Grid Scheduling Algorithm based on Dynamic Time Quantum

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Software Engineering

Submitted By:
Tanu
(800831014)

Under the supervision of:

Dr. Inderveer Chana

Assistant Professor, CSED

Thapar Univerity, Patiala

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

July 2010

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Grid Scheduling Algorithm based on Dynamic Time Quantum "**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Inderveer Chana and refers other researcher's works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

Tanu

(Tanu)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Inderveer 12/7/2K

(Dr. Inderveer Chana)
Assistant Professor,
Computer Science and Engineering Department,
Thapar University,
Patiala.

Countersigned by

(Dr. RAJESH BHATIA) 12/07/10
Head,
Computer Science & Engineering, Department,
Thapar University,
Patiala.

RK Sharma 14.7.10

(Dr.R.K.SHARMA)
Dean (Academic Affairs),
Thapar University,
Patiala.

Acknowledgement

I wish to express my deep gratitude to Dr. Inderveer Chana, Assistant Professor and Dr. Rajesh Bhatia Head, Computer Science & Engineering Department for providing their uncanny guidance and support throughout the preparation of the thesis report.

I am also heartily thankful to Dr. A.K.Verma, Assistant Professor, Computer Science and Engineering Department for the motivation and inspiration that triggered me for my thesis work.

I would also like to thank all the staff members, T.U. Grid Group and all my friends who were always there at the need of the hour and provided all the help and support, which I required for the completion of the thesis.

Last but not the least; I would like to thank God for not letting me down at the time of crisis and showing me the silver lining in the dark clouds.

Tanu

Tanu

(800831014)

Abstract

Computational Grid is the next generation of distributed computing systems. They allow the sharing of geographically distributed resources in an efficient way, extending the boundaries of what is perceived as distributed computing. Various science applications can benefit from the use of grids to solve CPU-intensive problems, creating potential benefits to the entire society. With further development of grid technology, it is very likely that corporations, universities and public institutions will exploit grids to enhance their computing infrastructure. In recent years there has been a large increase in grid technologies research, which has produced some reference grid implementations. Grid Computing has progressed a lot, still the areas like resource management, resource scheduling, load balancing and security have many challenges that need to be addressed.

Scheduling is an integral part of Grid computing. Even though middleware support for grid computing has been the subject of extensive research, scheduling policies for the grid context have not been much studied. In addition to processor utilization, it is important to consider the waiting time, throughput, and response times of jobs in evaluating the performance of grid scheduling strategies. In this thesis a distributed scheduling algorithm has been proposed and designed that is based on the Dynamic time Quantum technique. Dynamic time Quantum technique improves the performance in terms of time delays. The algorithm has been implemented in Java and further validated in Condor scheduler. The experimental results depict the efficiency of the algorithm based on Dynamic Time Quantum technique.

Table of Content

List of Figures

List of Tables

Chapter 1
Introduction

Grid is emerging as a wide-scale infrastructure that promises to support resource sharing and coordinated problem solving in dynamic, multi-institutional virtual organization [1]. Grid computing applies the resources of many computers in a network for a single problem at the same time - usually to a scientific or technical one that requires a large number of computer processing cycles or access to large amounts of data. Grid computing can be thought of as distributed and large-scale cluster computing and as a form of network-distributed parallel processing.

This chapter introduces Grid computing, discusses about the history, types, and the motivation of this research work and finally gives the framework of this thesis.

1.1 Grid Computing

Grid computing is gaining a lot of attention within the IT industry. Grid computing is a distributed computing taken to the next evolutionary level. It provides the electronic foundation for a global society in business, government, research, science and entertainment. The goal is to create the illusion of a large and powerful self managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources. The Grid computing integrates networking communication, computation and information to provide a virtual platform for computation and data management in the same way that the Internet integrates resources to form a virtual platform for information [5].

Grid computing is analogous to a power grid. When you plug an appliance or other object requiring electrical power into a receptacle, the power is available. The vision of Grid computing is similar. Individual users can access computers and data, or even contribute its resources to the existing Grid, without having to consider location, operating system, account administration, and other details. The resources are not visible to the user, just as the consumer of electric power is unaware of how their electricity is being generated.

Grid infrastructure will provide us with the ability to dynamically link together resource as an ensemble to support the execution of large-scale, resource-intensive, and distributed applications [5].

Grid applications often involve large amounts of data and computing and often

require secure resource sharing across organizational boundaries. Sharing resources over the Internet raises many technical problems, such as large pool of resources, the heterogeneity of platforms, the diversity in user behaviors and the lack of reliability. In the next few years, the Grid holds the potential for fundamental infrastructure not only for e-Science but also for e-Business, e-Government, e-Science and e-Life. This emerging infrastructure will exploit the revolutions driven by Moore's law [2] for CPU's, disks and instruments as well as Gilder's law [3] for (optical) networks.

1.2 History of Grid

The term "Grid" was coined in the mid 1990s to denote a proposed distributed computing infrastructure for advanced science and engineering. The concept of computational Grid has been inspired by the 'electric power Grid', in which a user could obtain electric power from any power station present on the electric Grid irrespective of its location, in an easy and reliable manner. Whenever additional electricity is required, just plug into a power Grid to access additional electricity on demand, similarly for computational resources plug into a computational Grid to access additional computing power on demand using most economical resources [8].

Table-1.1 Historical Background of the Grid [8]

Technology	Year
Networked Operating Systems	1979-81
Distributed operating systems	1988-91
Heterogeneous computing	1993-94
Parallel and distributed computing	1995-96
The Grid	1998

Cluster and Intranet computing are the parents of grid computing as shown in Figure 1.1. They came in existence, but there were many drawbacks due to which both technologies couldn't survive for a long time. There is collection of interconnected computers which are tightly coupled in cluster computing. This computing is cost effective but communi- cation overhead, resource wastage and maintenance are its major drawbacks [14].

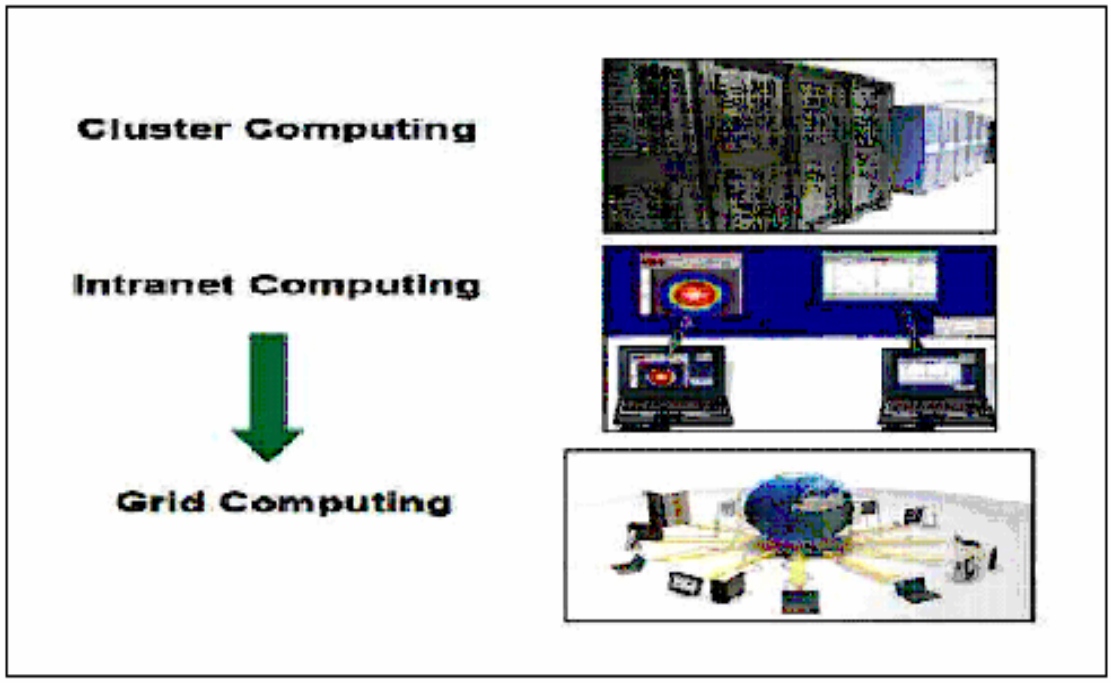


Figure 1.1: Evolution of Grid [14]

1.3 Characteristics of Grid Computing

In today's complex world computers have become extremely powerful and they are enough capable to run more complex problem, still there are many complex scientific experiments, advanced modeling scenarios, genome matching, astronomical research, a wide variety of simulations, complex scientific & business modeling scenarios and real time personal portfolio management, which require huge amount of computational resources. To satisfy some of these aforementioned requirements, Grid computing is being utilized [4]. Following are the benefits that Grid computing provides to user community, developer community and enterprise community as well [6].

(a) Exploiting Underutilized Resources

In most organizations, there are large amounts of underutilized computing resources. Most desktop machines are busy less than 5 percent of the time. In some organizations, even the server machines can often be relatively idle. Grid computing provides technique for exploiting these underutilized resources. Data Grid can be used to aggregate the un- used storage into a much larger virtual data store. Also a Grid can provide a consistent way to balance the loads on a wider federation of resources [10].

(b) Parallel CPU Capacity

The potential for massive parallel CPU capacity is one of the most attractive features of a Grid. In addition to pure scientific needs, such computing power is driving a new evolution in industries such as the bio-medical field, financial modeling, oil exploration, motion picture animation, and many others. The common attribute among such uses is that the applications have been written to use algorithms that can be partitioned into independently running parts [10]. Many industries and scientific communities require the use of parallel computing in order to run applications or solve certain problems. Grid computing provides a framework that allows jobs to be split up into multiple sub jobs, and each sub-job can be made to execute in parallel on different machines [10].

(c) Access to Additional Resources

Grid users can access resources that they might not normally have access to. For e.g., an organization might have a Grid machine connected to an electron microscope or a telescope. Some machines may have expensive licensed software installed that the user requires. Grid users who do not normally have access to such equipment can be given access to it, whether they belong to the organization or not [10].

(d) Resource Balancing

The Grid framework provides the proper utilization of resources. For e.g., jobs can be scheduled to run on idle machines or machines with low activity levels. Grids also offer load balancing. If jobs running on Grid require a high level of communication between each another, they could be scheduled in a manner that minimizes the cost of communication or the amount of traffic on their communication lines [10].

(e) Reliability

Grid provides reliability in terms of failure at one location; the other parts of the Grid are not likely to be affected. Grid management software can automatically or manually resubmit jobs most of the times to other machines on the Grid when a failure is detected. In critical, real-time situations, multiple copies of the important jobs can be run on different machines throughout the Grid. Their results can be checked for any kind of inconsistency, such as computer failures, data corruption, or tampering; thus offering much more reliability [10].

1.4 Types of Grid

Grid can be classified on the basis of two parameters namely functional and topology as discussed below:-

(i) Functional classification

According to the distinctly targeted application realms, Grid systems can be classified into two categories. But there are actually no hard boundaries between these Grid categories. Real Grids may be combination of two of these types. The two categories of grid systems are described below:

(a) Computational Grid

Computational Grid is hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities and that enables coordinated resource sharing within dynamic organizations consisting of individuals, institutions, and resources. Thus computational Grid is designed so that users won't have to worry about where scientific and engineering computations are being performed [1].

(b) Data Grid

In present time Grid application areas are shifting from scientific computing to industry and business applications, that’s why data Grids is an enhancement of computational Grids, and have been designed to store, move, and manage large data sets exploited in distributed data-intensive applications [12]. Data Grid represents a combination of large data sets, currently terabytes and will grow to petabytes due to geographic distribution of users, resources and computational intensive analysis results in complex and strict performance demands that cannot be satisfied by ordinary data infrastructure [20].

(c) Semantic Grid

Semantic Grid is supported by two key building blocks – semantic web technologies for creating and maintaining models (ontology’s), and semantic aware services and protocols for exchanging metadata [7]. Semantic Grid is an extension of Grid and provides the infrastructure that systematically manages the complete lifecycle of metadata. Semantic Grid aims to provide an infrastructure within the Grid middleware providing a core set of services and protocols to support the sharing and management of all aspects of metadata. This will enable unanticipated reuse of Grid services and resources, better support for interoperability, and flexible Grids [9].

(d) Knowledge Grid

Knowledge Grid is an intelligent, sustainable internet application environment that enables people or virtual roles (mechanisms that facilitate interoperation among users, applications, and resources) to effectively captures, publish, share, and manage explicit knowledge resources [11].

(e) Service Grid

Service Grids are created in order to realize the business potential of web services. Service Grids represent distributed architectural component that realize an array of service business or utilities owning and deploying specific enabling services. Services businesses on the other hand, may either be specialized and independent businesses or revenue centers within larger enterprises offering their specialized enabling services to other enterprises [19].

(ii) Topology classification

Grids can be built in all sizes, ranging from just a few machines in a department to groups of machines organized as hierarchy spanning the world. Grids can be classified into three categories according to the topology of grid; these are intra grid, extra grid and inter grid.

(a) Intragrid

A typical intragrid topology, as illustrated in Figure 1.2, exists within a single organization, providing a basic set of Grid services.

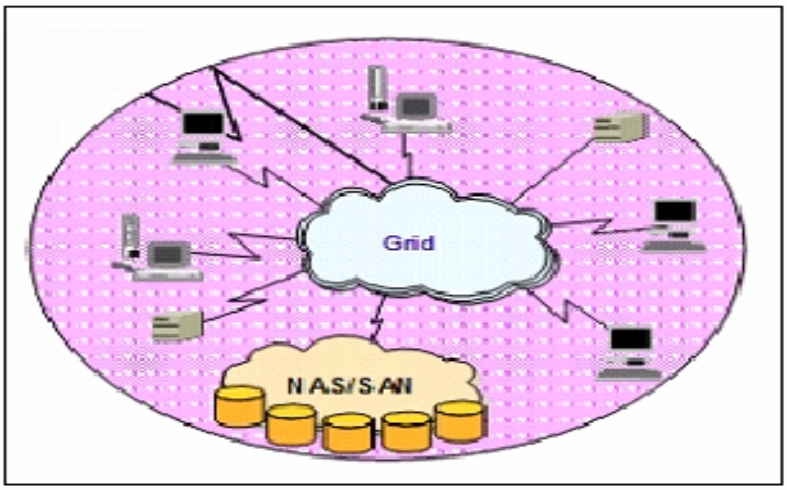


Figure 1.2: Intragrid [10]

The single organization could be made up of a number of computers that share a common security domain, and share data internally on a private network. The primary characteristics of an intragrid are a single security provider, bandwidth on the private

network is high and always available, and there is a single environment within a single network. Within an intragrid, it is easier to design and operate computational and data Grids. intragrid provides a relatively static set of computing resources and the ability to easily share data between Grid systems [10].

(b) Extragrid

Based on a single organization, the extragrid expands on the concept by bringing together two or more intragrids. An extragrid, as illustrated in Figure 1.3 typically involves more than one security provider, and the level of management complexity increases. The primary characteristics of an extragrid are dispersed security, multiple organizations, and remote/WAN connectivity Within an extragrid, the resources become more dynamic [10].

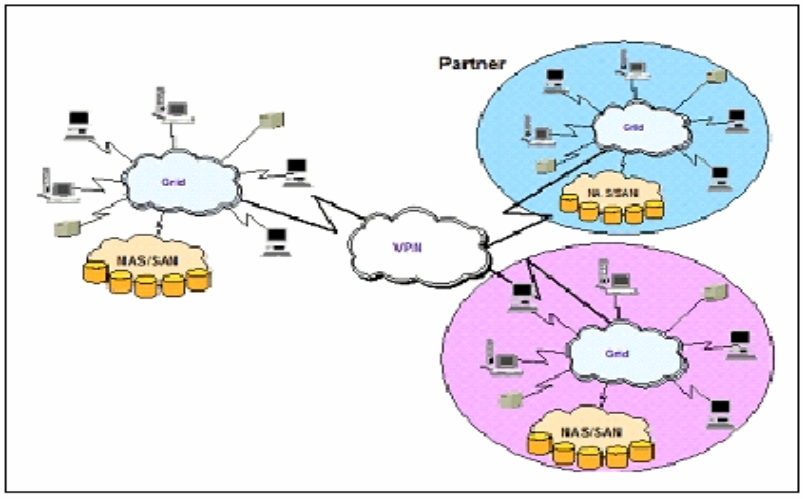


Figure 1.3: Extragrid [10]

(c) Intergrid

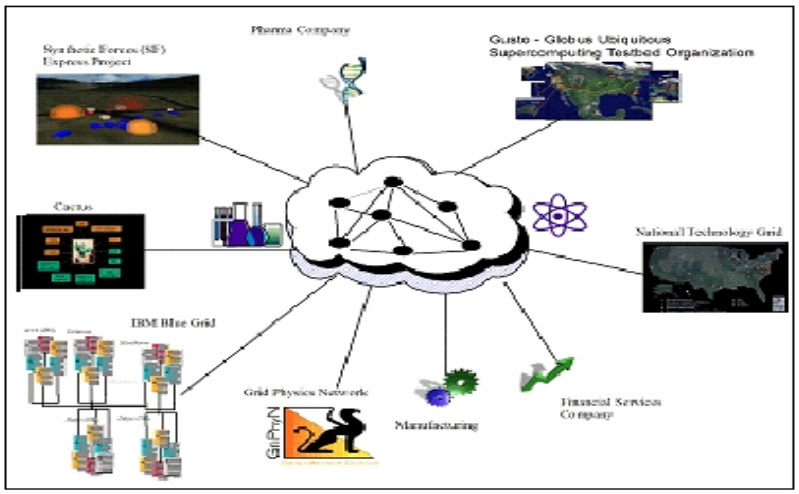


Figure 1.4: Intergrid [10]

An Intergrid has an analogy with the internet. It is the most complicated form of Grid topology. The primary characteristics of an intergrid include dispersed security, multiple domains and WAN connectivity [10].

1.5 Grid Resource Management System

A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities [10]. Grid Resources can be computers, storage space, instruments, software applications, and data, all connected through the Internet and a middleware software layer that provides basic services for security, monitoring, resource management, and so forth.

To manage Grid resources a Grid Resource Management System is required which deals with the process of identifying requirements, matching resources to applications, allocating those resources, and scheduling and monitoring Grid resources over time in order to run Grid applications as efficiently as possible.

1.6 Grid Scheduling

Grid scheduling is defined as the process of making scheduling decisions involving resources over multiple administrative domains. This process can include searching multiple administrative domains to use a single machine or scheduling a single job to use multiple resources at a single site or multiple sites.

Job: anything that needs a resource - from a bandwidth request, to an application, to a set of applications (for example, a parameter sweep).

Resource: anything that can be scheduled: a machine, disk space, a QoS network etc.

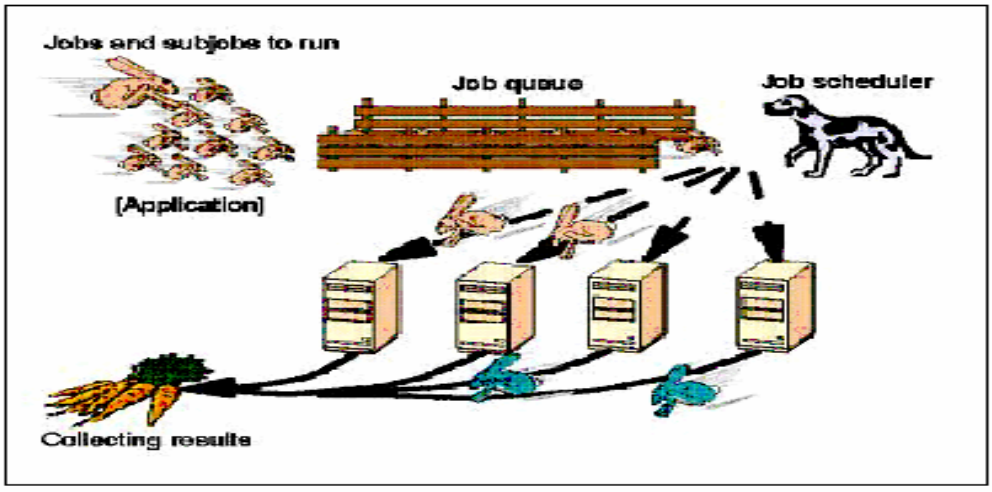


Figure 1.5:Application is one or more jobs that are scheduled to run on grid [10]

Grid Scheduling is a software framework with which the scheduler collects resource state information, selects appropriate resources, predicts the potential performance for each candidate schedule, and determines the best schedule for the applications to be executed on a Grid System subject to QoS goals.

Grid Resource Management involves several different layers of schedulers. At the highest level are Grid-level schedulers that may have a more general view of the resources but are very "far away" from the resources where the application will eventually run. At the lowest level is a local resource management system that manages a specific resource or set of resources [10].

1.7 Research Motivation

Grid Computing enables aggregation and sharing of geographically distributed computational, data and other resources as single, unified resource for solving large scale compute and data intensive computing application. Management of these resources is an important infrastructure in the Grid computing environment. It becomes complex as the resources are geographically distributed, heterogeneous in nature, owned by different individual or organizations with their own policies, have different access and cost models, and have dynamically varying loads and availability. The conventional resource management schemes are based on relatively static model that have centralized controller that manages jobs and resources accordingly. These management strategies might work well in those scheduling regimes where resources and tasks are relatively static and dedicated. However, this fails to work efficiently in many heterogeneous and dynamic system domains like Grid where jobs need to be executed by computing resources, and the requirement of these resources is difficult to predict.

Due to highly heterogeneous and complex computing environments, the chances of faults increases [15], and therefore number of resources available to any given application highly fluctuates over time which reduces the performance and the efficiency. Therefore it is necessary to design a mechanism for scheduling to improve the efficiency in such infrastructure.

This work focuses on scheduling in a Grid environment. Grid application performance is critical in Grid computing environment. So to achieve high performance there is a need to understand the factors that can affect the performance of an application due to Scheduling, which is one of most important factors that influence the overall performance of application.

1.8 Thesis Organization

Chapter 2 explains what Grid scheduler is, its architecture and various challenges in grid scheduling. It also describes the Grid scheduling process, types of Grid scheduling and various approaches associated with it.

Chapter 3 describes the problem formulation.

Chapter 4 describes implementation detail of the Grid Scheduler .

Chapter 5 describes the experimental results of proposed algorithm and Comparisons of proposed algorithm with the existing algorithms

Chapter 6 summarizes the work presented in this thesis followed by future scope of this work

Chapter 2

Literature survey

Grid scheduling is defined as the process of making scheduling decisions involving resources over multiple administrative domains. This process may include searching multiple administrative domains to use a single machine or scheduling a single job to use multiple resources at a single site or multiple sites. This chapter surveys Grid scheduling procedure, Architecture and Challenges in Grid scheduling [10].

2.1 Grid Scheduling Procedure

A user goes through three stages to schedule a job when it involves multiple Grid sites. Phase one is resource Discovery, in which the user makes a list of potential resources to use. Phase two involves gathering information about those resources and choosing a best set to use. In the phase three the user runs the job.

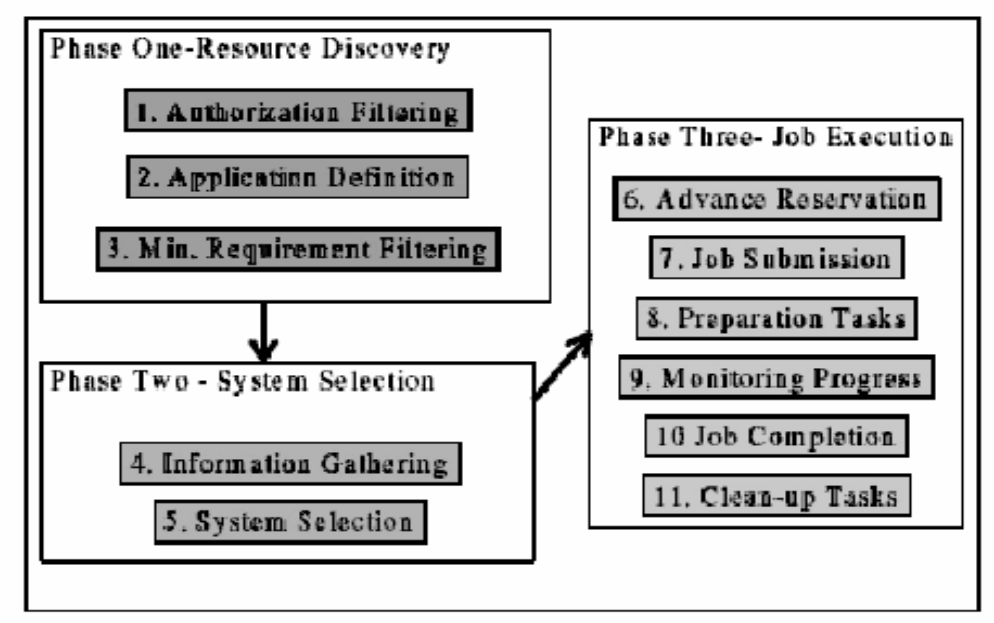


Fig 2.1: Resource Scheduling Phases [10]

Phase 1: Resource Discovery

Resource discovery is the process that takes as input a user request and returns a list of resources or services that can possibly fulfill the given request. Resource discovery is the first phase of resource scheduling. It involves the user selecting a set of resources. At the beginning of this phase, the potential set of resources is the empty set, and at the end of this phase, the potential set of resources is some set that has passed a minimal feasibility requirement [10].

Steps in Resource Discovery

- Authorization filtering,
- Application requirement definition
- Minimal requirement filtering

Step 1: Authorization Filtering

It is generally assumed that a user will know which resources he or she has access to in terms of basic services. At the end of this step the user will have a list of machines or resources to which he or she has access.

Step 2: Application requirement definition

In order to proceed in Resource Discovery, the user must be able to specify some minimal set of job requirements in order to further filter the set of feasible resources. The set of possible job requirements can be very broad, and vary significantly between jobs. This may include static details such as operating system or hardware for which a binary of the code is available. Dynamic details are also possible e.g. a minimum RAM requirement, connectivity needed. This may include any information about the job that should be specified to make sure that the job could be matched to a set of resources.

Step 3: Minimal requirement filtering

Given a set of resources to which a user has access and the minimal set of requirements the job has, the third step in the Resource Discovery phase is to filter out the resources that do not meet the minimal job requirements. The user generally does this step by going through the list of resources and eliminates the ones that do not meet the job requirements. It could also be combined with the gathering of more detailed information about each resource. However, when being done by hand, if a user can eliminate an inappropriate resource it is done at this stage to simplify the information gathering in the next phase [10].

Phase 2: System Selection

Given a group of possible resources (or a group of possible resources set), all of which meet the minimum requirements for the job, a single resource (or single resource set) must be selected on which to schedule the job.

Steps in System Selection

- Information Gathering
- System selection

Step 4: Gathering Information (QUERY)

In order to make the best possible resource match, a user needs to gather dynamic information about the resources in question. Depending on the application and resource in question, different information may be needed. Take for instance the simple case of finding the best single resource for a job to run on. A user might want to know the load on the various machine(s) and queue lengths if the machine has queues. In addition, physical characteristics and software requirements play a role, is the disk big enough for the data etc. then there are location/connectivity issues is the machine close enough to the data store. All of these issues are multiplied in the case of multiple resources. Making an advance reservation may or may not be a part of this step.

Step 5: Select the system(s) to run on

Given the information gathered by the previous step, a decision of which resource (or set of resources) should the user submit a job is made in this step. This can be done in variety of ways. Note that this does not address the situation of speculative execution, where a job is submitted to multiple resources and when one begins to run the other submissions is cancelled [10]

Phase 3: Job Execution

The third phase of scheduling is running a job. This involves a number of steps [23]:

- Advance Reservation
- Job Submission
- Preparation Tasks
- Monitoring Progress
- Job Completion
- Clean-up Tasks

Step 6: Advance Reservation (Optional)

In order to make the best use of a given system, part or all of the resources may have to be reserved in advance. Depending on the resource, an advance reservation can be easy or hard to do and may be done with mechanical means or human means. Moreover, the reservations may or may not expire with or without cost. One issue in having advance reservations become more common is the need for the lower-level resource to support the fundamental services on the native resources [22].

Step 7: Submit Job to Resources

Once resources are chosen the application must be submitted to resources. This may be easy as running a single command or as complicated as running a series of scripts, and may or may not include setup or staging.

Step 8: Preparation Tasks

The preparation stage may involve setup, claiming a reservation, or other actions needed to prepare the resource to run the application. One of the first attempts at writing a scheduler to run over multiple machines at America's National Aeronautics and Space Agency (NASA) was considered unsuccessful because it did not address the need to stage files automatically [22].

Step 9: Monitoring Progress

Depending on the application and its running time, users may monitor the progress of their application and possibly change their mind about where or how it is executing. Such monitoring is done by repetitively querying the resource for status information. If a job is not making sufficient progress, it may be rescheduled. Such rescheduling is significantly harder on a Grid system than on a single parallel machine because of the lack of control. It may be possible to develop additional primitives for interactions between local systems and Grid Schedulers to make this behavior more straight forward.

Step 10: Job Completion

When the job is finished, the user needs to be notified. Often, submission scripts for parallel machines will include an e-mail notification parameter. For fault-tolerant reasons, however, such notification can prove surprisingly difficult. Moreover, with so many interacting systems one can easily envision situations in which a completion state cannot be reached. And of course, end-to-end performance monitoring to ensure job completion is a very open research question [22].

Step 11: Cleanup Tasks

After a job is run, the user may need to retrieve files from that resource in order to do data analysis on the results, remove temporary settings, and so forth. Any of the current systems that do staging (Step 8) also handle cleanup. Users generally do this by hand after a job is run, or by including clean-up information in their job submission scripts [22].

2.2 Grid Scheduler Architecture

The generic Grid Scheduler architecture provides a high-level view of Grid Schedulers. Every component seems necessary for any comprehensive Grid scheduling System [16]. Generic Scheduler does the following:

- Interact with local resource managers
- Interact with external services that are not defined in the Grid scheduling

Architecture, like information, forecasting, submission, security or execution services

- Receive a scheduling Problem
- Calculate a schedule, and return a scheduling decision

Real resources lay on the bottom of the architecture in the Figure 2.2 each being Managed by a Local Resource Manager (LRM). An LRM act as the interface between a Grid Scheduler and a local autonomous site. An Local Resource Manager (LRM) is responsible for :

- local scheduling inside a resource domain, where not only jobs from exterior Grid users, but also jobs from the domain's local users are executed, and
- reporting resource information to GIS.

Examples of such local schedulers include OpenPBS and Condor. An LRM also collects local resource information by tools such as Network Weather Service [17].

(a) Information Service (IS)

Information about the status of available resources is very important for a Grid Scheduler to make a proper schedule, especially when the heterogeneous and dynamic nature of the Grid is taken into account. The role of the Grid information service (GIS) is to provide such information to Grid Schedulers. GIS is responsible for collecting and predicting the resource state information, such as CPU capacities, memory size, network bandwidth, software availabilities and load of a site in a particular period. GIS can answer queries for resource information or push information to subscribers. The Globus Monitoring and Discovery System (MDS) [18] is an example of GIS [17].

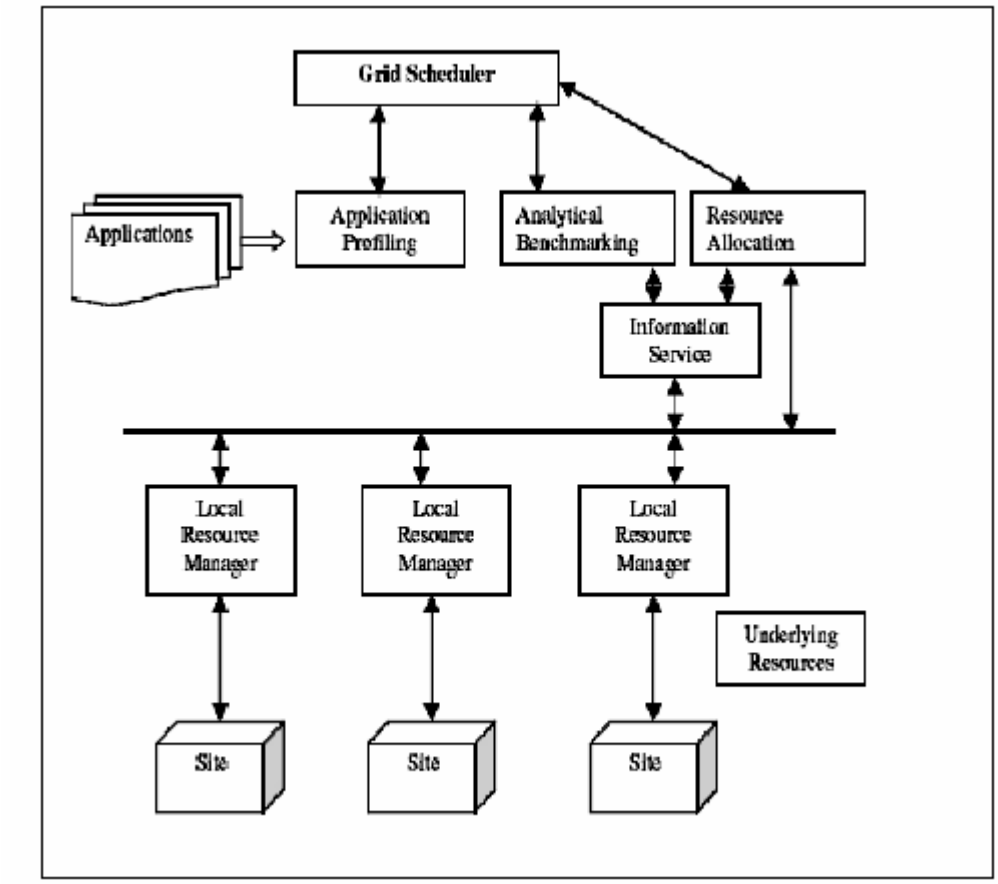


Figure 2.2: General Grid Scheduler Architecture [16]

(b) Application Profiling and Analytical Benchmarking (AP and AB)

Application profiling (AP) is used to extract properties of applications, while analogical benchmarking (AB) provides a measure of how well a resource can perform a given type of job. On the basis of knowledge from AP and AB, and following a certain performance model, cost estimation computes the cost of candidate Schedules, from which the Scheduler chooses those that can optimize the objective functions [17].

(c) Grid Scheduler (GS)

This is the core component in the architecture. The GS needs to do two jobs: one is resource selection and the other one is mapping. Resource selection is the process of selecting feasible and available resources for a given application to be scheduled. Mapping is the process of placing the jobs and communications of the application onto the resources and networks. Each mapping of jobs to feasible resources produces a candidate schedule. Each candidate schedule is then estimated for its performance

potential based on the performance model.

(d) Resource Allocation (RA)

This component implements a finally determined schedule through allocating the resources to the corresponding jobs. Resource allocation may involve data staging and binary code transferring before the job starts to execute on the computational resource.

2.3 Challenges in Grid Scheduling

Although Grids fall into the category of distributed parallel computing environments, they have a lot of unique characteristics, which make the Scheduling in Grids highly difficult. An adequate Grid scheduling system should overcome these challenges to leverage the promising potential of Grid systems, providing High-Performance services [21].

2.3.1 Resource Heterogeneity

A Grid has mainly two categories of resources: networks and computational resources. Heterogeneity exits in both of the two categories of resources. First, networks used to interconnect these resources may differ significantly in terms of their bandwidth and communicational protocols. Second, computational resources are usually heterogeneous in that these resources may have different hardware, such as instruction set, computer architecture, number of processors, physical memory size, CPU speed and so on and also different software such as different operating systems, file systems, cluster management software and so on. The heterogeneity results in differing capability of processing jobs. Resources with different capacity can not be considered uniformly. An adequate scheduling system should address the heterogeneity and further leverage different computing powers of diverse resources [16].

2.3.2 Site Autonomy

Typically a Grid may compromise multiple administrative domains. Each domain shares a common security and management policy. Each domain usually authorizes a group of users to use the resources in the domain. Thus applications from unauthorized user should not be eligible to run on the resources in some specific

domains. Furthermore, a site is an autonomous computational entity. A shared site will result in many problems. It usually has its own scheduling policy, which complicates the prediction of a job on the site. A single overall performance goal is not feasible for a Grid system since each site has its own performance goal and scheduling decision is made independently of other sites according to its own performance goal.

2.3.3 Local Priority

It's another important issue. Each site within the Grid has its own scheduling policy. Certain classes of jobs have higher priority only on certain specific resources. For example, it can be expected that local jobs will be assigned higher priorities such that local jobs will be better served on the local resources [16].

2.3.4 Resource Non-Dedication

Because of non-dedication of resources, resource usage contention is a major issue. Competition may exist for both computational resources and interconnection networks. Due to non-dedication of resources, a resource may join multiple Grids simultaneously. The workloads from both local users and other Grids share the resource concurrently. The underlying interconnection network is shared as well. One consequence of contention is that behavior and performance may vary over the time; Contention free at the guaranteed level Schedulers must be able to consider the effects of contention and predict the available resource capabilities.

2.3.5 Application Diversity

This problem arises because the Grid applications are from a wide range of users, each having its own special requirements. For example, some applications may require sequential execution, some applications may consist of a set of independent jobs, and others may consist of a set of dependent jobs. In this context, building a general-purpose scheduling system seems extremely difficult. An adequate scheduling system should be able to handle a variety of applications [16].

2.3.6 Dynamic Behavior

In a Grid Environment, dynamics exists in both the networks and computational resources. First, a network shared by many parties cannot provide guaranteed

bandwidth. This is particularly true when wide areas networks such as the internet are involved. Second, both the availability and capability of computational resources will exhibit dynamic behavior. On one hand new resources may join the Grid and on other hand, some resources may become unavailable due to problems such as network failure. The capability of resources may vary overtime due to the contention among many parties who share the resources. An adequate Scheduler should adapt to such dynamic behavior. After a new resource joins the Grid, the Scheduler should be able to detect it automatically and leverage the new resources in the later scheduling decision making. When a computational resource becomes unavailable resulting from an unexpected failure, mechanisms such as rescheduling should be used to guarantee the reliability of Grid systems [16].

These challenges pose significant obstacles on the problem of designing an efficient and effective scheduling system for Grid environments.

2.4 Grid Scheduling Algorithms

The Grid Scheduling Algorithms can be classified as:

2.4.1 Local vs. Global

The local Scheduling discipline determines how the processes resident on a single CPU are allocated and executed; a global Scheduling policy uses information about the system to allocate processes to multiple processors to optimize a system wide performance objective. Grid Scheduling falls into the Global Scheduling [17].

2.4.2 Static vs. Dynamic

The next level in the hierarchy (under the Global Scheduling) is a choice between static and dynamic Scheduling. This choice indicates the time at which the Scheduling decisions are made. In case of static Scheduling, information regarding all resources in the Grid as well as all the tasks in an application is assumed to be available by the time the application is scheduled. By contrast, in the case of dynamic Scheduling, the basic idea is to perform task allocation on the fly as the application executes.

2.4.3 Optimal vs. Suboptimal

In the case that all information regarding the state of resources and the jobs is known, an optimal assignment could be made based on some criterion function, such as minimum makespan and maximum resource utilization. But due to difficulty in Grid scenarios to make reasonable assumptions which are usually required to prove the

optimality of an algorithm, current research tries to find suboptimal solutions, which can be further divided into the following two general categories

2.4.4 Approximate vs. Heuristic

The approximate algorithms use formal computational models, but instead of searching the entire solution space for an optimal solution, they are satisfied when a solution that is sufficiently "good" is found. The factors, which govern their decision, are:

- Availability of a function to evaluate a solution.
- The time required evaluating a solution.
- The ability to judge the value of an optimal solution according to some metric.
- Availability of a mechanism for intelligently pruning the solution space

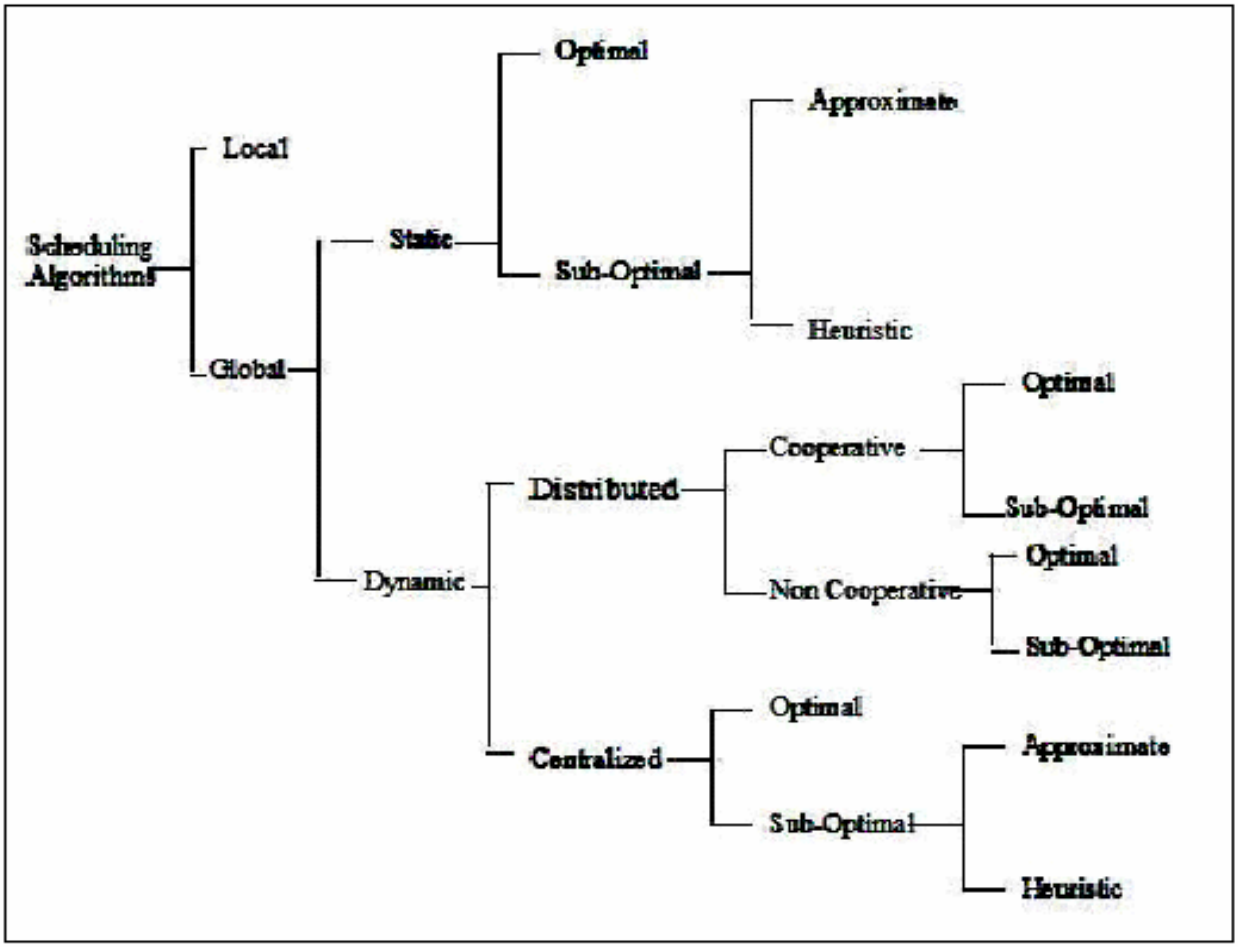


Figure 2.3: A Hierarchical taxonomy for scheduling algorithms [17]

Heuristic represents the class of algorithms, which make the most realistic assumptions about a priori knowledge concerning process and system loading characteristics. It represents the solutions to the Scheduling problem, which cannot give optimal answers and require the most reasonable amount of cost and other

system resources to perform their function. The evaluation of this kind of solution is usually based on experiments in the real world or on simulation. Heuristic algorithms are more adaptive to the Grid scenarios.

2.4.5 Cooperative vs. Non-cooperative

If a distributed Scheduling algorithm is adopted, the next issue that should be considered is whether the nodes involved in job Scheduling are working cooperatively or non-cooperatively. In the non-cooperative case, individual schedulers act alone as autonomous entities and arrive at decisions regarding their own optimum objects independent of the effects of the decision on the rest of system e.g. application-level schedulers. In the cooperative case, each Grid Scheduler has the responsibility to carry out its own portion of the Scheduling task, but all schedulers are working toward a common system-wide goal.

2.4.6 Distributed vs. Centralized

In dynamic Scheduling scenarios, the responsibility for making global Scheduling decisions may lie with one centralized Scheduler, or be shared by multiple distributed schedulers. The centralized strategy has the advantage of ease of implementation, but suffers from the lack of scalability, fault tolerance and the possibility of becoming a performance bottleneck [17].

2.5 Types of Grid Schedulers

Some schedulers are there which are popular. These can be discussed in the following way:-

2.5.1 Condor

Condor is workload management software that is developed by a research team located at the University of Wisconsin in Madison. The Condor research project started in 1988. There are three basic components in a Condor installation: a central manager, execution hosts, and submission hosts. A checkpoint server is an optional fourth component. The central manager serves two main functions in a condor cluster. The first function is collecting the status of all of the nodes in a Condor cluster. The second function is to match up resource requests for job submissions with a Condor node that can fulfill these requirements. Any node in a Condor cluster can be configured to be an execution machine and a submission machine, including the central manager. Execution machines are those nodes that can run Condor jobs and submission machines are those machines where Condor jobs can be submitted. An

optional checkpoint server can be added to a cluster to store all of the checkpoint files for the jobs in the cluster [10].

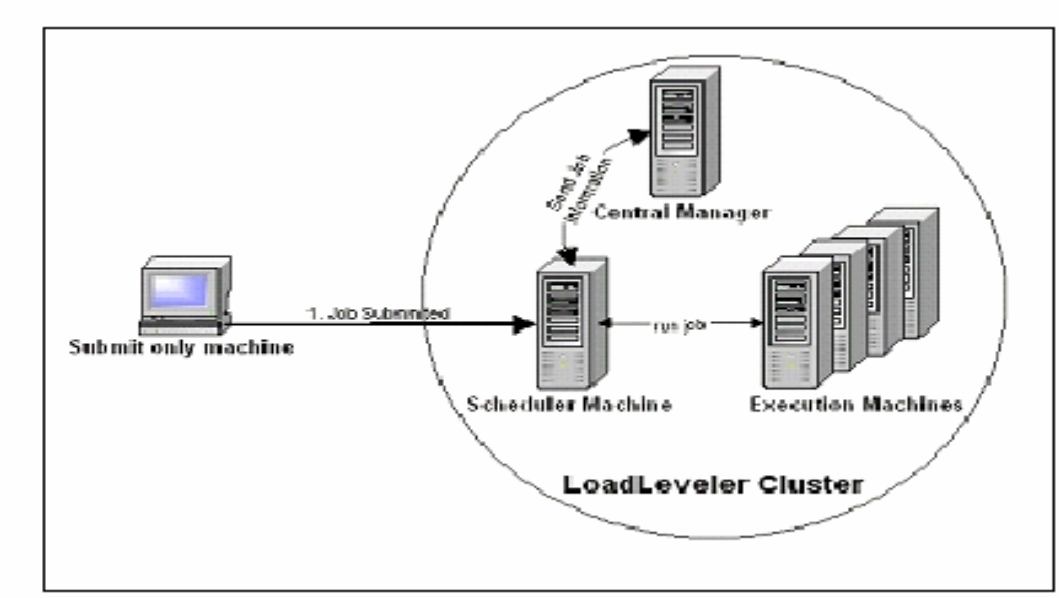


Figure 2.4: Condor Scheduler [10]

2.5.2 Load Leveller

Load Leveller is job management software available from IBM that can be used to schedule and load balance jobs across a cluster. Load Leveller is supported on AIX.. Load Leveller cluster can be made up of four different types of machine servers: scheduling, central manager, executing, and submitting machines. One machine in a cluster is designated as the central manager. It is responsible for monitoring the status of the other nodes and also matching up job requirements with nodes that satisfy them. Any number of machines may be designated as submit only machines. This is a workstation, which is only able to submit jobs to a cluster and run commands that monitor the status of jobs and kill jobs. A submit only machine is not able to run any jobs submitted to a Load-Leveler cluster. Scheduling machines are cluster nodes that manage a queue and are responsible for scheduling submitted jobs. An executing machine is the node were a given job is run. A single machine may have more than one role [10].

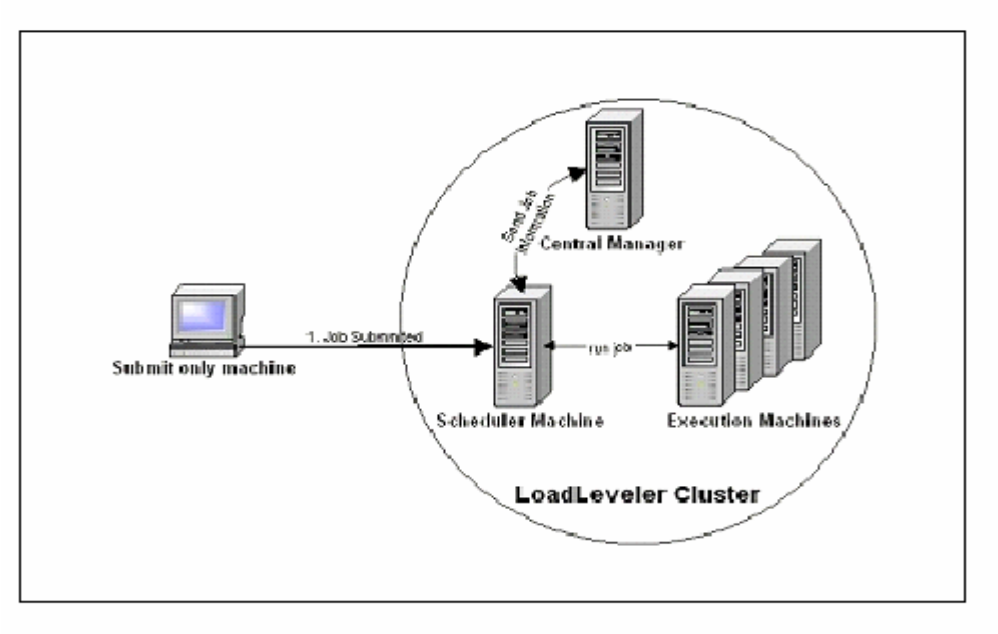


Figure 2.5: Load Leveller Scheduler [10]

2.5.3 PBS

Portable Batch System (PBS) is a scheduler that was developed for NASA Ames Research Center by Veridian. There is an open source version available, Open PBS, and a commercial version available, PBSPro. There is support available with the commercial version as well as additional features, such as better fault tolerance, advance reservation, and desktop cycle scavenging.

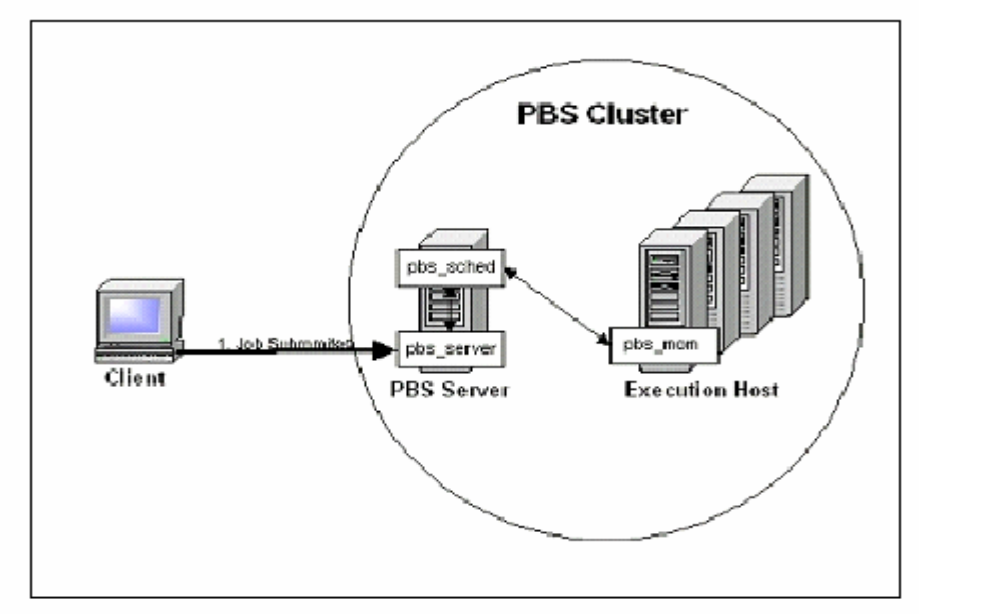


Figure 2.6: PBS Scheduler [10]

The source is also available for the commercial version. There are three main components that make up a PBS cluster:

- Job server
- Job executor
- Job scheduler

The job server is responsible for all batch processes in PBS, including creation, modification, running, and monitoring of batch jobs. A PBS cluster contains only one server machine. The job executer is the machine in the cluster where a batch job is running. The job scheduler is responsible for scheduling jobs on the cluster; it can query the status of execution nodes and also query the server to determine what jobs need to be run [10].

2.5.4 Sun Grid Engine (SGE)

Sun Grid Engine (SGE), previously known as CODINE (Computing in Distributed Networked Environments) or GRD (Global Resource Director), is an open source batch-queuing system, developed and supported by Sun Microsystems. Sun also sells a commercial product based on SGE, also known as N1 Grid Engine (N1GE).SGE is typically used on a computer farm or high-performance computing (HPC) cluster and is responsible for accepting, scheduling, dispatching, and managing the remote and distributed execution of large numbers of standalone, parallel or interactive user jobs. It also manages and schedules the allocation of distributed resources such as processors, memory, disk space, and software licenses.SGE is the foundation of the Sun Grid utility computing system, made available over the Internet in the United States in 2006, later becoming available in many other countries [36].

2.5.5 Globus Toolkit

The Globus Toolkit is an open source software toolkit used for building Grid systems and applications. It is being developed by the Globus Alliance and many others all over the world. A growing number of projects and companies are using the Globus Toolkit to unlock the potential of grids for their cause. The toolkit includes software services and libraries for resource monitoring, discovery, and management, plus security and file management. The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or together to develop applications [28].

Table 2.1: Comparison between Various Schedulers

Features	PBS	SGE	Loadleveller	Condor	Globus Toolkit
Single process preemptive jobs	No	Yes	Yes	Yes	Yes
Multi-process preemptive jobs	No	Yes	No	Yes	No
Costs	Open Source	Open Source	Open Source	Open Source	Open Source
Cycle Scavenging Grid	Yes	Yes	Yes	Yes	Yes
Policy based scheduling	Yes	Yes	Yes	Yes	Yes
Queue configuration	Minimal	Six	High	Very high	Medium
Backfill problem	No	Yes	Yes	No	No
Flocking technology	No	No	No	Yes	No
Queue optimization	Auto-matic	Not automat ic	Not automatic	Auto-matic	Not auto-matic

2.6 Fundamental Scheduling Algorithms of Operating System

There are various scheduling algorithms which are used by the operating system. Some of these algorithms are explained as:-

2.6.1 First Come First Serve (FCFS)

Also known as First-In, First-Out (FIFO). It is a Non-preemptive scheduling. In this scheduling algorithm the first process to request the CPU is the one that is allocated the CPU first. It is Very simple but it creates problem like Long average and worst-case waiting times, Poor dynamic behavior (convoy effect)

2.6.2 Shortest-Job-First (SJF)

Also known as Shortest remaining time. It may be either preemptive or Non-preemptive. Preemptive SJF scheduling is sometimes called shortest time first scheduling (STF). In this scheduling scheme the process with the shortest next CPU burst will get the CPU first. It minimizes average waiting times. But in this algorithm it is difficult to determine length of next CPU burst? Starvation of jobs with long CPU bursts [34].

2.6.3 Round Robin scheduling (RR)

It is a Preemptive scheduling. The Round-Robin scheduling scheme is similar to that of FCFS except preemption is added to it. In the RR scheduling scheme the CPU picks a process from the ready queue and sets a timer to interrupt after one time quantum. During this scheme two things may happen. The process may need less than one time quantum to execute. The process needs more than one time quantum.

2.6.4 Priority Scheduling (PS)

It can be Preemptive scheduling or Non-Preemptive. In this scheduling scheme a priority is associated with each process. Depending on the implementation of the algorithm there can be a range of priorities. The job that has the highest priority will be the one that is selected from the ready queue. This process will be allocated the CPU and the lower prioritized jobs will have to wait. But there is problem of Starvation arises. Starvation is caused when a process that is ready to be executed is not because it is still waiting for the CPU. The solution of this problem is Aging. Aging increases the priority of a process gradually [33].

Table 2.2: Comparison between Various Scheduling Algorithms

Features	**FCFS**	**RR**	**PS**	**SJF**
CPU Utilization	Low	High	Medium	Medium
Throughput	Low	Medium	Low	High
Turn around time	High	Medium	High	Medium
Response time	Poor	Good	Good	Medium
Waiting time	Low	Reduced	Depends	Depends upon burst length
Starvation	No	No	Yes	Yes
Overhead	Low	High	High	Medium

2.7 Conclusion

This chapter discussed all the existing schedulers and various scheduling algorithms, working under different strategies, policies and categories. Both static and dynamic algorithms give different performance results in customized environment. Static algorithms always give the constant performance but dynamic algorithms are preferred over static due to heterogeneous and dynamic environment. Next chapter, chapter 3 discusses the problem formulation.

Chapter 3

Proposed Scheduling Algorithm

In Grid environments, the shared resources are dynamic in nature, which inturn affects the application performance. Resource management and scheduling are the two main functions for this. To improve the throughput of the environment, an effective Scheduler is important and for the effective Scheduler an efficient algorithm is also fundamentally important.

3.1 Proposed Approach

In this thesis, the approach followed consists is based on Dynamic Time Quantum for the Round Robin Scheduling to improve the performance. The Scheduler will create a new Pool for the processes and execute them by using the Dynamic time Quantum scheme in which time Quantum is allocated to the processes dynamically. The major advantage of using Dynamic Time Quantum technique is that it will help in improving the performance and give the efficient results because an equal time of processor is allocated to each process which are in the Pool and according to the requirement of remaining processes a new time is allocated to them in order to improve the performance and produce the efficient results.

3.2 Round Robin Scheduling

Round-robin (RR) is one of the simplest scheduling algorithms for processes in an operating system, which assigns time slices to each process in equal portions and in circular order, handling all processes without priority. Round-robin scheduling is both simple and easy to implement, and starvation-free. Round-robin scheduling can also be applied to other scheduling problems, such as data packet scheduling in computer networks. But there is a problem in Round Robin scheduling as it is not applicable for the distributed environments. Therefore, Dynamic Time Quantum is an alternative of the Round Robin scheduling in the distributed environments. Equal time is allocated to all the processes as shown in Figure 3.1. All the processes execute completely with this time only.

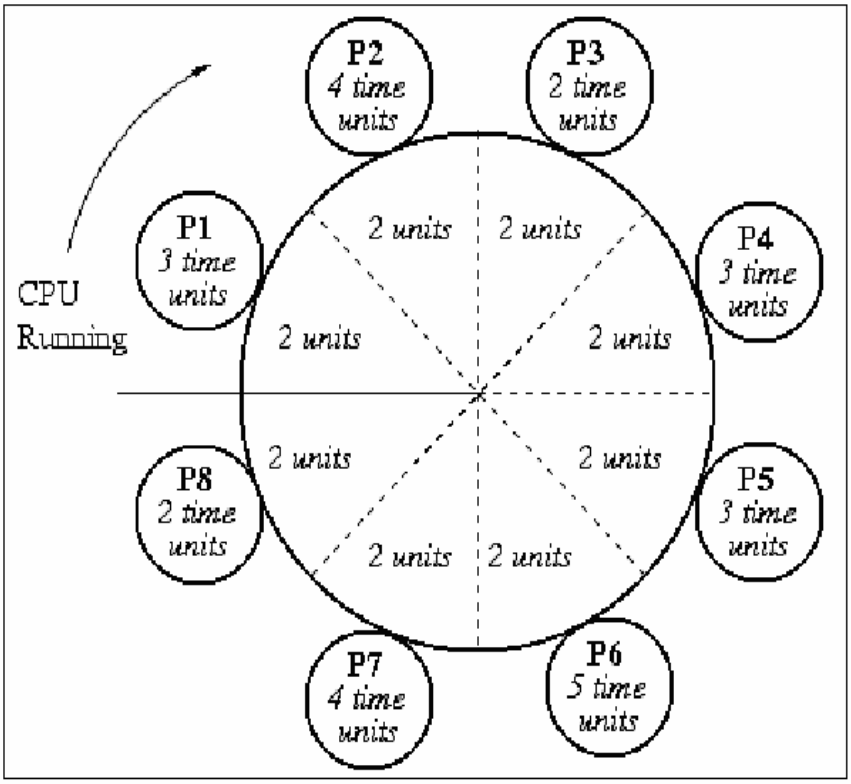


Figure 3.1: Round Robin Scheduling [34]

3.3 Dynamic Time Quantum Scheduling

In the Dynamic time Quantum scheduling, same time of processor is allocated to all the processes. After the execution a new time is allocated to all the remaining processes dynamically according to their requirements. In the dynamic time Quantum, equal priority is assigned to each process and then according to that another time Quantum is assigned for all the processes. Due to this every process get the processor time and gives the results in an appropriate time which improves the performance.

3.4 Proposed Algorithm

In this algorithm firstly all the processes are executed on the basis of given time Quantum. After that the remaining time of all the processes is calculated. On the basis this calculation a new time Quantum is allocated to the processes in order to execute them. After this, again calculation is required to calculate how much processor time is required by all the processes. On the basis of this calculation a mew time Quantum is allocated to all the processes. All processes get equal amount of time for their execution. Equal priority is considered for all the processes. This process is continued until all the processes complete their process.

3.5 Flow Chart of Proposed Algorithm

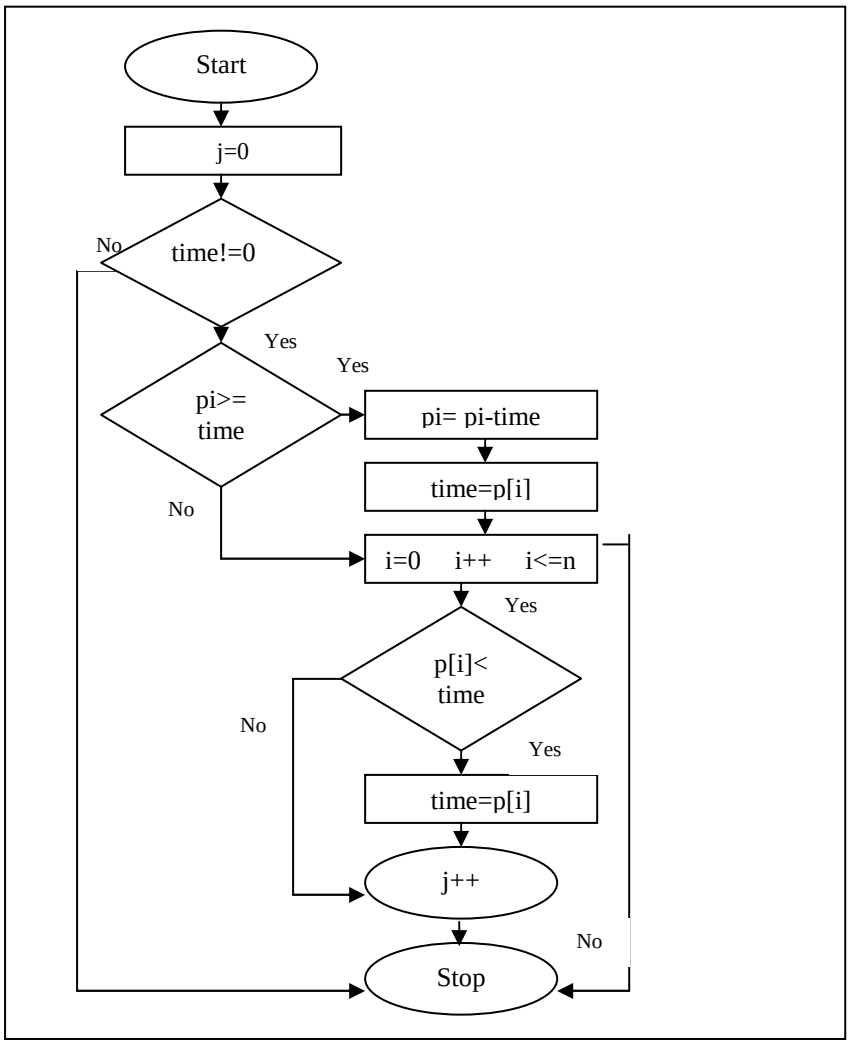


Figure 3.2: Flow chart for Dynamic time Quantum algorithm

Step1:- Jobs are allocated to the processor.

Step 2:- Then give the Time Quantum to all the processes for the execution.

Step 3:- Execute all the processes.

Step 4:- Check which processes are finished and which require more processor time.

Step 5:- Then according to the remaining processes requirement allocate dynamically time Quantum.

Step 6:- Repeat steps 3-5 until all jobs are not completed.

3.6 Pseudo code for Dynamic Time Quantum Scheduling algorithm

```
Begin
        Set Pi= Value,   where i is an array of
                          size n
        Set time= Value
        Set j=0
        While (time!=0)
                If (pi>=time)
                        Pi=pi-time
                End if
        Time=p[1]
        for i=1 to n by 1  do
                if(p[i]<time)
                    time=p[i]
                endif
        endfor
        j++
        endwhile
END
```

Firstly an array of processes are to be submitted.. Then assign the time Quantum to all the processes. Then allocate the processor to all the processes for equal interval of time. After that a new Quantum is assigned dynamicaaly to all the remaining processes. This step is continued until all the processes will be completed.

To allocate these jobs to the scheduler, firstly define the universe then define the path for the execution of the processes.. Then define output file, error report file and log file. After that there is a need to set the path for the execution. Then by using certain condor commands results of particular accounting domain or the pool can be seen.

3.7 Complexity of Proposed Scheduling Algorithm

Complexity is a measure of the performance of an algorithm in term of CPU time and memory usage. In this case computational complexity has been considered as this algorithm is for grid environment.

Computational Complexity:

Complexity = No. Of closed loop = 4;

Another formula to find out Complexity of algorithm

Complexity = No. of decision making statements + 1;

Complexity = 3 +1 =4;

3.7 Conclusion

This chapter discusses comparative study of Proposed algorithms, design and workflow of proposed algorithm. Next chapter discusses the installation steps of the scheduler and the experimental results of the algorithm.

Chapter 4

Implementation and Experimental Results

Dynamic Time Quantum Round Robin Algorithm has been implemented in JAVA programming language on Condor-7.5.1-winnt50-x86. The steps followed for the implementation are described below in detail with appropriate screenshots.

4.1 Installation of Pre-requisites and Necessary Components

4.1.1 Installation of Condor

In the current implementation of the Dynamic time Quantum Round Robin Algorithm we have used Condor-7.5.1-winnt50-x86. This open source JAVA based Scheduler can be downloaded from the page of download for Condor [26].

To use the Condor for the implementation of the Dynamic time Quantum Round Robin Algorithm we need to follow these steps; The Dynamic time Quantum Round Robin Algorithm is implemented in JAVA and run on Condor-7.5.1-winnt50-x86.

To install the Condor double click on Condor-7.5.1-winnt50-x86.exe file. A welcome screen will appear as shown in Figure 4.1. Follow the instructions and install the Condor.

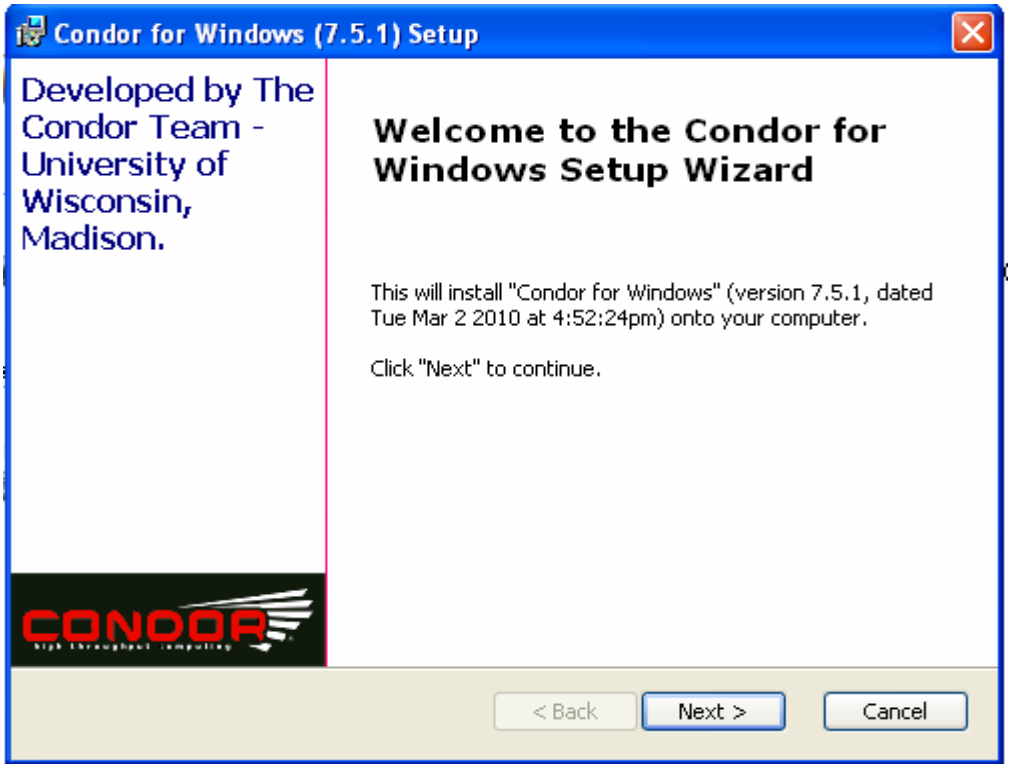


Figure 4.1: Welcome window of Condor

By default Condor installs in the C;\Condor directory. After installation, start the task manager. There are Condor files added like Condor_startd.exe under processes tasks as shown in Figure 4.2. If it appears then Condor is installed.

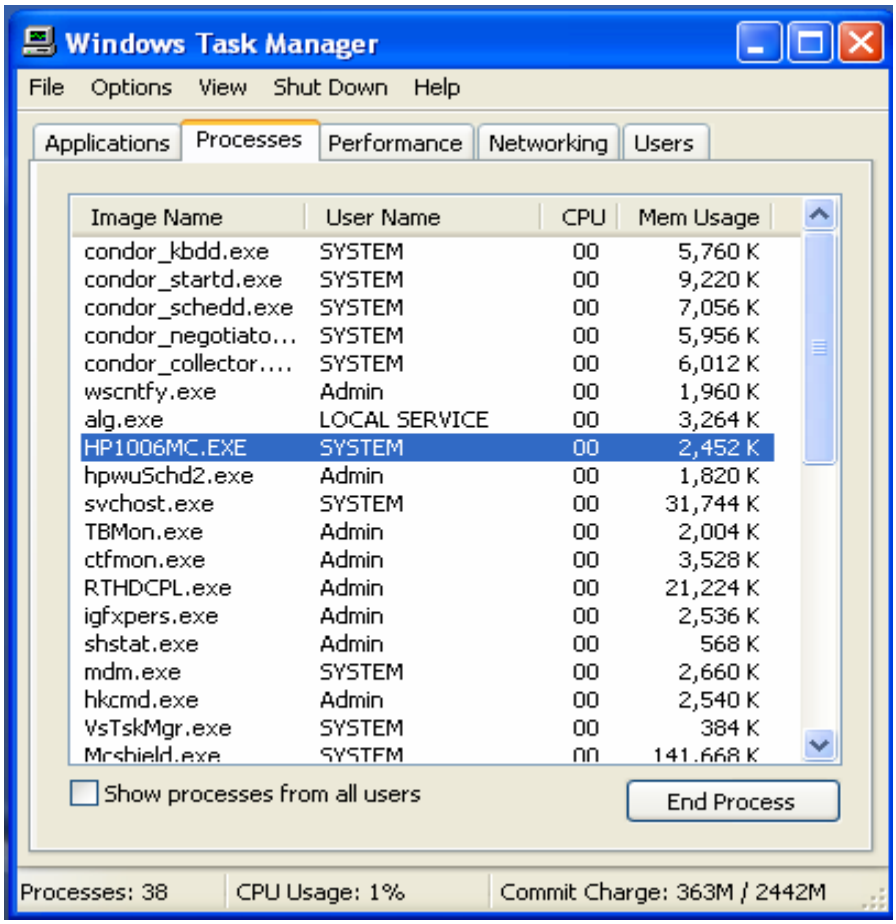


Figure 4.2: Showing the Condor files at startup

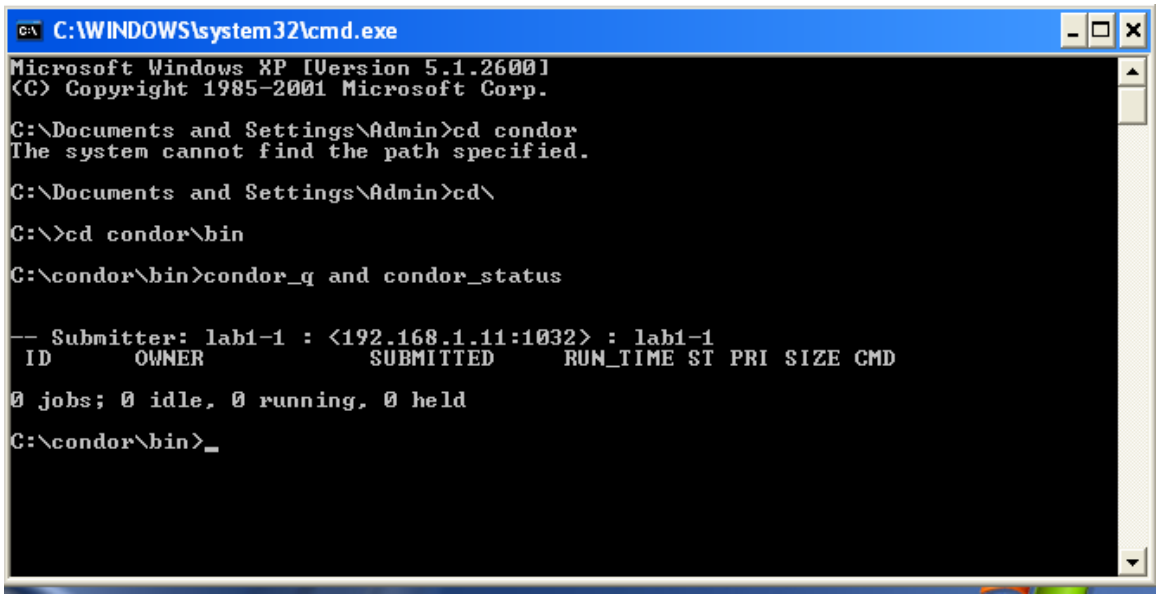


Figure 4.3:- Running view of Condor

4.2 Test Results

CASE 1:-Assume four processes arrived at time = 0, with burst time (P1 = 20, P2 = 40, P3 = 60, P4 = 80). Table 4.1 gives this description of all the processes in which includes the burst time and arrival time for all the processes. Table 4.2 shows the average waiting time and the average turnaround time for all the processes and also shows the comparison of the existing algorithms with the new proposed algorithm in terms of average waiting time and the average turnaround time. Then Figure 4.4 shows the average waiting time comparisons between all the existing algorithms and the new proposed algorithm graphically and Figure 4.5 shows the average turnaround time comparison between all the existing algorithms and the new proposed algorithm graphically

Table 4.1: Process description

Process	**Arrival Time**	**Burst Time**
P1	0	20
P2	0	40
P3	0	60
P4	0	80

Table 4.2: Average waiting time and turnaround time of all the Scheduling Algorithms

Scheduling Algorithms	**Average Waiting Time**	**Average Turnaround Time**
First-Come-First-Serve	50	100
Shortest-Job-First	50	100
Round Robin	70	120
Priority Scheduling	65	115
Dynamic time Quantum Round Robin	62.5	65

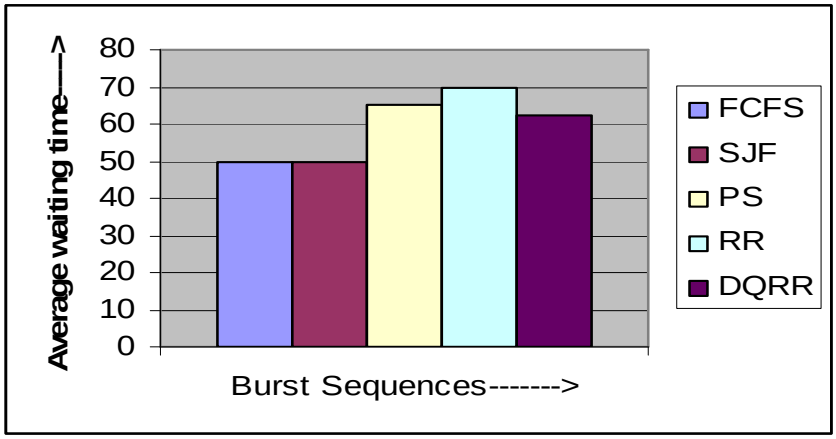


Figure 4.4:- Comparison of average waiting time

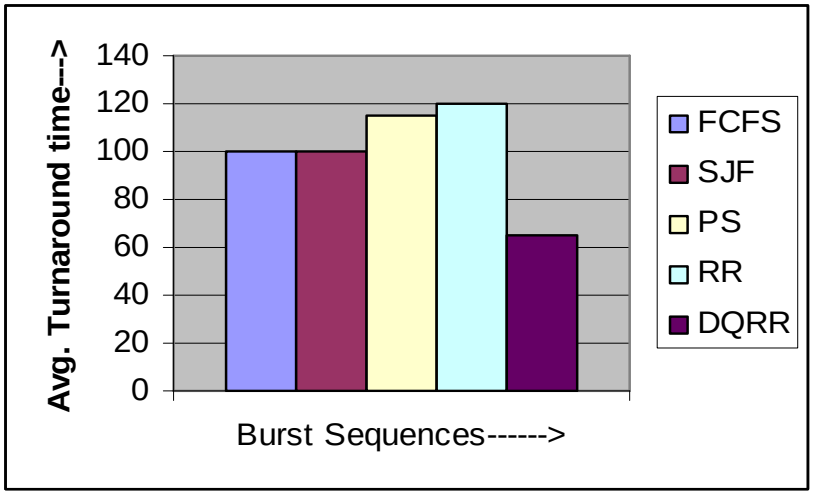


Figure 4.5:- Comparison of average turnaround time

CASE 2:-Assume three processes arrived at time = 0, with burst time (P1 = 5, P2 = 8, P3 = 4). Table 4.3 gives this description of all the processes in which includes the burst time and arrival time for all the processes. Table 4.4 shows the average waiting time and the average turnaround time for all the processes and also shows the comparison of the existing algorithms with the new proposed algorithm in terms of average waiting time and the average turnaround time. Then Figure 4.6 shows the average waiting time comparisons between all the existing algorithms and the new proposed algorithm graphically and Figure 4.7 shows the average turnaround time comparison between all the existing algorithms and the new proposed algorithm graphically

Table 4.3: Process description

Process	Arrival Time	Burst Time
P1	0	5
P2	0	8
P3	0	4

Table 4.4: Average waiting time and turn around time of all the Scheduling Algorithms

Scheduling Algorithms	Average Waiting Time	Average Turnaround Time
First-Come-First-Serve	6	11.66
Shortest-Job-First	4.33	10
Round Robin	7	14
Priority Scheduling	7	12.66
Dynamic time Quantum Round Robin	14	8.33

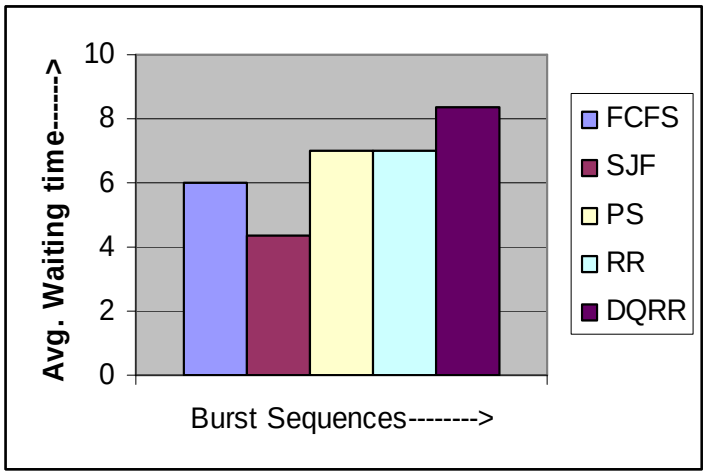


Figure 4.6:- Comparison of average waiting time

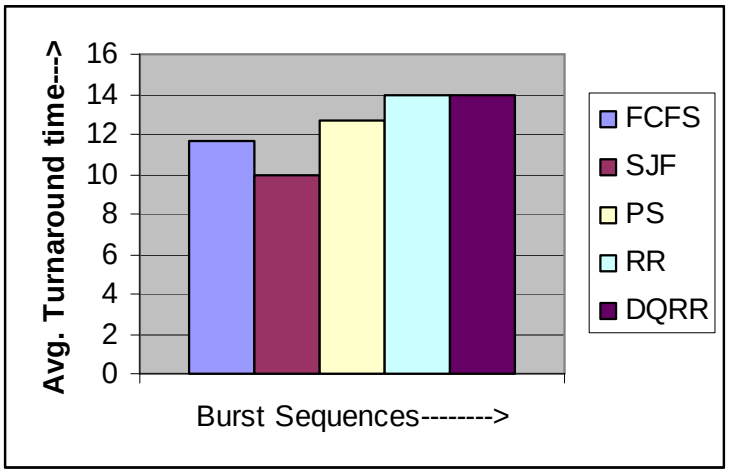


Figure 47:- Comparison of average turnaround time

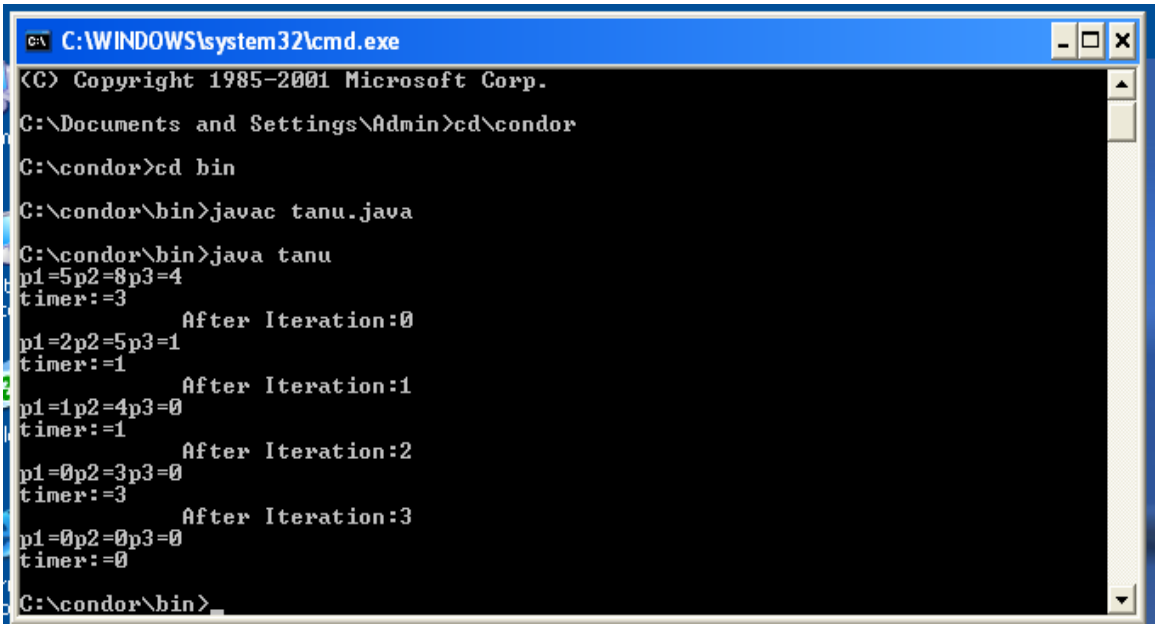


Figure 4.8: Result on Condor

CASE 3:-Assume three processes arrived at time = 0, with burst time (P1 = 15, P2 = 4, P3 = 2). Table 4.5 gives this description of all the processes in which includes the burst time and arrival time for all the processes. Table 4.6 shows the average waiting time and the average turnaround time for all the processes and also shows the comparison of the existing algorithms with the new proposed algorithm in terms of average waiting time and the average turnaround time. Then Figure 4.9 shows the average waiting time comparisons between all the existing algorithms and the new proposed algorithm graphically and Figure 4.10 shows the average turnaround time comparison between all the existing algorithms and the new proposed algorithm graphically

Table 4.5: Process description

Process	Arrival Time	Burst Time
P1	0	15
P2	0	4
P3	0	2

Table 4.6: Average waiting time and turn around time of all the Scheduling Algorithms

Scheduling Algorithms	Average Waiting Time	Average Turnaround Time
First-Come-First-Serve	11.33	8.33
Shortest-Job-First	2.66	9.66
Round Robin	6	13
Priority Scheduling	7.66	14.66
Dynamic time Quantum Round Robin	4.66	11.66

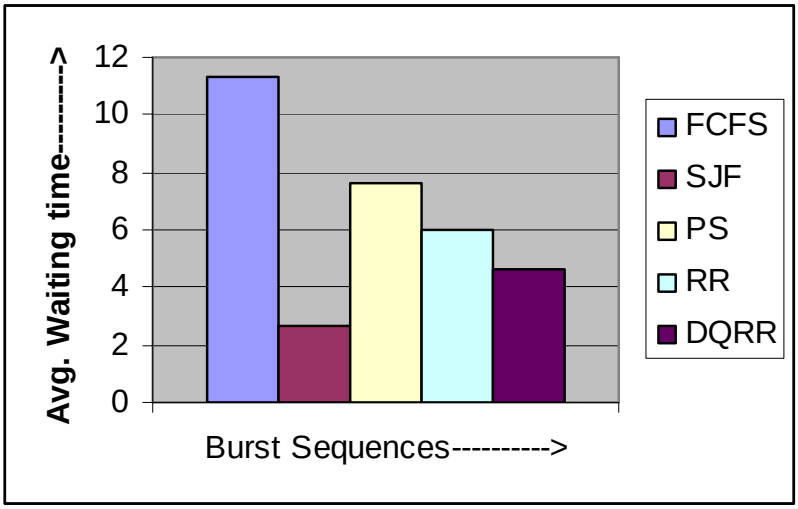


Figure 4.9:- Comparison of average waiting time

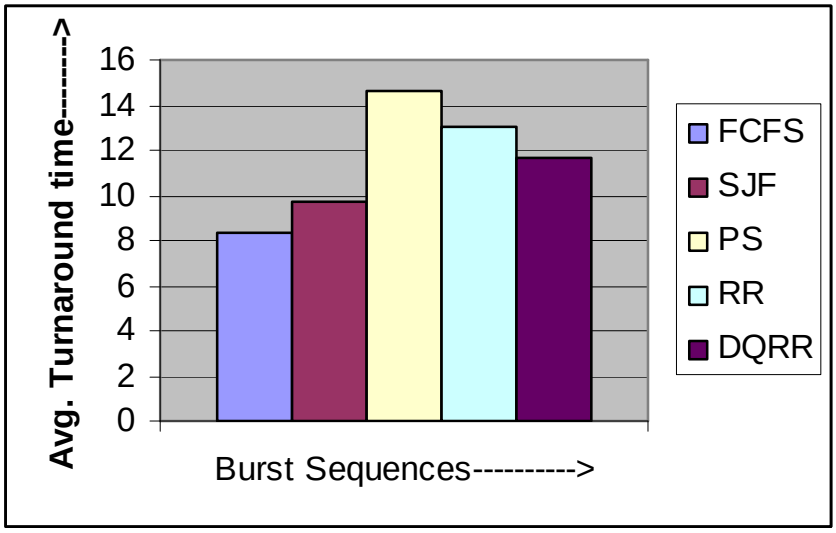


Figure 4.10:- Comparison of average turnaround time

CASE 4:-Assume four processes arrived at time = 0, with burst time (P1 = 14, P2 = 10, P3 = 20, P4 = 5). Table 4.7 gives this description of all the processes in which includes the burst time and arrival time for all the processes. Table 4.8 shows the average waiting time and the average turnaround time for all the processes and also shows the comparison of the existing algorithms with the new proposed algorithm in terms of average waiting time and the average turnaround time. Then Figure 4.11 shows the average waiting time comparisons between all the existing algorithms and the new proposed algorithm graphically and Figure 4.12 shows the average turnaround time comparison between all the existing algorithms and the new proposed algorithm graphically

Table 4.7: Process description for

Process	Arrival Time	Burst Time
P1	0	14
P2	0	10
P3	0	20
P4	0	5

Table 4.8: Average waiting time and turn around time of all the Scheduling Algorithms

Scheduling Algorithms	Average Waiting Time	Average Turnaround Time
First-Come-First-Serve	20.5	32.75
Shortest-Job-First	12.25	24.5
Round Robin	18.5	35.75
Priority Scheduling	13.75	26
Dynamic time Quantum Round Robin	18.75	35.75

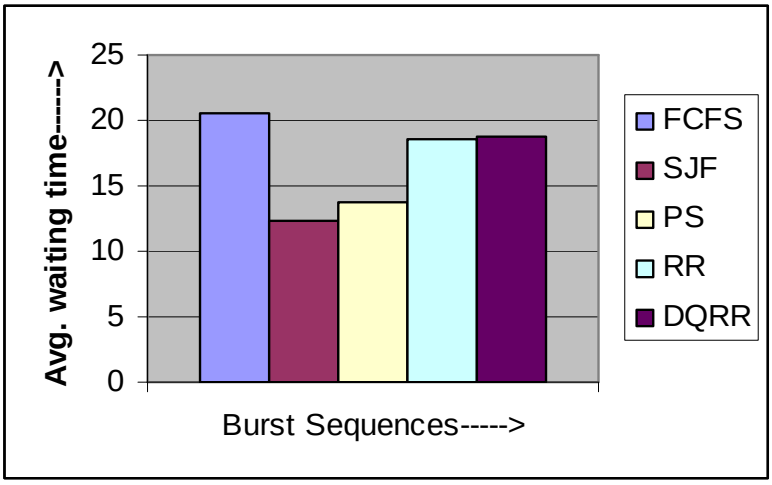


Figure 4.11:- Comparison of average waiting time

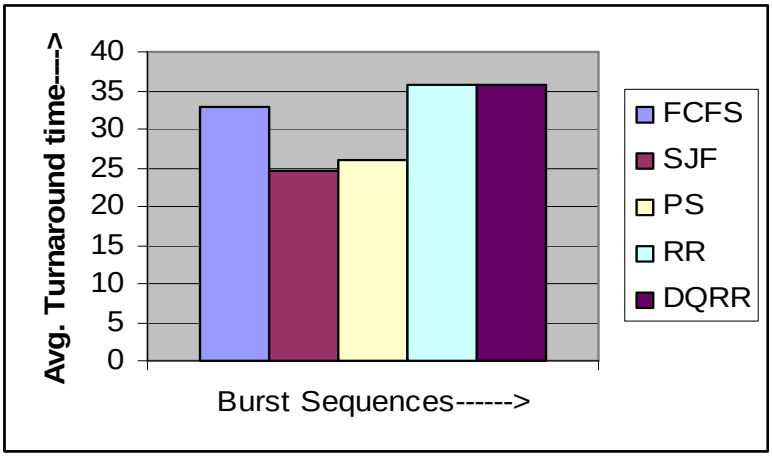


Figure 4.12:- Comparison of average turnaround time

4.3 Conclusion

This chapter discussed the steps of installing the Condor Scheduler and experimental results of proposed algorithm.

Chapter 5

Conclusion and Future Scope

This chapter discusses the conclusions of the work presented in this thesis. The chapter ends with a discussion of the future directions.

5.1 Conclusion

The work presented in this thesis describes multiple aspects of grid computing and introduces numerous concepts which illustrate its grand capabilities. Grid Computing is definitely a promising tendency to solve high demanding applications and its related problems. Main objective of the grid environment is to achieve high performance.

Dynamic nature and complexity of Grid makes scheduling very complex There are a number of factors, which can affect the grid application performance like Scheduling, heterogeneity of resources and resource sharing in the Grid environment.

This Thesis primarily focuses on scheduling the jobs in an optimized manner. The user submits the numbers of jobs to the scheduler. Then these jobs are executed by the scheduler on the basis of Dynamic time Quantum. This technique allocates the best time Quantum to the processes for improving their performance. At the end of the execution all the processes have to be executed completely which are submit by the users. This work demonstrates the efficiency of the proposed algorithm through experimental results and validation with traditional algorithms.

5.2 Future Scope

- Instead of Condor scheduler, this algorithm can be run on another scheduler. After the installation of the scheduler the same algorithm can be run on it and the results can be matched.
- The same algorithm can be coded in other languages. The performance of that new algorithm can be validated with the performance this existing algorithm.
- There are various operating systems on which the Condor scheduler can be installed. This can be done by following the instructions given in the manual and the same work can be done on it.

References

[1] Foster, I., Kesselman, C. and Tuecke, S. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations" International Journal of High Performance Computing applications, Vol. 15, No. 3, pp. 200-222, 2001.

[2] Moore's Law Explained by Intel
http://www.intel.com/research/silicon/mooreslaw.htm>

[3] "Gilder's law on network performance. Telecosm: The World After Bandwidth Abundance", Gilder, G. ed., Touchstone Books, ISBN: 0743205472, 2002

[4] Klein rock, L. "MEMO on Grid Computing", University of California, Los Angeles, 1969.

[5] "Grid Computing - Making the Global Infrastructure a Reality", F. Berman, G. Fox and T. Hey, John Wiley & Sons, Ltd, ISBN: 0-470-85319-0,2002.

[6] Ya-Fing Zhung, Jizhou Sun, Jian-Bo Ma, "Self Management Model Based on multiagent and Worm Techniques" CCECE 2004

[7] Li, L. and Horrocks, I, "A Software Framework for Matchmaking based on Semantic Web Technology", In Proceedings of the 12th international conference on World Wide Web (WWW'03), Budapest, Hungary, May 2003.

[8] C.Kesselman. I. Foster. Computational grids. In The Grid: Blueprint for a New Computing Infrastructure., chapter 2. Morgan-kaufman edition, 1999.

[9] Lord, P.Alper, P.Wroe, C. and Goble, C., "Feta: A lightweight Architecture for User Oriented Semantic Service Discovery", In Proceedings of The Semantic Web: Research and Applications: Second European Semantic Web Conference (ESWC 2005), Heraklion, Crete.

[10].Ferreira, L., Bieberstein, N., Berstis, V., Armstrong, J., "Introduction to Grid Computing with Globus," Redbook, IBM Corp.,
http://www.redbooks.ibm.com/redbooks/pdfs/sg246895.pdf.

[11]. Hai Zhuge "Special Section: Semantic Grid and Knowledge Grid" 17 July 2006; accepted 20 July 2006.

[12]. Ann Chervenak and others, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets", Journal of Network and Computer Applications, 2001.

[13] Jean Christophe Durand, "Grid Computing: A Conceptual and Practical Study", University de Lausanne, Nov 8, 2004.

[14].María S. Pérez "Grid Computing: A Ten Years Look Back",Facultad de Informática Universidad Politécnica de Madrid mperez@fi.upm.es.

[15] J. Joseph, C. Fellenstein, "Grid Computing", Prentice Hall/IBM Press, Edition 2004

[16] <http://www.unix.cs.anl.gov/~schopf/ggf-sched/wd/scedwd.8.5.pdf>

[17] Fangpeng Dong and Selim G. Akl, "Scheduling Algorithms for Grid Computing: State of the Art and Open Problems", School of Computing, Queen's University Kingston, Ontario, Technical Report No. 2006-504, January 2006.

[18] http://jcharts.sourceforge.net/>

[19]. John Hagel, III and John Seely Brown, "Service Grids: The Missing Link in Web Services", A Working Paper,

http://www.johnhagel.com/paper_servicegrid.pdf, 2002.(accessed:18September 2005).

[20].Srikumar Venugopal, Rajkumar Buyya and Ramamohanarao Kotagiri "A Taxonomy of Data Grids for Distributed Data Sharing, Management and Processing" Grid Computing and Distributed Systems Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Australia Email:fsrikumar, raj, raog@cs.mu.oz.au.

[21] N. Tonellotto, Information Science and Technologies Institute Italian National Research Council Italy, R. Yahyapour Institute for Robotics Research University of Dortmund Germany, "A Proposal for a Generic Grid Scheduling Architecture".

[22] Jarek Nabrzyski, Jennifer M. Schopf & Jan Weglarz, Grid Resource Management- State of the art and Future trends, Kluwer Academic Publisher.

[23] Klaus Krauter, Rajkumar Buyya and Muthucumaru Maheswaran, Taxonomy and survey of Grid Resource Management Systems for Distributed Computing. Software Practice and Experience Softw. Pract. Exper. 2002.

[24] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, "Operating System Principles Seventh Edition".

[25] Yanmin ZHU, A Survey of Grid Scheduling, Department of Computer Science Hong Kong University of Science and Technology.

[26] http://www_cs.wisc.edu/condor

[27].http://www.globus.org/toolkit/docs/development/4.1.2.

[28] Allen, G., Angulo, D., Foster, I., and others: "The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment. Journal of

High-Performance Computing Applications", Volume 15, no. 4 (2001).

[29] Rafael A. Moreno "Job Scheduling and Resource Management Techniques in Dynamic Grid Environments".

[30] Lichen Zhang, "Scheduling Algorithm for Real-Time Applications in Grid Environment", 2002 IEEE SMC TP1K5.

[31] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, "Operating System Principles Seventh Edition".

[32] Ann Chervenak and others, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets", Journal of Network and Computer Applications, 2001.

[33] Stallings, William (2004). Operating Systems Internals and Design Principles (fifth international edition). Prentice Hall. ISBN 0-13-147954-7.

[34] http://images.google.co.in/images.

[35] http://en.wikipedia.org/wiki/Sun_Grid_Engine

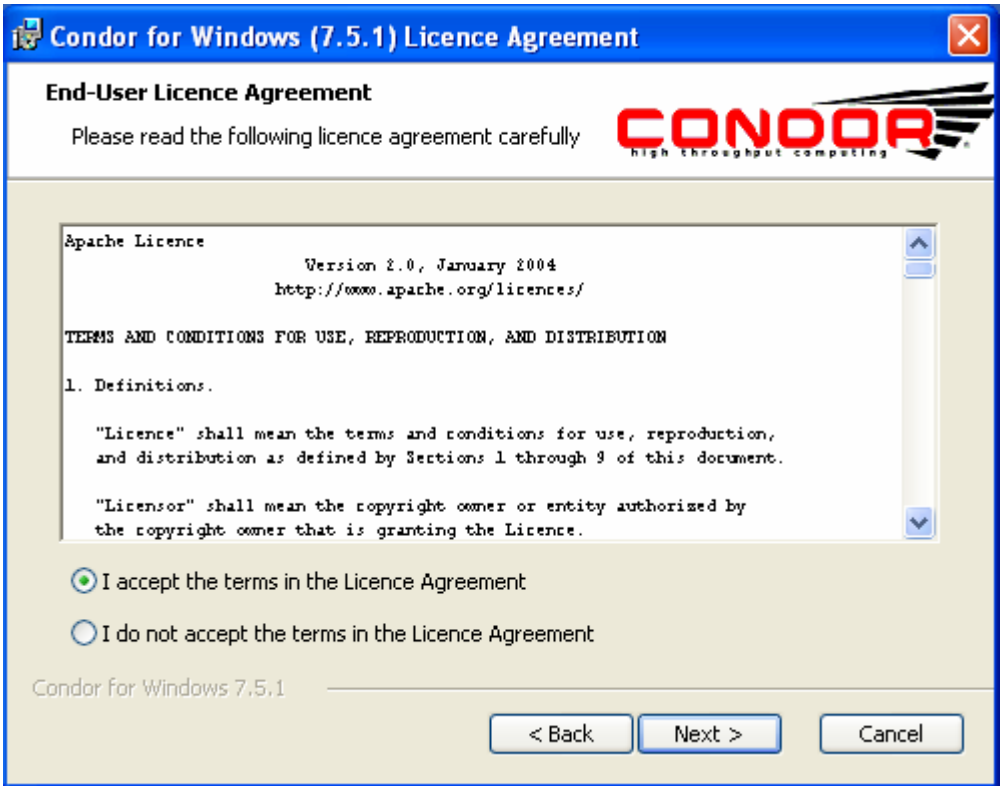
Communicated Papers

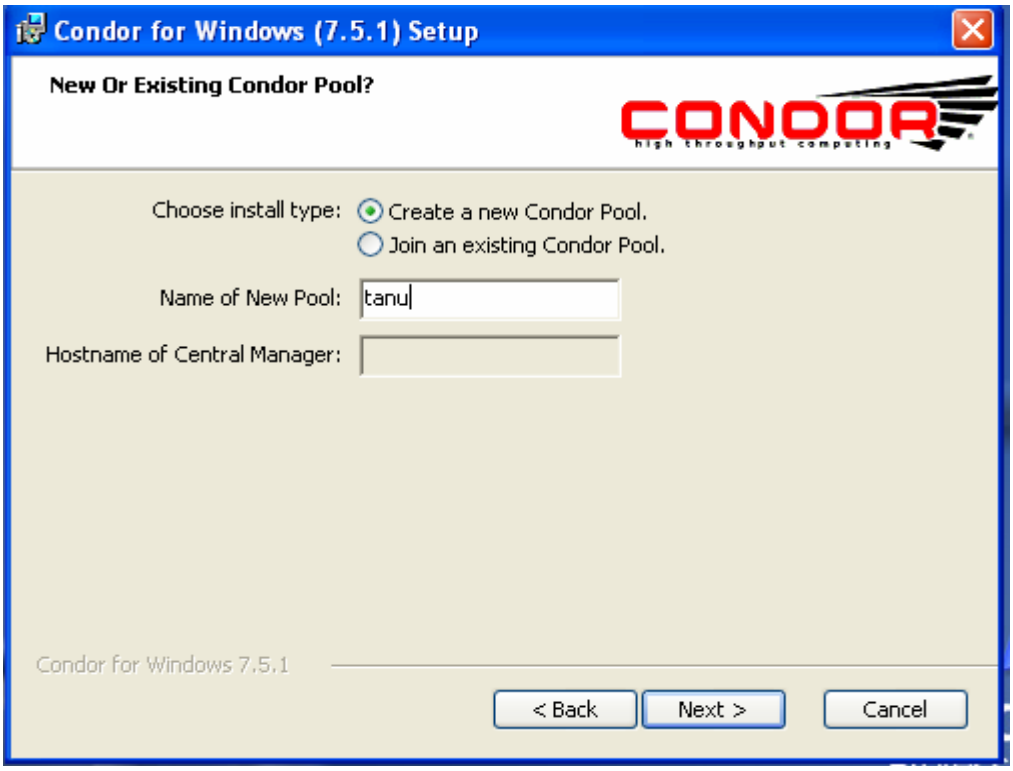
1. Tanu, Dr. Inderveer chana, "Round Robin with varying Quantum scheduling algorithm adapted for Grid environment", International Conference on Computer ad communication Technology, on Sept 17-19, 2010, Allahabad, India.
Proceedings would be published.

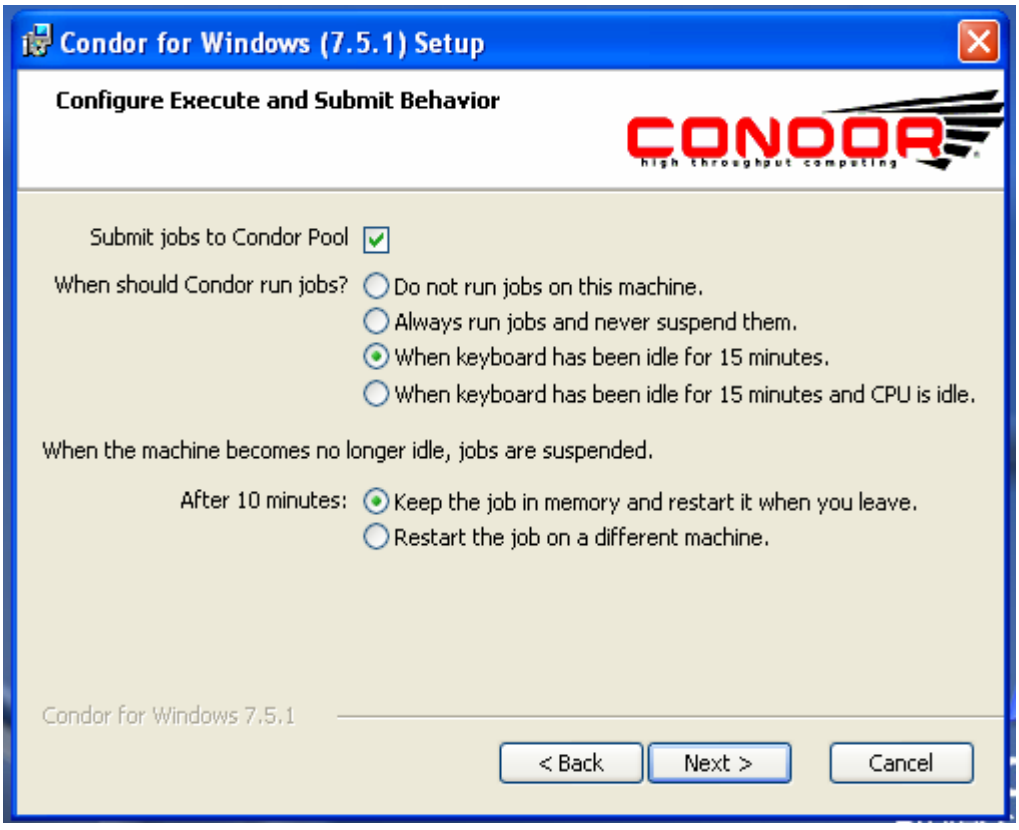
2. Tanu, Dr. Inderveer Chana, "Analysis and design of a efficient Dynamic time Quantum algorithms adapted for Grid schedulers", 1st International Conference on Parallel, Distributed and Grid Computing (PDGC), on Oct 28-30, 2010, JUIT, Solan, India.
Proceedings would be published.

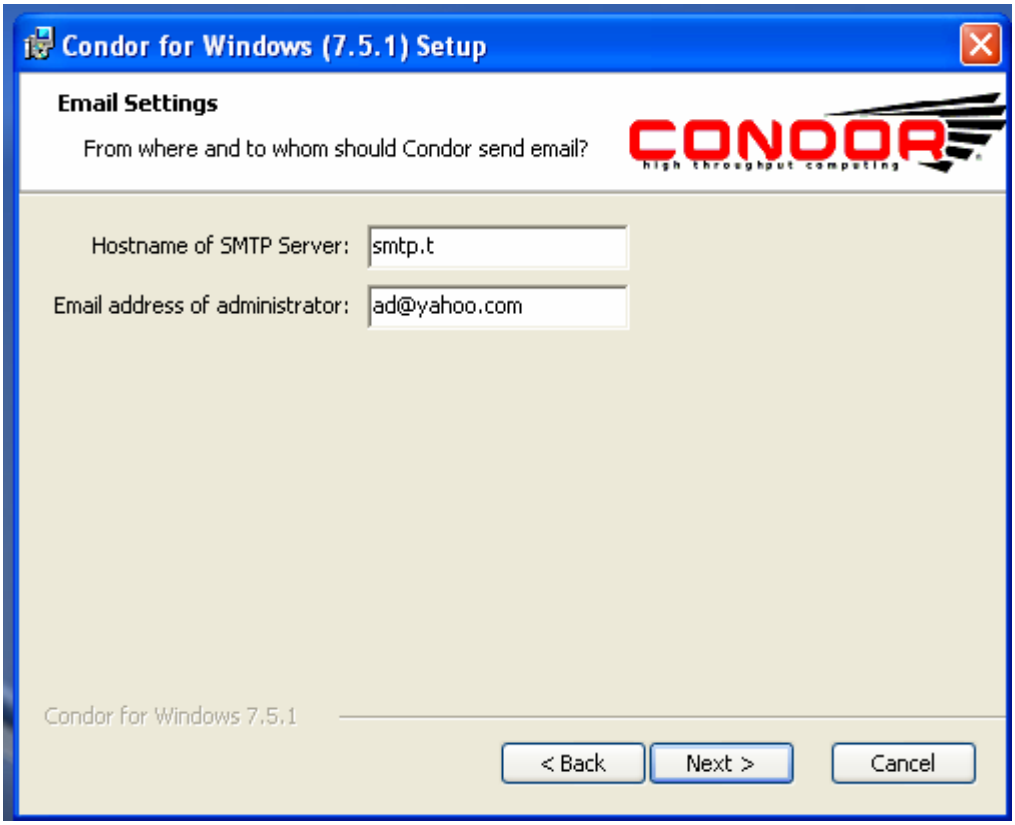
Appendix-A

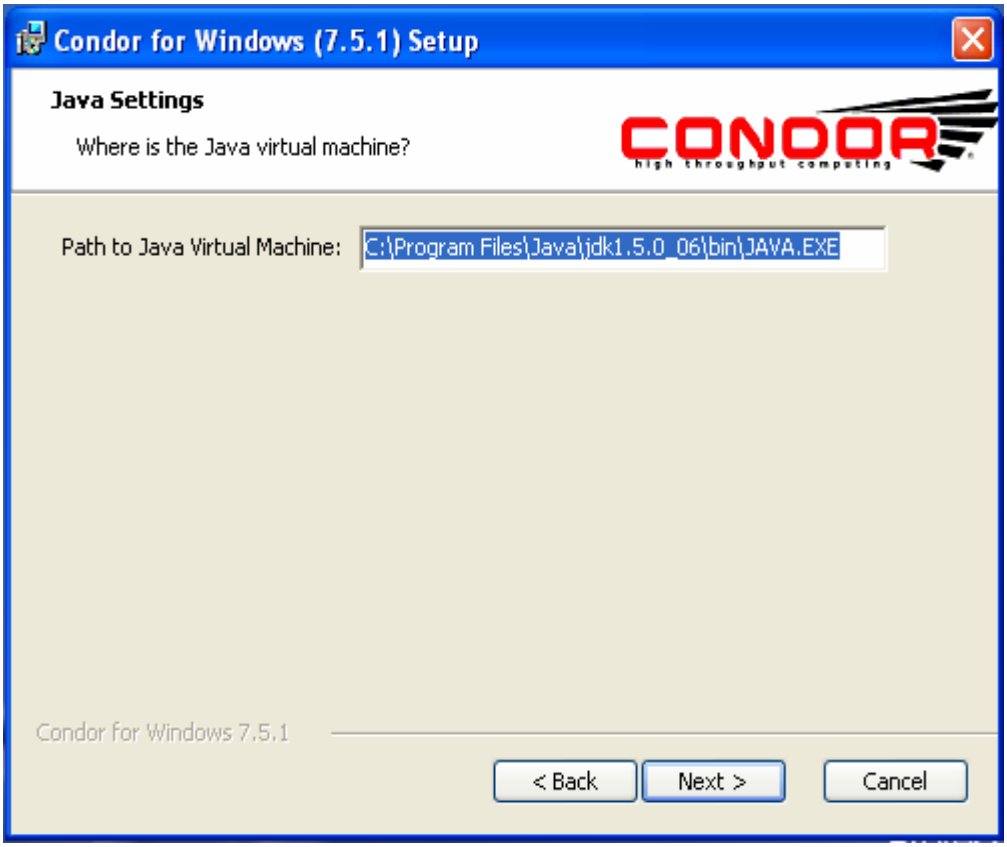
Installation Steps of Condor

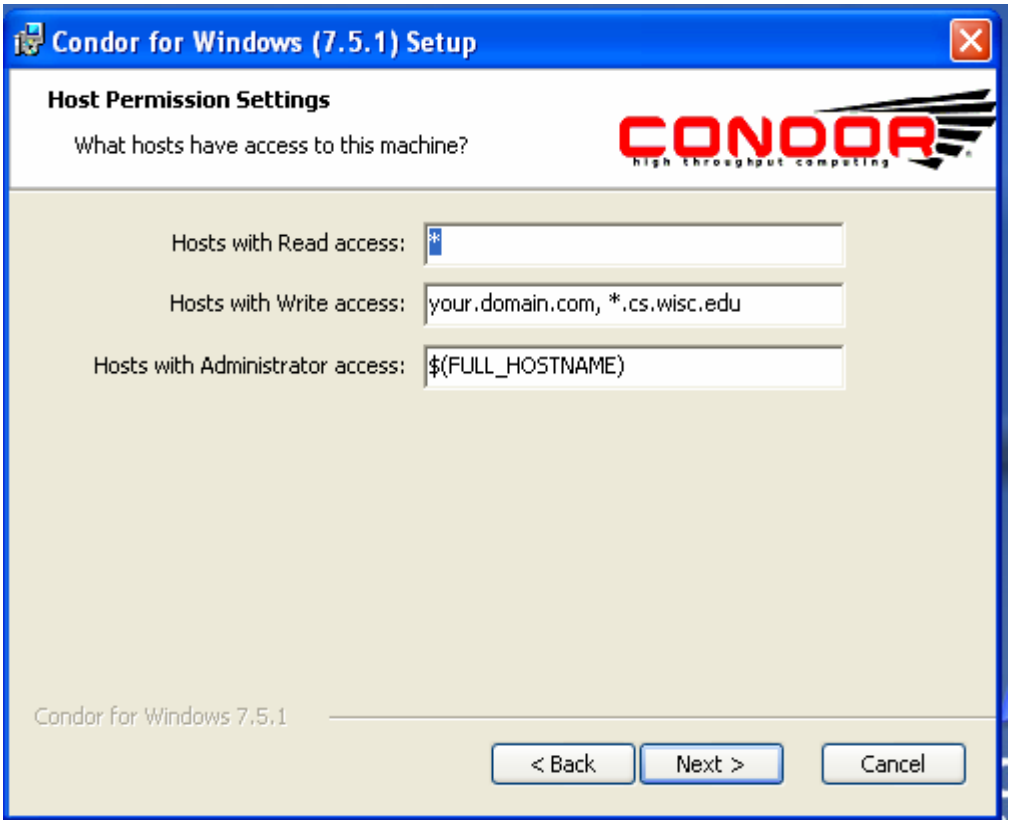


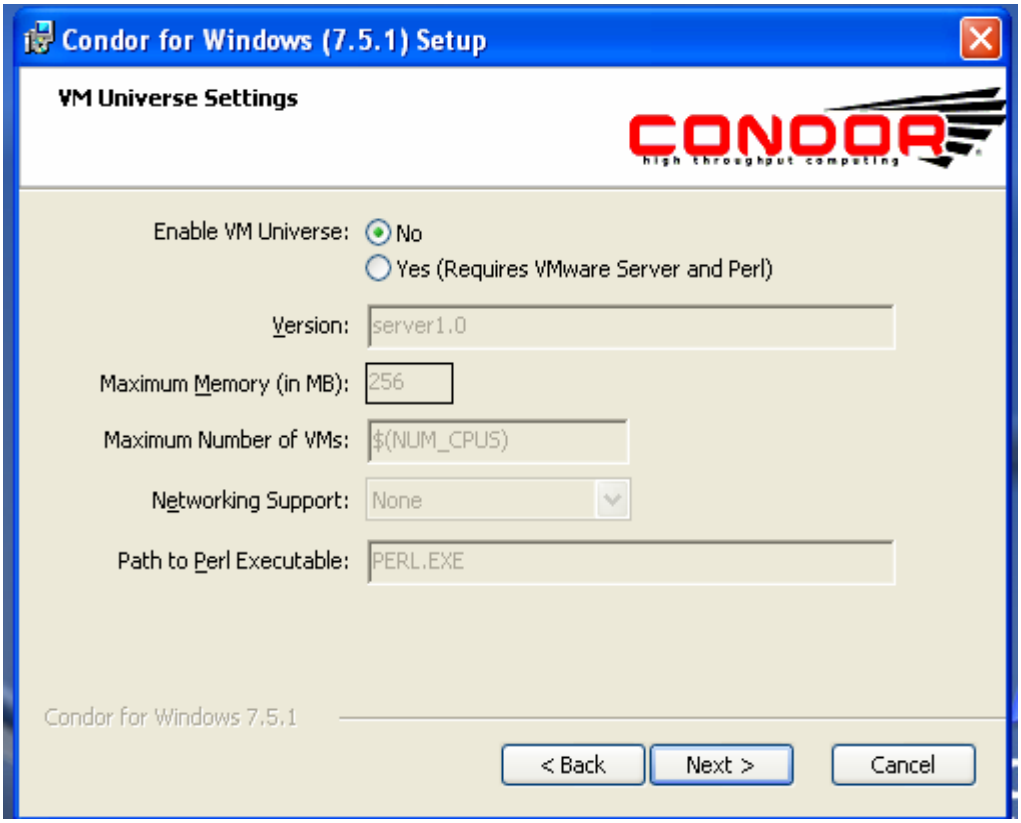

Condor for Windows (7.5.1) Setup
New Or Existing Condor Pool?
CONDOR
high throughput computing
Choose install type:
Create a new Condor Pool.
Join an existing Condor Pool.
Name of New Pool:
tanu
Hostname of Central Manager:
Condor for Windows 7.5.1
< Back
Next >
Cancel

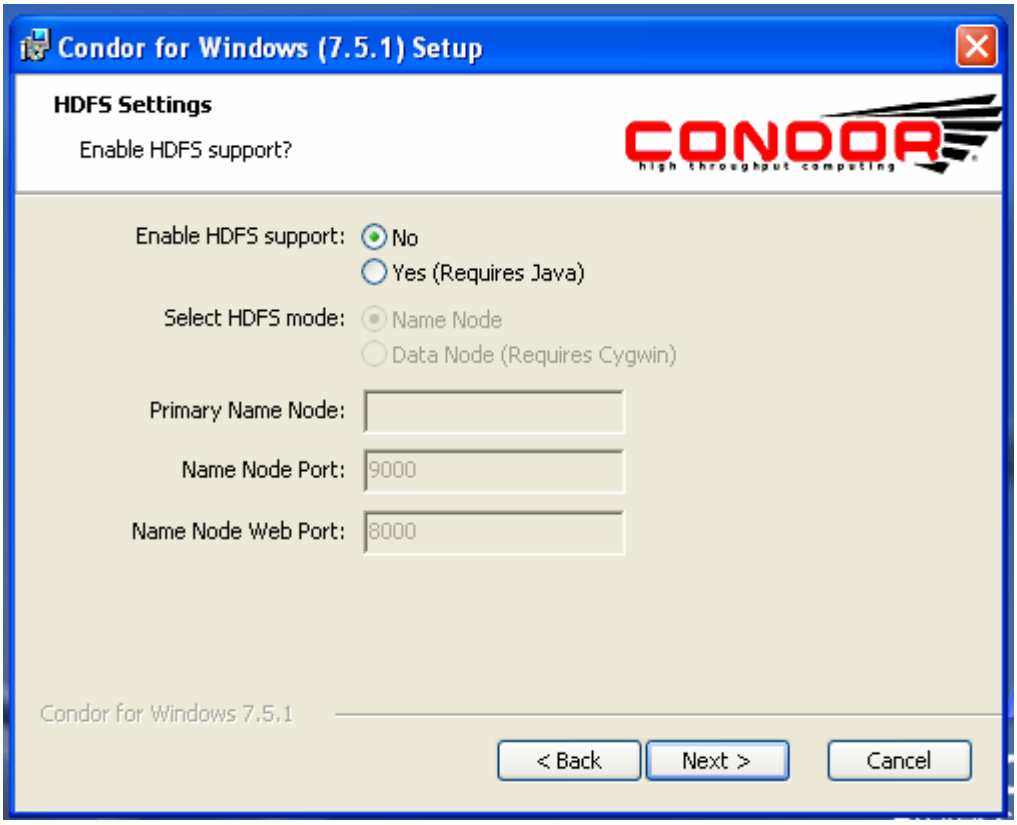

Condor for Windows (7.5.1) Setup
Configure Execute and Submit Behavior
CONDOR
high throughput computing
Submit jobs to Condor Pool
When should Condor run jobs?
Do not run jobs on this machine.
Always run jobs and never suspend them.
When keyboard has been idle for 15 minutes.
When keyboard has been idle for 15 minutes and CPU is idle.
When the machine becomes no longer idle, jobs are suspended.
After 10 minutes:
Keep the job in memory and restart it when you leave.
Restart the job on a different machine.
Condor for Windows 7.5.1
< Back
Next >
Cancel

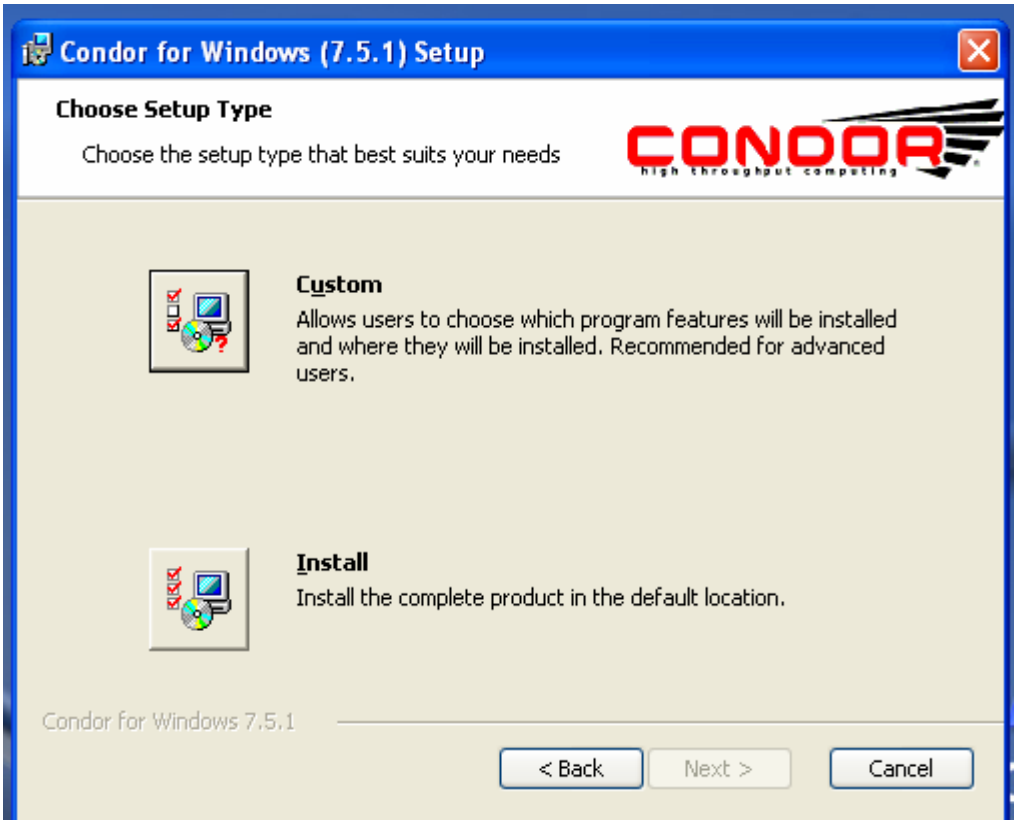

Condor for Windows (7.5.1) Setup
Email Settings
From where and to whom should Condor send email?
CONDOR
high throughput computing
Hostname of SMTP Server:
smtp.t
Email address of administrator:
ad@yahoo.com
Condor for Windows 7.5.1
< Back
Next >
Cancel

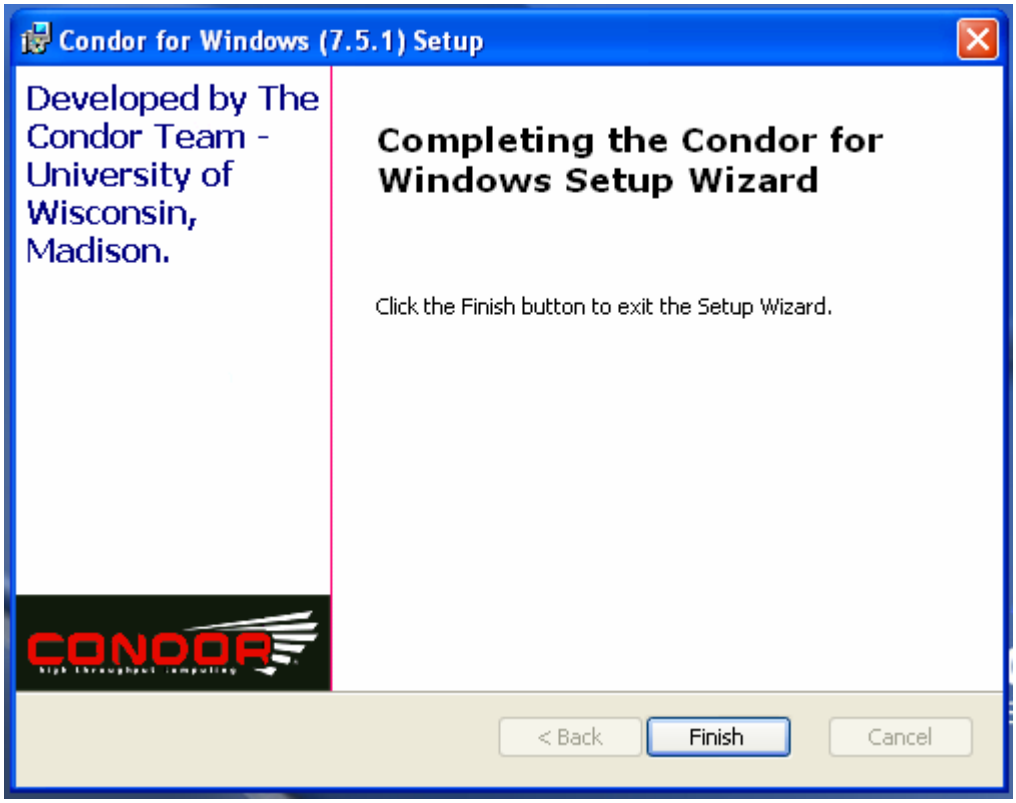

Condor for Windows (7.5.1) Setup
Java Settings
Where is the Java virtual machine?
CONDOR
high throughput computing
Path to Java Virtual Machine:
C:\Program Files\Java\jdk1.5.0_06\bin\JAVA.EXE
Condor for Windows 7.5.1
< Back
Next >
Cancel

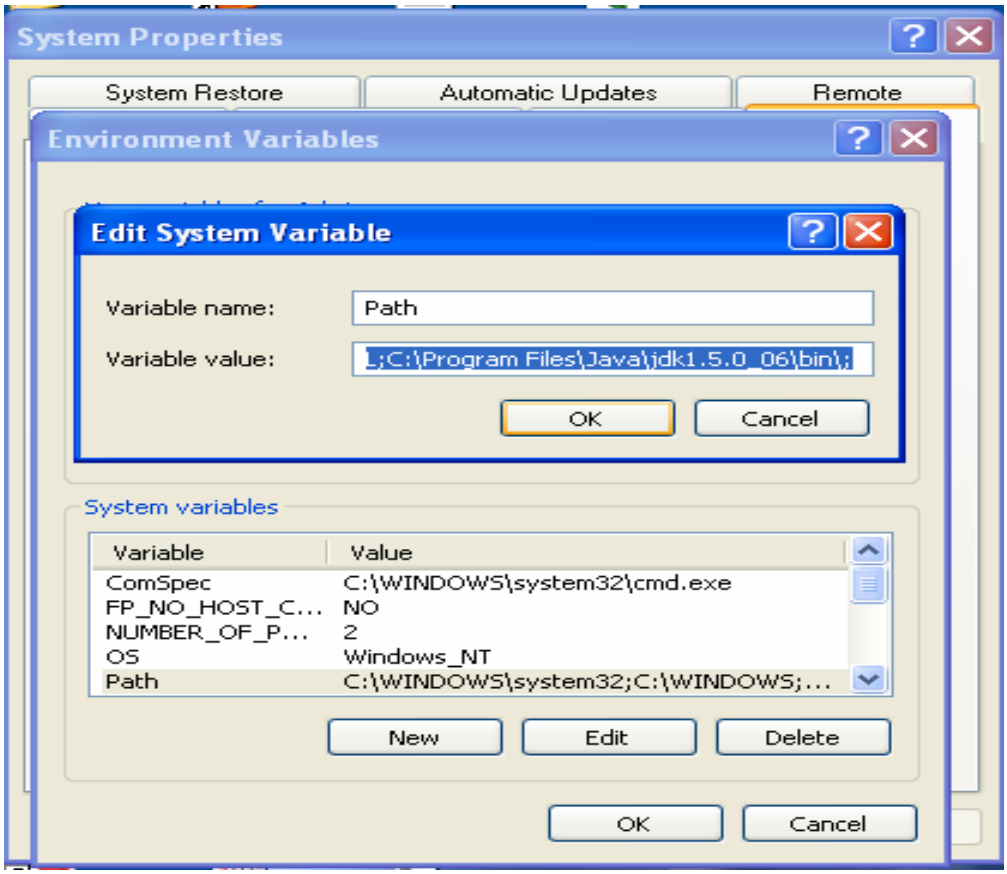

Condor for Windows (7.5.1) Setup
Host Permission Settings
What hosts have access to this machine?
CONDOR
high throughput computing
Hosts with Read access: *
Hosts with Write access: your.domain.com, *.cs.wisc.edu
Hosts with Administrator access: $(FULL_HOSTNAME)
Condor for Windows 7.5.1
< Back
Next >
Cancel


Condor for Windows (7.5.1) Setup
VM Universe Settings
CONDOR
high throughput computing
Enable VM Universe: No
Yes (Requires VMware Server and Perl)
Version: server1.0
Maximum Memory (in MB): 256
Maximum Number of VMs: $(NUM_CPUS)
Networking Support: None
Path to Perl Executable: PERL.EXE
Condor for Windows 7.5.1
< Back
Next >
Cancel


Condor for Windows (7.5.1) Setup
HDFS Settings
Enable HDFS support?
CONDOR
high throughput computing
Enable HDFS support:
No
Yes (Requires Java)
Select HDFS mode:
Name Node
Data Node (Requires Cygwin)
Primary Name Node:
Name Node Port:
9000
Name Node Web Port:
8000
Condor for Windows 7.5.1
< Back
Next >
Cancel


Condor for Windows (7.5.1) Setup
Choose Setup Type
Choose the setup type that best suits your needs
CONDOR
high throughput computing
Custom
Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.
Install
Install the complete product in the default location.
Condor for Windows 7.5.1
< Back
Next >
Cancel


Condor for Windows (7.5.1) Setup
Developed by The Condor Team - University of Wisconsin, Madison.
Completing the Condor for Windows Setup Wizard
Click the Finish button to exit the Setup Wizard.
CONDOR
< Back
Finish
Cancel


System Properties
System Restore
Automatic Updates
Remote
Environment Variables
Edit System Variable
Variable name:
Path
Variable value:
;C:\Program Files\Java\jdk1.5.0_06\bin\;
OK
Cancel
System variables
Variable
Value
ComSpec
C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_C...
NO
NUMBER_OF_P...
2
OS
Windows_NT
Path
C:\WINDOWS\system32;C:\WINDOWS;...
New
Edit
Delete
OK
Cancel