

# **OPTIMIZING QoS PARAMETERS IN COMPUTATIONAL GRID USING ACO**

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

**MASTER OF TECHNOLOGY**

**In**

**COMPUTER SCIENCE AND TECHNOLOGY**

**By**

**Pawan Kumar Tiwari**

**Enrollment No.08/10/MT/22**

**Under the Supervision of**

**Dr. D.P. Vidyarthi**



**SCHOOL OF COMPUTER & SYSTEMS SCIENCES**

**JAWAHARLAL NEHRU UNIVERSITY**

**NEW DELHI- 110067, INDIA**

**JULY 2010**



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

---

**Certificate**

This is certify that the dissertation entitled “*Optimizing QoS Parameters in Computational Grid using ACO*” is being submitted by **Mr. Pawan Kumar Tiwari** to the **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



**SCHOOL OF COMPUTER & SYSTEMS SCIENCES  
JAWAHARLAL NEHRU UNIVERSITY  
NEW DELHI-110067, INDIA**

**DECLARATION**

I hereby declare that the dissertation work entitled “*Optimizing QoS Parameters in Computational Grid using ACO*” 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 by me elsewhere, in part or full for the award of any other degree or diploma.

**Pawan Kumar Tiwari**

Enroll No. 08/10/MT/21

M.Tech (2008-2010)

SC & SS, JNU

New Delhi - 110067

# For

Papa

*Even though you left this mortal world,  
I still feel your presence and that you stood behind me whenever I needed you the most.  
It is impossible to forget your love, care and your dreams for us all. You were, you are  
and you will be an inspiration to me forever.*

Mom

*Who sacrifices her life for her children while nurturing us and incorporating values of  
life and responsibilities towards others.*

Brother & Sisters

*We should intact in the bright and dark period of our lives. You were and are the best  
friends in whom I could always confide and count on.*

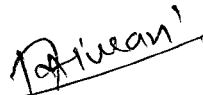
## Acknowledgement

First and foremost, I express my deep sense of gratitude to my supervisor **Dr. D.P. Vidyarthi** for giving me an opportunity to work with him. His invaluable guidance has not only helped to complete me my dissertation work, but also enabled me to develop understanding of the interesting subjects like parallel and distributed systems, graph theory and grid computing. I am highly indebted to him for his concerns and putting extra effort and time to grown me despite of his busy schedule.

I would like to express my thanks to Dean and to all the teachers of the SC&SS JNU, in support to pursue my work in the School. I would specially like to thank Dr. Zahid Raza for their encouragement and inspiration throughout the research work. I thank Mr. Sushil Kumar Dohre for providing me the hostel facility etc., at their personal level during M.tech. first year. 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.

I would like to thanks my friends Shiv Prakash, Pushpendra, Gyani Umesh, Hanuman, Md. Tanveer, Akshansh, and Md. Anbar for their unselfish help whenever I was needed. In the last one year life in hostel become joyous because of friends in the corridor.

And last but not the least, I thank God for giving me courageous and loving mother who encouraged me all the times. I immensely thank my mother for her confidence in me.

  
Pawan Kumar Tiwari

# Table of Contents

Certificate.....	ii
Declaration.....	iii
Acknowledgement.....	iv
Contents.....	v
List of Figures.....	viii
List of Tables.....	x
Abbreviations.....	xi
Abstract.....	xiii
<b>Chapter 1 Introduction.....</b>	<b>1</b>
1.1 Computational Grid.....	5
1.2 QoS Parameters in Computational Grid.....	6
1.3 ACO.....	9
1.4 ACO as a Soft Computing Tool.....	11
1.5 Organization of Dissertation.....	16
<b>Chapter 2 Computational Grid.....</b>	<b>18</b>
2.1 Distributed Systems.....	18
2.2 Grid computing.....	22
2.3 Types of Grid.....	24
2.3.1 Computational Grid.....	26
2.3.2 Data Grid.....	26

2.4	Limitations of Grid.....	27
2.5	Issues in Computational Grid.....	27
2.6	The Problem.....	31
2.7	Concluding Remarks.....	34
 <b>Chapter 3 Ant Colony Optimization .....</b>		<b>35</b>
3.1	ACO.....	35
3.1.1	AS.....	36
3.1.2	ACS.....	37
3.2	Data Structure for AS and ACS in TSP .....	40
3.2.1	Psuedo Code for ACO.....	41
3.2.2	Terminating Condition of ACO.....	42
3.3	Important Steps to Solve a Problem by ACO .....	42
3.4	Challenges and Future Scope of Research .....	43
 <b>Chapter 4 Proposed Model.....</b>		<b>44</b>
4.1	QoS Parameters.....	44
4.2	Fitness Function.....	44
4.3	The Model.....	48
4.3.1	The ACO Algorithm for Computational Grid.....	50
4.4	Concluding Remarks.....	51
 <b>Chapter 5 Experimental Evaluation.....</b>		<b>52</b>

5.1 Parallel Tasks Without Communication Cost.....	53
5.2. Observation With Communication Cost.....	63
5.3 Comparisons between ACO Based Model and GA Based Model .....	77
5.4 Concluding Remark.....	79
<b>Chapter 6 Conclusion and Future Scope.....</b>	<b>80</b>
6.1 Conclusion.....	80
6.2 Future Scope.....	81
<b>References.....</b>	<b>82</b>



## List of Figures

<b>Fig. 2.1</b> Architecture of a Multi-processor System.....	20
<b>Fig. 2.2</b> Architecture of a Multi-computer System.....	21
<b>Fig. 2.3</b> Architecture of a Distributed Computing System .....	21
<b>Fig. 2.4</b> Cluster Architecture.....	22
<b>Fig. 2.5</b> Architecture of Grid.....	25
<b>Fig. 2.6</b> Grid Scheduling Architecture.....	32
<b>Fig. 2.7</b> Queuing Architecture of Grid.....	33
<b>Fig. 2.8</b> Workflow Management.....	34
<b>Fig. 5.1.1</b> Turnaround time Observation Graph ( $l=50$ ).....	54
<b>Fig. 5.1.2</b> Turnaround time Observation Graph ( $l=60$ ).....	54
<b>Fig. 5.1.3</b> Turnaround time Observation Graph ( $l=70$ ).....	55
<b>Fig. 5.1.4</b> Turnaround time Observation Graph ( $l=80$ ).....	55
<b>Fig. 5.1.5</b> Turnaround time Observation Graph ( $l=160$ ).....	56
<b>Fig. 5.1.6</b> Turnaround time Observation Graph ( $l=250$ ).....	56
<b>Fig. 5.1.7</b> Turnaround time Observation Graph ( $l=300$ ).....	57
<b>Fig. 5.1.8</b> TAT Comparison Graph with Different Number of Tasks (I).....	58
<b>Fig. 5.1.9</b> TAT Comparison Graph with Different Number of Tasks (II).....	58
<b>Fig. 5.1.10</b> TAT Comparison Graph with Different Number of Tasks (III).....	58
<b>Fig. 5.1.11</b> Comparison of Min TAT with Increase in Number of task.....	59
<b>Fig. 5.1.12</b> Comparison of Speedup with Increase in Number of task.....	59
<b>Fig. 5.1.13</b> Turnaround time Observation Graph ( $\bar{m}=60$ ).....	60
<b>Fig. 5.1.14</b> Turnaround time Observation Graph ( $m=70$ ).....	60
<b>Fig. 5.1.15</b> Turnaround Time Observation Graph ( $m=100$ ).....	61
<b>Fig. 5.1.16</b> TAT Comparison Graph with Different Number of Machine.....	62
<b>Fig. 5.1.17</b> Speedup Graph with Increased Number of Machine.....	62

<b>Fig. 5.1.18</b> Min TAT with Different Number of Machine.....	63
<b>Fig. 5.2.1</b> TAT Observation Graph (UMR=15, MHD=4).....	63
<b>Fig. 5.2.2</b> TAT Observation Graph (UMR=30,MHD=10).....	64
<b>Fig. 5.2.3</b> TAT Observation Graph (UMR=80, MHD=15).....	65
<b>Fig. 5.2.4</b> Comparison Graph of Turnaround Time .....	66
<b>Fig. 5.2.5</b> TAT Observation Graph ( $l=50$ , U MR=30, MHD=6).....	67
<b>Fig. 5.2.6</b> TAT Observation Graph ( $l=60$ , UMR=30,MHD=6).....	67
<b>Fig. 5.2.7</b> TAT Observation Graph ( $l=70$ , UMR=30,MHD=6).....	68
<b>Fig. 5.2.8</b> TAT Observation Graph ( $l=100$ ,UMR=30,MHD=6).....	69
<b>Fig. 5.2.9</b> TAT Observation Graph ( $l=150$ , UMR=30,MHD=6).....	69
<b>Fig. 5.2.10</b> TAT Observation Graph ( $m=60$ ,UMR=15,MHD=4).....	71
<b>Fig. 5.2.11</b> TAT Observation Graph ( $m=70$ ,UMR=15, MHD=4).....	71
<b>Fig. 5.2.12</b> TAT Observation Graph ( $m=80$ , UMR=15,MHD=4).....	72
<b>Fig. 5.2.13</b> TAT Observation Graph ( $m=90$ , UMR=15,MHD=4).....	72
<b>Fig. 5.2.14</b> Speedup and Min TAT with Increased Number of Machine.....	74
<b>Fig. 5.2.15</b> TAT Observation Graph with Increased Range of Tasks.....	75
<b>Fig. 5.2.16</b> TAT Observation Graph.....	76
<b>Fig. 5.2.17</b> TAT Observation Graph with Increased Arrival Rate.....	76
<b>Fig. 5.2.18</b> Comparison Graph of TAT with Increased Arrival Rate.....	77
<b>Fig. 5.3.1</b> Comparison Graph of TAT with ACO and GA (I).....	78
<b>Fig. 5.3.2</b> Comparison Graph of TAT with ACO and GA (II).....	78

## **List of Tables**

<b>Table 5.1</b> Description of variables used in program.....	53
<b>Table 5.1.1</b> Comparison of TAT and Speedup without Communication .....	61
<b>Table 5.2.1</b> Comparison Table of Task with Communication in 50 Machine.....	70
<b>Table 5.2.2</b> Comparison Table with different Number of machine.....	73

## Abbreviations

<b>ACO</b>	Ant Colony Optimization
<b>ACS</b>	Ant Colony System
<b>AS</b>	Ant System
<b>CERN</b>	European Centre for Nuclear Research
<b>CMS</b>	Compact Muon Solenoid
<b>COP</b>	Combinatorial Optimization Problem
<b>CPU</b>	Central Processing Unit
<b>DB</b>	Data Base
<b>DCS</b>	Distributed Computing System
<b>FCFS</b>	First Come First Served
<b>FIFO</b>	First In First Out
<b>GA</b>	Genetic Algorithm
<b>LHC</b>	Large Hadron Collider
<b>LIFO</b>	Last In First Out
<b>MBPS</b>	Million Bits per Second
<b>MHD</b>	Maximum Hamming Distance
<b>MIPS</b>	Million Instructions per Second
<b>MPI</b>	Message Passing Interface
<b>M/M/1</b>	Single Server Multiple Client
<b>M/M/S</b>	Multiple Server Multiple Client
<b>NUMA</b>	Non Uniform Memory Access
<b>OS</b>	Operating System
<b>OSGA</b>	Open Service Grid Architecture
<b>QoS</b>	Quality of Service
<b>RAM</b>	Risk Assessment Model

<b>RB</b>	Resource Broker
<b>RPC</b>	Remote Procedure Call
<b>SJF</b>	Shortest Job First
<b>SRTF</b>	Shortest Remaining Time First
<b>SSI</b>	Single System Image
<b>TAT</b>	Turnaround Time
<b>TSP</b>	Traveling Salesman Problem
<b>UMA</b>	Uniform Memory Access
<b>UMR</b>	Upper Limit of Message Range
<b>VO</b>	Virtual Organizations.

## **Abstract**

Computational Grids are a new trend in distributed computing systems, which allows the sharing of geographically distributed resources in an efficient, reliable and secure manner as well as extended the boundaries of what we perceive as distributed computing. It supports both wide area parallel and distributed computing and many more issues. Uses of Computational Grids is growing very rapidly in distributed heterogeneous computing environment, for utilizing and sharing large scale resources to solve complex intensive scientific and industrial problems.

Grid computing is gaining more significance in the high performance computing world. Basically this concepts leads to provide the solution of complicated large, complex and intensive scientific problems. There are many constraints associated with the computational grid like job scheduling, resource management, information service, information security, routing, fault tolerance and many more.

Scheduling of the job on proper grid nodes with the appropriate resources is a NP-class problem even in already existed parallel and distributed computing systems. Scheduling problem adds a further dimensionality of complexity in this new platform since the capability and availability of the resources varies dynamically in the computational grid. We, in our proposed work, have applied a metaheuristic technique in particular Ant colony optimization to solve this NP-class problem.

This dissertation includes brief discussion about Grid and ACO technique. The proposed work presented a model for scheduling problem in computational Grid and also formulated the objective function as well as fitness function in a nice mathematical form for the turnaround time of the submitted job and speedup achieved by the grid. The proposed formula supports two different type of scenario. One in which different tasks of the job are parallel and not interactive and in the other one interactive modules of the job has been considered. The model focuses on problem solving using Ant Colony

Optimization technique. For optimizing QoS parameters (turnaround time and speedup) simulation of the proposed model has been done in matlab 8.4 by writing the program for scheduling by ACO. For analyzing the results and extracting the conclusion of our proposed model different graphs has been plotted for both types of job via varying the different scenarios in computational grid perspective e.g. increasing the number of resources or workload (i.e. increasing the arrival rates) as well as in user perspective like increasing the number or size of tasks in the submitted job.

The dissertation also compares the result with one improvement heuristic technique (GA). We found that the ACO not only gives the better solution in comparison to GA but, also converges in lesser no of iteration since started solution itself is very good due to the constructive and decision based policy adopted by ACO.

# Chapter 1

## Introduction

The World Wide Web (www) provides seamless access to information that is stored in millions of different geographical locations. Whereas, Grid is an emerging infrastructure that provides seamless access to computing power and data storage capacity distributed over the infrastructure across the globe. In fact, Computational grids are on its way to become just as natural as electrical grid [IFC 1998].

The objectives of parallel computing were to minimize the job/task execution time, and at the same time maximize the reliability and throughput of the system [DFL 1999]. Distributed Computing system was the integration of already existing computing systems which had the objective as resource sharing and cooperative engineering apart from exploring the parallel computing. 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 is because of various reasons. Reasons may be affordability or availability of the resources. A possible solution for this is to integrate the computing resources available at various places to enable authorized users to use it in a well-defined transparent manner [AT 2002, DF 1989]. Same concept is applied in *Grid*, a technology that is one step forward to that of the distributed systems [FBG 2003].

Scheduling over the grid is altogether different from the scheduling of a uniprocessor system which has a single objective of utilizing CPU maximally by keeping it busy as much as possible [ZSJ 2006], however the grid has inherited the concept of scheduling from uniprocessor system. Though it incorporates all the aspect of challenges and constraint that occurs in the distributed environment e.g. availability of resources, security and reliability as well as other user perspective like deadline and cost and many others. Often in grid system two types of scheduler



are prescribed; one is local and the other is global scheduler. The second one tries to meet the requirement of the different users and then send the job to different clusters by keeping in mind the load balancing factor of different clusters [JYR 2005, YYX 2006]. Local scheduler schedules the job/tasks on different processors by adopting some scheduling policy taking into consideration the local factors at instantaneous time. Scheduling over the grid is a well known NP class problem having a very huge search space of possible solution [MGD 1979]. To find the near optimal solution there are many soft computing technique that have been proposed in the literature. Thus, there is a possibility of applying soft-computing techniques to optimize characteristic parameters of user preference keeping different constraints in minds. In literature different models have been proposed for solving this problem. We, in our work, have considered the finishing time of the job to be optimized in a computational grid by using Ant Colony Optimization (ACO) based scheduling policy. We have proposed two fitness functions for two totally different scenario; one is applicable when there is no network congestion and the other in which we are considering network bandwidth and load factor. 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 by considering existing modules over the node, load on the link connecting the nodes and job's characteristics in form of cost<sup>2</sup> of communication incurred due to the interactive modules have been accounted for in our quantification.

A scheduling model using ACO has been proposed. ACO, a Soft-Computing technique, inspired by behavior of real ants was proposed by Dorigo [MDV 1991]. It can provide a better optimal solution for all combinatorial optimization problems in comparison to other but the mapping of the parameters of the techniques with the tuned constraint of the problem is a tough task.

In the following sections, we will discuss briefly about the grids, motivation from other distributed systems, types, requirement and its future. After that we will

introduce Ant Colony Optimization and its applications. Finally, the organization of the dissertation is presented.

Grid is an example of distributed systems, so to understand it we should introduce the infrastructure and development of other distributed systems like railroads, telephone, telegraph, power and electricity. These are few examples of distributed systems that have different end users intermediate reservoirs and created hundreds of millions of dollars in today's world market. We have seen that success of any one of the distributed systems depends on its infrastructure, setupcast, and its overall impact on the society in terms of many thing , that may be what facilities are being provided and at which cost? Same is also true for grids. If nobody is going to use the grid it will be a failure. So, on hand we are improving technologies and its architecture and on the other way round we are searching where it is really needed. Some well known areas where it can help are as follows [IFC 2003]:

1. Computational Scientists and Engineers needs the grid
2. Experimental Scientists need the grid
3. Corporation needs the grid
4. The environment needs the grid
5. Training and education need the grid
6. Nations and states need the grid
7. World needs the grid
8. Consumer needs the grid.

As mentioned earlier the idea of the grid is quite similar to that of electrical grid. Computational Grid is an analogous to a power grid that provides consistent, pervasive, dependable, transparent access to electricity irrespective of its source. End users just use the computational power, but don't know (or care) where from the computational energy comes. Allowing people, without any knowledge, to share advanced things like CPU time, storage and algorithms just by plugging in their computer-device forces the service to be simple. Things like how the technology works and where the resources are located must be hidden from the end users. Many research projects requires huge CPU time,

some requires a lot of memory and some projects need the ability to communicate in real time. Today super computers are not enough to solve those needs even if they have the capacity, it would not be economically justifiable to use these resources.

Computational grids are the solution to all these problems and many more. They offer a convenient way to connect many devices (e.g., processors, memory and I/O-devices) so that end users are able, if allowed, to use combined devices computational powers for a certain period of time. For example, if a researcher need to make some huge CPU consuming calculations, he could occasionally borrow CPU time from a grid on much lower cost than borrowing the time on super-computer. A grid could be created in all environments where end users have a computer with memory and CPU. Computers are often in idle state and over time they only use 5% of their capacity. Even in some organizations servers are also often idle. Hence a lot of computational power goes waste. On a computational grid each end user would be able to enjoy a lot of computational power that otherwise would have gone waste.

Grid can be classified into the following categories on the basis of their uses [IFC 1998, IFC 2003, J 2005].

- Scavenging Grid: In this type of grid system the grid management node spontaneously gets the report from each individual machine as soon as it becomes idle. Grid management node allocates a new job on this machine accordingly. Scavenging is done in a way that normal user does not know it.
- Data Grid: It provides an infrastructure to support data storage, their discovery, handling and manipulation of large volume of data actually stored in various heterogeneous data bases and file system.
- Network Grid: In this type of grid each node-work as a data router between two communication points for providing fault tolerant and high-performance communication services.
- Utility Grid: In this type of grid not only data and computation cycles are shared but it will also provide the facility to share software as well as any resource.

- **Storage Grid:** This type of grid is used for providing user to large storage space. One of the examples is Amazon S3.
- **Equipment Grid:** Grids which were used to access physical resources are coming under this category. One of the examples is e-star project which shares astronomical telescopes.
- **E-Science Grid:** CMS is a high energy physics detector planned for the LHC at CERN (the European centre for nuclear research) in Switzerland. It can read the data from the highest proton –proton collision. Data from this collision will shed light on many fundamental scientific issues like definitive research about higgs particle, possible origin of mass in the universe, existence of new fundamental symmetry of nature called super symmetry, possible symmetry of new spatial dimension. The data will contain potential millions of individual elements within the detector itself. CMS is producing several petabytes of data per year itself .It is a good example of understanding the e-science grid also can be given as a example of many more.
- **Collaboration Grid:** CMS is also a good example of collaboration grid. Scientists and institutions participating in the project CMS are located throughout the world and they are not expected to live at CERN for its expected time period of 10 years. It is only possible since it contains all the facility of collaborative grid.

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

## **1.1 Computational Grid**

A computational grid deals primarily with the computational requirement 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 participant 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 resource of the grid.

A computational grid emphasizes on the allocation and execution aspect of the submitted job in order to optimize one or more characteristics of the job like minimizing the turn around, maximizing the reliability, maximizing the security or maximizing other quality of service parameters. It does so by proper load balancing of the jobs amongst the system's resources while meeting the desired objective.

## 1.2 QoS Parameter in Computational Grid

Grid is used for dynamic sharing of resources for Virtual Organization (VO). Ian Foster defines Grid as a system that coordinates distributed resources using standard, open, general purpose protocol and interfaces to deliver non trivial quality of service [IFC 2003]. According to this definition, a distributed architecture is a grid if it satisfies the following properties:

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

Evaluation of performance of different systems can be done on different scale of parameters. In the case of computing system these parameters may be **turnaround time, reliability, security, speedup** etc. Collectively all those parameters which play a major/minor role in evaluating the performance of Computational Grid in any one perspectives of user are known as quality of service parameter [LV 2005].

**Turnaround time** of a job is defined as the time taken by the job immediately after the submission upto its final execution. The first thing which has motivated the scientists to move from uniprocessor system to parallel/distributed system was to

minimize the turnaround time as much as possible for the job execution. As mentioned earlier grid is one step forward solution of distributed system towards the next generation, we can obviously expect grid scheduler to allocate the job to faster cluster/nodes (having lesser load) so that the turnaround time is reduced upto some extent. Scheduler of the grid should always keep the following point in consideration while allocating the job on a particular node.

- No of processor in the node
- Speed of the node
- Existing Workload
- Local Scheduling Policy of the node
- Degree of interaction in the Job
- Load on the network link

A fault is a mechanical or algorithmic defect that may generate an error. A fault in a system causes failure. Depending on the manner in which a failed system behaves, system failures are of two types; fail-stop failure and Byzantine failure. In fail stop, system halts once a fault is detected whereas in Byzantine failure the system continues to function but produces wrong result. To deal this type of failure is much more difficult in comparison to the first one [PS 2005] Significantly, incorrect performance of the computers may lead to several devastating effects. For obtaining the higher reliability, the fault handling mechanisms of computational grid must be designed properly to avoid faults, to tolerate faults, and to detect recover from faults. In fact it is always desired to have a reliable system that is able to digest any type of system failure (hardware/software).

Mathematical definition of **reliability** has been given as “The probability that a system works properly over a given period of time”. Reliability is the ability of the system to perform and maintain its functions in routine circumstances, as well as hostile or unexpected circumstances. During the construction of a grid, software and hardware components which are used either in the middleware structure or at the

end system, have a specified rate of failure. 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. To provide highest level of **security** in grid is a highly challengeable job since different types of works are dealt by grid simultaneously e.g. sharing data, executing the code, migrating the jobs as well as communication between them. Any compromise at any level either at the construction or at the association level can provide a fertile ground for viruses, torjan horses, and other attacks. Certificate authority is one of the important aspects of maintaining strong grid security which will check about confidentiality, authentication and authorization. Replication and integrity are the other important issues of security [LFV 2003, RBD 2005, ECR 2008].

**Authentication:** Authentication deals with the problem of verifying the identity of a user (person or program) before permitting access to the required resources. This process will prohibit the use of grid (some resources) by unauthorized users (programs) verifying the identity or password.

**Confidentiality:** Confidentiality assures that data or message transmitted is handling by only authorized users. For implementing this, we use encryption and description algorithms.

**Authorization:** A remote shared resources are protected by only allowing authorized access.

**Throughput** is defined as work accomplished in unit time by the Grid as a single unit.

**Availability** 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.

**Waiting time** is amount of time to wait for a particular process in the system. In other words waiting time for a job is estimated as the time taken by the job from its submission to the end excluding the execution. The waiting time depend on the parameters similar to turnaround time.

**Response time** is amount of time to get first response in a time sharing system. The response time depend on the parameters similar as turnaround time.

**Cost /Budget/Deadline** are the user preferred parameters. Different users have different type of constraint. These are also important parameters while scheduling the job on different machine to make the grid much more useful and attract the larger market. For example suppose Mukesh Ambani submitted his job in the grid and payed 1 million dollar then we will give the preference to this task even ignoring the throughput of whole Grid.

We have considered only two basic parameter, turnaround time and speedup in our dissertation work.

### **1.3 ACO**

Most of the social insect societies, found in the nature, behave like a distributed system. We see that in a group they completed some complicated tasks very nicely in spite of the less capability and simplicity of each individual. All the ant algorithms are based on the behaviors of real ants. Initial ant algorithm was known as Ant System (AS). ACO is one of the latest successful algorithms in the ant algorithm category. In this algorithm artificial ants work as an artificial agents and they select their new paths/tasks according to some given probabilistic rule. Selection of path by artificial ant in ACO is a stochastic constructive phenomenon that will build the solution step by step by obtaining the feedback from partial solution construction.



There are two types of heuristic algorithms [LTS 2006, TM 1997]; one is constructive heuristic and the other is improvement heuristic. Metaheuristic algorithm uses both of them at different steps in the whole process. The metaheuristic part permits the low level heuristic to obtain better solution. The controlling mechanism is done by different methods; constraining or randomizing the set of local neighbor solution or combining elements taken by different solutions. ACO metaheuristic can be applied to any combinatorial optimization problem for which a constructive heuristic can be defined. Only challenge is that in applying ACO method how to map the considered problem to a representation that can be used by artificial ants to build a solution.

The essential of ACO algorithm is the combination of prior information about the structure of a solution with posterior information about the structure of a previously obtained good solution. The first algorithm which can be classified within this framework was presented in 1991 by Dorigo [MDV 1991] and after that so many diverse variants and combination of ACO with different heuristic methods have been reported in the literature by many authors including Dorigo [MDL 1997A, MDL 1997B, TSH 2000, MDT 2006]. This is a distributed learning process and good quality solution can emerge as the result of collective interaction among the ants.

Informally, we can say that ACO is the interplay of three procedures: ConstructAntsSolutions, UpdatePheromones, and DaemonActions [MDT 2006].

ConstructsAntSolutions manages a colony of ants that concurrently and asynchronously visit adjacent states of the considered problem by moving through neighbor nodes of the problem construction graph  $G_C$ . Ants will move by applying a local stochastic decision which is dependent on the value of pheromone trails and heuristic information. Ant incrementally build solution to the COP. Path of each ant against the fitness function are tested after the completion of the whole tour and accordingly the decision of amount of pheromones deposition on the best paths should be taken.

UpdatePheromones is the process by which the pheromones trails are modified. The trail value can increase (due to deposit) or decrease (due to evaporation). The deposition of new pheromones increases the probability of choosing that paths by future ants. On the other hand evaporation implements a useful form of forgetting and thus avoids quick convergence towards a local minimum.

DaemonActions: It is used to implement centralized actions which can't be performed by single ants.

The pseudo-code of the ACO is given below.

```
ACO ( )
{
  While (terminating condition not reached)
  {
    While there is still ants in colony
    {
      For each ant, do
        Pick up a path based on local heuristic and pheromone trail
        Path is added into the partial solution
        Until the tour is completed
        Apply local decay
      }
    Evaluate the fitness of path of each ant
    Update Global deposit
    Apply evaporation
  }
}
```

#### **1.4 ACO as a Soft Computing Tool**

A straight forward approach to the solution of Combinatorial Optimization Problems (COP) would be exhaustive search, i.e. to find the best solution among all possible solution of that problem. An instance of a combinatorial optimization problem  $\pi$  is a triple  $(S, f, \Omega)$ , where  $S$  is a set of candidate solutions,  $f$  is the objectives function which assigns an objective function value  $f(s,t)$  to each candidate solution  $s \in S$ , and  $\Omega$  is a set of constraints. Parameter  $t$  indicates that the objective function and constraint can be time dependent (for example in network routing problem cost of link is proportional to the traffic). The solutions belonging to the set  $B$  which is a proper subset of  $S$  and satisfy the constraint are called feasible solution. The objective is to find a globally optimal feasible solution  $s^*$ . In the case of minimization problem we find  $s^*$  such that  $f(s^*) \leq f(s) \forall s \in B$ , where as in the case of maximization problem  $f(s^*) \geq f(s) \forall s \in B$ . ACO as a soft computing tool is used to find the closest optimal solution of many NP-complete problems in different fields like Routing (TSP, Vehicle Routing), Assignment (Quadratic assignment, Graph Coloring) Scheduling (Job Shop, Open shop, Flow shop, Total tardiness, group shop), Subset (Multiple knapsack, Max independent set, set covering, Maximum clique), Machine learning (Classification rules, Bayesian networks, Fuzzy systems) and in many others [MDT 2006].

The COP  $(S, f, \Omega)$  is mapped on a problem that can be characterized by the following list of items [MDT 2006, JCH 1998]:

- A finite set  $C = \{c_1, c_2, \dots, c_N\}$  of components is given, where  $N$  is the total no of component.
- The states of the problem are defined in terms of sequences  $x = \langle c_1, c_2, \dots, c_h \rangle$  of finite length over the element of  $C$ .
- Number of elements in a sequence is denoted by  $|x|$ .
- The set of all possible states is denoted by  $X$ . The set of solution  $S$  is a subset of  $X$ .

- A set of feasible states  $B$  which is a proper subset of  $X$ , defined via a problem dependent set that verifies that it is not impossible to complete a sequence  $x \in B$  into a solution satisfying the constraint  $\Omega$ .
- $S \subseteq X$ . And a non empty set  $S^*$  of optimal solutions, with  $S^* \subseteq B$  and  $S^* \subseteq S$ .
- A cost  $g(s,t)$  is associated with each candidate solution  $s \in S$ . In most cases  $g(s,t) \equiv f(s,t), \forall s \in \bar{S}$ , where  $\bar{S} \subseteq S$  is the set of feasible candidate solution, obtained from  $S$  via the constraint  $\Omega(t)$ .
- In some cases a cost or the estimate of a cost,  $J(x,t)$  can be associated with states other than candidate solution.
- $J(x_i,t) \leq J(x_j,t)$  whenever we add solution component  $x_j$  to a state  $x_i$ .
- $J(s,t) \equiv g(s,t)$ .

After formulating the problem this way, we construct a graph  $G_C=(C,L)$ , where  $C$ , the set of components described above serves as a node of the graph and the set  $L$  fully connects the component  $C$ . Now, we allow artificial ants to construct the solution by randomized walk on this graph, or it can move according to some policy which can take care of the constraints  $\Omega(t)$  depending upon the COP considered. In most cases we will try to construct feasible solution but some time it is necessary or beneficial to allow them to construct infeasible solution.

Depending upon the problem some time we will associate pheromone trail  $\tau_i$  with each component  $c_i$  or we can associate pheromone trail  $\tau_{ij}$  with each connections  $l_{ij}$ . It encodes the long term memory about the entire ant search process and updated by ants themselves.

We will associate heuristic information  $\eta_i$  or  $\eta_{ij}$  to the corresponding part as earlier depending upon the problem. It is the prior information about the problem instance or run time information. It may be the estimation of cost, of adding the component or connection to the solution under construction.

Each individual ant  $k$  has the following properties:

1. It exploits the construction graph  $G_C=(C, L)$  to search for optimal solutions.
2. A memory  $M^k$  will be associated to each individual ants which will be used to:
  - Build feasible solutions.
  - Compute the heuristic values  $\eta$
  - Evaluate the solution against fitness function
  - Retracing the path backward.
3. Each individual ant has a start state  $x_s^k$  and one or more termination conditions  $e^k$ .
4. It moves to a node  $j$  in its neighborhood  $N^k(x_r)$  i.e. to  $x_{r+1} = \langle x_r, j \rangle$ , if termination condition is not satisfied in the state  $x_r = \langle x_{r-1}, i \rangle$ .
5. Move of ant will be decide by probabilistic transition rule which is a function of the following components
  - Pheromone value  $\tau$
  - Heuristic information  $\eta$
  - Ants private memory storing its current state
  - Problem constraints.
6. As soon as the ant has added the component  $c_j$  to the current state, it can update the pheromone trail  $\tau$  associated with it or with the corresponding connection.
7. Ant can retrace the path in backward direction after building the solution if it is found suitable against the fitness function and updates the value of  $\tau$  of used component.
8. Each ants act concurrently and independently.
9. Good quality solution will be obtained after the interaction among the ants.

As said earlier, challenging part in ACO for solving COP is how suitably we map the COP parameters to the ACO parameter. We will illustrate it by taking the example of TSP. The TSP is a well known NP-hard COP which has attracted a very

significant amount of research [MBG 1968, MGD 1979]. It has interesting relationship with ACO because it was the application problem chosen when first Ant Algorithm (AS) was proposed [MDV 1991]. TSP is also used as a test problem for almost all ACO algorithms [MDT 2006, JYX 2008].

- TSP can be represented by a complete weighted graph  $G=(N, A)$  where  $N$  is the no cities and  $A$  being the set of arcs fully connecting the nodes. Each arch  $(i, j) \in A$  is assigned a weight  $d_{ij}$  which represents the distance between cities  $i$  and  $j$ .
- The solution of TSP is to find a minimum length Hamiltonian circuit.
- A solution to the TSP can be represented as the permutation of city indices. Since only relative order of city is important there will exist  $n$  permutations that will map to the same solution.
- Construction graph  $G_C = (C, A)$  is identical to the problem graph  $G=(N, A)$ .
- The set of component  $C$  corresponds to set of nodes  $N$  i.e.  $C=N$ . The connections correspond to the set of arcs i.e.  $L=A$ . The states of the problem are the set of all possible partial tours.
- Only constraint in TSP is that all cities should be visited and that each city is visited at most once. This can be enforced if feasible neighborhood  $M_i^k$  of ant  $k$  in city  $i$ , will contains only those cities which are still unvisited after each construction step.
- Pheromone trail  $\tau_{ij}$  will be the desirability of visiting city  $j$  after  $i$ .
- The heuristic information:  $\eta_{ij} = \frac{1}{d_{ij}}$ .
- Initially any ant can choose any city randomly.
- Solution construction terminates when all cities have been visited.

## **1.5 Organization of the Dissertation**

This dissertation has been organized as follows.

In the first chapter the impact of computational grid on our society and why the researchers are interested to improve the qualities of computational grid have been discussed. In section 2 we have elaborated different parameters affecting the quality and requirements and their brief definitions of our concerned. In section 3 and 4 there is a brief introduction of ACO and types of problems where it is successfully used.

In Chapter 2, firstly we have discussed about Computational Grid and their construction in brief. In section 1 of chapter 2, we will make a very brief discussion about the distributed system. Discussion started with Distributed Computing System followed by Multi processor, Multi computer and Compute cluster. We elaborated grid in general along-with specific discussion about computational grid. In next two sections, the type of grids used in current scenario in different Grid environment and some important issues in the computational grid in the current competitive environment is discussed. In the last section concluding remarks of chapter 2 is made.

In Chapter 3, first section mentions the historical background of Ant algorithms and the chronological order of appearances of different ACO techniques have been given. In this section a detailed discussion of two ACO techniques over TSP have been elaborated. In section 2, data structure and pseudo code of ACO technique over TSP is discussed. In section 3, important steps for solving NP-Class combinatorial optimization problem by ACO techniques is notified. In last section challenges occurred in ACO techniques is also mentioned.

In chapter 4 we have presented our proposed model and the QoS parameter addressed. In section 2 of the chapter, the fitness functions are derived and shown how it can be formulated for different type of graphs. In next two sections the

proposed model and the algorithm is described. Last section makes the concluding remarks.

In chapter 5 the experimental results of the model on varying input is detailed. A comparative study of the proposed method with another soft computing method is also done.

Chapter 6 is about the conclusion and the future work.



# Computational Grid

This chapter discusses about various types of distributed systems including grid. Also, it elaborates about types of grid and research issues in computational grid.

A computational grid is a large scale, heterogeneous collection of autonomous systems, geographically distributed and interconnected by low latency and high bandwidth networks. The sharing of computational jobs is a major application of the grid. Grid resource management provides functionality for discovery and publishing of resources as well as scheduling, submission and monitoring of jobs. However, computing resources are geographically distributed under different ownerships each having their own access policy, cost and various constraints.

Grid computing is still in the development stage, and most projects are still from academia and large IT enterprises; however, it has developed very quickly and more and more scientists are currently engaged to deal with the challenges in grid computing. Among these, improving its efficiency is imperative! The big question is: “How to make use of millions of computers world-wide, ranging from simple laptops, to clusters of computers and supercomputers connected through heterogeneous networks in an efficient, secure and reliable manner?”

The above question is a real challenge for grid computing community.

### 2.1 Distributed Systems

After 90's, advancements in technologies in the area of microelectronics and communication lead the use of interconnected multiple processors rather than single high speed processors for optimizing the cost performance ratio. Other dependant parameters have also been increased tremendously. Computer architecture which

consists of interconnected, multiple processors can be divided into two types viz., tightly coupled systems roughly referred as parallel processing systems and loosely coupled systems roughly referred as distributed computing systems. In the former one sharing is tight by means of a single clock and single memory and in the latter system; processors do not share clock and memory.

A distributed computing system is a collection of multiple processors interconnected by a communication network in which each processor has its own local memory and other peripherals and communication between processors is possible by message passing over communication network. We refer a processor and all its resources jointly as a site or machine of the distributed computing system.

There are mainly five categories in which distributed computing systems can be classified [PS 2005]:

1. Minicomputer Model: It is the extension of centralized time sharing system.
2. Workstation Model: It consists of several workstations interconnected by a communication network (via high speed LAN).
3. Workstation-Server Model: It consists of few mini computers and several workstations (most of which are diskless, but a few of which may be diskful).
4. Processor-Pool Model: In this a pool of all the computational processors are created to serve as a single computational unit.
5. Hybrid Model: It is based on the workstation server model with the addition of a pool of processors. The processors in the pool can be allocated dynamically at any time for computation whenever there is overload on workstation for better execution.

**Multi-processor system** 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 multi-processor system, it can run different processes (or different threads belonging to the same process) on different processors. . The architecture of the Multi-processor System is shown in Figure 2.1 [AT 2002].

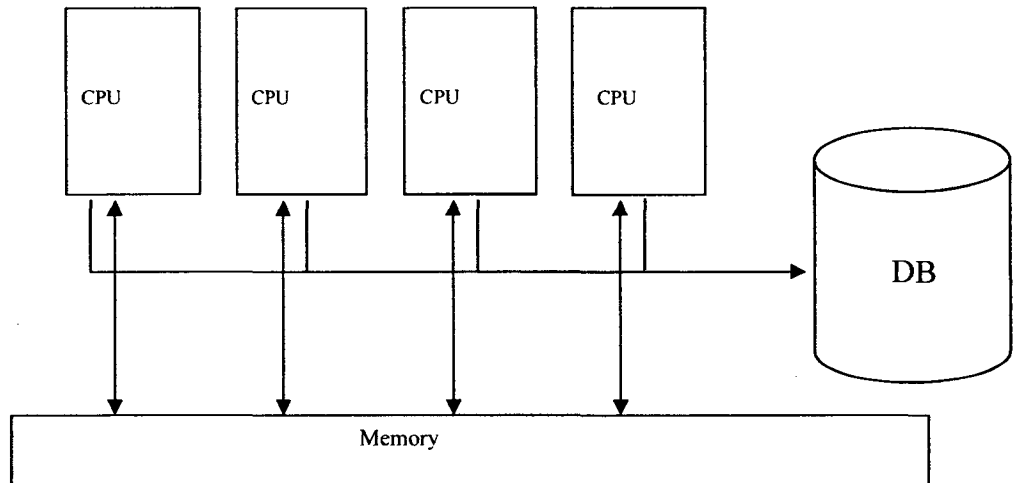


Figure 2.1 Architecture of a Multi-processor System

**Multi-computer systems** are a system made up of several independent computers interconnected by intercommunication network .Multi-computer systems can be homogeneous or heterogeneous: A homogeneous 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 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 [AT 2002, ATW 2004, MQ 2002, PS 2005].

TH-17491



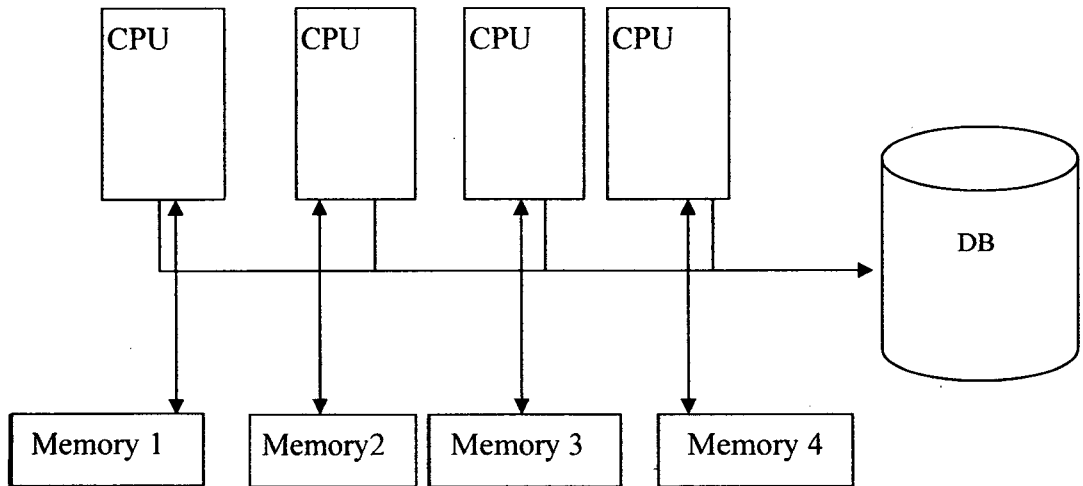


Figure 2.2 Architecture of a Multi-computer System

Distributed computing 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 [AT 2002].

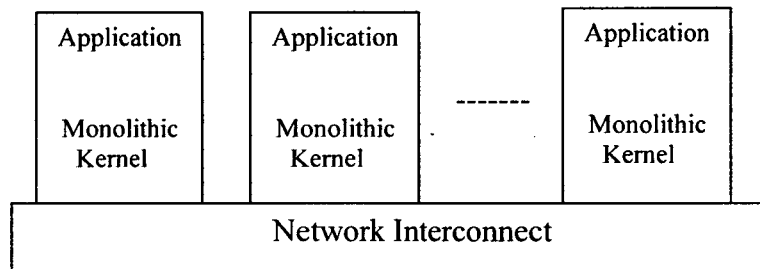


Figure 2.3 Architecture of a Distributed Computing System

003 T543 Op

Compute clusters 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 [AT 2002, ATW 2004, MQ 2002, PS 2005].

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 2.4 [AT 2002].

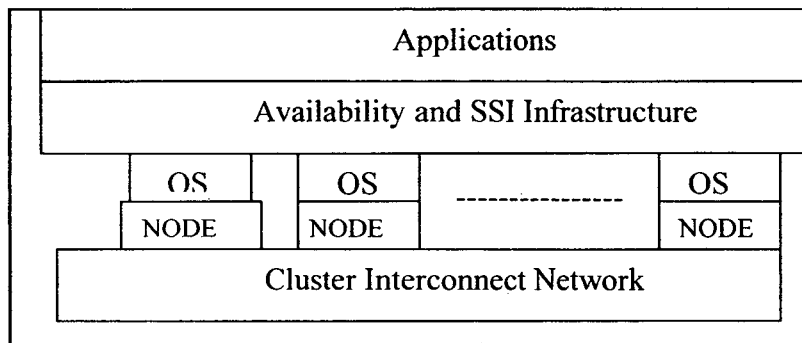


Figure 2.4 Cluster Architecture

## 2.2 Grid Computing

Grid computing is the extension of the distributed computing upto the next evolutionary level. Evolutionary steps in the field of technology on sharing the resources, availability of high bandwidth of the network and standardization of communication between heterogeneous systems by Internet have driven equally evolutionary steps in grid computing. In this section, we will discuss about the concepts, construction, ability, limitations as well as some good applications of the grid.

Logically the whole grid computing system can be considered as composed of the following three layers:

1. *Application layer*: It contains user interface and user will submit their job at this portal system.

2. *Grid resource management layer*: As soon as user submit their job, user interface portal send it to the resource management layer and resource management system will select the appropriate resources available in the grid and submit the job to it.

3. *Grid resource layer*: It contains the resources available in the grid and after completing the computation resources submit their result to the requesters.

We should always keep in mind that we are using grid computing for the fulfillment of the user requirement at different level.

- **Exploitation of unutilized processors**: First thing which we have gained from grid computing is exploitation of unutilized resources. In this system if any job is running on a machine and due to an unusual peak in activity it is busy then in this system it can run on any other machine which is idle. The simplest examples is batch job processing which take some significant time in reading the data as an input then produces output. If the amount of data is large more new ideas and planning are required to efficiently use the grid. This data can be replicated at different strategic point in the grid and then after doing this we don't need to pass the data if we want to execute on some remote machine. These replicated copies can be used as a backup in case of damage of the primary data.
- **Exploitations of extremely large amount of disk drive capacity**: This is one of the most important things which we are utilizing in the grid. Grid computing, specially a data grid is used to aggregate these unused spaces into a extremely large virtual data space.
- **Access to additional resources**: In addition to some regular resources computational grid have a list of reliable donor's and whenever there is a urgent need we can access these special equipment, software and other services for improving quality of service.

- **Balancing resource utilization:** In general a grid balances the loads on its geographically distributed federation of resources. Any unexpected load can be routed to relatively idle machine in the grid. Even in some grid enabled application partially completed jobs can also be migrated.
- **Enabling collaboration among wider audience:** With the evolution of distributed computing we have achieved this goal up-to some extent but with the emergence of grid computing we are achieving this goal up-to the next level to an even wider audience by offering a facility to different heterogeneous system to work together with the formation of a image of large virtual computing system.
- **Reserving resource facility:** To improve quality of service user can reserve some resources for certain time period after paying more cost.
- **Offering management of priorities among different projects:** In grid we are dealing instantaneously if a project finds itself in trouble by allocating it to other sites or by attaching some extra resources on this site.
- **Resource balancing, access to additional resources at the appropriate time, increased reliability and having massive parallel CPU capability are the most attractive features of the grid.**

Grid Computing can integrate computational resources from different networks or regional areas into a high computational platform and will be used to solve complex compute intensive problem. In grid computing, we utilize the idle time of thousands of CPU which are spreaded geographically at a large distance, as a result we can handle a large amount of data that would otherwise require the power of expensive supercomputers or would have been impossible to analyze.

### **2.3 Types of Grid**

The Grid is defined as a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high end computational capabilities. It became the main computing paradigm for resource intensive scientific application but it also promises to be a successful platform for

commercial environment. This trend is confirmed by an alliance of leading companies in different consortiums with the purpose to research and to develop enterprise grid solutions, and to accelerate the deployment of grid computing in enterprises.

The architecture of grid is multilayered in nature (Figure2.5)[GF, GP]. These layers are:

- Fabric layer
- Connectivity layer
- Resource layer
- Collective layer
- Application layer

Since the grid uses the internet infrastructure there is a relationship between the grid layers and the architecture of Internet Protocol (IP).

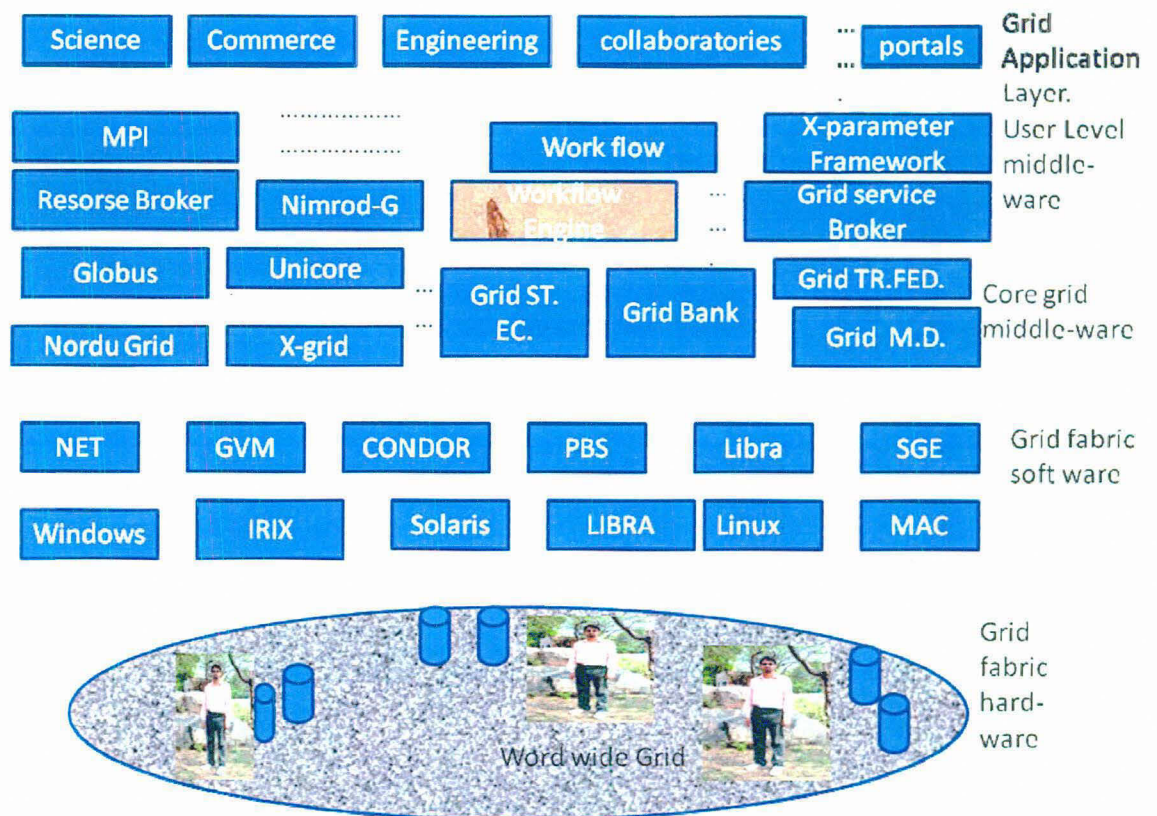


Fig 2.5 Architecture of Grid



Broadly, we identify two types of general purpose grid as follows.

### **2.3.1 Computational Grid**

The primary thing about computational grid is that how we manage the computational requirement of the job in the given system. In fact it is highly sophisticated juncture of hardware and software technologies that will provide economically bearable access to high end computational capabilities without seeing the geographical distribution of resources and users. These accesses of high end computational capabilities should also be dependable, consistent and pervasive in nature. A layman user after seeing the grid makes an idea that it is infinitely powerful source of computational power in which any job can be executed on appropriate resources. Users submit their job at one end of the given system and the middleware infrastructure from the given system decides the allocation of job on particular resources according to nature and requirement of the job without taking any further input from the user [IFC 2003, JJC 2003, J 2005, FBG 2003].

On allocating the job to the particular resources computational grid has the capabilities to take all the effective consideration of the user defined objective function into account [FNA 2006].

### **2.3.2 Data Grid**

The Grid environment has enabled us to produce more efficient computational tools which in turn has resulted more advanced research. These two factors have produced software tool and applications that can handle complex computation. 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. Apart from these big applications even applications like computational biology, remote sensing, environmental sciences, robotics etc. demands churning and managing large databases. Thus the quality of data involved

the scale of the demand and complexity of the infrastructure poses great challenge for the effectiveness of the current methods and tools. The integrating architecture that address all these problem is the data grid which provides the accelerated progress on petascale data intensive computing by enabling the integration of various approaches adopted today while encouraging the deployment of basic enabling techniques and revealing technology gaps that require further research and deployment [LFV 2003, LFC 2000].

## **2.4 Limitations of Grid**

Grid also has its own limitation. We can't expect that we will submit any job in the grid and it will return the result 1000 time faster always without adding any machine or software in the already existing system. Even there are limitations on many programs that how much we can parallelize it. It is also a considerable fact that it can affect the security of the donor's upto some extent. Also there is no practical tool for transforming arbitrary application to explore the parallel capability of the grid. Program can be written accordingly for the grid enabled application, yet automatic transformation of application is a very difficult job.

## **2.5 Issues in Computational Grid**

As we have seen there are many important issues in computational grid. These issues are categorized with respect to user's, developers, as well as administrator point of view. After emergence of OGSA (open grid service architecture) which incorporate web services into the grid architecture provides a new height to the grid technologies. In this new architecture many static and traditional algorithms failed to achieve the goal upto the desired level. In fact for providing computational resources there are many more grid service providers (GSPs) in the open market under OGSA. One of the most important issues in computational grid is that how we map each individual task to a appropriate service instances. Other important issues are security as well as reliability of the resources of different GSPs. In order to optimize the objectives of application, scheduler has to consider the tradeoff

between the grid environment and the need of the user i.e., deadline or cost into account.

Sometimes before, in some organizations for increasing reliability of high-end conventional computing system are using much expensive hardware. These hardwares are made up of chips with redundant circuits for graceful recovery in the case of hardware failure. Even machines are using duplicate processors with hot plug-ability as well as duplicate processors and cooling systems.

Grid has given an alternate approach to reliability that relies more on software technologies than expensive hardware. In this system since the different machines are geographically distributed and dispersed so power failure or data tampering or any type of inconsistency at one location does not affect the other so we are achieving high reliability in relatively less expensive manner.

So increasing reliability in an inexpensive manner in grid system by applying new software technology is also an issue in grid system. The other 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 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.

Although after submitting the job, Computational grid provides authority to the user for queering the progress of the job. But to date, as an output the computational grid produces one graph/bar showing some averaged progress metric. It is an important issue to tell exactly the user instantaneously that, which job/subjobs has failed to execute and due to which what type of error has occurred. Possible errors are as follows [LFV 2003]:

1. Programming error: Tells the user that error is due to user itself so that he can modify their program accordingly.
2. Hardware or Power failure: Suppose due to any reason after allocation of the job, hardware is failed. It is important to deal with this problem instantaneously so that it will have less negative effect on the allocation of other dependent subjobs. At least there should be a facility that tells the user that upto this level the program is executed.
3. Communication problem: Due to excessive network load at the path used in communication.
4. Excessive slowness: Due to any other type of problem etc.

From administrator point of view there are also few important issues in grid computing. Some of them are:

- Addition and removal of heterogeneous institutional resources in a dynamic virtual organization in a reliable and secure manner.
- For ensuring high level security, Certificate authority, which is based on public key encryption system, have the following responsibilities in VO systems:
  1. Issuing, removing and archiving certificates
  2. Positively identifies entities certificates for donor's as well as users.
  3. Logging activity etc.

It may be useful for developers to develop a small grid of their own so that they can use debuggers on each machine to control and watch the detailed working of applications.

## **2.6 The Problem**

Scheduling of task is an integrated part of parallel and distributed computing systems. An intensive research has been done in this area and the results of those were widely accepted. But, with the emergence of OGSA in computational grid, new scheduling models and algorithms are in demand for addressing new concerns in grid environment. The complexity of the scheduling problem increases with the size of the grid and becomes highly difficult to solve it effectively. In grid there are three main phases of scheduling. Phase one discover the set of different resources according to the specialty of the job/task. Phase two collect the information of queuing length, reliability and the speed of the node and matching have been done according to application. In third phase allocation and execution takes place.

A Global Grid scheduler, often called resource broker, acts as an interface between the user and distributed resources and hide the complexities of grid computing. It performs all the three phases of scheduling in grid environment. In spite of this it is also responsible for monitoring and tracking the progress of application execution along with adapting to the changes in the run time environment of the Grid,

variation in resource share availability, and failures. Grid environments are composed by different types of processing nodes, storage units, scientific instrumentation and information belonging to different research organizations. Each time a VO user submits a job in the grid, it has to make a reference with Resource Broker (RB). There is also a local scheduler on each site of the nodes which take care the scheduling inside the element. In our problem we have assumed that local scheduler works in FCFS manner [ZXX 2003, WCJ 2009, JYR 2005, YYX 2006, KE 2005, JA 2005].

The parameters which we have considered for scheduling problem are turnaround time and speedup. 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.

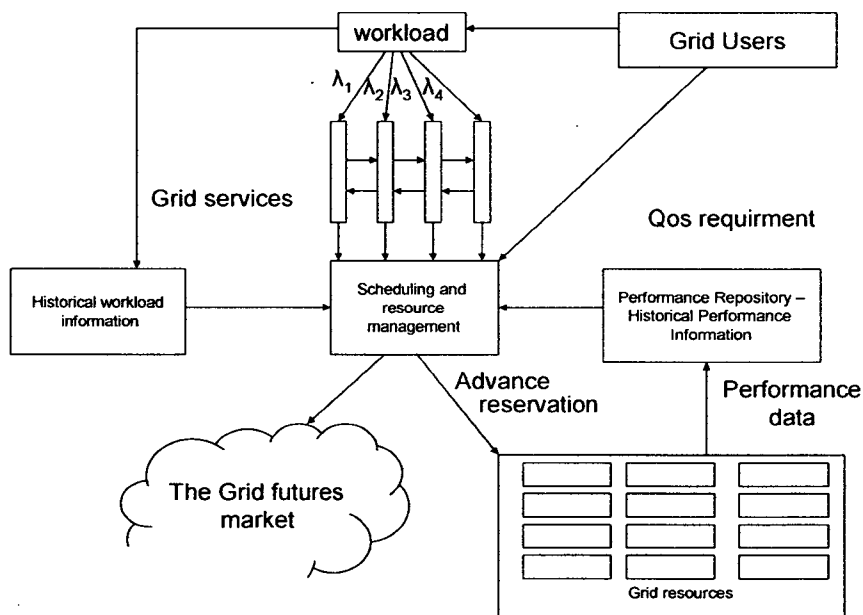


Figure 2.6 Grid Scheduling Architecture [GP].

For understanding scheduling in grid we apply Queuing Theory [RC 1981, CH 1997]. Scheduling problem under computational grid falls in category of M/M/G 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. If all the computing units have the same rate  $\mu$  then the model degenerate to the M/M/n model, and the results of multi-server queue with different service rate should be identical with M/M/n model. In our simulation part we are assuming that arrival of job/task on on each node follow a poisson distribution with arrival rate  $\lambda_i$ , their execution process follow an exponential distribution with service rate  $\mu_i$ , and the task process in each individual machine is an M/M/1 queuing system.

Assume that average arrival rate and average service rate of the system are  $\lambda_i$  and  $\mu_i$  MIPS respectively. 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 is shown below.

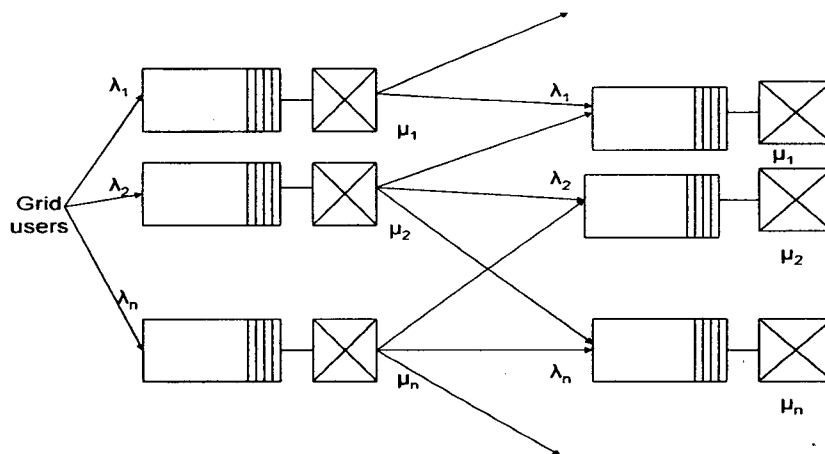


Figure 2.7 Queuing Architecture of Grid

The workflow of tasks is to be linear or hybrid or parallel. If workflow is linear then it executed in sequential otherwise we apply parallel execution model. In my



problem the workflow is parallel. The workflow management is shown in figure 2.8 [WCJ 2009].

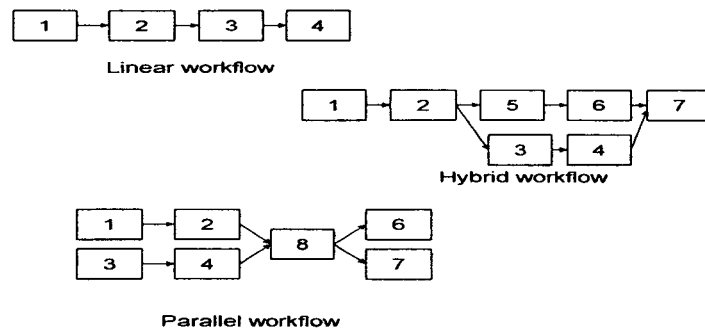


Figure 2.8 Workflow Management

## 2.7 Concluding Remarks

Concepts of grid is still evolving and most attempts to define it precisely end up excluding implementation that many would consider to be grids. Today, grid systems are still at the early stages of providing a reliable, well performing, and automatically recoverable virtual data sharing and storage. A good mathematicians and programmers are needed for giving techniques and programming for managing resources, scheduling and scavenging. OGSA is an open standard at the base of all of these future grid enhancements. Globus toolkit is a set of tools useful for building toolbox. We can take more information about this from the websites <http://www.globus.org/ogsa>.

This chapter deals thoroughly the concept of computational grid and research issues involve in this.

# Chapter 3

## Ant Colony Optimization

ACO is an emerging soft computing technique having the capability of its greater application for combinatorial problems. This chapter discusses the ACO along with its application domain with specific application to TSP problem. The challenges in applying ACO are also detailed in this chapter. The pseudo code of ACO and the example of collaboration with other metaheuristics have also been given.

Many social insects like ants, bees, birds have attracted the attention of human being due to their complex social behavior. The most noticeable examples are ant street, and the social democratic behavior of bees. The field of ant algorithm derives models for complex problems taking inspirations from real ants behaviors and then by transforming it into mathematical forms. ACO is one of the most successful algorithms of the category “ant algorithms”. Marco Dorigo gave the first ACO algorithm with the collaboration of Alberto and Vittorioo in about 90’s [MDV 1991]. Further, the field was not only strengthened by many researchers but they also applied the algorithm in variety of fields [MDT 2006].

### 3.1 ACO

ACO algorithms according to their chronological order of appearances are listed below [MDV 1991, LGM 1995, MDT 2006]:

System	Year
Ant System (AS)	1991
Elitist AS	1992
Ant-Q	1995
Max-Min AS	1996
ACS	1997
Rank-based AS	1997

Some of these are discussed in the following sections.

### 3.1.1 AS

The behavior of real ants has inspired Ant System (AS), an algorithm in which a set of artificial ants cooperate towards the solution of problem by exchanging information with pheromone deposited on graph edges. They move by stochastic local decision policy based on two parameters, trails and attractiveness. Furthermore, an ACO algorithm includes two more mechanisms: trail evaporation and, optionally, daemon actions. Trail evaporation decreases all trail values over time, in order to avoid unlimited accumulation of trails over some component. An ant is a computational agent which iteratively constructs a solution for the instances of problem to solve. Partial problem solutions are seen as states. Core of the ACO algorithm contains a loop where in each iteration, each ant move from a state  $i$  to state  $\Psi$ , which corresponds to a more complete partial solution. At each step  $\sigma$  each ant  $k$  compute a set  $A_k^\sigma(t)$  of feasible extension to its current state, and moves to one of these as per the computed probability. For each ant  $k$ , the probability  $p_{i\Psi}^k$  of moving from state  $i$  to state  $\Psi$  is given by [MDL 1997A, MDL1997B].

$$p_{i\Psi}^k = \begin{cases} \frac{\tau_{i\Psi}^\alpha + \eta_{i\Psi}^\beta}{\sum_{l \in \text{tabu}_k} (\tau_{il}^\alpha + \eta_{il}^\beta)} \dots \dots \dots \text{if } \notin \text{tabu}_k & \text{-----} \text{(A)} \\ 0 \dots \dots \dots \text{otherwise} \end{cases}$$

In eqn (A)  $\text{tabu}_k$  is the list containing all moves which are infeasible for ant  $k$  starting from instantaneous state  $i$ . While parameter  $\alpha$  and  $\beta$  specify the impact of trail and attractiveness, respectively.

After each iteration trail are updated by the following formula (B) when all agents have completed their desired work.

$$\tau_{i\psi}(t) = (1 - \gamma)\tau_{i\psi}(t-1) + \Delta\tau_{i\psi} \text{-----(B)}$$

Here  $\Delta\tau_{i\psi}$  represents the sum of contribution of all agents that used move (i $\psi$ ) to construct their solution  $\gamma$ ,  $0 \leq \gamma \leq 1$ , is the user defined parameter called evaporation coefficient.

One point should be noted here that agent contributions are proportional to the quality of solution achieved.

### 3.1.2 ACS

ACS differs from AS in three main aspects [MDL 1997 B]:

A new state transition rule provides a direct way to balance between exploration and exploitation of new edges. A new Pseudo Random Proportional Rule (PRPR) is introduced. PRPR is a compromise between the pseudo random state choice rule typically used in Q- learning and random proportional action choice rule typically used in Ant System.

The best state is chosen with probability  $q_0$  (usually fixed from 0.7 to 0.9) and with probability  $(1 - q_0)$  the next state is chosen randomly with a probability distribution based on  $\eta_{i\psi}$  and  $\tau_{i\psi}$  weighted by  $\alpha$ ,  $0 \leq \alpha \leq 1$  and  $\beta$ ,  $0 \leq \beta \leq 1$ .

$$s = \begin{cases} \arg \max_{(i\psi \in tabu_k)} \{ \tau_{i\psi}^\alpha \cdot \eta_{i\psi}^\beta \} & \text{if } q \leq q_0 \\ RWS & \text{otherwise} \end{cases}$$

Global updating rule is applied only to edge which belongs to the best ant tour.

While ants construct a solution, a local pheromone updating rule is applied.

Each ant generates a complete tour by choosing the cities according to a probabilistic state transition rule. Ants prefer to move to cities which are connected by short edges with a high amount of pheromone. Once all ants have completed their tours a *global pheromone updating rule* is applied. A fraction of the pheromone evaporates on all edges and edges that are not refreshed become less desirable. Each ant deposits an amount of pheromone on edges which belong to its tour in proportion to how short its tour was. The process is then iterated.

The state transition rule with which ant  $k$  in city  $r$  chooses to move the city  $s$  is given by

$$P_k(r,s) = \begin{cases} \frac{[\tau(r,s)]^\alpha \cdot [\eta(r,s)]^\beta}{\sum_{u \in J_k(r)} [\tau(r,u)]^\alpha \cdot [\eta(r,u)]^\beta}; & \text{if } s \in J_k(r) \\ 0 & \text{otherwise} \end{cases} \quad \text{-----(3.1)}$$

where  $\tau$  is the pheromone,  $\eta = \frac{1}{d}$  is the inverse of the distance  $d(r,s)$ ,  $J_k(r)$  is the set of cities that remains to be visited by ant  $k$  positioned on city  $r$  (to make the solution feasible).

In ant system, when all ants completed their tour the global updating rule is implemented as follows.

$$\tau(r,s) \leftarrow (1-\gamma)\tau(r,s) + \sum_{k=1}^m \Delta \tau_k(r,s) \quad \text{----- (3.2)}$$

Where,

$$\Delta \tau_k(r,s) = \begin{cases} \frac{1}{L_k} & \text{if } (r,s) \in \text{tour done by ant } k. \\ 0 & \text{otherwise} \end{cases}$$

$0 < \gamma < 1$  is the pheromone decay parameter,  $L_k$  is the length of the tour performed by ant  $k$ , and  $m$  is the number of ants. Pheromone updating is intended to allocate a greater amount of pheromone to shorter tours. This is similar to reinforcement learning in which better solution get a higher reinforcement. This is good for small TSP with (30) cities.

#### ACS State Transition rule:

An ant positioned on node  $r$  chooses the city  $s$  to move by applying the rule given by(3.3)

$$s = \begin{cases} \arg \max_{u \in J_k(r)} [\tau(r,u)]^\alpha \cdot [\eta(r,u)]^\beta & \text{if } q \leq q_0 \\ S & \text{otherwise} \end{cases} \quad \text{-----(3.3)}$$

Where  $q$  is a random number uniformly distributed in  $[0,1]$ ,  $q_0$  is a parameter ( $0 < q_0 < 1$ ) and  $S$  is a random variable selected according to the probability distribution (3.1).

### ACS Global Updating Rule

Here Global updating is performed by equation (3.4)

$$\tau(r,s) \leftarrow (1-\gamma) \cdot \tau(r,s) + \gamma \cdot \Delta\tau(r,s) \text{ -----(3.4)}$$

Where,

$$\Delta\tau(r,s) = \begin{cases} (L_{gb})^{-1}, & \text{if } (r,s) \in \text{Global best tour}; \\ 0 & \text{otherwise} \end{cases}$$

Where  $L_{gb}$  is the length of the globally best tour from the beginning of the trial. Eqn (3.4) dictates that only those edges belonging to the globally best tour will receive reinforcement in comparison to (3.2). Another Global updating rule is called iteration best denoted by  $L_{ib}$  (the length of the best tour in the current iteration of the trial). In this case edges which receive reinforcement are those belonging to the current tour of the best iteration.

### ACS Local Updating rule

While building a solution of the TSP, ants visit edges and change their pheromone level by applying the local updating rule given by the following eqn (3.5).

$$\tau(r,s) \leftarrow (1-\rho) \cdot \tau(r,s) + \rho \cdot \Delta\tau(r,s) \text{ -----(3.5)}$$

Where  $0 < \rho < 1$  is a parameter,  $\Delta\tau(r,s)$  can be defined in many ways which are inspired from different evolutionary method for improving the results.

$\Delta\tau(r,s) = \kappa \cdot \max_{z \in J_k(s)} \tau(s,z)$  which is exactly the same formula used in Q-learning,  $0 < \kappa < 1$  is a parameter,  $\Delta\tau(r,s) = \tau_0$ , where  $\tau_0$  is the initial pheromone level,

$\Delta\tau(r,s) = \tau_0 \left( 1 - \frac{z_{curr} - LB}{\bar{z} - LB} \right)$ , where  $\bar{z}$  is the average of the last  $k$  solutions

and  $LB$  is the lower bound on optimal solution cast. If  $z_{curr}$  is lower than  $\bar{z}$ , the

trail level of the last solution move is increased, otherwise it is decreased.  $\Delta\tau(r, s) = 0$ , and sometime local updating is not applied in many problems.

The role of ACS local updating rule is to shuffle the tours so that early cities in one ants tour may be explored later in other ant tour. In other word, the effect of local updating is to make the desirability of edges change dynamically; every time an ant uses edge it becomes slightly less desirable (since it loses some of its pheromone). In this way ants will make a better use of pheromone information. Without local updating all ants would search in a narrow neighborhood of the best previous tour.

### 3.2 Data Structure and pseudo code for AS and ACS in TSP

TSP was the first Combinatorial Optimization problem solved by ACO. We will explain it by taking the example of TSP.

Problem representation:

**Intercity Distances:** First, we compute the distances between each city and store them in a symmetric distance matrix with  $n^2$  entries. In fact in symmetric TSP we only need to store  $\frac{n(n-1)}{2}$  distinct distances.

**Nearest Neighbor Lists:** We will also provide it as an input for each city a list of its nearest neighbors. For each city  $i$ , let  $d_i$  be the list of distances from a city  $i$  to all cities  $j$ ,  $j=1,2,\dots,n$  and  $i \neq j$ . Sort the list according to non decreasing distances and obtained a sorted list  $ds_i$ , with the assumption that  $d_{ii}=9999$ . The  $nn\_list[i][r]$  gives the identifier of  $r^{\text{th}}$  nearest city to  $i$ . It can be constructed in  $O(n^2 \log n)$  since we have to repeat sorting algorithm over  $n-1$  cities for each cities.

**Pheromone Trails:** We store the initial pheromone value  $\tau_{ij}$  of each path  $(i,j)$  in a  $n^2$  matrix. However, we need only to store  $\frac{n(n-1)}{2}$  distinct values.

**Heuristic Information:** We store the heuristic information value  $\eta_{ij} = \frac{1}{d_{ij}}$  of each path  $(i,j)$  in a  $n^2$  matrix. However, we need only to store  $\frac{n(n-1)}{2}$  distinct values.

**Combining Pheromone and Heuristic Information:** An ant located on city  $i$  chooses the next city  $j$  with a probability proportional to  $[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta$ . This value will be used by each of  $m$  ants so we will store it in an additional matrix of order  $n^2$ .

**Representing Ants:** An ant contains at least three structures

One variable viz. `tour_length`, to store the ant's objective (fitness) function value.

An array of dimension  $n+1$  to store the ant's tour

A array of dimensional  $n$  to store the visited cities.

### 3.2.1 Pseudo code for ACO

Procedure Construct Solutions

```

{
  k=1 to Number of Ants do
  {
    for i=1 to Number of cities do
    {
      ant[k]visited[i] ← false
    }
  }
  step ← 1
  for k=1 to Number of Ants do
  {
    r ← rand{1,2,...n}
    ant[k].tour[step] ← r
    ant[k].visited[r] ← true
  }
  while (step < Number of cities)
  {
    step ← step+1
    for k=1 to Number of Ants do
    {

```



```

        AS Decisionrule (k,step)
    }
}
for k=1 to number of Ants do
{
    ant[k].tour[n+1] ← ant[k].tour[i]
    ant[k].tour_length ← compute tour length
}
}

```

The Roulette wheel scheme which is used in both ACO algorithms is described here. Procedure of AS decision rule is as follows: first, we determine the current city of the ant k. The choice of next city will be determined by RWS of evolutionary computation. For each value choice of a city j that ant k has not visited yet, construct a slice on a circular roulette wheel in proportional to their weight. After this, spun the wheel and the city to which marker points is chosen as next city. This can be done mathematically as follows:

- Summing the weight of the variable choices.
- Choose a uniform random number r from the interval 0 and sum.
- Going through the all feasible choices until the sum is greater or equal to r.
- Finally, the ant is moved to the chosen city, which is marked as visited.

### 3.2.2 Terminating condition of ACO

The possible termination condition of ACO algorithms can be as follows:

- Maximum number of iteration has reached.
- Maximum number of tour's construction can be given as a terminating condition. Stagnation behavior occurred.
- Near Optimal solution is found within a predefined distance.

### 3.3 Important Steps to Solve a Problem by ACO

There are some points that should be addressed before applying ACO. They can provide a rough guidance towards positive direction however real experience can be

obtained only after implementation which can enforce us to change the value of parameters taken initially. The key points are the following:

- Transform the problem in the form of set of components and transitions on which ant can build a solution
- A good definition of pheromone trail should be addressed and this part itself requires deep insight into the problem
- Equivalently appropriate value of heuristic information is a key point for the success of the algorithm
- For solving NP-hard COP, implement an efficient local search algorithm with ACO for best performance
- Choose the appropriate ACO algorithm according to the problem taking the previous aspect into account
- Tune the parameter of the ACO algorithm. A good starting point produces the good results for variety of problems.

### **3.4 Challenges and future scope of research**

There exist a variety of problems e.g. graph coloring, job shop problem for which other algorithms appear to be superior. It is the point of research that for which type of problem ACO can be applied and whether there is some problem with tuning parameters or not. There is no exact formula for estimating the optimal number of ants after seeing the nature and size of problem [MDL 1997B]. In recent, ACO is mixed with some other algorithm and they are giving good results for specific type of problem [JAG 2009, THZ 2010].

# Chapter 4

## The Proposed Model

The problem considered in this work is scheduling of job, consisting of tasks, to a computational grid for execution. Through scheduling the objective is to optimize the turnaround time of the job. An evolutionary technique, known as ACO, has been used for the purpose. This chapter discusses the QoS parameter, derives fitness function and deliberates over the proposed model. The algorithm to solve the problem is also elaborated. Finally this chapter is concluded.

### 4.1 QoS Parameter

We have already discussed about different QoS parameters in chapter 3. In this chapter, we concentrated on only two parameters; turnaround time and speed up, which we tried to optimize in our proposed model. Turnaround time is the time when a job is submitted for the execution in the grid and finally completes the execution. Through computational grid, the scheduler tries to allocate the job, consisting of tasks/subjobs, onto computing nodes and optimizes the turnaround time. Speed up of computational grid is defined as the ratio of the time taken by a single computer to that of the computational grid. Speedup always lies between 1 to  $n$ , where  $n$  is the number of resources available in the grid.

### 4.2 Fitness Function

In OGSA different users submit their jobs unaware of others. So there may be queue length on each site or nodes of the computational grid. Taking into consideration of queuing theory [RC 1981, CH 1997], scheduling problem in computational grid as a single system, will fall in general under M/M/G system. We are seeing it as a distributed systems and concentrated on processing of task on each individual machine and assuming that arrival of job/task on each node follow a poisson distribution with arrival rate  $\lambda_i$ , their execution process follow an

exponential distribution with service rate  $\mu_i$ , and the task process in each individual machine is an M/M/1 queuing system.

Let  $\lambda$  and  $\mu$  be the average arrival and service rate of an individual site at any given time respectively then average waiting time is given by  $\frac{\lambda}{\mu(\mu - \lambda)}$  and average waiting time plus service time is given by  $\frac{\lambda}{\mu(\mu - \lambda)} + \frac{1}{\mu}$  which after simplification gives  $\frac{1}{(\mu - \lambda)}$  known as little theorem.

Let us assume that a user submits a job which have  $\ell$  tasks, on a computational grid having  $m$  systems. Arrival and service rates of the  $i^{\text{th}}$  system is  $\lambda_i$  and  $\mu_i$  respectively. Average waiting time including service time of the  $i^{\text{th}}$  system can be given by :

$$T_i = \sum_{j=1}^{\ell} \left( \frac{\lambda_i}{\mu_i(\mu_i - \lambda_i)} + \frac{1}{\mu_i} \right) \cdot N_j \cdot \chi_{ji} \dots\dots\dots(4.1)$$

where  $\chi_{ji}$  is equal to 1 if  $j^{\text{th}}$  task is allocated on  $i^{\text{th}}$  machine otherwise 0 and  $N_j$  is the no of instructions in  $j^{\text{th}}$  task. So finishing time  $T$  of the whole job in terms of user according to this allocation will be following:

$$T = \max_{i=1}^m (T_i) \dots\dots\dots(4.2)$$

Substituting the values of  $T_i$  from eqn (4.1) into (4.2) we get the following eqn:

$$T = \max_{i=1}^m \left( \left( \sum_{j=1}^{\ell} \frac{\lambda_i}{\mu_i(\mu_i - \lambda_i)} + \frac{1}{\mu_i} \right) \cdot N_j \cdot \chi_{ji} \right) \dots\dots\dots(4.3)$$

Since our objective is to minimize finishing time so the objective function in our case will be

$$F = \text{Min}(T) \dots\dots\dots(4.4)$$

where value of T will be obtained from eqn (4.3) and minimum has taken over all possible schedule of tasks i.e. all possible values of allocation of tasks j with machine i.

Therefore fitness function for our problem can be written as:

$$L = \frac{1}{T} \dots\dots\dots(4.5)$$

According to eqn (4.5), L will be maximum for the schedule which have minimum corresponding value of T.

In the above formula we are not considering the communication cost between the different modules of the task. For this, we will provide as an input that what is the complete list of module which requires communication to others and how much bit of communications are required between them.

Let  $\lambda_{ki}$  and  $\mu_{ki}$  be the arrival rate and service rate of the network between the link of k<sup>th</sup> node and i<sup>th</sup> node of the grid then the average waiting time on this link for communication would be:

$$\frac{\lambda_{ki}}{\mu_{ki}(\mu_{ki} - \lambda_{ki})}; \text{Where, } \lambda_{ki} = \lambda_{ik}; \mu_{ki} = \mu_{ik} \dots\dots\dots(4.6)$$

So if r<sup>th</sup> task is allocated on k<sup>th</sup> node and s<sup>th</sup> task is allocated on i<sup>th</sup> node then total time on the link will be

$$\left( \frac{\lambda_{ki}}{\mu_{ki}(\mu_{ki} - \lambda_{ki})} + \frac{1}{\mu_{ki}} \right) * C_{rs} \dots\dots\dots(4.7)$$

where  $C_{rs}$  is the bit of communication between r<sup>th</sup> and s<sup>th</sup> task obtained from the communication cost matrix C,  $1 \leq r, s \leq l; 1 \leq i, k \leq m, C_{ii} = 0$ .

Total communication elapsed for all possible tasks between link of i<sup>th</sup> and k<sup>th</sup> node can be obtained as

$$\sum_{r=1}^l \sum_{s=r+1}^l \left( C_{rs} \cdot \chi_{kr} \chi_{is} \left( \frac{\lambda_{ki}}{\mu_{ki}(\mu_{ki} - \lambda_{ki})} + \frac{1}{\mu_{ki}} \right) \right) \dots\dots\dots(4.8)$$

Where  $\chi_{kr}$  assigns a value 1 if  $r^{\text{th}}$  task is assigned on machine  $k$  and 0 otherwise.  $l$  is the total number of module/tasks in the job submitted by the user.

Thus, total time  $\Omega$  taken by the computational grid for communication between the modules will be following:

$$\Omega = \underset{\substack{1 \leq i, k \leq m \\ i < k}}{\text{Max}} \left[ \sum_{r=1}^l \sum_{s=r+1}^l \left( \frac{\lambda_{ki}}{\mu_{ki}(\mu_{ki} - \lambda_{ki})} + \frac{1}{\mu_{ki}} \right) \cdot C_{rs} \cdot \chi_{kr} \chi_{is} \right] \dots \dots \dots (4.9)$$

Where  $m$  is the total number of machines in the given computational grid, and  $l$  is the total number of tasks in the submitted job.

In this case our objective function will be modified as:

$$F = \underset{\substack{1 \leq i, k \leq m \\ i < k}}{\text{Min}} [T + \Omega] \dots \dots \dots (4.10)$$

Standard fitness function according to eqn (4.10) for our problem will be

$$L = \frac{1}{T + \Omega} \dots \dots \dots (4.11)$$

According to eqn (4.11)  $L$  will be maximum for the schedule which have the minimum (execution + communication) time.

Let us consider that each node is not directly linked with each other. Further, let  $h(k, i)$  be the set which consist of the node (necessary for establishing communication between node  $k$  and  $i$ ) in an increasing order starting from node  $k$  and ends up with node  $i$ , in such a way that each predecessor in the set sequence is directly connected with its successor and let  $hd(k, i)$  be the hamming distance between  $k^{\text{th}}$  and  $i^{\text{th}}$  node. It can be written mathematically as

$$h(k, i) = \{h^0(k, i), h^1(k, i), \dots \dots \dots h^c(k, i)\}$$

Where  $c$  = hamming distance between node  $k$  and  $i$ , i.e.  $c = hd(k, i)$

$$h^0(k, i) = k; \quad h^c(k, i) = i;$$

In this case  $\Omega$  can be modified as follows:

$$\Omega = \underset{\substack{1 \leq i, k \leq m \\ i < k}}{\text{Max}} \left[ \sum_{r=1}^l \sum_{s=r+1}^l \left\{ \sum_{i=0}^{hd(k,i)-1} \left( \frac{\lambda_{h'(ki)h^{i+1}(ki)}}{\mu_{h'(ki)h^{i+1}(ki)} (\mu_{h'(ki)h^{i+1}(ki)} - \lambda_{h'(ki)h^{i+1}(ki)})} + \frac{1}{\mu_{h'(ki)h^{i+1}(ki)}} \right) \right\} \cdot C_{rs} \cdot \lambda \right] \dots \dots \dots (4.12)$$

### 4.3 The Model

Let us consider a computational grid which consists of n nodes/machines. These machines are distributed globally and situated at large distances. A user has submitted his job which consists of l parallel task and wants to finish his job in as minimum time as possible. Since in computational grid there are many jobs running on the machines, we have considered arrival rate of task coming on each machine and the service rate of the machine. According to arrival and service rate on each machine there will be queue on each machine of certain length so in the first part of our model we will allocate the tasks on different machines by applying ACO method and improve it with each iteration.

Some assumptions laid down for the model are as follows:

- Each node has a single processor
- arrival of job/task on on each node follow a poisson distribution with arrival rate  $\lambda_i$ , their execution process follow an exponential distribution with service rate  $\mu_i$ , and the task process in each individual machine is an M/M/1 queuing system.
- All the tasks in the submitted job are parallel
- We have not considered the setup time which will take place just before the scheduling process
- Arrival rate of packets on each individual machine may vary with respect to time however service rate of each node is same with respect to time

The various module of ACO for this purpose can be mapped as follows:

1. We construct the representing graph of our problem in the following notation:  
 $G=(T,N, L)$ , where  $T$  consist the set of subjobs/tasks i.e.,  $T=\{T_1, T_2, \dots, T_l\}$  and  $N$  will consist set of machines/nodes i.e.,  $N= \{1,2,\dots,n\}$  and,  $L =\{ T_{11}, T_{12}, \dots, T_{1n}, T_{21}, T_{22}, \dots, T_{2n}, \dots, T_{l1}, T_{l2}, \dots, T_{ln}\}$ , where  $L$  will denote the set of all possible move by the ant i.e.  $T_{ln}$  will denote that task  $l$  is assigned on machine  $n$ .
2. Assigning a task to a particular machine will be seen as one step move of the ant.
3. Path of a single ant will be completed when it will assign all the tasks to some machine.
4. Whenever the ant will allocate a task it will fall under allocated task, next task will be chosen from non allocated task just like in TSP from non visited cities.
5. Initially ant can take any task  $T_i$  from the set  $T$  randomly and allocate it on a machine  $m$  according to state transition rule. After allocation of the first task on machine any other task  $k$  will update the corresponding  $\eta_{kj}$  value in the heuristic matrix through the memory of ant  $s$  that one task is already allocated and decrease the desirability upto the considerable remark.

6. Initial pheromone value  $\tau_{ij} = \tau_0; \forall i, j$  where  $\tau_0 = \min_{1 \leq i \leq l} \left( \frac{T_i}{\sum_{i=1}^l T_i} \right)$ .

7. We set  $q_0 = .93$ .
8. Initially Heuristic information value  $\eta_{ij}$  of allocating the task  $i$  to machine  $j$  will be equal to  $\eta_{ij} = \frac{1}{(etm)_{ij}}. \forall i \in T, j \in N$ , where  $etm$  is the execution time of task  $i$  on machine  $j$ .
9. Each individual ant will reach to their destination i.e will construct the whole path and it can be seen as allocation of all the tasks to some machine. So there will be exactly  $l$  steps in completing the whole tour, where  $l = \text{cardinality}(T)$ .
10. Heuristic information value will be calculated in next step according to the following formula:



$\eta_{ij}(\text{step}(h)) = \frac{1}{(\text{btp})_j(\text{step}(h-1)) + (\text{etm})_{ij}}$ ; where btp is the busy time period of machine j upto step (h-1). Here we see that the changes will be reflected in only m values of  $\eta_{ij}$  out of (l.m) entries.

11. Each machine has a busy time period array. In each move of the ant out of n options only one btp of the corresponding machine will be updated which has been selected according to ACO choice rule.

$$S_i^j = \begin{cases} \arg \max_{(i| \in \text{tabu}_k)} \{ \tau_{ij} \cdot \eta_{ij} \} & \text{if } q \leq q_0 \\ RWS & \text{otherwise} \end{cases}$$

12. After each iteration calculate the fitness function for each complete tour constructed by the different ants. Update the value of pheromones which are responsible for the best tour by the amount  $\frac{1}{T_{\min}^{\text{ite}}}$  to the edges belonging to the best schedule in each iteration from the following eqn:

$$\tau_{ij} \leftarrow (1 - \gamma) \cdot \tau_{ij} + \gamma \cdot \frac{1}{T_{\min}^{\text{ite}}}; \quad 0 \leq \gamma \leq 1 .$$

#### 4.3.1 ACO Algorithm for Computational Grid

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

*ACO Algorithm ( )*

```
{
    for (i=1; i<=no_of_ iterations; i++)
    {
        For each ant do
        {
            Generate sequences of task randomly
            Ant will consider the preferences of task according to their order in
            the sequence
            Task will be allocated on the machine according to the State
            transition rule
        }
    }
}
```

```

    When tour of the ant is completed that each task has been allocated
    on the machine we calculate the fitness function
    }
    By applying sorting techniques we see that which ant provides the
    better solution
    Update Pheromones value of edges belonging to the best schedule
    }
}

```

#### **4.4 Concluding Remarks**

In this chapter, ACO is used to solve the scheduling problem of computational grid in an efficient manner. A detailed description of the matching of the different parameters of ACO has been notified with the computational grid problem in our concern. It is beyond doubts that it needs a deep insight through the problem to solve it using ACO. In most of the literature [YYX 2006, SFM 2006, SLM 2007, WCJ 2009, JAG 2009] workload on the machine have been assumed to be static i.e. already a fixed workload is assumed. They have done the experiment by providing it as a static input. In the proposed work, we are assuming queuing theory approach and derived the objective function and fitness function accordingly. In the next chapter we will observe that only after change of range of arrival rates of machine would increase or decrease the workload on the machine accordingly.

### Experimental Evaluation

After presenting the model and algorithms in the previous chapter, a large set of experiments has been conducted to study the performance of the proposed model. Different range of parameters is taken to plot the graphs between turnaround time and number of iteration. In this dissertation work only two QoS parameters; turnaround time and speedup have been considered. Different values of the input parameters are generated randomly between its lower limit and upper limit. Summary of all the observations and effect of varying the range of one parameter while keeping other constant over the turnaround time have been studied.

Two different cases, given below, have been considered broadly for various experiments.

Case 1: In this case scheduling of only parallel independent tasks have been considered. For evaluating the quality of solution of different schedules generated by different ants, fitness function (4.5) as described in chapter 4, have been used in this case.

Case 2: This case considers different dependant parallel tasks of the job submitted by the users. Thus in this routine, communication between different tasks of the job is effective. Fitness function (4.11) of chapter 4 is used for evaluating the different solutions generated by ants.

Description of different variables their notations, meanings, and prescribed range which are used in implementation of the algorithms (described in chapter (4)) have been listed below in the table 5.1.

Sr. No.	Variable Name	Meaning	Range
1	m	Number of nodes in the grid	20 to 600
2	<i>l</i>	Number of tasks in the job	50 to 1000
3	Ant	Number of artificial ants	3 to 10
4	ltn	Number of iteration	20 to 200
5	$\mu_i$ []	Speed of processors in( MIPS)	(200-900)
6.	$\lambda_i$ []	Load of processors in (MIPS)	(1-600)
7.	$\lambda_{rs}$ []	Network load in (MBPS)	60-90
8.	$\mu_{rs}$ []	network bandwidth (in MBPS)	100-120
10	Task[].size	Task size ( in Million Instructions)	(2000-5000) (100-90000)
11	$C_{rs}$ []	Message size ( in Mega Bits)	0-90
12	$\gamma$	Pheromone decay parameter	0.1-0.3
13	$q_0$	State selection threshold point	0.90 to .95

Table 5.1-Description of variables used in program

### 5.1 Parallel Task without Communication Cost

**Case 1:** This experiment considers number of machines in grid to be fixed, and number of tasks varying. Various experiments with varying tasks are as follows. Number of Machine (m) =50, Ant =3, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300 MIPS, Range of Task Size 2000-5000 Million Instructions.

### 5.1.1 Number of Tasks =50.

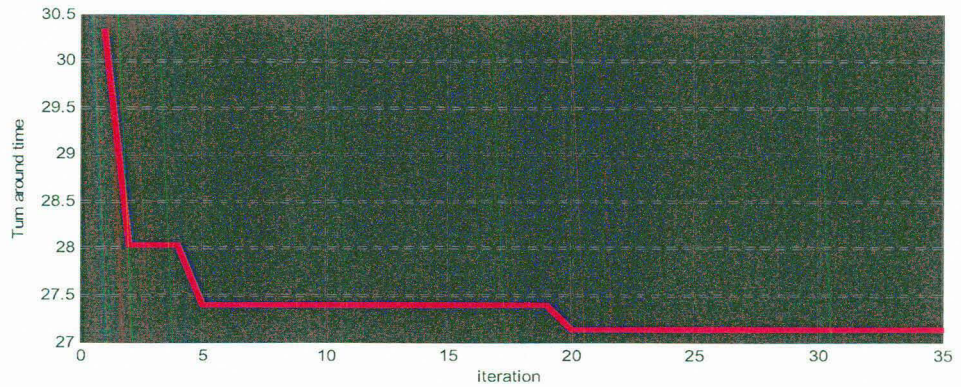


Fig. 5.1.1 Turnaround time Observation Graph ( $l=50$ ).

**OUTPUT:** Min TAT without communication=27.131131

Average TAT without communication=2.727794e+001

Speed UP without communication=2.145632e+001

### 5.1.2 With other input parameters to be same and number of Tasks =60

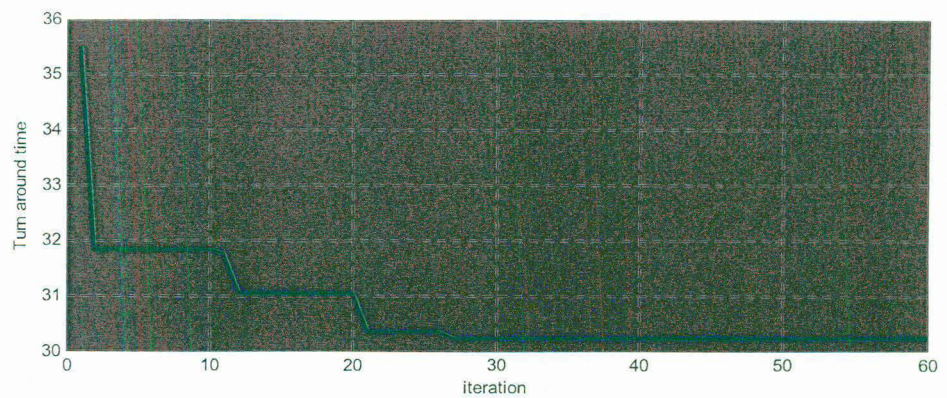


Fig. 5.1.2 Turnaround time observation Graph ( $l=60$ ).

**OUTPUT:** Min TAT without communication=30.224597

Average TAT without communication=3.071358e+001

Speed UP without communication=2.286744e+001.

### 5.1.3 Number of Tasks =70

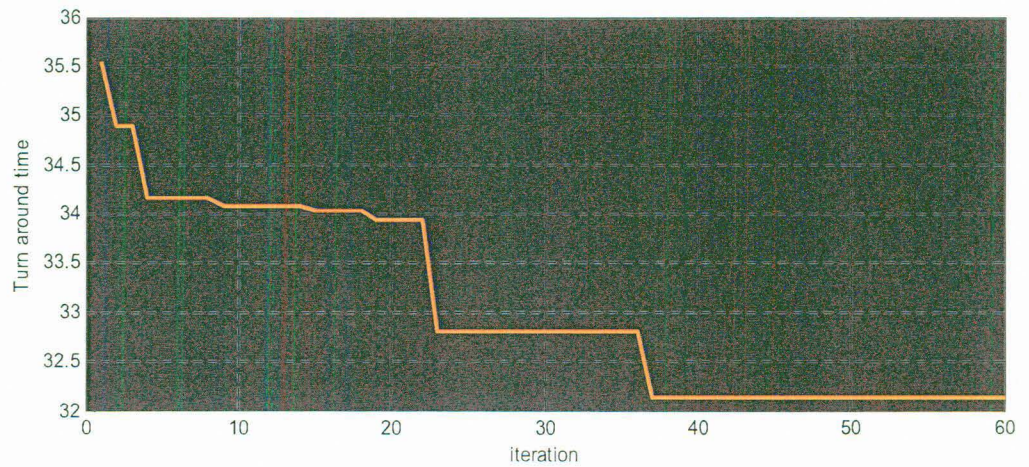


Fig. 5.1.3 Turnaround time Observation Graph ( $l=70$ ).

**OUTPUT:** Min TAT without communication=32.137871  
Average TAT without communication=3.304940e+001  
Speedup without communication=2.479312e+001

### 5.1.4 Number of Tasks =80

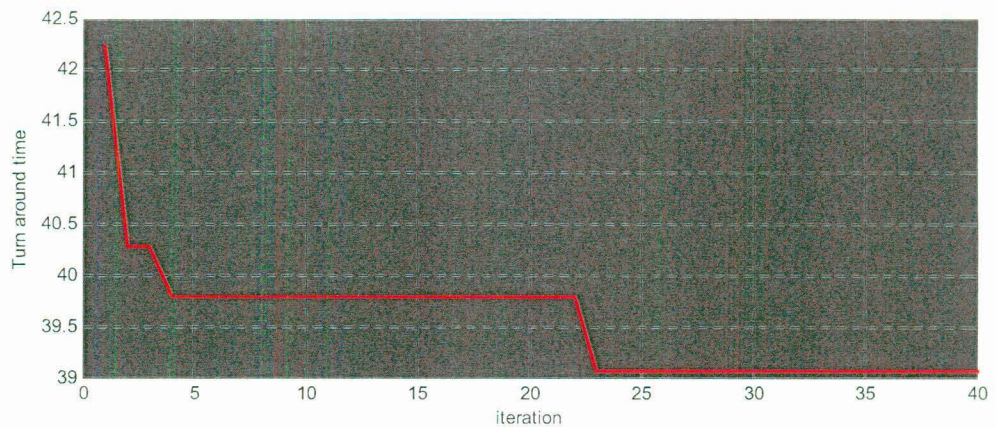


Fig. 5.1.4 Turnaround time Observation Graph ( $l=80$ ).

**OUTPUT:** Min TAT without communication=37.565280,  
Average TAT without communication=3.788680e+001  
Speedup without communication=2.471718e+001.

### 5.1.5 Number of Tasks =160

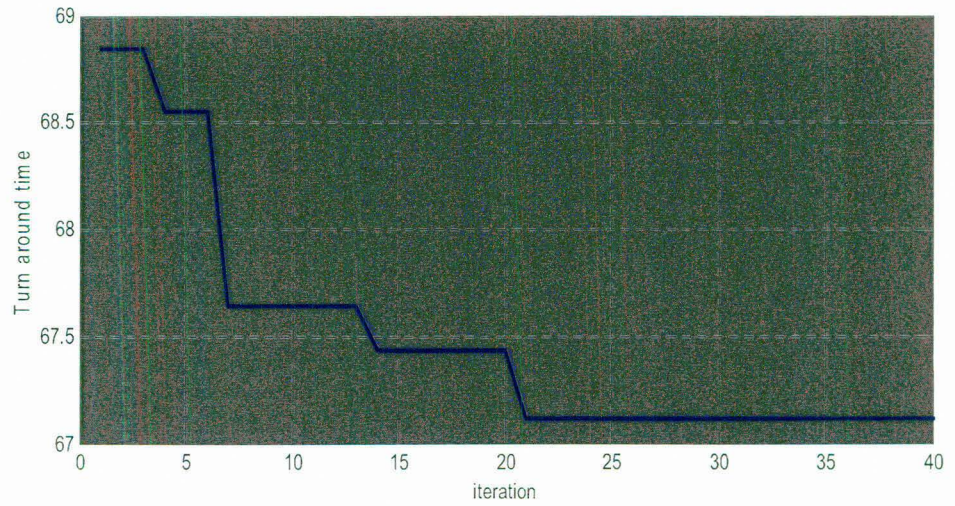


Fig. 5.1.5 Turnaround time Observation Graph ( $l=160$ ).

**OUTPUT:** Min TAT without communication=67.114948

Average TAT without communication=6.737207e+001

Speedup without communication=2.779950e+001

### 5.1.6 Number of Tasks=250

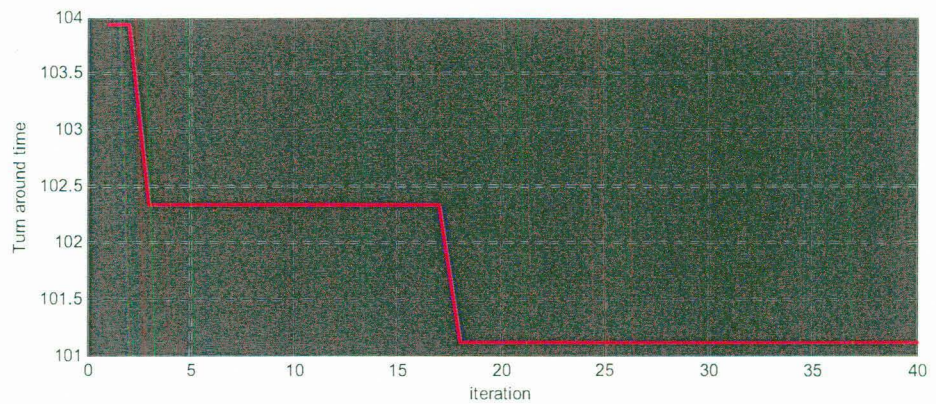


Fig. 5.1.6 Turnaround time Observation Graph ( $l=250$ ).

**OUTPUT:** Min TAT without communication=100.661579  
Average TAT without communication=1.014198e+002  
Speedup without communication=2.885453e+001

### 5.1.7 No of Tasks=300

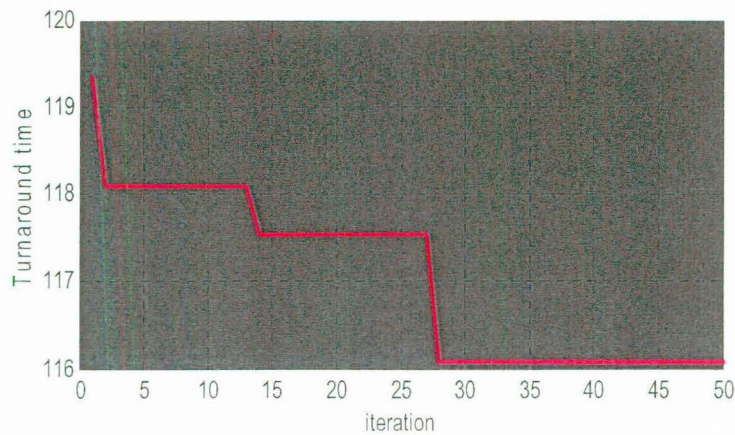


Fig. 5.1.7 Turnaround time Observation Graph ( $l=300$ ).

**OUTPUT:** Min TAT without communication=109.122812  
Average TAT without communication=1.096821e+002  
Speed UP without communication=3.201712e+001

### 5.1.8 Comparison of All Results

We observe that as the number of tasks in the job submitted by the user increase Turnaround time also increases in the computational grid. Speed up of computational grid also increase with the increase in no of tasks upto certain stage but after a critical stage it starts decreasing slightly or the rate of increase of speed up is moderated and almost becomes constant. We have depicted the comparison graph corresponding to the different tasks on the same number of machine in the following three figures.



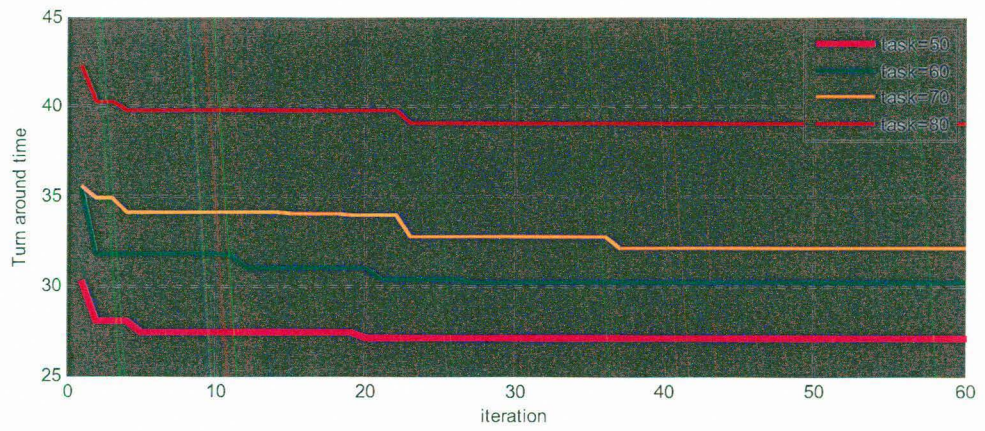


Fig. 5.1.8 TAT Comparison Graph with Different Number of Tasks ( I).

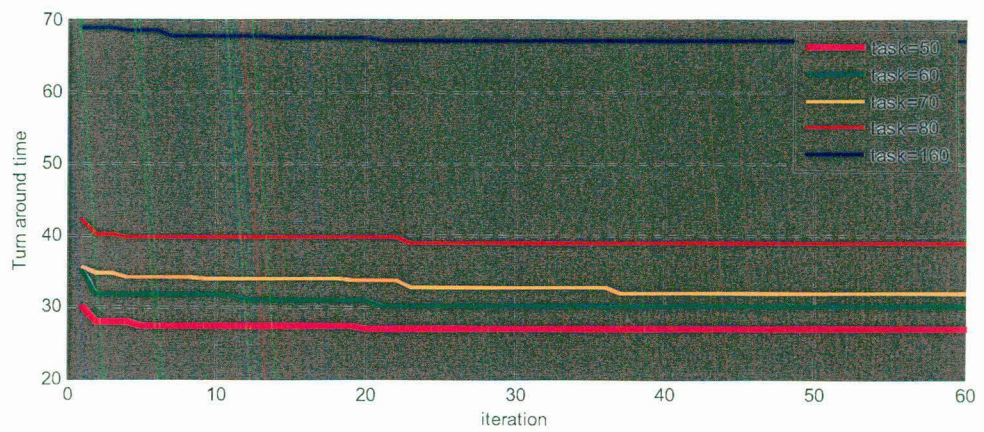


Fig. 5.1.9 TAT Comparison Graph with Different Number of Tasks (II).

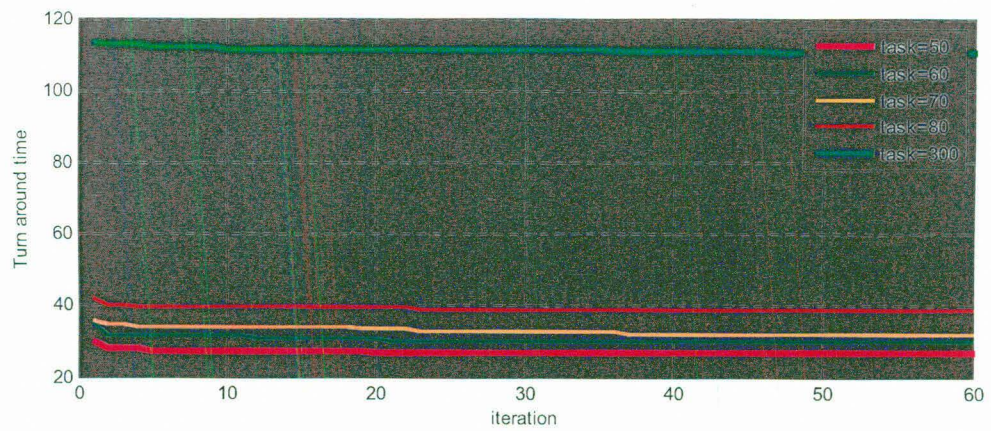


Fig. 5.1.10 TAT Comparison Graph with Different Number of Tasks (III).

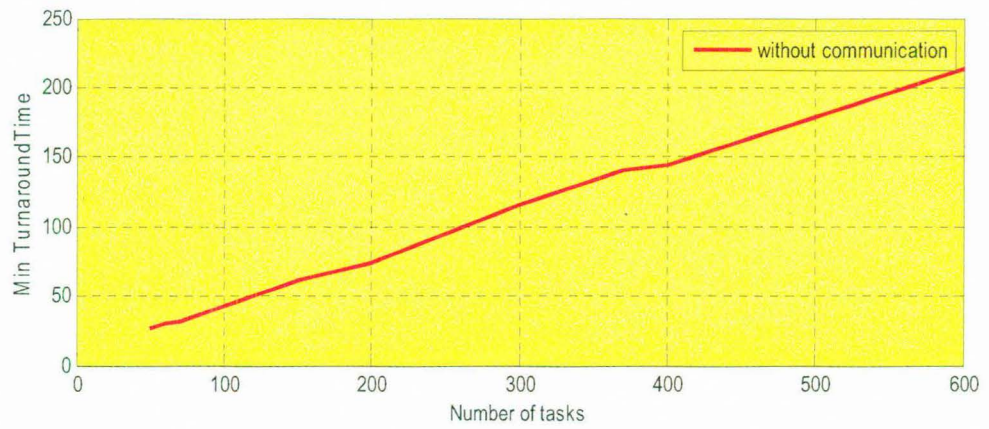


Fig. 5.1.11 Comparison of Min TAT with Increase in Number of Task.

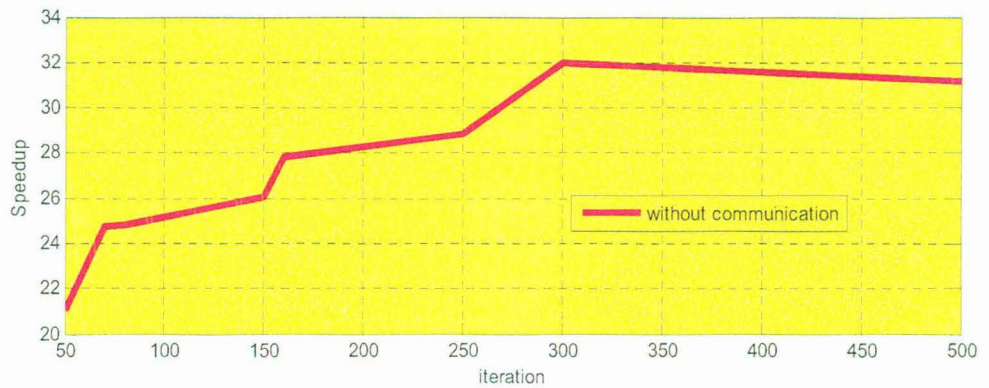


Fig. 5.1.12 Comparison of Speedup with Increase in Number of Task.

**Case 2** In this case number of machines varies. Other parameters are given below.

Number of Tasks =300, Ant =5, Range of Arrival Rate (1-100) MIPS, Range of Processing Speed (200-300) MIPS, Range of Task Size (2000-5000) Million Instructions for each experiment from section 5.1.9 to 5.1.12. Min Turnaround, Average Turnaround, and Speedup of each experiment for different number of machines are written just below each individual graph. From comparison point of view, Fig 5.1.16 also takes into account previous graphs. In these graphs each line is represented by different color for more visibility.

### 5.1.9 Number of Machines=60

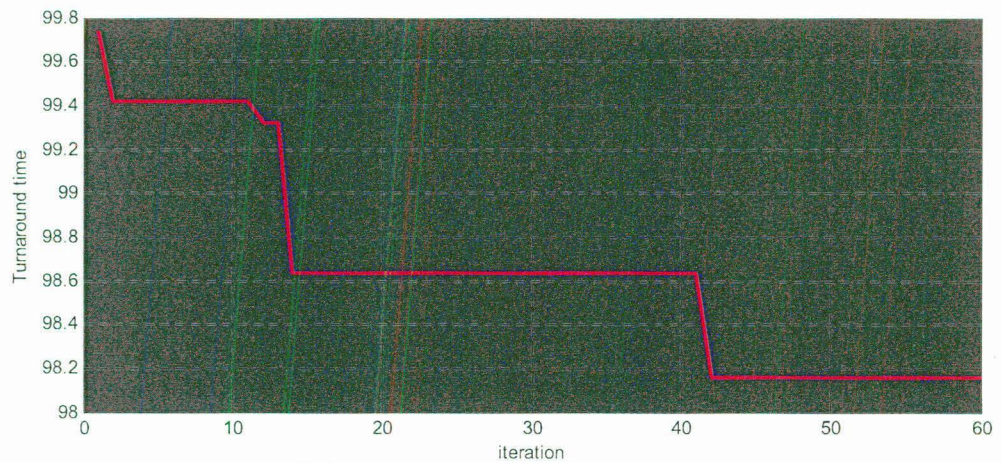


Fig. 5.1.13 Turnaround time Observation Graph (m=60).

**OUTPUT:** Min TAT without communication=98.158341

Average TAT without communication=9.865624e+001

Speed UP without communication=3.559537e+001

### 5.1.10 Number of machine=70

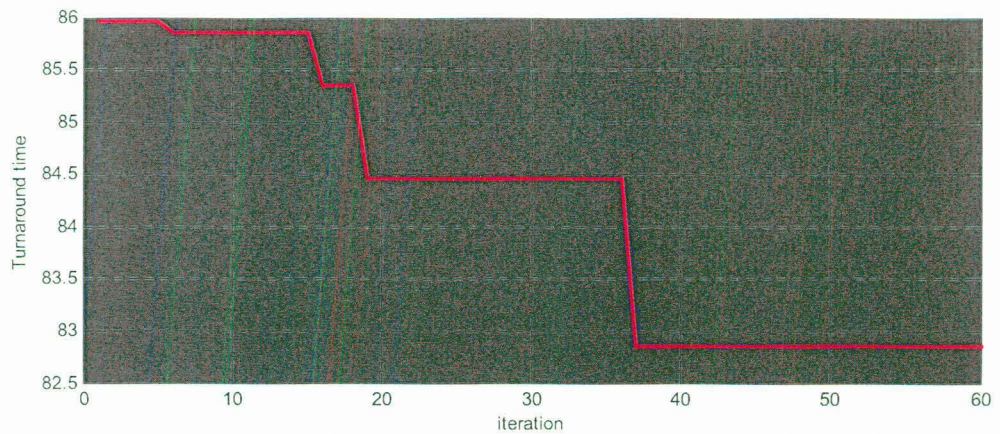


Fig. 5.1.14 Turnaround time Observation Graph (m=70).

**OUTPUT:** Min TAT without communication=82.865116

Average TAT without communication=8.403102e+001

Speed UP without communication=4.179059e+001

### 5.1.11 Number of machine=100

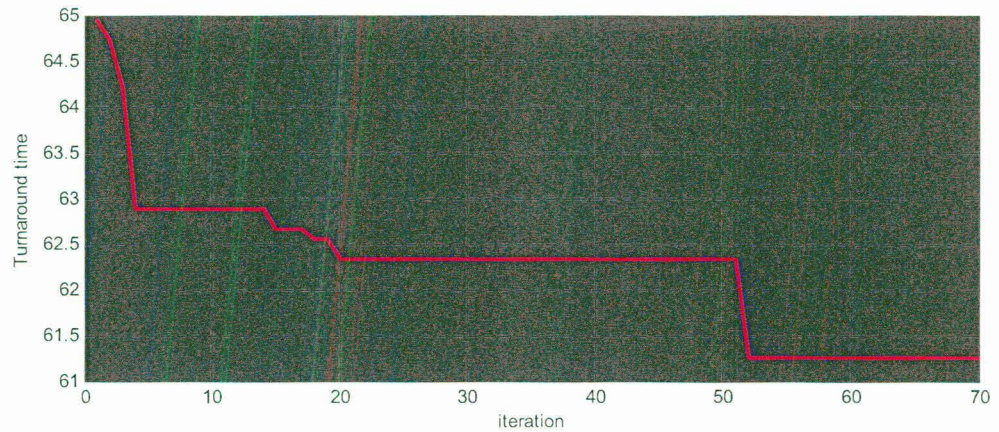


Fig. 5.1.15 Turnaround time Observation Graph (m=100).

**OUTPUT:** Min TAT without communication=61.253834

Average TAT without communication=6.224690e+001

Speed UP without communication=5.641575e+001

5.1.12 Rest of the outputs of the experiments are given in the form of table given below

No of Machine	Average TAT	Min TAT	Speed-Up
110	57.9622	57.1981	60.5860
120	54.3105	54.0031	64.6597
200	37.3908	36.5697	93.9186
250	31.9877	31.9380	109.7829
300	28.7960	28.5673	121.9523
350	27.7861	27.1698	126.3832
400	25.7606	25.4873	136.3241
500	24.4598	24.4534	142.7920
600	23.8753	23.4875	147.3882

Table 5.1.1- Comparison of TAT and Speedup without Communication

From the above experiments we observe that the Turnaround time of a job (which consists of fixed number of tasks) submitted by the user in the computational grid decreases with the increase in the number of machines/nodes (of the same range of speed and load) very rapidly but after a certain number of increase in the machines (usually after  $m > l$ ) rate of decrease of Turnaround time is very slow. The above situation is depicted in the following two figures given below. Speed up increase initially rapidly with the increase in the number of machines but after a certain stage Speed up increases very slightly (figure 5.1.17).

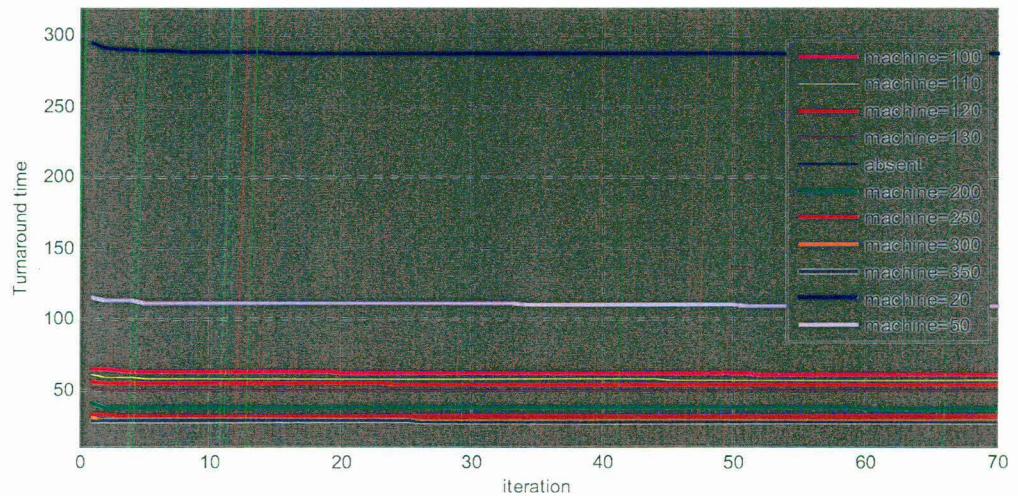


Fig. 5.1.16 TAT Comparison Graph with Different Number of Machine.

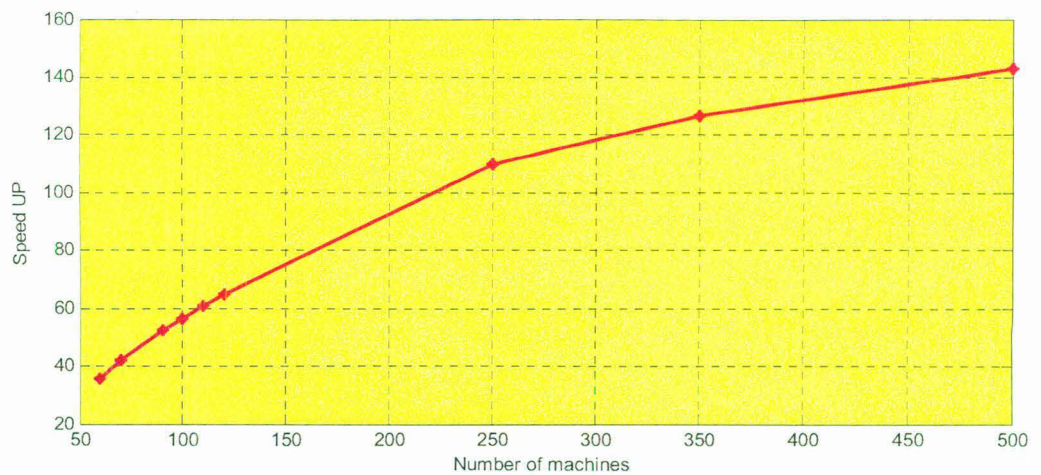


Fig. 5.1.17 Speedup Graph with Increased Number of Machine.

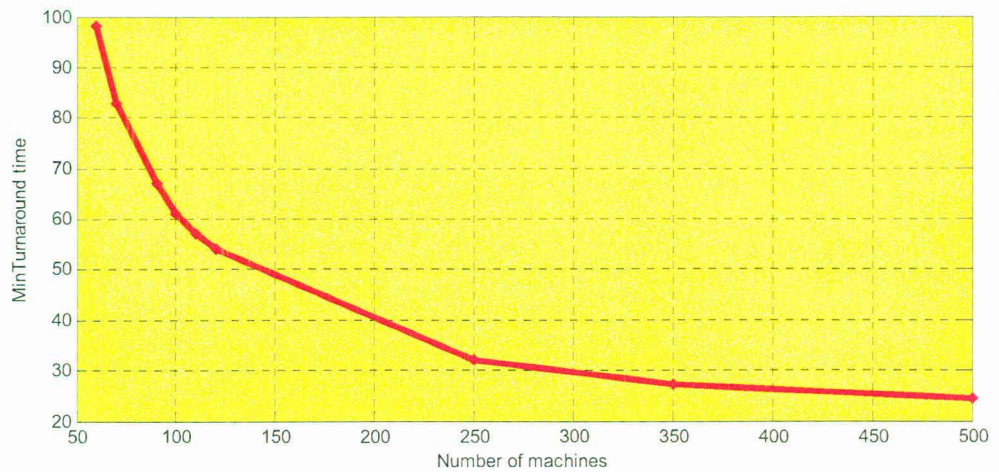


Fig. 5.1.18 Min TAT with Different Number of Machine.

## 5.2 Observation with communication cost

**Case 1** Range of communication and Hamming distance are varying. Other parameters are as below.

Number of Machines =60, Number of task=300, No of Ants=5, Range of Arrival Rate 1-100 MIPS, Range of Processing Speed 200-300 MIPS, Range of Task Size 2000-5000 Million instructions, Range of Load of Network 60-90MBPS, Range of Load of Bandwidth 100 -120MBPS.

### 5.2.1 Range of Message bit 0-15 MB, Hamming Distance Range 0-4

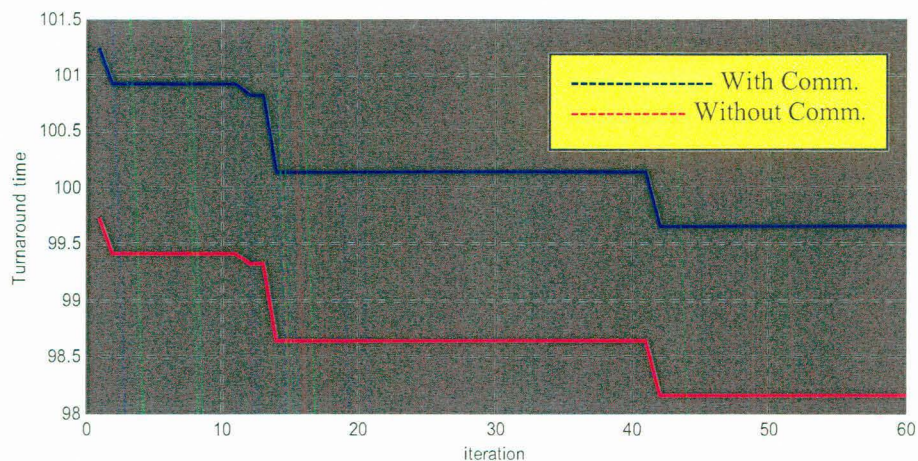


Fig. 5.2.1 TAT Observation Graph (UMR=15, MHD=4).