use. In a multilayer context such as the one described for grid architecture, the notions of availability differs for each layer.

**Security** [11] shows the dynamic nature of grid environments need secure communication. Therefore it requires challenging security issues. Key points for secure communication are the confidently, authentication, replication, integrity, and authorization.

**Reliability** [1, 7, 16, 17, 26] is the probability of system works correctly for a given time period. In reliability we check how system is working reliably. In other words Reliability is the ability of a system to perform and maintain its functions in routine circumstances, as well as hostile or unexpected circumstances.

TH-17481

004.36  
S9416  
AA



*Cost/Budget/Deadline* [1, 7] is the user preferred quality of service parameter. In Grid environment applications are often described as workflows. A workflow is composed of atomic tasks that are processed in specific order to fulfill a complicated goal. The development of the OSGA however makes the workflow management more intractable (OSGA introduces Web services into the grid interoperability model and soon become the predominant technology for grids). In the new architecture of OSGA, a task can be executed by any one of a set of service instances provided by different grid service providers. One of the most challenging problems is to map each task to corresponding service instance to achieve the customer quality of service requirement as well as to accomplish high performance of the work flow. This problem is found to be NP-Complete. Under the OSGA, the workflow scheduler has to balance several QoS parameter such as make-span, cost, reliability, security and deadline at different prospective.

## **2.5 The problem**

Scheduling [9, 10, 12] is a fundamental issue for getting high performance in computational grid. In computation grid there are large numbers of resources so that main objective is optimizing global use of resources. The problem of local use resources is already almost solved. We consider here two types of scheduler first is local scheduler and second is global scheduler. Local scheduler takes care about the scheduling inside each element of grid. In my problem we assume that local scheduler works in FCFS [12] manner.

Here we consider few parameters of scheduling which are the turnaround time and speed up. We have already described the parameter turnaround time in 2.4.

Initially global scheduler randomly distributes the task to nodes in grid environment. But generally distribution of accordingly the processing speeds of each machine in grid. We randomly generate the population i.e. initial population. We calculate the fitness of the population using fitness function. Sort the population using sorting algorithm and select

the population using roulette-wheel selection. In this way half the population is selected. After that crossover is performed 95% of time and mutation 5% of time of total number of generations. In this way new population is generated which includes the old population also. Now arrange the new population according to its fitness value and make it as old population. Repeat this process until stopping criteria meets.

The scheduling problem in which average arrival rate and average service rate of the systems are  $\lambda$  and  $\mu$  MIPS respectively. The grid scheduling architecture is given in figure 2.6 in which we allocate the grid resources in such way that the scheduling and resource parameters should be optimized. The figure 2.6 is shown below [7]

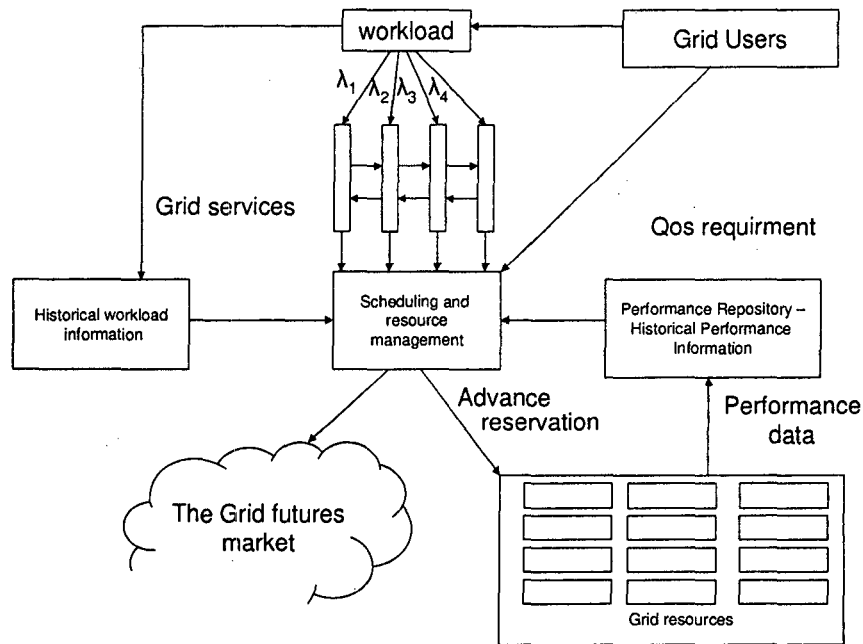


Figure 2.6 Grid Scheduling Architecture

For understanding scheduling in grid we should apply Queuing Theory [30, 32, 34, 35]. Scheduling problem falls in category of M/M/S Model. It is because we assume that there are many systems/nodes/machines and many jobs/sub-jobs to execute. Every system has its own capability to process the job.

Assume that average arrival rate and average service rate of the system are  $\lambda$  and  $\mu$  MIPS respectively. The queuing model [36] based on principle of real world queue which is based on FCFS [12]. The queuing grid architecture is given in figure 2.7. In this figure various tasks are distributed in various resources among the grid. The figure 2.7 [7] is shown below.

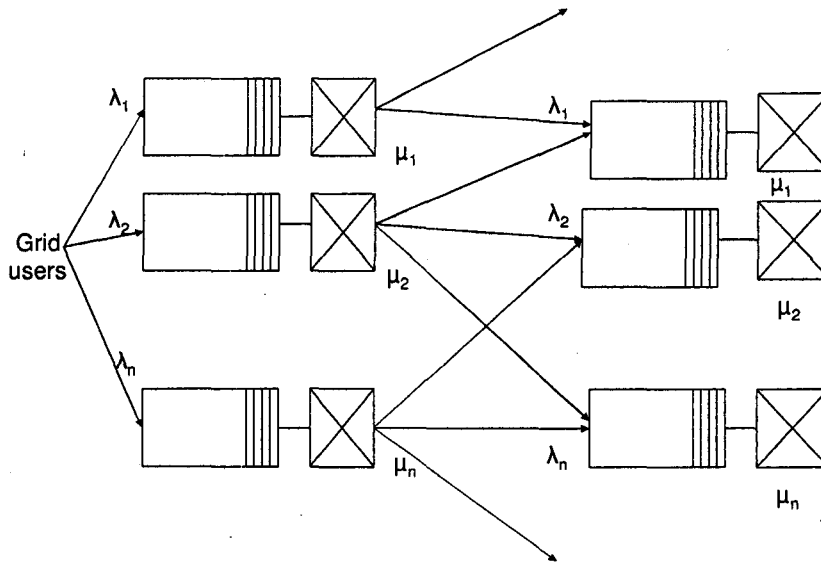


Figure 2.7 Queuing Architecture of Grid

The workflow of tasks is to be linear or hybrid or parallel. If workflow is linear then it executed by sequential otherwise we can apply parallel execution model. In my problem the workflow is parallel. The workflow management is shown in figure 2.8 [7].

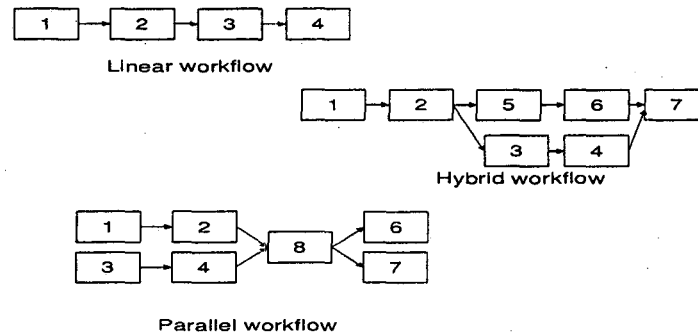


Figure 2.8 Workflow Management

## 2.6 Concluding Remarks

Here we have discussed about Multiprocessor System, Distributed Computing System, Cluster and Grid. A discussion about grid is in general before emphasizing about computational grid. Here we find that there are two types of grid. The most important grid is Computational Grid in which we identify the issues. The most important and challenging issues are Resource Management, Scheduling, Latency, Throughput, Reliability and Security. We discuss the scheduling in detail. We also have deliberated about types of grid, issues in computational grid before making concluding remarks.

## Chapter 3

### Genetic Algorithm

Scheduling has been identified to be an NP-class of problem as stated in chapter 1. Evolutionary techniques are best suited for such problems. We have applied Genetic Algorithm to solve scheduling problem of computational grid. This chapter makes discussion about the Genetic Algorithm.

Few questions that have been dealt with in this chapter are as follows. What is GA? How it solves the problems? What are the GA Operators? Why it is used for solving hard computing problems?

Genetic Algorithms are evolutionary algorithm derived from the Darwin's theory of natural genetics. It is based on the principle of "Survival of the Fittest". A best solution is derived out of number of solutions in GA.

Genetic algorithm generates population of potential solution and explores the best solution of the problem. The father of the GA is John Holland. He, along with his student De Jong, introduced the concept of GA in the year 1975 at the University of Michigan. He was influenced by the natural system. GA is derived from natural system. He thought to involve computing in natural system and derived it. He was of the opinion that as the natural system evolves, any computation may also evolve. This was the basis for the development of the GA.

Most of the terminology in GA [2, 3, 4] has been borrowed from genetic engineering. GA uses the biological concept of "Natural Selection" and "Genetic Inheritance". For most of the optimization problems, where a little knowledge about problem solving approach is available, GA is beneficial. It is widely used for NP-class of problems. The examples are

TSP, Vertex Cover, Coloring, MST Problem and Scheduling Problems for which GA has been used widely.

### 3.1 GA Operators

GA comprised of many operators that forms the basis for evolution. The following are some operators [2, 3, 25, 29, 45] used in GA.

#### 3.1.1 Crossover Operator

Crossover is a concept of genetics and has analogy of sexual reproduction. Crossover chooses two individuals and swap segments of their code (bit-value). It produces children that are combinations of their parents.

There are many types of crossover [2, 3, 4] discussed in literature.

1. In a **single-point crossover** one common point in the chromosomes is chosen randomly. Chromosomes of the parents are cut at that point and the resulting sub-chromosomes are swapped.

Parent1:      00000000|000000000000000000000000

Parent2:      11111111|111111111111111111111111

| is the crossover point chosen.

After the crossover the children produced are as follows.

Child1:      00000000111111111111111111111111

Child2:      11111111000000000000000000000000

2. In **Multipoint Crossover** more than one crossover point is chosen. For example in **two-point crossover** two sites are selected and the sub-string between the bits is swapped.

Parent1:      00000 | 000 | 000000 | 0000

Parent2:      11111 | 111 | 111111 | 1111

After the crossover the following children are produced.

Child1:      00000 | 111 | 000000 | 1111

Child2:        11111 | 000 | 111111 | 0000

3. In **Uniform crossover** [53] each gene of the offspring is selected randomly from the corresponding genes of the parents. Single-point and two-point crossover produce two children, whereas uniform crossover produces only one because bits in string are compared with the two parents. The bits are swapped with fixed probability (typically 0.5). The example for uniform crossover is given below.

|           |                      |          |                    |
|-----------|----------------------|----------|--------------------|
| Parent 1: | 00000000000000000000 | Child 1: | 010010110001011001 |
| Parent 2: | 11111111111111111111 | Child 2: | 101100001110100110 |

### 3.1.2 Selection Operator

Selection [2] is a procedure of picking best individuals from a pool of population based on their fitness value to reproduce offspring. It is an important step in GA. Many different approaches for selection schemes have been proposed. Some of the approaches are Roulette-wheel, Sigma, Scaling, Elitism, Boltzmann Selection, Rank Selection, Tournament Selection and Steady State Selection.

One of the popular selection methods in Fitness-proportionate selection scheme is roulette-wheel method often used in problem solving using GA. Holland's original GA uses fitness-proportionate selection, in which the "expected value" of an individual that is the expected number of times an individual occurs, will be selected to reproduce. The expected value is the individual's fitness divided by the average fitness of the population. To implement this method a slice, circular in the size of circle being proportion to the individual's fitness is assigned. The wheel is circulated N times, where N is the number of individual in the population. In each circular motion the individual under wheel makers is selected to be the pool of parents for the next generation. The algorithm is as follows.



The pseudo-code [2] of the Roulette-wheel Selection Method is given below.

*Roulette-wheel Selection Algorithm ( )*

```
{  
    Sum the total expected value of the individuals in the population and this sum is S.  
    for i= 1 to N  
        {  
            Choose a random integer r between 0 and S.  
            Repeat through the individuals in the population, summing the  
            expected values until the sum is greater than or equal to r. The  
            individual whose expected value puts the sum over this limit is the  
            one selected.  
        }  
    }  
}
```

For example roulette-wheel [2, 3] method, the chance of an individual's being selected is proportional to the amount by which its fitness is greater or less than its competitors' fitness. Each individual gets a slice of the wheel. More fit ones get larger slices than less fit ones. The wheel is then spun and an individual "owns" the section on which it lands. Suppose that there are four populations A, B, C and D having their fitness value 4, 6, 2 and 8 respectively. The roulette wheel for selection of these populations is as given in Fig. 3.1. Obviously D has more probability to be chosen than A, B and C as this has wider allocation in wheel.

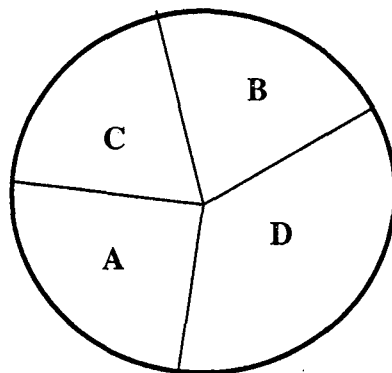


Figure 3.1 Roulette Wheel Selection Method

### 3.1.3 Mutation Operator

This operator randomly flips some of the gene in a chromosome. For example, the string 00000100 might be mutated in its second position to yield 01000100. Mutation [2, 3] can occur at each bit position in a string with some probability, usually very small (e.g., 0.001).

### 3.1.4 Inversion Operator

Inversion operator [2] was introduced in 1975 by Holland. It is similar to real genetic operator. This is inverting part of chromosome. Used for single point crossover it deals with the linkage problem of fixed length strings.

The detailed pseudo-code [2] for GA is as follows.

*Genetic Algorithm ( )*

```
{  
    Generate the population of solutions randomly. //using any approach  
    Evaluate the fitness value of each individual against the fitness function.  
    Repeat while termination condition does not hold  
    {  
        Randomly pick two parents from the population  
        Perform the crossover over the parents to produce offspring  
        Mutate any offspring if required // the probability is often very less  
        Evaluate the fitness of each new individual  
    }  
}
```

The above algorithm is simple GA to solve the problems that falls in category of NP-class computing problems. In genetic algorithm the initial population is generated randomly. After that the crossover is performed and then mutation is applied to generate new population. Selection of the new population from the best population is done thereafter. The process is repeated until stopping criteria is met.

### **3.2 Why to use GA**

GA is a soft computing technique. QoS parameters optimization in Computational Grid is an NP class of problem and thus GA may be applied to solve this problem. For the scheduling problem the GA is not the best choice because of its decision making. But due to the simplicity of GA people generally apply it. This method gives us nearly optimal solution.

### **3.3 Concluding Remarks**

GA is a soft computing technique. Scheduling in Grid for QoS Parameters Optimization is an NP class problem and thus GA can be used for this. In this chapter, we have discussed about GA and various GA operators. Crossover is done more frequently than other GA operator mutation. Generally two GA operators are enough for solving soft-computing problem which are crossover and mutation. Selection plays an important role in GA. A brief discussion over various selection methods of GA has been discussed. Roulette-wheel selection is one of the often used selection method and has been described in this chapter. We also have deliberated why to use GA in this chapter.

## Chapter 4

### The Proposed Model

This chapter discusses about Scheduling Problem [9, 10, 12], QoS parameters expected from the scheduler [9, 10, 12] of the grid. It also discusses about the fitness function used in the model. The proposed model is presented thereafter. Also the algorithm to solve the scheduling problem of computational grid is described. Finally, we make the concluding remarks.

Scheduling is a fundamental issue for getting high performance in computational grid. In computational grid, there are large numbers of resources and the optimal global uses of the resources affect both the grid system and the applications both. The problem of local resources utilization is already solved to a greater extent. We consider here two types of scheduler first is local scheduler and second is global scheduler. Local scheduler takes care about the scheduling inside each individual node/processor of the grid. In the proposed problem we assume that local scheduler works in any manner as described by the various operating systems [12].

Here we consider only two parameters with scheduling which are turnaround time and speed up. We have already described the parameter turnaround time in 2.4.

Initially global scheduler randomly distributes the task to the nodes in grid environment. Generally distribution of the tasks is accordingly to the processing speeds of each machine in the grid. This gives us a solution, called as population in GA terminology. We randomly generate this type of many solutions i.e. initial population. We calculate the fitness of the population using fitness function. We select half of the population using roulette-wheel selection method. After that crossover is performed 95% of time and mutation 5% of time of total number of generations. In this way new population is generated which includes the old population also. Now arrange the new population

according to its fitness value and make it as old population. Repeat this process until stopping criteria meets.

## 4.1 QoS Parameters

Various QoS Parameters have already been described in Chapter 3. Here, we concentrated only on two QoS Parameters. One of them is Finishing Time of the job. It is also known as Turnaround time. Another QoS Parameter is Speed Up. The Speed Up is defined as the ratio of the time taken by a single processor to that of many processors of the computational grid. It lies between 1 to n where n is the number of processors in the computational grid.

## 4.2 Fitness Function

To derive the fitness function we have applied Queuing Theory [30, 31, 32, 33, 34, 35, 36]. Our problem falls in category of M/M/S Model. It is because we assume that there are many systems/nodes/machines and many jobs/sub-jobs to execute. Every system has its own capability to process the job.

Assume that average arrival rate and average service rate of the system are  $\lambda$  and  $\mu$  MIPS respectively. The average waiting time using M/M/1 is given below

$$\lambda/(\mu - \lambda) \text{--- eqn.4.1}$$

The average Service Time is

$$1/\mu \text{--- eqn.4.2}$$

The total average waiting time in the system is given below

$$\lambda/(\mu - \lambda) + 1/\mu \text{--- eqn.4.3}$$

When it is simplified, it gives total average waiting time by the system is given below

$$1/(\mu - \lambda) \text{--- eqn.4.4}$$

The scheduling problem discussed here falls in M/M/S Model. Therefore we assume that the arrival rate and service rate of  $i^{\text{th}}$  system is  $\lambda_i$  and  $\mu_i$  MIPS respectively. The waiting time using M/M/S is given below

$$\lambda_i / (\mu_i - \lambda_i) \text{-----eqn.4.5}$$

The Service Time of  $i^{\text{th}}$  system is  $((\sum_{k=1}^{k=r} \text{Task}_k) / \mu_i)$ .

Therefore total average waiting time  $i^{\text{th}}$  system is given below

$$[(\lambda_i (\mu_i - \lambda_i) + (1 / \mu_i)) * (\sum_{k=1}^{k=r} \text{Task}_k)] \text{---eqn.4.6}$$

Where r is total no of tasks allocated in a machine.

Now total time taken by the grid is given below

$$\max_{i=1}^{i=m} [(\lambda_i (\mu_i - \lambda_i) + (1 / \mu_i)) * (\sum_{k=1}^{k=r} \text{Task}_k)] \text{--- eqn.4.7}$$

Where i vary from 1 to m and m is the total no of machines. Assume that

$$T_j = \max_{i=1}^{i=n} [(\lambda_i (\mu_i - \lambda_i) + (1 / \mu_i)) * (\sum_{k=1}^{k=r} \text{Task}_k)] \text{--- eqn.4.8}$$

$$\min_{j=1}^{j=n} [T_j = \max_{i=1}^{i=n} [(\lambda_i (\mu_i - \lambda_i) + (1 / \mu_i)) * (\sum_{k=1}^{k=r} \text{Task}_k)]] \text{-----eqn.4.9}$$

Where, j varies from 1 to n, n being the population size.

Here in above fitness function we are not considering the communication cost. Adding communication costs between various modules/subtasks of the application, we can modify the above function.

Assume that average arrival rate and average network load in system are l and b MBPS respectively. Similarly average waiting time using M/M/1 is given below.

$$1/(b-l) \text{-----eqn.4.10}$$

The scheduling problem discussed here falls in M/M/S Model. Therefore, we assume that the arrival rate and service rate of  $i^{\text{th}}$  link is  $l_i$  and  $b_i$  MBPS respectively. The waiting time using M/M/1 is given below

$$[b_i / (b_i - l_i) * (\sum_{k=1}^{k=r} \text{message}_k)] \text{-----eqn.4.11}$$

Here r is total number of messages in  $i^{\text{th}}$  link.

The Service Time of  $i^{\text{th}}$  link is  $((\sum_{k=1}^{k=r} \text{message}_k) / b_i)$ .

Therefore total average waiting time  $i^{\text{th}}$  system is given below

$$[(l_i (b_i - l_i) * \sum_{k=1}^{k=r} \text{message}_k + (1 / b_i) * \sum_{k=1}^{k=r} \text{message}_k)] \text{--- eqn.4.12}$$

Where, r is total no of messages in the network.

Now, the total time (the optimal function) is given below [55, 56]

Minimize

$$N_j = \max_{i=1}^{i=n} [(l_i(b_i - l_i) * \sum_{k=1}^{k=r} message_k) + (d * \sum_{k=1}^{k=r} message_k) / b_i] - eqn.4.13$$

Where, i varies from 1 to n. n is the total no of links and d is the hamming distance between the nodes of the grid. The problem thus becomes as

$$\text{minimize}_{j=1}^{j=n} [T_j + N_j] - eqn.4.14$$

Where n is population size and j varies from 1 to n.

### 4.3 The Model

The model proposed minimizes the turnaround time of the job submitted for execution. It applies GA for this purpose. The various modules of the GA specific to this problem are as follows.

**Chromosome Structure** of the problem is given below

```
Struct task {int task_no; double task_size ;} task;
Struct machine {int machine no; double_machine ;} machine;
Struct chromosome {task []; machine ;};
```

**Initial Population** is generated randomly. Each Chromosome of the population is generated randomly. The random generation is based on the random permutation concept in which we generate the permutation randomly. If we want to improve performance of the algorithm then we generate the population based on minimum time which we will discuss in future work.

**Crossover**, used single-point crossover one common site in the chromosomes is chosen randomly. Chromosomes of the parents are cut at that point and the resulting sub-chromosomes are swapped.

**Selection**, used roulette-wheel selection method. In this chance of an individual's being selected is proportional to the amount by which its fitness is greater or less than its competitors' fitness. Each individual gets a slice of the wheel. More fit ones get larger slices than less fit ones. The wheel is then spun and an individual "owns" the section on which it lands.

### 4.3.1 Genetic Algorithm for Computational Grid

The pseudo-code of the algorithm for the proposed model is as follows.

*Algorithm ( )*

```
{   Intialize_population ( );
    Evaluate the fitness value of each individual against the fitness function using
    calculate_fitness ( ). //fitness is calculated according its turnaround time
    Sort the population. //Sort Population using bubble sort according its fitness
    Select the half best population
    for (i=1; i<=no_of_generations; i++)
    {
        Initial individual of new population are mixed with old population
        Randomly pick two parents from the population
        Perform the crossover over the parents to produce offspring based on the
        probability of applying crossover (95%)
        Mutate an offspring based on the probability of applying mutation (5%).
        Store it in the rest part of new population
        Evaluate the fitness of each new individual
        Sort the new population
        Select the half best population
    }
}
```



#### **4.4 Concluding Remarks**

In this chapter, GA is applied to solve the scheduling problem of the Computational Grid. Scheduling problem in computational grid is considered to be the most challenging problem. Through scheduling the turnaround time offered to the job is optimized. Fitness function considering the two situations; one in which non-interactive jobs are considered and the other in which interactive jobs are considered have been described. The structure of chromosome, initial population, crossover applied and selection method applied is also discussed in this chapter. Finally, the pseudo code of the proposed scheduling problem in grid using GA is presented.

## Chapter 5

### Experimental Evaluation

This chapter discusses about the experiments carried out for the model presented in this work followed by the observations derived from the experiments.

The experiments have been performed for turnaround time and speedup. To represent the result graph is plotted between generation verses turnaround time.

Based on the two fitness functions, proposed in the previous chapter, we have performed the experiments for the two cases as follows.

Case1: We use fitness function described in equation 4.9.

Case2: We use fitness function described in equation 4.14.

Description of various input parameters used in the implementation of the algorithm described in chapter 4 is as follows.

| Sr. No. | Variable Name | Meaning                      | Range     |
|---------|---------------|------------------------------|-----------|
| 1       | popsize       | population size              | 10-50     |
| 2       | generations   | no of generation             | 100-500   |
| 3       | old pop       | old population               | 10-50     |
| 4       | new pop       | new population               | 20-100    |
| 5       | tempo         | temporary population         | 10-50     |
| 6       | P[.miue       | speed of processor (in MIPS) | 200-300   |
| 7       | P[.lamda      | load of processor (in MIPS)  | 1-100     |
| 8       | l             | network load(in MBPS)        | 60-90     |
| 9       | b             | bandwidth(in MBPS)           | 100-120   |
| 10      | Task[.size    | Task Size(in MIPS)           | 2000-5000 |

Table 5.1-Description of variables used in program

## 5.1 Parallel Task without Communication Cost

Our first set of experiments does not consider the interactive modules i.e. the execution of the job without communication amongst them.

### 5.1.1 Observation with 50 Machines

Experiments have been carried out with 50 nodes/machines in the grid and variable number of tasks as follows. Other parameters of the experiment are also given.

Number of Tasks =50, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS.

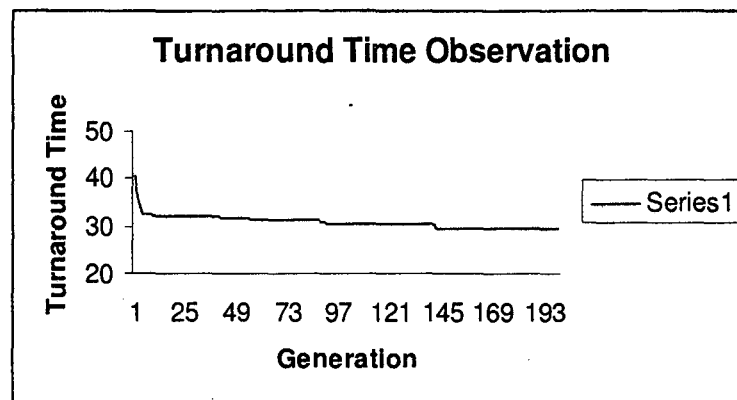


Figure 5.1.1.1 Turnaround Time Observation with 50 Tasks

The achieved Minimum Turnaround time (TAT) =29.6295, Average TAT =30.7614, and the Speed Up =37.8686.

Number of Tasks =60, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS,Range of Task Size 2000-5000 MIPS.

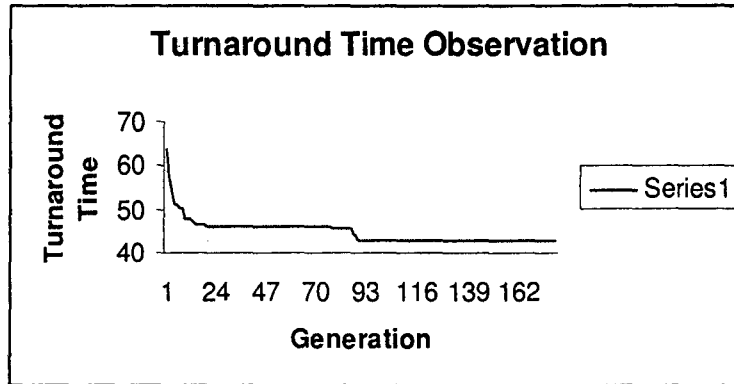


Figure 5.1.1.2 Turnaround Time Observation with 60 Tasks

The achieved Minimum Turnaround time (TAT)=42.9450, Average TAT =44.6660, Speed Up =31.3683

Number of Tasks =70, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS,Range of Task Size 2000-5000 MIPS.

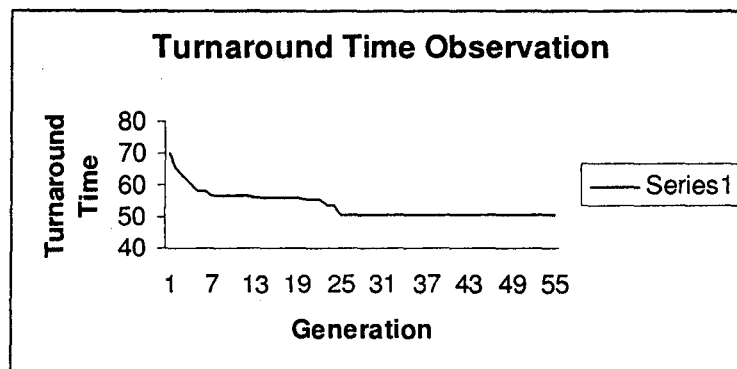


Figure 5.1.1.3 Turnaround Time Observation with 70 Tasks

The achieved Minimum Turnaround time (TAT)=50.3140, Average TAT =51.2004, Speed Up =31.0707

Number of Tasks =80, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS,Range of Task Size 2000-5000 MIPS.

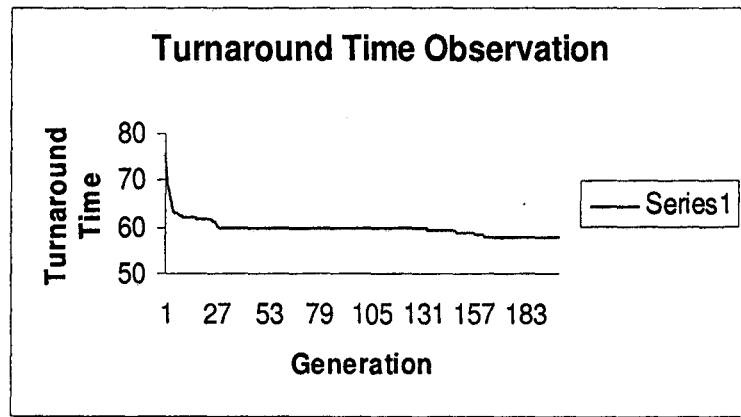


Figure 5.1.1.4 Turnaround Time Observation with 80 Tasks

The achieved Minimum Turnaround time (TAT) =57.9150, Average TAT =59.8608, Speed Up =31.1236

Number of Tasks =90, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS,Range of Task Size 2000-5000 MIPS.

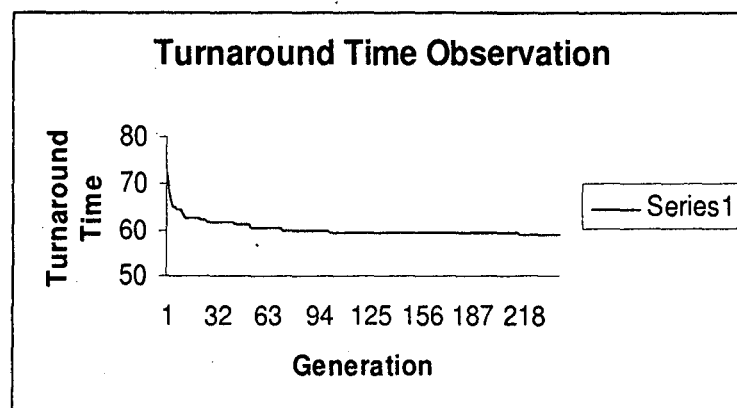


Figure 5.1.1.5 Turnaround Time Observation with 90 Tasks

The achieved Minimum Turnaround time (TAT) =57.9314, Average TAT =60.0707, Speed Up =34.8792

Number of Tasks =100, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS,Range of Task Size 2000-5000 MIPS.

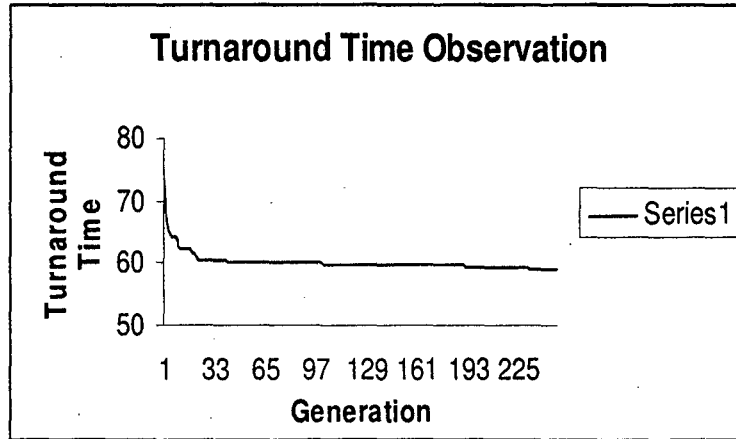


Figure 5.1.1.6 Turnaround-Time Observation with 100 Tasks

The achieved Minimum Turnaround time (TAT) =58.9151, Average TAT =60.1472, Speed Up =38.7071

Number of Tasks =110, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS,Range of Task Size 2000-5000 MIPS.

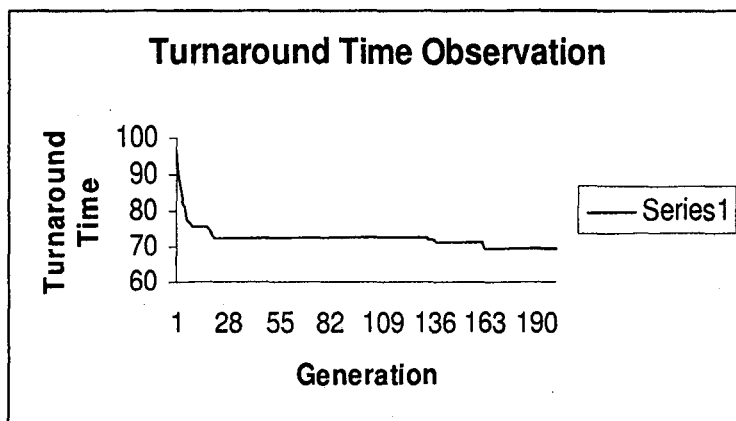


Figure 5.1.1.7 Turnaround-Time Observation with 110 Tasks

The achieved Minimum Turnaround time (TAT) =69.6059, Average TAT =71.8027, Speed Up =35.5703

Number of Tasks =120, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS,Range of Task Size 2000-5000 MIPS.

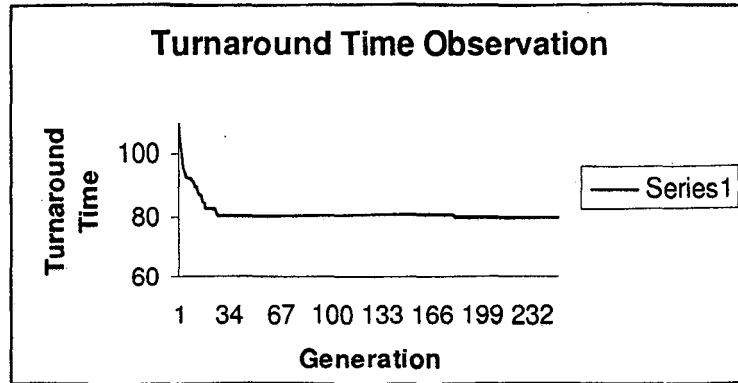


Figure 5.1.1.8 Turnaround-Time Observation with 120 Tasks

The achieved Minimum Turnaround time (TAT) =79.4821, Average TAT =80.7782, Speed Up =33.8668

Number of Tasks =130, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS,Range of Task Size 2000-5000 MIPS.

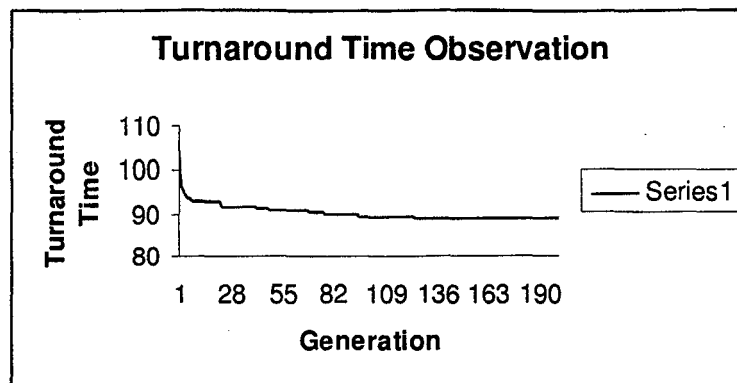


Figure 5.1.1.9 Turnaround-Time Observation with 130 Tasks

The achieved Minimum Turnaround time (TAT) =88.9918, Average TAT =90.2364, Speed Up =32.6597

### 5.1.2 Observation with 60 Machines

Experiments have been carried out making number of nodes/machines to 60 in the grid. Other parameters are given.

Number of Tasks =50, Number of Machines =60, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS

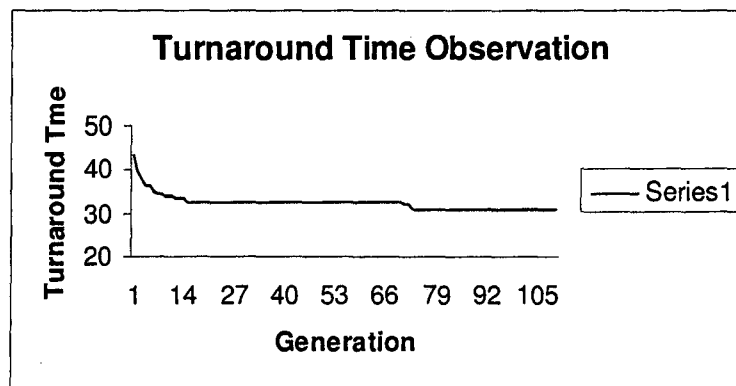


Figure 5.1.2.1 Turnaround-Time Observation with 50 Tasks

The achieved Min TAT =30.5804, Average TAT =31.5365, Speed Up =43.5056

Number of Tasks =60, Number of Machines =60, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS

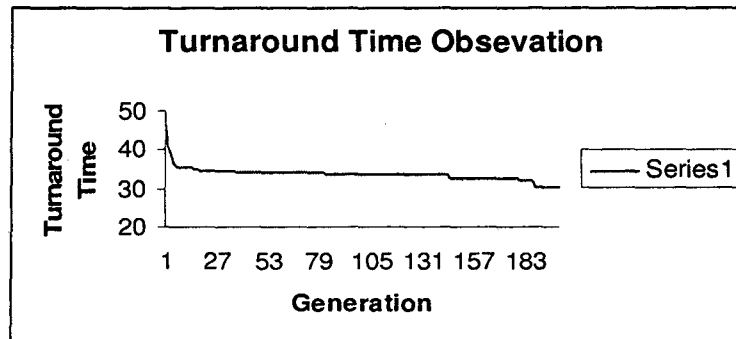


Figure 5.1.2.2 Turnaround-Time Observation with 60 Tasks,

The achieved Min TAT =30.1619, Average TAT =33.5950, Speed Up =53.5217



Number of Tasks =70, Number of Machines =60, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS,Range of Task Size 2000-5000 MIPS

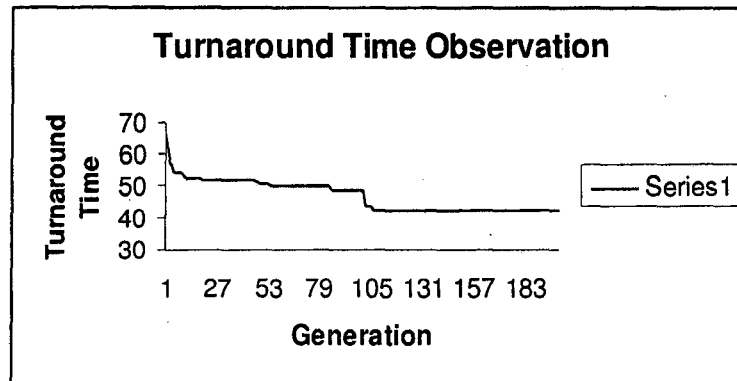


Figure 5.1.2.3 Turnaround-Time Observation with 70 Tasks

The achieved Min TAT =42.7544, Average TAT =46.1048, Speed Up =43.6025

Number of Tasks =80, Number of Machines =60, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS,Range of Task Size 2000-5000 MIPS

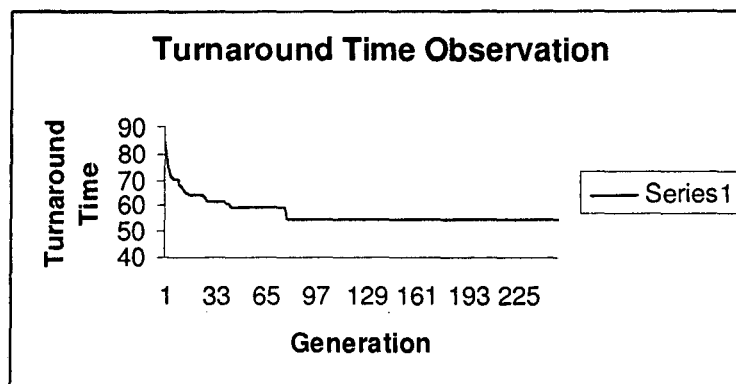


Figure 5.1.2.4 Turnaround-Time Observation with 80 Tasks

The achieved Minimum Turnaround Time (TAT) =54.9845, Average TAT =56.5972, Speed Up =43.6325

Number of Tasks =90, Number of Machines =60, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS, Min TAT

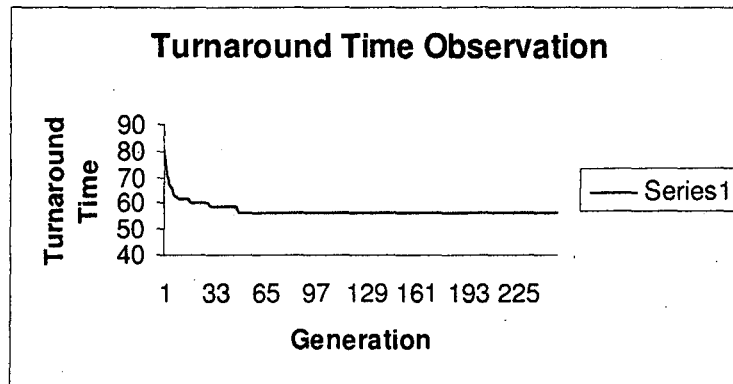


Figure 5.1.2.5 Turnaround-Time Observation with 90 Tasks

The achieved Minimum Turnaround Time (TAT) =56.7327, Average TAT =57.611, Speed Up =42.3705

From the above experiments we derive the following conclusions

1. Through the experiments we found that solution saturated average around 50 generations and also the saturation of result depends upon the initial solution.
2. If number of tasks increases, turnaround time also increases.
3. If number of machines increases then turnaround time decreases with same number of tasks.
4. If number of task increases then speed-up increases but after certain time period it starts decreasing. It is because now the grid is highly loaded and is not able to offer much for the task execution.

## 5.2 Parallel Tasks with Communication Cost

We have preformed the same set of experiments considering the interactive modules i.e. communication between the modules/subtasks of the task is effective.

### 5.2.1 Observation with 50 Machines

Number of Tasks =50, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS, Load of Network Link 60 MBPS, Load of Bandwidth 100 MBPS, Maximum Hamming Distance-10, Maximum Communication Cost -30 MBPS

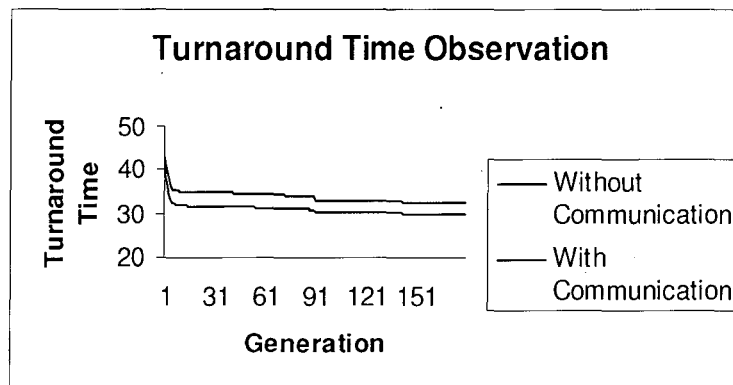


Figure 5.2.1.1 Turnaround Time Observation with 50 Tasks

OUTPUT: Minimum Turnaround Time =32.6295, Average Turnaround Time =33.7614, Speed Up =34.3869

Number of Tasks =60, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS, Load of Network Link 60 MBPS, Load of Bandwidth 100 MBPS, Maximum Hamming Distance-10, Maximum Communication Cost -30 MBPS

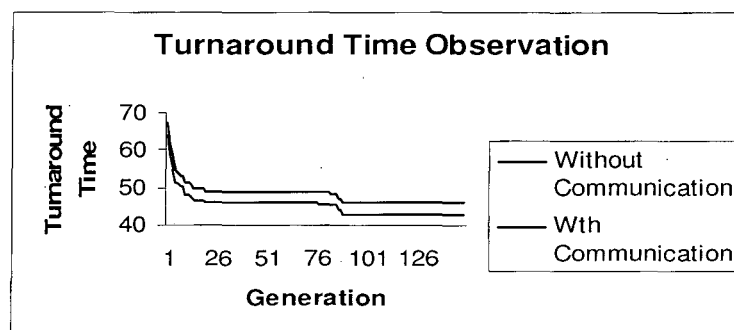


Figure 5.2.1.2 Turnaround Time Observation with 60 Tasks

OUTPUT: Minimum Turnaround Time =45.9450, Average Turnaround Time =47.6660, Speed Up =29.3201

Number of Tasks =70, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS, Load of Network Link 60 MBPS, Load of Bandwidth 100 MBPS, Maximum Hamming Distance-10, Maximum Communication Cost -30 MBPS

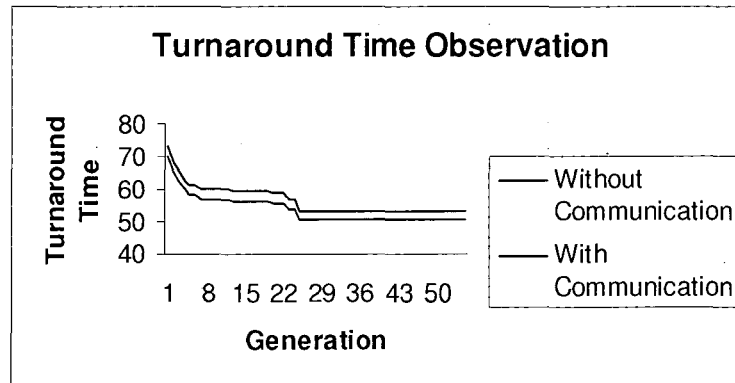


Figure 5.2.1.3 Turnaround Time Observation with 70 Tasks  
 OUTPUT: Minimum Turnaround Time =53.3140, Average Turnaround Time =54.2004, Speed Up =29.3223

Number of Tasks =80, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS, Load of Network Link 60 MBPS, Load of Bandwidth 100 MBPS, Maximum Hamming Distance-10, Maximum Communication Cost -30 MBPS

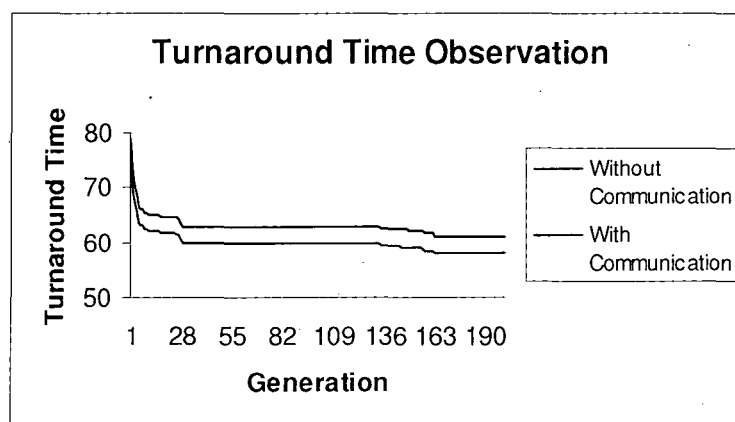


Figure 5.2.1.4 Turnaround Time Observation with 80 Tasks  
 OUTPUT: Minimum Turnaround Time =60.9950, Average Turnaround Time =62.8608, Speed Up =29.5928

Number of Tasks =90, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS, Load of Network Link 60 MBPS, Load of Bandwidth 100 MBPS, Maximum Hamming Distance-10, Maximum Communication Cost -30 MBPS

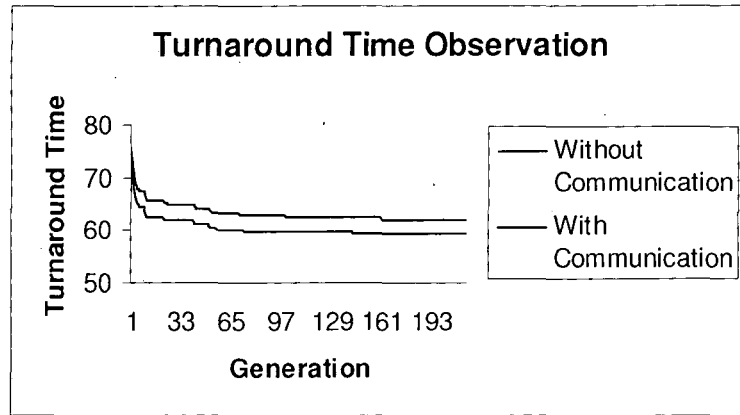


Figure 5.2.1.5 Turnaround Time Observation with 90 Tasks  
 OUTPUT: Minimum Turnaround Time =60.8314, Average Turnaround Time =63.0707, Speed Up =31.6815

Number of Tasks =100, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS, Load of Network Link 60 MBPS, Load of Bandwidth 100 MBPS, Maximum Hamming Distance-10, Maximum Communication Cost -30 MBPS

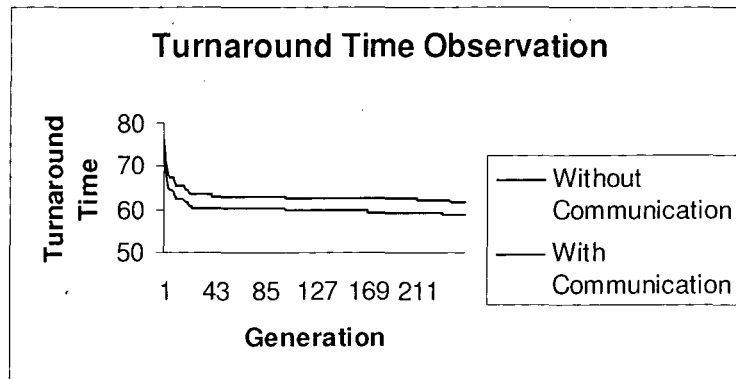


Figure 5.2.1.6 Turnaround Time Observation with 100 Tasks  
 OUTPUT: Minimum Turnaround Time =61.9151, Average Turnaround Time =63.1472, Speed Up =36.8259

Number of Tasks =110, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS, Load of Network Link 60 MBPS, Load of Bandwidth 100 MBPS, Maximum Hamming Distance-10, Maximum Communication Cost -30 MBPS

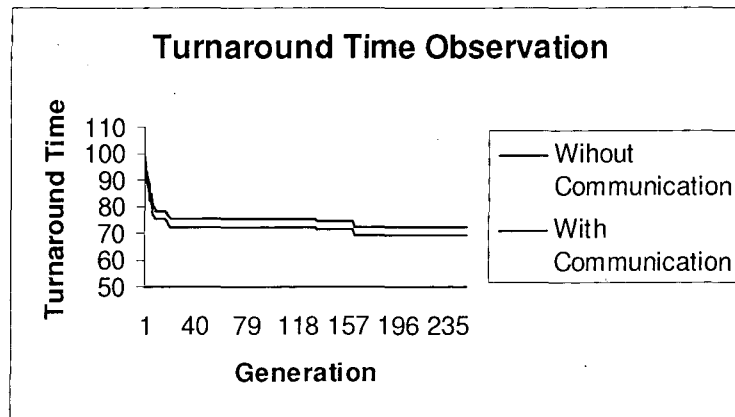


Figure 5.2.1.7 Turnaround Time Observation with 110 Tasks

OUTPUT: Minimum Turnaround Time =72.6059, Average Turnaround Time =74.8027, Speed Up =34.1006

Number of Tasks =120, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS, Load of Network Link 60 MBPS, Load of Bandwidth 100 MBPS, Maximum Hamming Distance-10, Maximum Communication Cost -30 MBPS

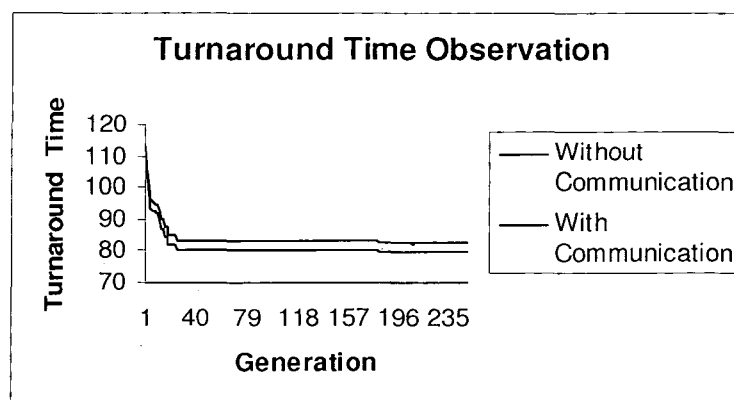


Figure 5.2.1.8 Turnaround Time Observation with 120 Tasks

OUTPUT: Minimum Turnaround Time =82.4821, Average Turnaround Time =83.7642, Speed Up =32.6350

Number of Tasks =130, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS, Load of Network Link 60 MBPS, Load of Bandwidth 100 MBPS, Maximum Hamming Distance-10, Maximum Communication Cost -30 MBPS

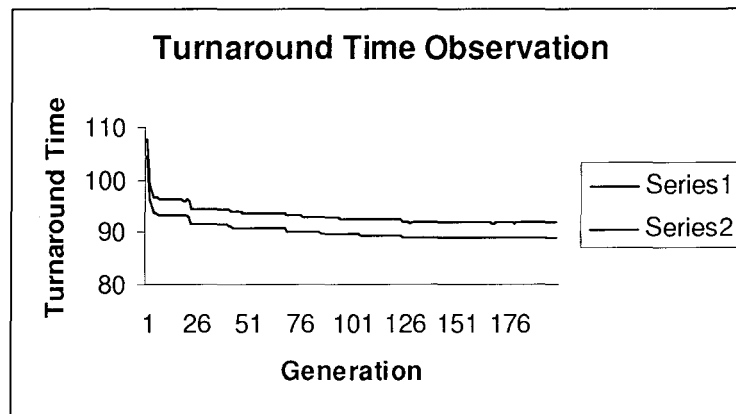


Figure 5.2.1.9 Turnaround Time Observation with 130 Tasks

OUTPUT: Minimum Turnaround Time =91.8618, Average Turnaround Time =93.2304, Speed Up =31.5942

### 5.2.2 Observation with 60 Machines

Number of Tasks =50, Number of Machines =60, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS, Load of Network Link 60 MBPS, Load of Bandwidth 100 MBPS, Maximum Hamming Distance-10, Maximum Communication Cost -30 MBPS

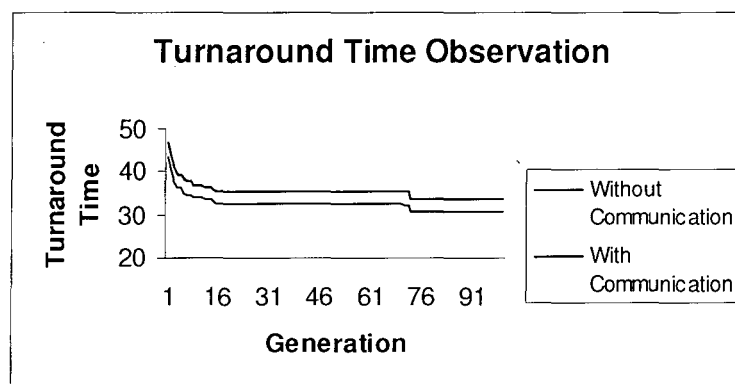


Figure 5.2.2.1 Turnaround Time Observation with 50 Tasks

OUTPUT: Minimum Turnaround Time =33.5804, Average Turnaround Time =34.5345, Speed Up =39.6198

Number of Tasks =60, Number of Machines =60, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS, Load of Network Link 60 MBPS, Load of Bandwidth 100 MBPS, Maximum Hamming Distance-10, Maximum Communication Cost -30 MBPS

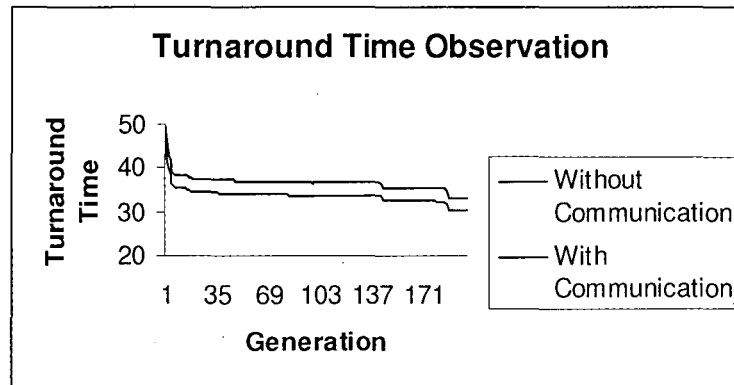


Figure 5.2.2.2 Turnaround Time Observation with 60 Tasks

OUTPUT: Minimum Turnaround Time =33.1699, Average Turnaround Time =36.5930, Speed Up =48.6810

Number of Tasks =70, Number of Machines =60, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS, Load of Network Link 60 MBPS, Load of Bandwidth 100 MBPS, Maximum Hamming Distance-10, Maximum Communication Cost -30 MBPS

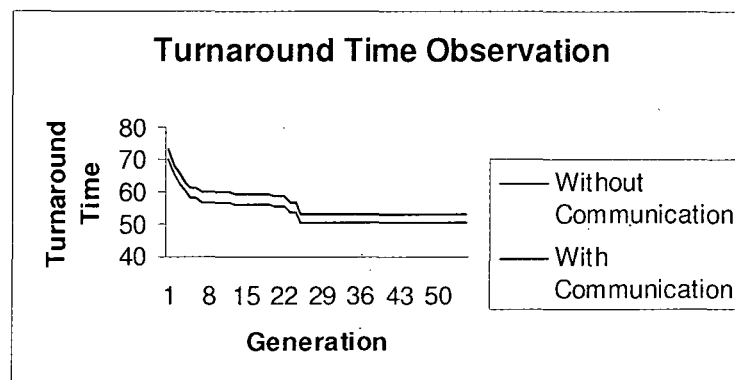


Figure 5.2.2.3 Turnaround Time Observation with 70 Tasks

OUTPUT: Minimum Turnaround Time =45.7544, Average Turnaround Time =49.1028, Speed Up =40.7436



Number of Tasks =80, Number of Machines =60, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS, Load of Network Link 60 MBPS, Load of Bandwidth 100 MBPS, Maximum Hamming Distance-10, Maximum Communication Cost -30 MBPS

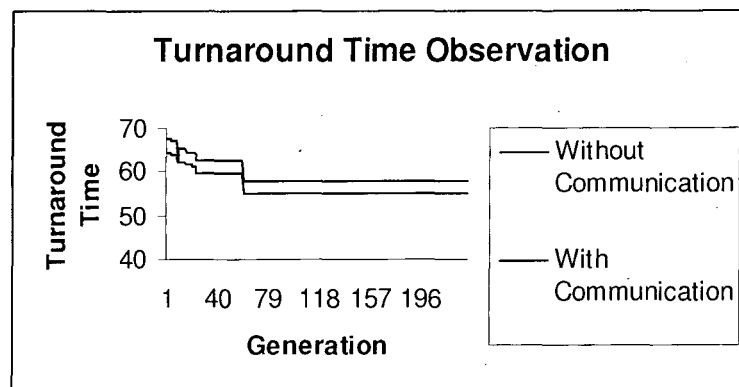


Figure 5.2.2.4 Turnaround Time Observation with 80 Tasks

OUTPUT: Minimum Turnaround Time =57.9845, Average Turnaround Time =59.4936, Speed Up =40.8715

Number of Tasks =90, Number of Machines =60, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS, Load of Network Link 60 MBPS, Load of Bandwidth 100 MBPS, Maximum Hamming Distance-10, Maximum Communication Cost -30 MBPS

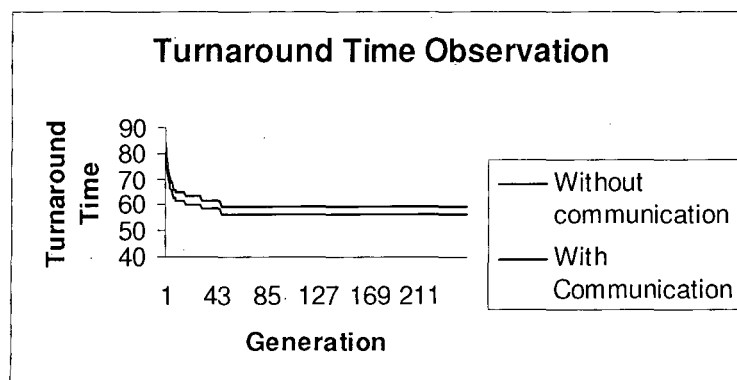


Figure 5.2.2.5 Turnaround Time Observation with 90 Tasks

OUTPUT: Minimum Turnaround Time =59.7327, Average Turnaround Time =60.6110, Speed Up =40.2425

### 5.2.3 Observation with 50 Machines and More Communication Cost

Number of Tasks =100, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS, Load of Network Link 60 MBPS, Load of Bandwidth 100 MBPS, Maximum Hamming Distance-10, Maximum Communication Cost -50 MBPS

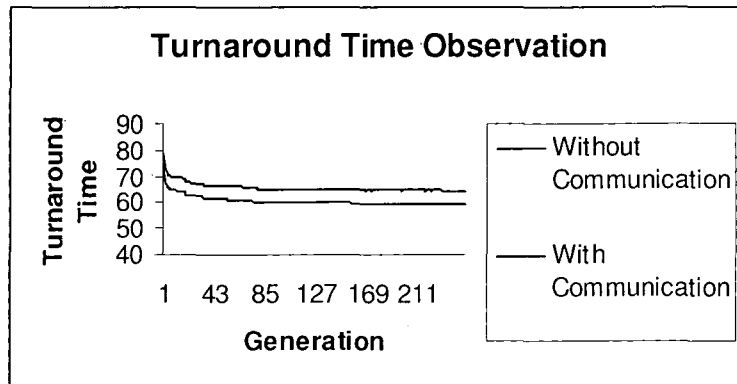


Figure 5.2.3.1 Turnaround Time Observation with 50 MBPS  
OUTPUT: Minimum Turnaround Time =64.4570, Average Turnaround Time =65.7607, Speed Up =35.3736

Number of Tasks =100, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS, Load of Network Link 60 MBPS, Load of Bandwidth 100 MBPS, Maximum Hamming Distance-10, Maximum Communication Cost -80 MBPS

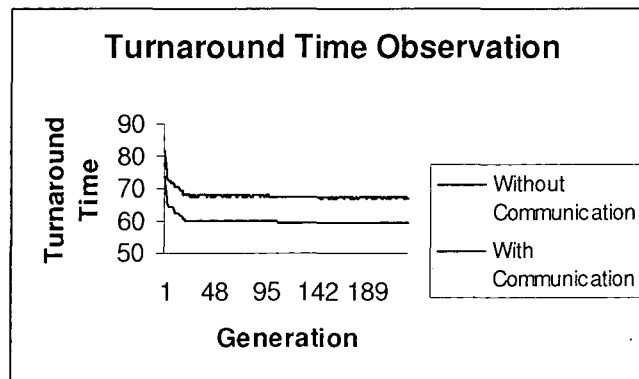


Figure 5.2.3.2 Turnaround Time Observation with 80 MBPS  
OUTPUT: Minimum Turnaround Time =66.5064, Average Turnaround Time =67.7245, Speed Up =35.3736, Speed Up =34.4859

### 5.2.4 Observation with 50 Machines with increase in Hamming Distance

Number of Tasks =100, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS, Load of Network Link 60 MBPS, Load of Bandwidth 100 MBPS, Maximum Hamming Distance-20, Maximum Communication Cost -30 MBPS

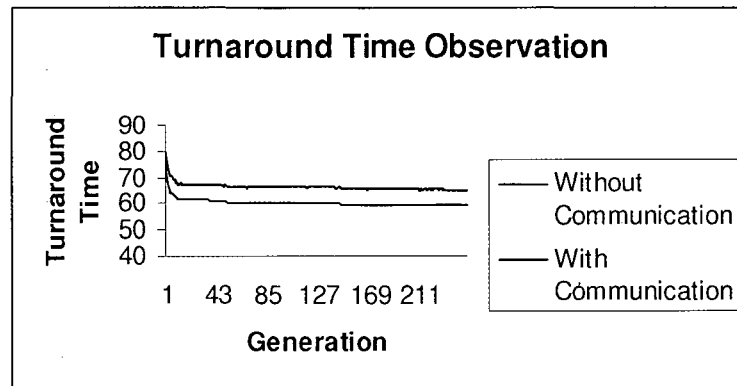


Figure 5.2.4.1 Turnaround Time Observation with Hamming Distance 20

OUTPUT: Minimum Turnaround Time =65.2589, Average Turnaround Time =66.4273, Speed Up =34.9389

Number of Tasks =100, Number of Machines =50, Population Size =50, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300MIPS, Range of Task Size 2000-5000 MIPS, Load of Network Link 60 MBPS, Load of Bandwidth 100 MBPS, Maximum Hamming Distance-30, Maximum Communication Cost -30 MBPS

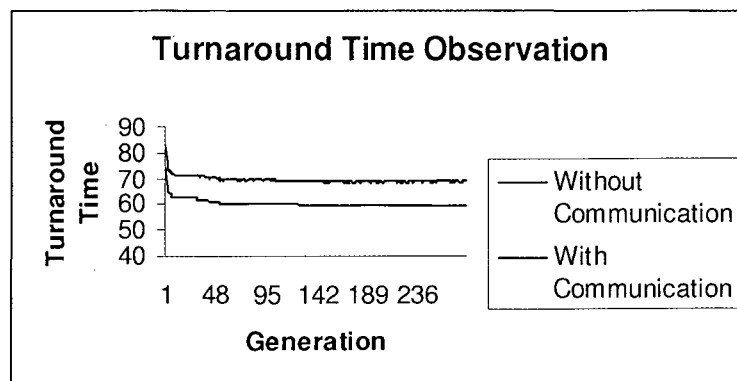


Figure 5.2.4.2 Turnaround Time Observation with Hamming Distance 30

OUTPUT: Minimum Turnaround Time =68.4882, Average Turnaround Time =69.3383, Speed Up =33.2916,

### 5.3 Summary Result

Experiments with varying number of tasks with 50 machines and then 60 machines have been conducted. Results have been summarized in the tables given below. We have considered two cases for obtaining turnaround- time; without communication and with communication.

#### Case 1: Parallel Tasks without Communication

| No of Tasks | Generations | Average TAT | Min TAT | Speed-Up |
|-------------|-------------|-------------|---------|----------|
| 1. 50       | 145         | 30.7614     | 29.6295 | 37.8686  |
| 2. 60       | 90          | 44.6660     | 42.9450 | 31.3683  |
| 3. 70       | 30          | 51.2004     | 50.3140 | 31.0707  |
| 4. 80       | 160         | 59.8608     | 57.9150 | 31.1236  |
| 5. 90       | 220         | 60.0707     | 57.9314 | 34.8792  |
| 6. 100      | 220         | 60.1472     | 58.9151 | 38.7071  |
| 7. 110      | 170         | 71.8027     | 69.6059 | 35.5703  |
| 8.120       | 165         | 80.7782     | 62.3883 | 33.8668  |
| 9.130       | 115         | 90.2364     | 88.9918 | 32.6597  |

Table 5.2- Comparison Tasks without Communication in 50 Machines

| No of Tasks | Generations | Average TAT | Min TAT | Speed-Up |
|-------------|-------------|-------------|---------|----------|
| 1. 50       | 70          | 31.5365     | 30.5804 | 43.5056  |
| 2. 60       | 190         | 33.5950     | 30.1619 | 53.5217  |
| 3. 70       | 110         | 46.1048     | 42.7544 | 43.6025  |
| 4. 80       | 70          | 56.5972     | 54.9845 | 43.6325  |
| 5. 90       | 50          | 57.611      | 56.7327 | 42.3705  |

Table 5.3- Comparison Tasks without Communication in 60 Machines

## Case 2: Parallel Tasks with Communication

| No of Tasks | Generations | Average TAT | Min TAT | Speed-Up |
|-------------|-------------|-------------|---------|----------|
| 1. 50       | 150         | 33.7614     | 32.6295 | 34.3869  |
| 2. 60       | 95          | 47.6660     | 45.9450 | 29.3201  |
| 3. 70       | 35          | 54.2004     | 53.3140 | 29.3223  |
| 4. 80       | 165         | 62.8608     | 60.9950 | 29.5928  |
| 5. 90       | 162         | 63.0707     | 60.8314 | 31.6815  |
| 6. 100      | 250         | 63.1472     | 61.9151 | 36.8259  |
| 7. 110      | 173         | 74.8027     | 72.6059 | 34.1006  |
| 8. 120      | 165         | 83.7642     | 82.4821 | 32.6350  |
| 9. 130      | 190         | 93.2304     | 91.8618 | 31.5942  |

Table 5.4- Comparison Parallel Tasks with Communication

| No of Tasks | Generations | Average TAT | Min TAT | Speed-Up |
|-------------|-------------|-------------|---------|----------|
| 1. 50       | 75          | 34.5345     | 33.5804 | 39.6198  |
| 2. 60       | 192         | 36.5930     | 33.1699 | 48.6810  |
| 3. 70       | 30          | 49.1028     | 45.7544 | 40.7436  |
| 4. 80       | 50          | 59.4936     | 57.9845 | 40.8715  |
| 5. 90       | 50          | 60.6110     | 59.7327 | 40.2425  |

Table 5.5- Comparison Parallel Tasks with Communication

From the above experiments we derive the following conclusions.

1. Through the experiments we observe that the solution converged around 50 generations and saturation of result depends upon the initial solution.
2. If number of tasks increases then turnaround time increases.
3. If number of machines increases then turnaround time decreases.
4. If number of task increases then speed-up increases but after a certain time it starts decreasing.
5. If hamming distance increases then TAT increases and speedup decreases and solution is saturated after more number of generations.

6. If communication cost increases then TAT increases and speedup decreases and solution is saturated after more number of generations.

## 5.4 Comparisons between ACO and GA

We make a comparison between the two methods to solve the same problem. One using GA (proposed and experimented by me) and the other using ACO (proposed and experimented by my batch-mate).

### 5.4.1 Number of Tasks =12 Number of Machines =4, Arrival Rates are 0 MIPS

Processing Speed 1,2,3,4 MIPS

Tasks Size are 5,10,12,14,8,6,5,10,20,18,15 MIPS

Population Size =10, Generation =50 using GA

No of Ants =2, Iteration=50 using ACO

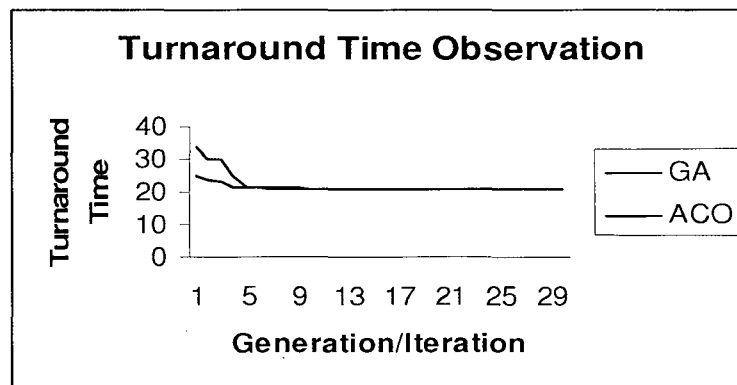


Figure 5.4.1 Turnaround Time Observation with 12 Tasks

OUTPUT: Minimum Turnaround Time =20.5, Speed Up =1.656

**5.4.2 Number of Tasks =18, Number of Machines =4, Arrival Rates are 0 MIPS**

Processing Speed 1,2,3,4 MIPS

Tasks Size are 2,4,6,8,10,12,14,14,16,18,18,18,20,20,22,24,24,4,6 MIPS

Population Size =10, Generation =50 using GA

No of Ants =2, Iteration=50 using ACO

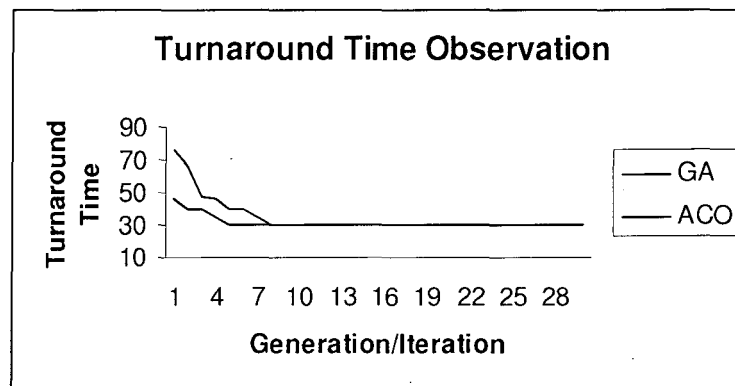


Figure 5.4.2 Turnaround Time Observation with 18 Tasks

OUTPUT: Minimum Turnaround Time =30, Speed Up =1.7976

In above two experiments we took same number of tasks, machines, same  $\lambda$  &  $\mu$  and same workload. We observe that the ACO [54] performs better than GA. The reason may be drawn as because ACO is decision based technique and scheduling is decision based problem it performs better. We also find that ACO converges faster than GA. Therefore ACO performs better than GA. We are not considering communication cost based experiments for comparison.

Finally we conclude from above observation and above experiments we conclude that ACO gives better performance than GA. Because of ACO is decision based technique and scheduling is decision based problem. For detail about ACO we will discuss in future work.

# Chapter 6

## Conclusion & Future Scope

Over past few years, developments in Grid Computing are enormous. Still, being a highly heterogeneous system, Grid poses a lot of constraints. The research in the area of grid computing is going on. If the problems pertaining to the grid are handled effectively the grid will be widely available for solving the future high performance demanding applications. In fact the future will be governed by the grid.

This dissertation work deals with QoS parameters optimization with scheduling in Computational Grid. This dissertation handles only two QoS parameter of Computational Grid; turnaround time and speed up. Two cases have been considered for obtaining turnaround time.

Case 1: Parallel Tasks without Communication

Case 2: Parallel Tasks with Communication

The following conclusions are derived from the overall study.

1. Most of the experiments depicts that solution converges at around 50 generations and convergence of result depends upon initial solution.
2. When number of tasks increases, turnaround time also increases.
3. When number of machines increases, turnaround time decreases.
4. Most of the results show that when number of task increases, speed-up linearly increases, though after certain time it starts decreasing.
5. The effect of hamming distance is also observed. When this distance is more the Turnaround time increases and speedup decreases.
6. Communication Cost between interactive modules increases the Turnaround Time and decreases the speedup.
7. ACO based scheduling model performs better than GA based model.



## **6.1 Future Scope**

There are many QoS parameters that can be handled while scheduling the job for execution on computational grid. We are able to handle only two in this work. In future, we plan to consider other QoS parameters and therefore many more models for this purpose. Also, there is vast possibility of applying other evolutionary methods to solve the scheduling problem. We also plan to apply other evolutionary methods towards this. A comparative study will give us the insight for which type of problem which method will be more appropriate. The comparison of ACO and GA based models shows that ACO gives better performance than GA. It is to be explored further in our future work.

## References

1. Daniel A. Menasce and Emillano Casalicchiop, "QoS in Grid Computing", IEEE Computer Society, 2004.
2. Melanie Mitchell, "An Introduction to Genetic Algorithms", PHI, Third Edition, 1996.
3. D. Goldenberg, "Genetic Algorithms in Search Optimization and Machine Learning", Pearson Education, 2005.
4. David Abramson, Rajkumar Bunya and Jonathan Giddy, "An Architecture for Resource Management and Scheduling system in Global computational Grid", HPC Asia 2000,China,IEEE CS Press,USA,2000.
5. Michael J. Quinn, "Parallel Computing: Theory and Practices", TMH, Second Edition, 2002.
6. Andrew S. Tanenbaum, "Distributed Systems: Principles and Paradigms", PHI, Second Edition, 2002.
7. Daniel A. Menasce and Emillano Casalicchiop, "Quality of Service Aspects and Metric in Grid Computing", IEEE Computer Society, 2004.
8. Andrew S. Tanenbaum "Modern Operating Systems", PHI, Second Edition, 2004.
9. J.H. Abawajy, "Automatic Job Scheduling Policy for Grid Computing", Springer-Verlag Berlin Heidelberg, 2005.
10. Keqin Li, "Job Scheduling for Grid Computing on Meta-computers", IEEE Computer Society, 2005.
11. Kaufmann, Perlman and Speciner, "Network Security", PHI, Second Edition, 2004.
12. A. Silberschatz, G. Ganges and Peter Galvin, "Operating Systems Concepts", Wiley & Wiley, Sixth Edition, 2006.
13. Behrouz A. Forouzan "Data communication and Networking", PHI, Fifth Edition, 2005.
14. Andrew S. Tanenbaum "Computer Networks", PHI, Second Edition, 2004.
15. R.Buyya, D.Abramson and S.Venugopal, "The Grid Economic," Proc. IEEE, vo.93, no.3, pp.698-714, Mar.2005.

16. Xian-He Sun & Ming Wu "QoS of Grid Computing: Resource sharing", The Sixth International Conference on Grid cooperative Computing IEEE 2007.
17. Luca Valcarenghi "QoS in Global Grid Computing", Proceeding of the 13<sup>th</sup> Symposium on High Performance Interconnects IEEE 2005.
18. I. Foaster and C. Kesselman, Grid: Blueprint for Computing Infrastructure, San Mateo, A Morgan Kaufmann, ISBN July 1998.
19. R. Stienmetz and K Wehrley "P2P Systems and Applications LNCS 3485" pp 193-207 2005, Springer –Verlag Berlin Heidelberg 2005.
20. J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut, "Quality of service for workflows and web service processes," J. Web Semantics, vol. 1, pp. 281–308, 2004.
21. Wei-Neng Cheng "Ant Colony Optimization Approach to Grid work Flow scheduling Problem with various QoS Requirements," IEEE Trans Jan 2009.
22. Syd Chapman, Alistair Dunlop, Peter Henderson and Steven Newhouse "OMII Grid Security Technology Overview", Open Middleware Infrastructure Institute University of Southampton UK, July 2005.
23. Butler, R. Engert, D.Foster, I Kesslman ,Tuckie ,S. Walmer and J. Welch " VA National authentication Infrastructure", IEEE Computer 33(12) 60-66 2000.
24. Charlie Kaufman, Radian Perlman and Mike Speciner, "Network Security", Pearson Education, 2004.
25. Rich E. and Knight "Artificial Intelligence" Second Edition TMH 1991.
26. R.Buyya, D.Abramson and Jonathan Giddy "An Economy Driven Resource Management Architecture for Global Computational Power Grids", IEEE, vo.93, no.3, pp.698-714, Mar.2005.
27. N. J. Nilsson "Artificial Intelligence New Synthetic Approach", Morgan Kaufmann Publishers Inc., 1998.
28. Russell and Novel "Artificial Intelligence, a Modern Approach", Pearson Education Inc. Third Edition 1995.
29. Michael Paul and team "Grid Services Programming and Applications" IBM Cooperation May 2004.
30. J.N. Sharma "Operation Research," Macmillan Publication Third Edition 2004.

31. Gupta and V. Kapoor "Mathematical Statistics" S. Chanda Publications Eleventh Edition 2003.
32. S.D. Sharma "Operation Research," Macmillan Publication 2004.
33. A. Grama, A. Gupta, V. Kumar and G. "Introduction to Parallel Computing" Addison Wesley, Second Edition, 2003.
34. A.Hamdy Taha "Operation Research: An Introduction", Pearson Education Seventh Edition 2005.
35. Robert B. Cooper "Introduction to Queuing Theory", Second Edition North Holland Publications.
36. Kanti Swarup and Gupta "Operation Research" Second Edition, PK & Man Mohan Publication 2004.
37. R.G. Dromey "How to solve it by computer", Third Edition, PHI, 2004.
38. Niklas Wirth "Data Structures + Algorithms=Programs", PHI, 2003.
39. E.Balaguruswamy "Object Oriented Programming using C++", TMH, Third Edition, 2006.
40. B. Lipman & J. Lasoie "C++ Primer" TMH, Third Edition, 2004.
41. B. Stroustrup "Programming Principles and Practices using C++", TMH, Third Edition.
42. Dennis Ritchie and B. Kernighan "C Programming Language", PHI, 2002.
43. E.Balaguruswamy "Programming in C", TMH, Third Edition, 2006.
44. Pradeep .K. Sinha "Distributed Operating Systems Concepts and Design" PHI, 2005.
45. Zbigruiew Michalewicz "GeneticAlgorithms+Data Structures=Evolution Programs", Springer-Verlag Berlin Heidelberg, 1992.
46. Aaron M. Tanenbaum "Data Structures using C" PHI, Second Edition, 2004.
47. Kenneth E. Kendall and Julie E. Kendall "System Analysis and Design", Pearson Education, Third Edition, 2003.
48. Jaraskiram "Grid Computing" Pearson Education, Third Edition, 2005.
49. Ian Foster & Karl Kesselman "Grid 2: Blueprint for New Grid Computing Infrastructure", Morgan Kaufmann Publishers Inc. San Francisco, CA, USA an Imprint of Elsevier, Second Edition, 2003.

50. Fran Berman, Geoffrey Fox, Anthony J.G.Hey, "Grid Computing: Making the Global Infrastructure a reality", John Wiley & Sons, 2003.
51. Joshy Joseph and Craig Fellenstein, "Grid Computing", Pearson Education, 2003.
52. P. Henry Winston "Artificial Intelligence" Pearson Education, Third Edition, 2004.
53. Kalyanmoy Deb "Multi-Objective Optimization using Evolutionary Algorithms" John Willey & Sons First Edition pp-81.
54. Pawan K. Tiwari "Optimizing QoS Parameters using ACO" M.Tech Dissertation SC&SS JNU New Delhi India.
55. Khalid Sayood "Introduction to Data Compression", Morgan Kaufmann Publishers an Imprint of Elsevier, Third Edition, 2006.
56. Ranjan Bose "Information Theory, Coding and Cryptography", Tata McGraw-Hill Companies, Second Edition, 2010.
57. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein "Introduction to Algorithms" PHI, Second Edition, 2001.
58. Andrew S. Tanenbaum and Wood Hull "Operating Systems: Design and Implementation", Pearson Education, Third Edition, 2005.
59. Andrew S. Tanenbaum "Data Structures using C and C++" PHI, Second Edition, 2004.
60. Frederic Margoles, Jie Pan, Kiat-An Tan and Abhinit Kumar "Introduction to grid computing" CRC press, 2009.

