

P-2831



A SIMPLE SCHEDULING HEURISTICS FOR HETEROGENEOUS ENVIRONMENTS

A PROJECT REPORT

Submitted by

S.PREM SAGAR

71205104034

R.RAJA

71205104037

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY, CHENNAI-600 025

MAY 2009



BONAFIDE CERTIFICATE

Certified that this project report entitled “**A Simple Scheduling Heuristics For Heterogeneous Environments**” is the bonafide work of S. Prem Sagar and R. Raja, who carried out the research under my supervision. Certified also, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.


SIGNATURE

Dr.S.Thangasamy, PhD

Head Of the Department


SIGNATURE

Mrs.P.Devaki

Asst.Professor


Department of

Computer Science & Engineering,

Kumaraguru College of Technology,

Coimbatore -641006.

The candidates with University Register Nos. 71205104034 and 71205104037 were examined by us in the project viva-voce examination held on 28.04.2009


INTERNAL EXAMINER


EXTERNAL EXAMINER

DECLARATION

We hereby declare that the project entitled " **A Simple Scheduling Heuristics For Heterogeneous Environments**" is a record of original work done by us and to the best of our knowledge, a similar work has not been submitted to Anna University or any Institutions, for fulfillment of the requirement of the course study.

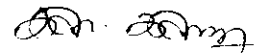
The report is submitted in partial fulfillment of the requirement for the award of the Degree of Bachelor of Computer Science and Engineering of Anna University, Chennai.

Place: Coimbatore

Date:



(S.Prem Sagar)



(R.Raja)

ACKNOWLEDGEMENT

We extend our sincere thanks to our Vice Principal, **Prof. R.Annamalai, M.E.**, Kumaraguru College of Technology, Coimbatore, for being a constant source of inspiration and providing us with the necessary facility to work on this project.

We would like to make a special acknowledgement and thanks to **Dr. S. Thangasamy, Ph.D.**, Dean, Professor and Head of Department of Computer Science &Engineering, for his support and encouragement throughout the project.

We express deep gratitude and gratefulness to **our Guide Mrs.P.Devaki (M.E)** Department of Computer Science &Engineering, for her supervision, enduring patience, active involvement and guidance.

We would like to convey our honest thanks to **all Faculties** of the Department for their enthusiasm and wealth of experience from which we have greatly benefited.

We also thank our **friends and family** who helped us to complete this project fruitfully.

Abstract

Efficient task scheduling of computationally intensive applications is one of the most essential and difficult issues when aiming at high performance in heterogeneous computing environments. The application scheduling problem has been shown to be NP-complete in general cases. Although a large number of scheduling heuristics have been presented in the literature, most of them target only homogeneous computing systems. In this project **Heterogeneous Task Scheduling (HTS)**, we present a simple Job Scheduling algorithm for any number of dependent tasks which finds the better makespan than the existing algorithms [1][2][4][5]. The analysis and experiments have shown that HTS provides comparable or even better results together with low complexity. In order to provide a robust and unbiased comparison with the existing algorithms, a parametric graph generator was designed to generate weighted directed acyclic graphs with various characteristics [1]. The weight matrix is generated using simulation model [3]. The comparison study, based on the above said randomly generated graphs. The parametric study involves speedup, frequency of best results, and average make span.

TABLE OF CONTENTS

CHAPTER -- I

1. Overview of Grid Computing	-	1
1.1 Introduction	-	1
1. 2. Grid Construction	-	3
1.3 Grid Components	-	4
1.4 Grid Computing Lexicon	-	6
1.5 Types of Grids	-	8
1.6. Key benefits of the Grid Computing Model	-	9
1.7 Characteristics and Capabilities	-	10
1.8 Grid computing Vs Cluster computing	-	11
1.9 Distributed vs. Grid Computing	-	12
1.10 Grid Concepts and Components	-	14
1.11 Job Scheduling	-	16
1.12. Issues in Grid Computing	-	19
1.13 The Grid Taxonomy and Grid Architecture	-	20

CHAPTER II

2. Overview of Job Scheduling	-	26
2.1 Job Scheduling in Grid Computers	-	26
2.2 Classification of Static task-Scheduling algorithms	-	27
2.3 Job Scheduling in a Heterogeneous Grid Environment	-	28
2.4 Job Scheduling Policy for High Throughput	-	28

CHAPTER III

3. Problem Overview	-	29
3.1 Problem Definition	-	29
3.2 Task Graph Representation	-	29
3.3 Application Representation using Task Graph	-	30
3.4 Representation of Weight Matrix	-	30
3.5 Existing Algorithms taken up for comparison	-	31
3.5.1 Critical Path On a Processor (CPOP)	-	31
3.5.2 Heterogeneous Critical Parent Trees (HCPT)	-	33
3.6 Proposed Algorithm (HTS)	-	35

CHAPTER IV

4. Implementation	-	37
4.1 Input weight matrix generation	-	37
4.2 Graph construction	-	40

CHAPTER V

5. Experimental Results and Discussion	-	41
5.1. Comparison Metrics	-	41

CHAPTER VI

6. Comparison Graphs	-	42
6.1 Makespan Comparison	-	42
6.2 Speedup Comparison	-	46
6.3 Number of favorable cases for 1000 trails	-	50
6.4 Conclusion	-	54
6.5 References	-	54

Abbreviations:

HTS	-	Heterogeneous Task Scheduling
HCPT	-	Heterogeneous Critical Parent Trees
CPOP	-	Critical Path on a Processor
AEST	-	Average Earliest Start Time
ALST	-	Average Latest Start Time
EEFT	-	Earliest Execution Finish Time
ETC	-	Expected Time to Compute
CCR	-	Communication to Computation Ratio
QoS	-	Quality of Service
OGSA	-	Open Grid Service Architecture
GRAM	-	Grid Resource Allocation Manager
MDS	-	Monitoring and Directory Service
GRIS	-	Grid Resource Information Service
GIIS	-	Global Index Information Service
GASS	-	Global Access to Secondary Storage

CHAPTER – I

1. Overview of Grid Computing

1.1 Introduction

1.1.1 Grid Computing

Grid computing is a form of distributed computing that involves coordinating and sharing computing, application, data, storage, or network resources across dynamic and geographically dispersed organizations. Grid technologies promise to change the way organizations tackle complex computational problems.

Grid computing enables the virtualization of distributed computing and data resources such as processing, network bandwidth and storage capacity to create a single system image, granting users and applications seamless access to vast IT capabilities.

Grid computing is based on an open set of standards and protocols — e.g., **Open Grid Services Architecture (OGSA)** — that enable communication across heterogeneous, geographically dispersed environments

1.1.2 What is Grid Computing?

"A Grid is a collection of distributed computing resources available over a local or wide area network that appears to an end user or application as one large virtual computing system." – **IBM**

"Conceptually, a grid is quite simple. It is a collection of computing resources that perform tasks. In its simplest form, a grid appears to users as a large system that provides a single point of access to powerful distributed resources." - **Sun**

"Grid computing is computing as a utility - you do not care where data resides, or what computer processes your requests. Analogous to the way utilities work, clients request information or computation and have it delivered - as much as they want, and whenever they want." - **Oracle**

Grid computing represents an enabling technology that permits the dynamic coupling of geographically dispersed resources (machines, networks, data storage, visualization devices, software and scientific instruments) for performance-oriented distributed applications in science, engineering, medicine and e-commerce. However, it is a difficult task to agree on a concrete definition of Grid Computing, as different commercial and academic implementations use the word for a fairly wide spectrum of architectures. It is generally agreed in the literature that there are two important goals which are the driving force behind grid computing.

The first goal is to build up a **computational and networking infrastructure** that is designed to provide pervasive, uniform and reliable access to data, computational and human resources distributed over wide area environments. So a grid should bring together a diverse collection of different hardware and software technologies, different corporations, people and procedures do build a shared pool of resources.

The second and more distant goal behind grid computing is the **delivery of computing power** as a utility, like the electrical system. Actually the name 'Grid' comes from an analogy from power grids that supply electricity. When somebody needs electricity, he plugs in a device to the system which uses as much resources as it needs. The end user is not concerned with the details like which power plant is supplying the electricity at that moment or lack of power if he buys a hi-fi system. By analogy the home computer in the future will consist only **Human Computer Interface (HCI)** and the computing power will be provided by the grid. If the user will use a word processor, it will draw 1 MIPS and if he needs to do an elaborate scientific calculation it will draw 100 MIPS. People will pay as much as they use, just like the electricity or the water system.

Computational Grid is a collection of distributed, possibly heterogeneous resources which can be used as an ensemble to execute large-scale applications.

Grid applications include,

- **Distributed Supercomputing**
 - Distributed Supercomputing applications couple multiple computational resources – supercomputers and/or workstations

- **High-Throughput Applications**

- Grid used to schedule large numbers of independent or loosely coupled tasks with the goal of putting unused cycles to work. High-throughput applications include RSA key cracking, detection of extra-terrestrial communication.

- **Data-Intensive Applications**

- Focus is on synthesizing new information from large amounts of physically distributed data. Examples include NILE (distributed system for high energy physics experiments using data from CLEO), SAR/SRB applications, digital library applications

1. 2. Grid Construction

There are three main issues that characterize computational grids:

Heterogeneity

A grid involves a multiplicity of resources that are heterogeneous in nature and might span numerous administrative domains across wide geographical distances.

Scalability

A grid might grow from few resources to millions.

Dynamicity or Adaptability

With so many resources in a Grid, the probability of some resource failing is naturally high.

The steps necessary to realize a computational grid include:

- The integration of individual software and hardware components into a combined networked resource.
- The implementation of middleware to provide a transparent view of the resources available.

- The development of tools that allows management and control of grid applications and infrastructure.

1.3 Grid Components

1.3.1 Grid Fabric

It comprises all the resources geographically distributed and accessible from anywhere on the Internet. They could be computers, clusters, storage devices, databases, and special scientific instruments such as a radio telescope.

1.3.2 Grid Middleware

It offers core services such as remote process management, co-allocation of resources, storage access, information, security, authentication, and Quality of Service (QoS) such as resource reservation and trading.

A Grid Middleware Component is a piece of software integrated in one or more grid middleware distributions which provides a service, implements protocols and algorithms with the objective of allowing users access to the grid's distributed resources.

1.3.2.1 Categories of Grid Middleware Components

- List of Security Infrastructure Grid Middleware components
- List of Information System Grid Middleware components
- List of Computing Element Grid Middleware components
- List of Job Management Grid Middleware components
- List of Data Management Grid Middleware components
- List of Monitoring Grid Middleware components
- List of Accounting Grid Middleware components
- List of other Grid Middleware components

1.3.3 Grid Development Environments and Tools

These offer high-level services that allow programmers to develop applications and brokers that act as user agents that can manage or schedule computations across global resources.

1.3.3.1 Sharing Resources

Several companies and organizations are working together to create a standardized set of rules called protocols to make it easier to set up grid computing environments. It's possible to create a grid computing system right now and several already exist. But what's missing is an agreed-upon approach. That means that two different grid computing systems may not be compatible with one another, because each is working with a unique set of protocols and tools.

In general, a grid computing system requires:

- At least one computer, usually a server, which handles all the administrative duties for the system. Many people refer to this kind of computer as a control node. Other application and Web servers (both physical and virtual) provide specific services to the system.
- A network of computers running special grid computing network software. These computers act both as a point of interface for the user and as the resources the system will tap into for different applications. Grid computing systems can either include several computers of the same make running on the same operating system (called a homogeneous system) or a hodgepodge of different computers running on every operating system imaginable (a heterogeneous system). The network can be anything from a hardwired system where every computer connects to the system with physical wires to an open system where computers connect with each other over the Internet.
- A collection of computer software called middleware. The purpose of middleware is to allow different computers to run a process or application across the entire network of machines. Middleware is the workhorse of the grid computing system. Without it, communication across the system would be impossible. Like software in general, there's no single format for middleware.

1.3.4 Grid Applications and Portal

They are developed using grid-enabled languages such as HPC++, and message-passing systems such as MPI. Applications, such as parameter simulations and grand-challenge server they are accessing on a UNIX or NT platform.

1.4 Grid Computing Lexicon

Terms involved in Grid Computing

- **Cluster**

A group of networked computers sharing the same set of resources.

- **Extensible Markup Language (XML)**

A computer language that describes other data and is readable by computers. Control nodes (a node is any device connected to a network that can transmit, receive and reroute data) rely on XML languages like the Web Services Description Language (WSDL). The information in these languages tells the control node how to handle data and applications.

- **Integrated Development Environment (IDE)**

The tools and facilities computer programmers need to create applications for a platform .The term for an application testing ground is sandbox.

- **Interoperability**

The ability for software to operate within completely different environments. For example, a computer network might include both PCs and Macintosh computers. Without interoperable software, these computers wouldn't be able to work together because of their different operating systems and architecture.

- **Open standards**

A technique of creating publically available standards. Unlike proprietary standards, which can belong exclusively to a single entity, anyone can adopt and use an open standard. Applications based on the same open standards are easier to integrate than ones built on different proprietary standards.

- **Parallel processing**

Using multiple CPUs to solve a single computational problem. This is closely related to shared computing, which leverages untapped resources on a network to achieve a task.

- **Server virtualization**

A technique in which a software application divides a single physical server into multiple exclusive server platforms (the virtual servers). Each virtual server can run its own operating system independently of the other virtual servers. The operating systems don't have to be the same system -- in other words, a single machine could have a virtual server acting as a Linux server and another one running a Windows platform. It works because most of the time, servers aren't running anywhere close to full capacity. Grid computing systems need lots of servers to handle various tasks and virtual servers help cut down on hardware costs.

- **Service**

In grid computing, a service is any software system that allows computers to interact with one another over a network.

- **Simple Object Access Protocol (SOAP)**

A set of rules for exchanging messages written in XML across a network. Microsoft is responsible for developing the protocol.

1.5 Types of Grids

1.5.1 National-Grids

This will provide a strategic "computing reserve" and will allow substantial computing resources to be applied to large problems in times of crisis, such as to plan responses to a major environmental disaster, earthquake, or terrorist attack. Furthermore, such a Grid will act as a "national collaborators", supporting collaborative investigations of complex scientific and engineering problems, such as global climate change, space station design, and environmental cleanup.

1.5.2 Private-Grids

Private Grids can be useful in many institutions (hospitals, corporations, small firms, etc). They are characterized by a relatively small scale, central management and common purpose and, in most cases; they will probably need to integrate low-cost commodity technologies.

1.5.3 Project-Grids

Project Grids will likely be created to meet the needs of a variety of multi-institutional research groups and multi-company "virtual teams", to pursue short- or medium-term projects (scientific collaborations, engineering projects). A Project Grid will typically be built ad hoc from shared resources for a limited time, and focus on a specific goal.

1.5.4 Goodwill-Grids

Goodwill Grids are for anyone owning a computer at home who wants to donate some computer capacity to a good cause.

1.5.5 Peer-to-peer-Grids

Peer-to-peer technology depends on people sharing data between their computers. The name peer-to-peer suggests that there is no central control.

1.5.6 Consumer Grids

In a Consumer Grid, resources are shared on a commercial basis, rather than on the basis of goodwill or mutual self-interest. A big issue in such Grids will be "resource marketing": a user has to find the resources needed to solve his particular problem, and the supplier must make potential users aware of the resources he has to offer.

1.6. Key benefits of the Grid Computing Model

1.6.1 Consolidation

From servers to applications to whole sites, consolidation is a key benefit of the Grid computing model, especially in the data center. Consolidation not only minimizes the infrastructure necessary to meet an enterprise's business demands, but also reduces costs by migrating from proprietary or single-use systems to **commercial off-the-shelf (COTS)**-based systems that can be shared by multiple applications.

1.6.2 Modular Computing

Modular computing, especially in the data center, minimizes and simplifies the infrastructure using building blocks that address higher density, lower power, lower thermals, simplified cabling, and ease of upgrading and management. Blade servers are an excellent example of modularity.

1.6.3 Virtualization

By creating pools of resources enabled by highly automated management capabilities, virtualization can enable an IT system administrator to utilize far more of the resources in the data center, making the resources accessible to more than a single application sitting on a single physical server.

1.6.4 Utility Computing

Utility Computing allows an infrastructure to be managed analogously to an electric utility, applying a pay-per-use model, thereby optimizing and balancing the computing needs of an enterprise, and allowing it to run at maximum efficiency.

1.7 Characteristics and Capabilities

A grid is created by installing software services, or middleware on a set of networked computers. The middleware provides facilities such as hardware and software resource location, user authentication, and distributed scheduling of resources and tasks.

1.7.1 Smart Grid

A smart grid delivers electricity from suppliers to consumers using digital technology to save energy, reduce cost and increase reliability. Such a modernized electricity network is being promoted by many governments as a way of addressing energy independence or global warming issues.

Basic functions of Smart Grid are to:

- Be able to heal itself
- Motivate consumers to actively participate in operations of the grid
- Resist attack
- Provide higher quality power that will save money wasted from outages
- Accommodate all generation and storage options
- Enable electricity markets to flourish
- Run more efficiently

1.7.2 Wireless Grid

Wireless grids are wireless computer networks consisting of different types of electronic devices with the ability to share their resources with any other device in the network in an ad-hoc manner. A definition of the wireless grid can be given as: "Ad-hoc, distributed resource-sharing networks between heterogeneous wireless devices".

The following are the key characteristics.

- No centralized control
- Small, low powered devices
- Heterogeneous applications and interfaces
- New types of resources like cameras, GPS trackers and sensors
- Dynamic and unstable users / resources

1.8 Grid computing Vs Cluster computing

Cluster computing focuses on platforms consisting of often homogeneous interconnected nodes in a single administrative domain.

- Clusters often consist of PCs or workstations and relatively fast networks
- Cluster components can be shared or dedicated
- Application focus is on cycle-stealing computations, high-throughput computations, distributed computations

Web focuses on platforms consisting of any combination of resources and networks which support naming services, protocols, search engines, etc.

- Web consists of very diverse set of computational, storage, communication, and other resources shared by an immense number of users
- Application focus is on access to information, electronic commerce etc.

Grid focus on ensembles of distributed heterogeneous resources used as a platform for high performance computing.

- Some grid resources may be shared; other may be dedicated or reserved.



- Application focus is on high-performance, resource-intensive applications.

Cluster computing can't truly be characterized as a distributed computing solution; however, it's useful to understand the relationship of grid computing to cluster computing. Grids consist of **heterogeneous** resources. Cluster computing is primarily concerned with computational resources; grid computing integrates storage, networking, and computation resources. Clusters usually contain a single type of processor and operating system; grids can contain machines from different vendors running various operating systems. Grids are **dynamic** by their nature. Clusters typically contain a static number of processors and resources; resources come and go on the grid. Resources are provisioned onto and removed from the grid on an ongoing basis. Grids are inherently distributed over a local, metropolitan, or wide-area network. Usually, clusters are physically contained in the same complex in a single location; grids can be (and are) located everywhere. Cluster interconnects technology delivers extremely low network latency, which can cause problems if clusters are not close together. Grids offer increased **scalability**. Physical proximity and network latency limit the ability of clusters to scale out; due to their dynamic nature, grids offer the promise of high scalability.

1.9 Distributed vs. Grid Computing

Grid Computing is the latest name for the hoped-for universal distributed computing facility. The promise of ubiquitous, cheap, and almost infinitely scalable computing is alluring, and many descriptions paint a future in which grid computing gives every user and every application access to "supercomputing on demand". The Beowulf project demonstrated a practical application of large numbers of PCs to solve grand challenge supercomputing problems. However, the early successes required applications to be highly scalable and custom tailored for the environment. In the years since, many researchers and vendors have searched for ways to apply ever larger collections of computers to a wide variety of computing problems. There are actually two similar trends moving in tandem: distributed computing and grid computing. Depending on how you look at the market, the two either overlap, or **distributed computing is a subset of grid computing**.

Grid Computing got its name because it strives for an ideal scenario in which the CPU cycles and storage of millions of systems across a worldwide network function as a flexible, readily accessible pool that could be harnessed by anyone who needs it, similar to the way power companies and their users share the electrical grid. Grid computing can encompass desktop PCs, but more often than not its focus is on more powerful workstations, servers, and even mainframes and supercomputers working on problems involving huge datasets that can run for days.

1.9.1 Capabilities of Grid Computing

When you deploy a grid, it will be to meet a set of customer requirements. To better match grid computing capabilities to those requirements, it is useful to keep in mind the reasons for using grid computing is to exploit **underutilized resources**. The easiest use of grid computing is to run an existing application on a different machine. The job in question could be run on an idle machine elsewhere on the grid. There are at least two prerequisites for this scenario. First, the application must be executable remotely and without undue overhead. Second, the remote machine must meet any special hardware, software, or resource requirements imposed by the application.

1.9.2 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. A CPU intensive grid application can be thought of as many smaller “sub jobs,” each executing on a different machine in the grid. To the extent that these sub jobs do not need to communicate with each other, the more “scalable” the application becomes.

Another important grid computing contribution is to enable and simplify collaboration among a wider audience. Grid computing takes these capabilities to an even wider audience, while offering important standards that enable very heterogeneous systems to work together to form the image of a large virtual computing system offering a variety of

virtual resources. The users of the grid can be organized dynamically into a number of virtual organizations, each with different policy requirements. Sharing starts with data in the form of files or databases. A “data grid” can expand data capabilities in several ways. First, files or databases can seamlessly span many systems and thus have larger capacities than on any single system. Such spanning can improve data transfer rates through the use of striping techniques. Data can be duplicated throughout the grid to serve as a backup and can be hosted on or near the machines most likely to need the data, in conjunction with advanced scheduling techniques. Sharing is not limited to files, but also includes many other resources, such as equipment, software, services, licenses, and others. These resources are “virtualized” to give them a more uniform interoperability among heterogeneous grid participants.

1.10 Grid Concepts and Components

Types of resources: A grid is a collection of machines, sometimes referred to as “nodes,” “resources,” “members”, “donors,” “clients,” “hosts,” “engines,” and many other such terms. They all contribute any combination of resources to the grid as a whole.

1.10.1 Computation

The most common resource is computing cycles provided by the processors of the machines on the grid. The processors can vary in speed, architecture, software platform, and other associated factors, such as memory, storage, and connectivity. There are three primary ways to exploit the computation resources of a grid. The first and simplest is to use it to run an existing application on an available machine on the grid rather than locally. The second is to use an application designed to split its work in such a way that the separate parts can execute in parallel on different processors. The third is to run an application that needs to be executed many times on many different machines in the grid.

1.10.2 Communications

The rapid growth in communication capacity among machines today makes grid computing practical, compared to the limited bandwidth available when distributed computing was first emerging. Therefore, it should not be a surprise that another important resource of a grid is data communication capacity. This includes communications within the

grid and external to the grid. Communications within the grid are important for sending jobs and their required data to points within the grid. Some jobs require a large amount of data to be processed and it may not always reside on the machine running the job. The bandwidth available for such communications can often be a critical resource that can limit utilization of the grid.

1.10.3 Intragrid to Intergrid

There have been attempts to formulate a precise definition for what a “grid” is. In fact, the concept of grid computing is still evolving and most attempts to define it precisely end up excluding implementations that many would consider to be grids. Grids can be built in all sizes, ranging from just a few machines in a department to groups of machines organized as a hierarchy spanning the world.

Some examples in this range of grid system topologies:

1.10.4 A simple grid

The simplest grid consists of just a few machines, all of the same hardware architecture and same operating system, connected on a local network. This kind of grid uses homogeneous systems so there are fewer considerations and may be used just for experimenting with grid software. The machines are usually in one department of an organization, and their use as a grid may not require any special policies or security concerns. Because the machines have the same architecture and operating system, choosing application software for these machines is usually simple. Some people would call this a “cluster” implementation rather than a “grid.”

The next progression would be to include heterogeneous machines. In this configuration, more types of resources are available. The grid system is likely to include some scheduling components. File sharing may still be accomplished using networked file systems. Machines participating in the grid may include ones from multiple departments but within the same organization. Such a grid is also referred to as an “Intragrid.”

1.11 Job Scheduling

The job scheduling system is responsible to select best suitable machines in a grid for user jobs. The management and scheduling system generates job schedules for each machine in the grid by taking static restrictions and dynamic parameters of jobs and machines into consideration.

Job Scheduling in Grids: In a Grid system

1. It arranges for higher utilization Complex as many machines with local policies involved.
2. Resources are fixed Resources may join or leave randomly.
3. One job scheduler or two job schedulers.

1.11.1 Types of Job Scheduling Infrastructures

Centralized

Single job scheduler on one instance, all information collected here.

Hierarchical

Two job schedulers, one at global and other at local level.

Decentralized

No central instance, distributed schedulers interact and commit resources.

1.11.1.1 Centralized Job Scheduling

Multi Site Scheduling

A job can be executed on more than one machine in parallel. As job-parts are running on different machines, latency is important, different job-parts are started synchronously on all machines.

Single Site Scheduling

A job is executed on a single parallel machine. This means that system boundaries are not crossed. Efficient as communication inside a machine is fast.

Centralized Job Scheduling – Advantages

Efficiency

The scheduler is conceptually able to produce very efficient schedules, because the central instance has all necessary information on the available resources. Centralization is useful e.g. at a computing center, where all resources are used under the same objective. Due to this fact even communication bottlenecks can be ignored.

1.11.1.2 Hierarchical Job Scheduling

Jobs are submitted to the central scheduler; in turn jobs are submitted to low level machines. In addition, every machine uses a local job Scheduler. Basically centralized as there is one global instance .

Main advantage

Different policies can be used for global and local scheduling. Meta-scheduler redirects submitted jobs to local schedulers for resources based on some policy.

1.11.1.3 Decentralized Scheduling

No central instance is responsible. Distributed schedulers interact with each other and decide the allocations for each job to be performed. Information about state of all systems is not collected at a single point. Local job schedulers may have different but compatible scheduling policies.

1.11.1.3.1 Advantages of Decentralized Scheduling

- No communication bottleneck.
- Scalable to greater extent.
- Failure of single component doesn't affect whole metasystem.

- Better fault tolerance and reliability than centralized systems which have no back-ups.
- Site-autonomy for scheduling can be achieved easily as the local schedulers can be specialized on the needs of the resource provider or the resource itself.

1.11.1.3.2 Disadvantages of Decentralized Scheduling

- The lack of a global scheduler, which knows all job and system information at every time instant, usually leads to sub-optimal schedules.
 - The support for multi-site applications is rather difficult to achieve.
 - As all parts of a parallel program must be active at the same time, the different schedulers must synchronize the jobs and guarantee simultaneous execution.
- Decentralized Scheduling with Direct Communication

- The local schedulers can send/receive jobs to/from other schedulers directly. Either schedulers have a list of remote schedulers they can contact or there is a directory that provides information of other systems. If a job start is not possible on the local machine immediately, the local scheduler is searching for an alternative machine.
- If a system has been found, where an immediate start is possible, the job and all its data is transferred to the other machine/scheduler.
- It can be parameterized which jobs are forwarded to another machine, this affects the local queue. This can also affect the performance of some scheduling algorithms.

1.11.1.3.3 Decentralization via Central Job Pool

Jobs that cannot be executed immediately are sent to a central job pool instead of a remote machine, the local schedulers can pick suitable jobs for their schedules. Jobs can be pushed into or pulled out of the pool. A policy is required that all jobs from the pool are executed at some time

1.11.1.3.4 Scheduling Algorithms

First-Come-First-Serve

The scheduler starts the jobs in the order of their submission. If not enough resources are currently available, the scheduler waits until the job can be started.

Random

The next job to be scheduled is randomly selected among all jobs that are submitted.

Backfill

Special FCFS algorithms that prevent unnecessary idle time caused by wide jobs.

1.12. Issues in Grid Computing

A grid is a **distributed and heterogeneous** environment. Both of these issues require are the source of challenging design problems.

Being heterogeneous inherently contains the problem of managing multiple technologies and administrative domains. The computers that participate in a grid may have different hardware configurations, operating systems and software configurations. This makes it necessary to have right management tools for finding a suitable resource for the task and controlling the execution and data management.

A grid may also be distributed over a number of administrative domains. Two or more institutions may decide to contribute their resources to a grid. In such cases, security is a main issue. The users who submit their tasks and their data to the grid wish to make sure that their programs and data is not stolen or altered by the computer in which it is running. Of course the problem is reciprocal. The computer administrators also have to make sure that harmful programs do not arrive over the grid.

Another important issue is **scheduling**. Scheduling a task to the correct resource requires considerable effort. The picture is further complicated when we consider the need to access the data.

1.13 The Grid Taxonomy and Grid Architecture

The grids are categorized as departmental, enterprise and global grids. Although this is not a formal taxonomy all three levels present a different problem in the areas of data transfer and administration.

A departmental grid is putting together the resources of a department or small institution together. This means that usually the computers are on the same local high speed network and under one administrative domain. So the only problem remains is the correct administration of this single domain to allow users to have access to most suitable resource from a heterogeneous resource pool.

An enterprise grid on the other hand, is usually distributed over many geographical locations. At each location a departmental grid may be running but between them usually there is a lower bandwidth communication line, making transfer of large amounts of data not very feasible. In such grids, an efficient data transfer strategy is very important. However such grids usually belong to a single organization and usually run by a single administration. This means that some long term static planning can be made and some administrative procedures may be enforced.

A global grid, on the other hand, is a collection of enterprise grids. It is a pool of resources that are distributed globally and usually under different administrative domains. The diversity of administrative domains brings the problems of security and management. It is important that the system administrators can monitor and manage the job requests coming over the grid, so that nobody can execute a harmful code. Of course, the users who submit their jobs to the grid also have to make sure that their programs and data is not misused by the owners of the computer that the program is executing.

1.13.1 Applications Suitable for Grid Computing

As a general rule, applications that work well on clusters are also suitable for grids. Such applications can be partitioned to be executed on several processors and the results of these separate processes yield to the desired output. But such jobs are rare and most of the applications do not fall into this category.

Also the heterogeneous structure of grids do not allow for a fine grained parallelism. Therefore large batch jobs and long running processes that require minimum inter-process communication and synchronization are better suitable for grids. These jobs can be scheduled to processors that have low workloads making use of the off-peak cycles available.

Another important aspect is the data access. The smaller the size of data needed by the application is the better one. Because the applications that need large amount of data creates either a scheduling problem by forcing the application run on a processor closer to the data or requires movement of large chunks of data across the network. It is best to avoid both cases.

The interactive jobs are not suitable for grids. Because it put pressure on the scheduler to run the job when the user is available, which is usually means immediately. However in a grid environment, the networks quality of service cannot be trusted. Non-interactive jobs on the other hand are easier to schedule as they can wait for the off-peak periods of a processor within the grid.

1.13.2 Grid Management

One of the major problems in grid computing is to be able to schedule jobs and data to a suitable resource. As a grid may contain many different hardware and software configurations, a standard has to be agreed upon. The most widely used product for managing a grid is called Globus Toolkit. Supported by many large vendors, Globus offers all the functionality needed to manage a grid system.

Grid Resource Allocation Manager (GRAM) allows users to select a specific resource in the grid to run their jobs on. It has a client side module that allows user to schedule jobs at a specific server in the grid and a gatekeeper module that is running in each server to schedule arriving jobs. GRAM makes use of **Monitoring and Directory Service (MDS)**. MDS manages a directory of local and global resources. **Grid Resource Information Service (GRIS)** collects local resource information. **Global Index Information Service (GIIS)** collects GRIS information from all servers and provides a centralized resource directory for the whole grid. The movement of data in the grid is managed by **Global Access to Secondary Storage (GASS)**.

Apart from these basic services, Globus provides security functions and packaging tools to deploy software in a format that would work in any server. Globus is preferred by many grid implementations and supported by vendors like IBM. The reason behind the success of Globus is the open source approach and use of standards. For example, Globus uses SSL for secure data transfer, **Light-Weighted Directory Access Protocol (LDAP)** for directory information. By using these standard protocols it ensures that it is compatible with many operation environments.

1.13.3 Hardware Trends – Blade Servers

Grid computing and Clusters are two important architectural trends that demand a different approach from hardware vendors. Clusters are tightly coupled and homogenous structures. Grids on the other hand are lightly coupled and heterogeneous. However they both require a large number of servers to be at their disposal to deliver high performance. To deliver this in a feasible way, hardware vendors developed the concept of **Blade Servers**. A blade server is a server with processor, memory and secondary storage packed in a very small volume.

One of the main problems in operating a large number of machines is the space, power consumption and heat considerations. In order to answer these problems many vendors are now delivering blade computers.

As each blade is a server in itself, each may have their own operating systems, security privileges, data sets. Also it is possible to populate the slots in a chassis with blade servers of different capacity to create a heterogeneous server where scheduler may select the best server for the task at hand. This structure makes it possible that older server can still contribute in the blade form for low demand works protecting the investment made on them. With rack based servers, due to maintenance costs and physical limitations, older servers are usually phased out while they could still generate productive work.

1.13.4 Limitations of Grid Computing

Not every application is suitable or enabled for running on a grid. Some kinds of applications simply cannot be parallelized. For others, it can take a large amount of work to modify them to achieve faster throughput. The configuration of a grid can greatly affect the performance, reliability, and security of an organization's computing infrastructure.

To summarize we have shown how Grid Computing is becoming the preferred platform for next generation e- Science experiments that require Management of massive distributed data. It can be observed that while there have been a lot of development grid technologies for e-Science, there is still more to be achieved. This would require development of richer services and applications on top of already existing ones so that Grid Computing can move beyond Grid Computing can move beyond scientific applications and into mainstream IT infrastructure.

1.13.5 Grid Architecture

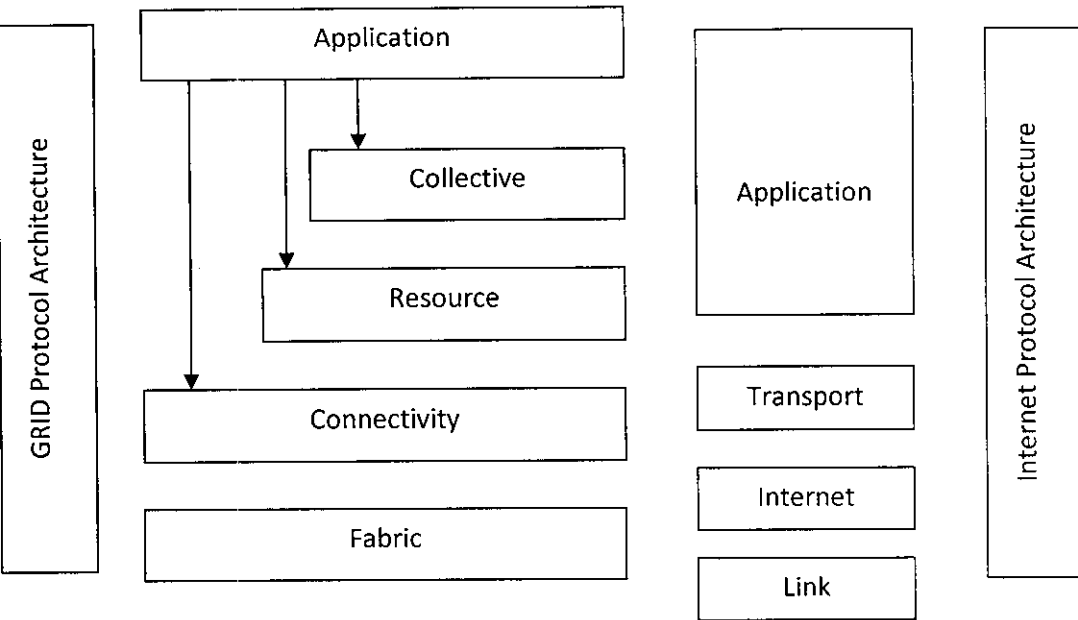


Figure 1.1 A layered grid architecture and its relationship to the Internet protocol architecture.

Figure 1.1 Illustrates the component layers of the architecture with specific capabilities at each layer. Each layer shares the behavior of the component layers. Each of these component layers is compared with their corresponding Internet protocol Layers, for purposes of providing more clarity in their capabilities.

1.13.5.1 Fabric Layer: Interface to Local Resources

This defines the resources that can be shared. This could include computational resources, data storage, networks, catalogs and other system resources. These resources can be physical resources or logical resources by nature. Example for logical resources are distributed file systems, computer clusters etc.,

Basic capabilities are

1. Provide an “inquiry” mechanism whereby it allows for the discovery against its own resource capabilities, structure and state of operations.
2. Provide appropriate “ resource management” capabilities to control the QoS the grid solution promises or has been contracted to deliver.

1.13.5.2 Connectivity Layer: Manages Communications

This defines the core communication and authentication protocol required for grid-specific networking services transactions. It includes networking transport, routing and naming. Characteristics to be considered are Single sign on, Delegation, User-Based trust relationships and Data Security.

1.13.5.3 Resource Layer: Sharing of a Single Resource

This utilizes the communication and security protocol defined by the networking communications layer, to control the secure negotiation, initiation, monitoring, metering, accounting, and payment involving the sharing of operations across individual resources.

1.13.5.4 The Collective Layer: Coordinating Multiple Resources

While the Resource layer manages an individual resource, the Collective layer is responsible for all global resource management and interaction with a collection of resources. Collective services are Discovery Services, Co allocation, Scheduling and Brokering Services, Monitoring and Diagnostic Services, Data Replication Services etc.,

1.13.5.5 Application Layer: User- Defined Grid Applications

These are user applications, which are constructed by utilizing the services defined at each lower layer. Such an application can directly access the resource, or can access the resource through the Collective service interface APIs (Application Provider Interface)

CHAPTER II

2. Overview of Job Scheduling

2.1 Job Scheduling in Grid Computers

Distributed computing utilizes a network of many computers, each accomplishing a portion of an overall task, to achieve a computational result much more quickly than with a single computer.

Distributed computing can be defined in many different ways. In distributed computing the task is split up into smaller chunks and performed by the many computers owned by the general public. Distributed computing is usually known as parallel computing. The key issue here is that we are using computing power that we don't own. These computers are owned and controlled by other people, who you would not necessarily trust.

Grid computing is a form of distributed computing that coordinates and shares computation, application, and data storage or network resources across dynamic and geographically dispersed organizations. One primary issue associated with the efficient utilization of heterogeneous resources in a grid is grid scheduling. Grid scheduling is a critical design issue of grid computing. It is a challenge because the capability and availability of resources vary dynamically. The complexity of scheduling problem increases with the size of the grid and becomes difficult to solve effectively.

Grid scheduling requires a series of challenging tasks. These include, searching for resources in the collection of geographically distributed heterogeneous computing systems and making scheduling decisions, taking into consideration quality of service. Grid scheduler does not have full control over the grid. The grid scheduler can not assume that it has a global view of the grid.

2.2 Classification of Static Task-Scheduling algorithms

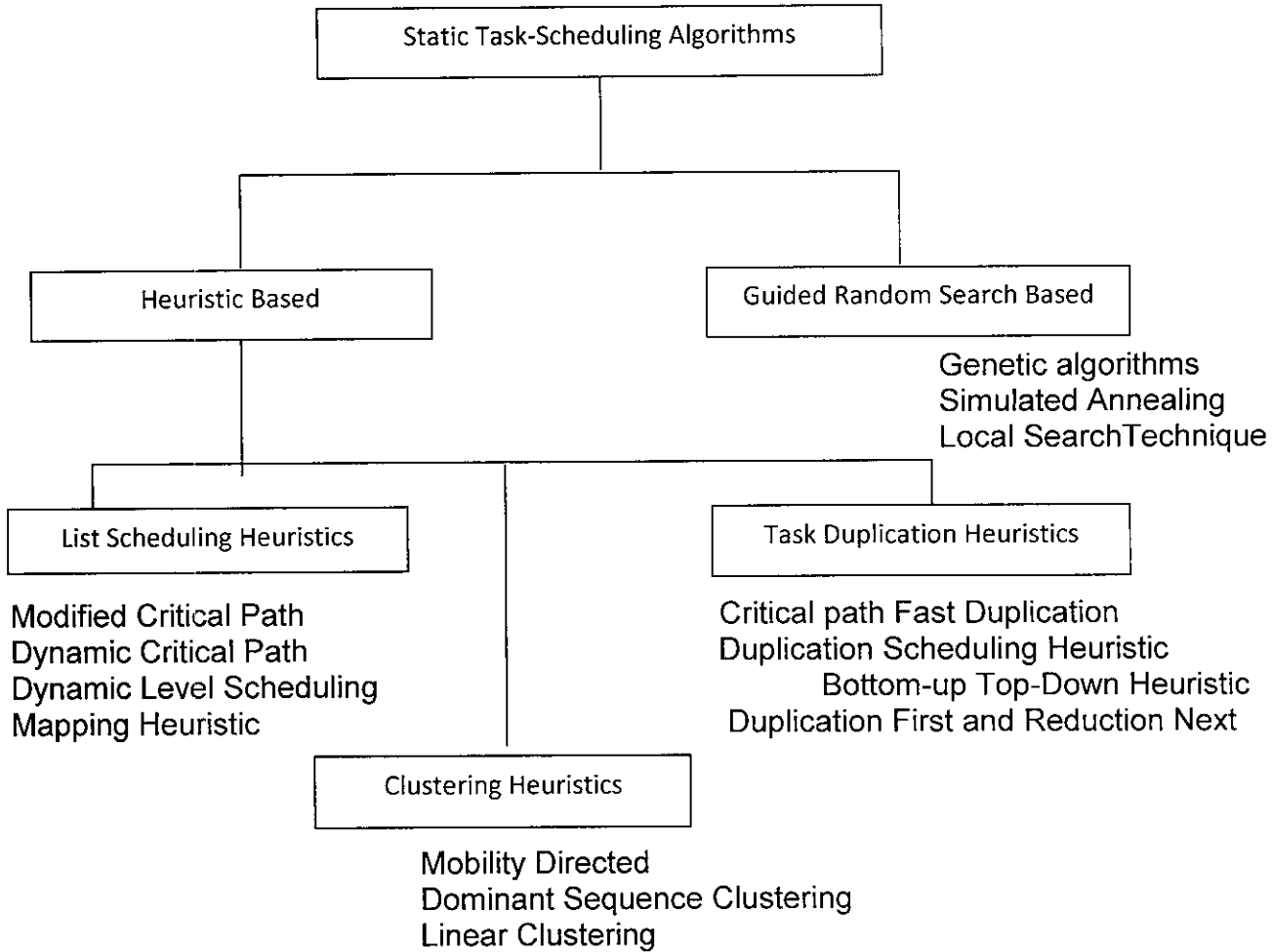


Figure 2.1 Classification of Static task-Scheduling algorithms

2.3. Job Scheduling in a Heterogeneous Grid Environment

Computational grids have the potential for solving large-scale scientific problems using heterogeneous and geographically distributed resources. However, a number of major technical hurdles must be overcome before this potential can be realized. One problem that is critical to effective utilization of computational grids is the efficient scheduling of jobs.

One of the primary goals of grid computing is to share access to geographically distributed heterogeneous resources in a transparent manner. There will be many benefits when this goal is realized, including the ability to execute applications whose computational requirements exceed local resources and the reduction of job turnaround time through workload balancing across multiple computing facilities. The development of computational grids and the associated middleware has therefore been actively pursued in recent years. However, many major technical (and political) hurdles stand in the way of realizing these benefits. Although numerous researchers have proposed scheduling algorithms for parallel architectures, the problem of scheduling jobs in a heterogeneous grid environment is fundamentally different.

2.4 Job Scheduling Policy for High Throughput

The growing computational power requirements of grand challenge applications has promoted the need for merging high throughput computing and grid Computing principles to harness computational resources distributed across multiple organizations.

First of all there is a lot of activity to bring standards to the field. Globus is a big step forward towards the formation of very large global grid systems.

Secondly, the hardware vendors are rushing to deliver the right kind of hardware for this new architecture. Blade servers will make it possible in the future that whenever we have a job, there will be an available server somewhere to execute it. Software vendors like Oracle are also delivering products that take advantage of these new architectures.

CHAPTER III

3. Problem Overview

3.1 Problem Definition

To determine the assignment of Tasks (N) of a given application to a given machine set P ($P < N$) such that

- The scheduling length (**Makespan** – overall completion time) is to be minimized
- All precedence constraints are to be satisfied.
- The application is represented by the Task Graph
- The Resources are scheduled in Batch Mode, where the jobs and resources are collected and mapped at prescheduled time.

3.2 Task Graph Representation

- Directed Acyclic Graph: $G(V,E)$
- V is the set of v nodes, each node $v_i \in V$ represents an application Task, which is a sequence of instructions that must be executed serially on the same machine.
- E is the set of communication edges. The directed edge $e_{i,j}$ joins nodes v_i and v_j , where node v_i is called the parent node and node v_j is called child node. This also implies that v_j cannot start until v_i finishes and sends its data to v_j .
- $C_{i,j}$ is the communication cost from the node n_i to the node n_j .

3.3 Application Representation using Task Graph

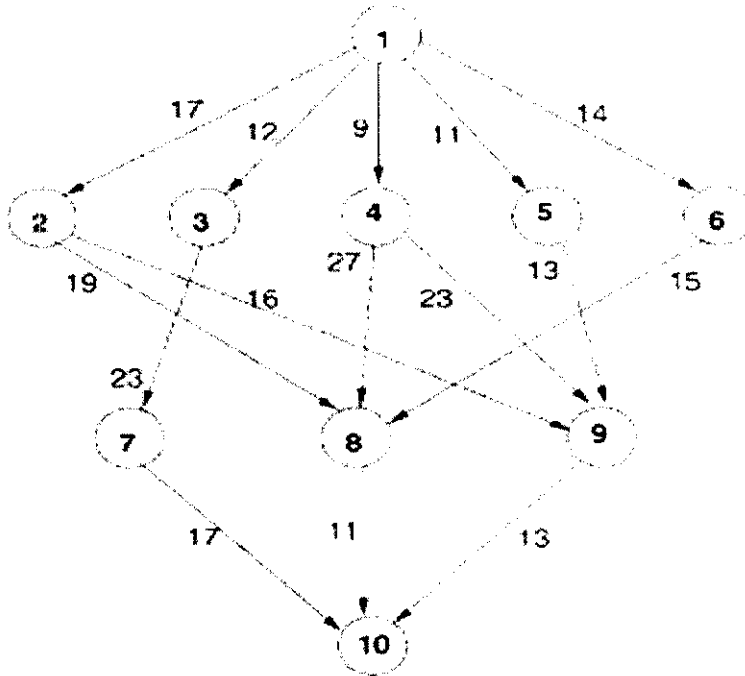


Figure 3.1 Task graph representation

3.4 Representation of Weight Matrix

TASK	P1	P2	P3
1	14	16	9
2	13	19	18
3	11	13	19
4	13	8	17
5	12	13	10
6	13	16	9
7	7	15	11
8	5	11	14
9	18	12	20
10	21	7	16

Table 3.1 Weight matrix representation

3.5 Existing Algorithms taken up for comparison

3.5.1 Critical Path on a Processor (CPOP)

The CPOP algorithm uses summation of upward and downward rank for prioritizing tasks.

Tasks are ordered in this algorithm by their scheduling priorities that are based on upward and downward ranking. The upward rank of a task n_i is recursively defined by

$$\text{Rank}_u(n_i) = \text{avg}(w_i) + \max(\text{avg}(c_{i,j}) + \text{rank}(n_j)), \text{ Where } n_j \in \text{succ}(n_i)$$

Where $\text{succ}(n_i)$ is the set of immediate successors of task n_i . $C_{i,j}$ is the average communication cost of edge (i, j) , and w_j is the average computation cost of task n_i . Since the rank is computed recursively by traversing the task graph upward, starting from the exit task, it is called upward rank. For the exit task n_{exit} , the upward rank value is equal to

$$\text{Rank}_u(n_{\text{exit}}) = \text{avg}(W_{\text{exit}})$$

Basically, $\text{rank}_u(n_i)$ is the length of the critical path from task n_i to the exit task, including the computation cost of task n_i . Similarly, the downward rank of a task n_i is recursively defined by

$$\text{Rank}_d(n_i) = \max\{\text{rank}(n_j) + \text{avg}(w_i) + \text{avg}(c_{i,j})\}, \text{ where } n_j \in \text{pred}(n_i)$$

where $\text{pred}(n_i)$ is the set of immediate predecessors of task n_i . The downward ranks are computed recursively by traversing the task graph downward starting from the entry task of the graph. For the entry task n_{entry} , the downward rank value is equal to zero. Basically, $\text{rank}_d(n_i)$ is the longest distance from the entry task to task n_i excluding the computation cost of the task itself. The priorities of each task n_i is calculated as

$$\text{Priority}(n_i) = \text{UpwardRank}(n_i) + \text{DownwardRank}(n_i)$$

Critical tasks are the tasks which are having equal upward and downward ranks.

- Critical tasks are scheduled in a critical path processor which will execute all the critical task fastly.
- Tasks are arranged in a queue in descending order of priority.
- Select the highest priority task and assign it to the processor which is executing the task fastly.
- Repeat the above step until the queue is empty.
- Calculate the exit task completion time which is the makespan.

Algorithm

Initialize the priority queue (PQ) with the entry task

While there is an unscheduled task in PQ do

 Select the highest priority task n_i from PQ

 if $n_i \in$ critical path task then

 Assign the task n_i to cpp

 else

 Assign the task n_i to processor p_j which minimizes the $EFT(n_i, p_j)$

 Update PQ with the successors of n_i , if they become ready tasks

End while

3.5.2 Heterogeneous Critical Parent Trees (HCPT)

This method consists of two phases

1. Listing phase
2. Assignment phase

Listing phase will list the order of execution of tasks. For ordering, it finds Critical nodes in the graph. Critical node is the node which has equal **Average Earliest Start Time (AEST)** and **Average Latest Start Time (ALST)**.

The AEST and ALST are calculated as follows,

$$\text{AEST}(v_j) = \max \{ \text{AEST}(v_m) + \text{avg}(w_m) + \text{avg}(c_{m,i}) \} \text{ where } v_m \in \text{pred}(v_j)$$

$$\text{avg}(w_i) = \sum (w_{i,j}) / p, \quad 0 < j \leq p$$

$$\text{ALST}(v_j) = \min \{ \text{ALST}(v_m) - \text{avg}(c_{i,m}) \} - \text{avg}(w_j), \quad \text{where } v_m \in \text{succ}(v_j)$$

Assignment phase will assign each task into a machine which minimizes the **Earliest Execution Finish Time (EEFT)** and it is calculated as,

$$\text{EEFT}(v_j, p_q) = \max \{ \text{AvailT}(p_q), \text{FT}(v_n) + k \cdot c_{n,i} \} + w_{i,q}$$

Algorithm

Traverse the graph downward and compute AEST for each node,

Traverse the graph upward and compute ALST for each node,

Push CNs on the stack S in the reverse order of their ALST,

While S is not empty do

 if there is an unlisted parent of top (S) then

 push the parent node on S

 else

 pop the top (S) and enqueue it to list L

While not the end of L do

 dequeue v_i from L

 for each machine p_q in the machine set P do

 compute EEFT (v_i, p_q)

 assign task v_i to the machine p_m that minimizes EEFT of v_i

3.5.4 Proposed method

Heterogeneous Task Scheduling (HTS)

Drawbacks of existing algorithms

Existing algorithms use almost 50% of its total execution time for computing listing phase.

In order to avoid this calculation we maintained a ready Queue dynamically. After executing a particular task, ready queue is updated with its children if they become ready tasks.

The algorithm starts from the entry node. Initially the entry node is available in ready queue. Until the queue is empty take the task of the nodes in ready queue. Select the node (let task t_i) which has earliest completion time at machine m_j and remove from the ready queue. Allocate the task t_i in m_j . Update the ready queue by adding the task which are ready due to t_i completion. The overall completion time (makespan) is calculated.

Algorithm

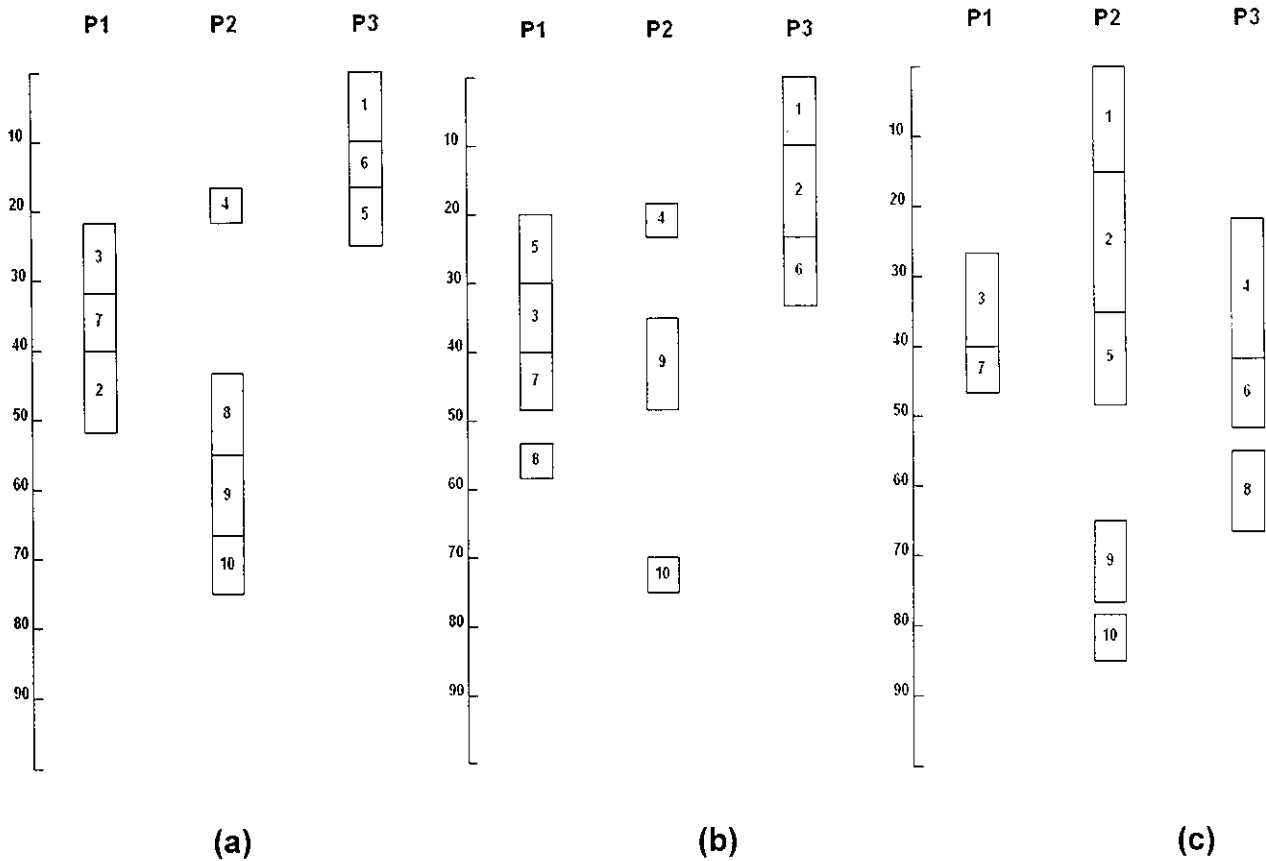
Initialize the Ready queue (RQ) with the entry task

While there is an unscheduled task in RQ do

 Assign the task n_i to processor p_j which minimizes the EEFT (n_i, p_j)

 Update RQ with the successors of n_i , if they become ready tasks

End while



**Scheduling of the task graph in the figure with: (a) HTS(makespan=75)
 (b)HCPT(makespan=76) (c) CPOP(makespan=86)**

Figure 3.2 Makespan comparison

CHAPTER IV

4. Implementation

4.1 Input Weight Matrix Generation

4.1.1 Simulation Model

Mixed-machine **heterogeneous computing (HC)** environments utilize a distributed suite of different high-performance machines, interconnected with high-speed links to perform different computationally intensive applications that have diverse computational requirements. The general problem of mapping (i.e., matching and scheduling) tasks to machines in an HC suite has been shown to be NP-complete. Heuristics developed to perform this mapping function are often difficult to compare because of different underlying assumptions in the original studies of each heuristic. To facilitate these comparisons, certain simplifying assumptions were made. For these studies, let a meta-task be defined as a collection of independent tasks with no (data dependencies) (a given task, however, may have subtasks and dependencies among the subtasks). For this case study, it is assumed that static: (i.e., off-line or predictive) mapping of meta-tasks is being performed. The goal of this mapping is to minimize the total execution time of the meta-task. It is also assumed that each machine executes a single task at a time (i.e., no multi-tasking), in the order in which the tasks are assigned. The size of the Meta task (i.e., the number of tasks to execute), l , and the number of machines in the HC environment, n are static and known a priori.

The **eleven static mapping heuristics** were evaluated using simulated execution times for a Heterogeneous Computing (HC) environment. Because these are static heuristics, it is assumed that an accurate estimate of the expected execution time for an each task on each machine is known prior to execution and contained within an **Expected Time to Compute (ETC)** matrix. One row of the ETC matrix contains the estimated execution times for a given task on each machine. Similarly, one of the ETC matrixes consists of the estimated execution times of a given machine for each task in the meta-task. Thus, for an arbitrary task t , and an arbitrary machine m , $ETC(t, m)$ is the estimated execution time of t on m .

For cases when inter-machine communications are required. ETC (t_i, m_j) could be assumed to include the time to move the executables and data associated with task t , from their known source to machine m ,. For cases when it is impossible to execute task t , on machine m_j (e.g., if specialized hardware is needed), the value of ETC (t_i, m) can be set to infinity, or some other arbitrary value. For this study , it is assumed that there are inter-task communication each task it can execute on each machine, and estimated expected execution time of each task on each machine following method are known. The assumption that these estimated expected execution times are known is commonly made when studying mapping heuristics for HC systems.

For the simulation studies, characteristics of the ETC matrices were varied in an attempt to represent a range of possible HC environments. The ETC matrices used were generated using the following method. Initially, a $t \times 1$ baseline column vector, B , of floating point values is created. Let $\$b$ be the upper-bound of the range of possible values within the baseline vector. The baseline column vector is generated by repeatedly selecting a uniform random number, $x_b^i \in [1, \$b]$, and letting $B(i) = x_b^i$ for $0 \leq i < t$. Next, the rows of the ETC matrix are created constructed. Each element ETC (t_i, m_j) in row i of the ETC matrix is created by taking the baseline value, $B(i)$, and multiplying it by a uniform random number, $x_r^{i,j}$, which has an upper-bound of $\$r$. This new random number $x_r^{i,j} \in [1, \$r]$, is called a row multiplier. One row requires m different row multipliers, $0 \leq j < m$. Each row i of the ETC matrix can then be described as $ETC(t_i, m_j) = B(i) \times x_r^{i,j}$ for $0 \leq j < m$. (The baseline column itself does not appear in the final ETC matrix). This process is repeated for each row until the $t \times m$ ETC matrix is full. Therefore, any given value in the ETC matrix is within the range $[1, \$b \times \$r]$.

To evaluate the heuristics for different mapping scenarios, the characteristics of the ETC matrix were varied based on several different methods from [Arm97]. The amount of variance among the execution times of tasks in the meta-task for a given machine is defined as task heterogeneity. Task heterogeneity was varied by changing the upper-bound of the random numbers within the baseline column vector. High task heterogeneity was represented by $\$b = 3000$ and low task heterogeneity used $\$b = 100$. Machine heterogeneity represents the variation that is possible among the execution times for a given task across all the machines. Machine heterogeneity was varied by changing the

upper-bound of the random numbers used to multiply the baseline values. High machine heterogeneity values were generated using $\sigma_r = 10$. These heterogeneous ranges are based on one type of expected environment for MSHN. The ranges were chosen to reflect the fact that in real situations there is more variability across execution times for different tasks on a given machine than the execution time for a single task across different machines.

To further vary the ETC matrix in an attempt to capture more aspects of realistic mapping situations. Different ETC matrix consistencies were used. An ETC matrix is said to be consistent if whenever a machine m_j executes any task t_i faster than machine m_k , then machine m_j executes all the task faster than m_k . Consistent matrices were generated by sorting each row of the ETC matrix independently, with machine m_0 always being the fastest and machine $m_{(m-1)}$ the slowest. In contrast: inconsistent matrices characterize the situation where machine m_j may be faster than the machine m_k for some tasks, may be slower for others. These matrices are left in the unordered, random state in which they were generated (i.e., no consistence is enforced). Partially-consistent matrices are inconsistent matrices that include a consistent sub matrix. For the partially-consistent matrices used here, the row elements in column positions $\{0,2,4,\dots\}$ of row l are extracted sorted, and replaced in order, while the row elements in column positions $\{1,3,5,\dots\}$ remain unordered (i.e., the even columns are consistent and odd columns are in general inconsistent).

While it was necessary to select some specific parameter values for t , m , and the ETC entries to allow implementation of a situation, the techniques presented here are completely general. Therefore, if these parameters values do not apply to a specific situation of interest, researchers may substitute in their own values and the evaluation software of this study will apply.

4.1.2 ETC Matrix Generation Algorithm

Step 1:

Construct $t \times 1$ baseline column vector B , by repeatedly selecting uniform random number $x \in [1, b)$ where b is upper bound,

Step 2:

Construct Row Multiplier by repeatedly selecting uniform random number $y \in [1, r)$, where r is upper bound,

Step 3:

Construct ETC matrix by multiplying Baseline column vector and Row multiplier

4.1.3 Sample ETC Matrix

Task 1	0.24243058	1.2949938	2.0064628	2.4523716	2.971872
	3.4139936	3.903617	4.9719954	5.5944223	6.371052
	6.597259	6.721537	7.0242734	7.8943787	8.861938
	10.836024				

4.2 Graph Construction

The random graph generator was implemented to generate application graphs with various characteristics. The generator requires the following input parameters:

- number of tasks in the graph v ,
- The computation cost w_i for each task t_i is generated using the simulation model.
- **Communication to Computation Ratio (CCR)**, which is defined as the ratio of the average communication cost to the average computation cost.
- Each node in the level l_i has half the number of nodes in the level l_{i-1} as parents.

In all experiments

- Only graphs with a single entry and a single exit node were considered.
- Graph levels $l=5$.

CHAPTER V

5. Experimental Results and Discussion

This section presents performances comparison of the proposed algorithm with the existing CPOP and HCPT algorithms.

5.1. Comparison Metrics

The comparisons of the algorithms are based on the following metrics:

5.1.1 Makespan

The makespan, or scheduling length, is defined as:

$$\text{Makespan} = FT(v_{\text{exit}}) ,$$

Where $FT(v_{\text{exit}})$ is the finishing time of the scheduled exit node.

5.1.2 Speedup

The speedup value is defined as the ratio of the sequential execution time (i.e., cumulative computation costs of all tasks) to the parallel execution time (i.e., the makespan). The sequential execution time is computed by assigning all tasks to a single machine, which minimizes the cumulation of the computation costs.

$$\text{SpeedUp} = (\min_{p_j \in Q} \{\sum_{n_i \in V} w_{i,j}\}) / \text{makespan}$$

CHAPTER VI

6 .Comparison Graphs

6.1 Makespan Comparison

6.1.1 Low Task Heterogeneity Low Machine Heterogeneity

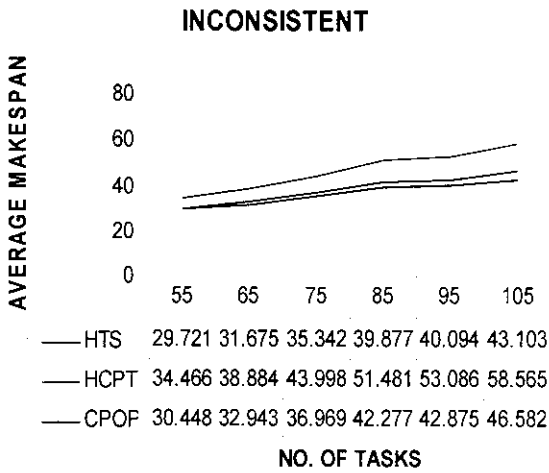


Figure 6.1

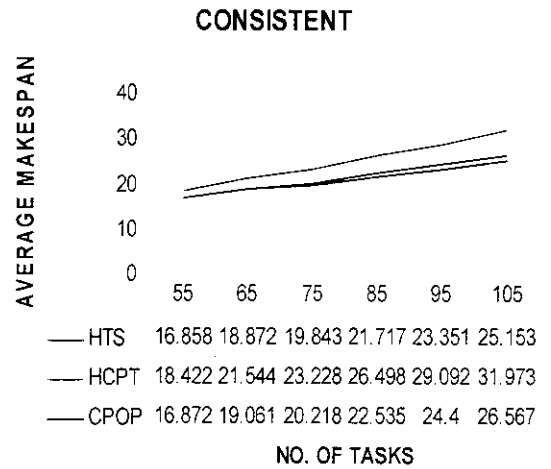


Figure 6.2

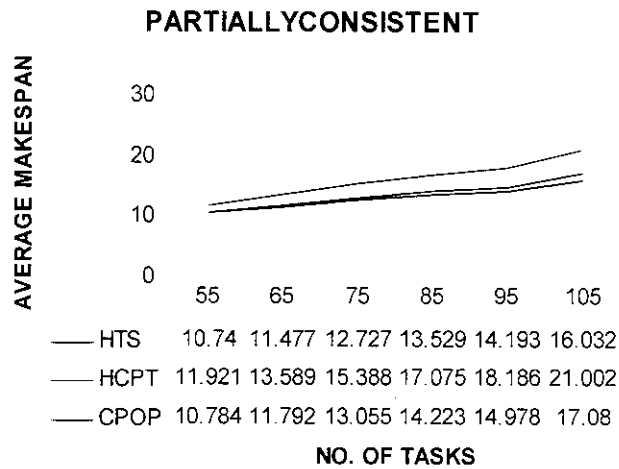


Figure 6.3

6.1.2 Low Task Heterogeneity High Machine Heterogeneity

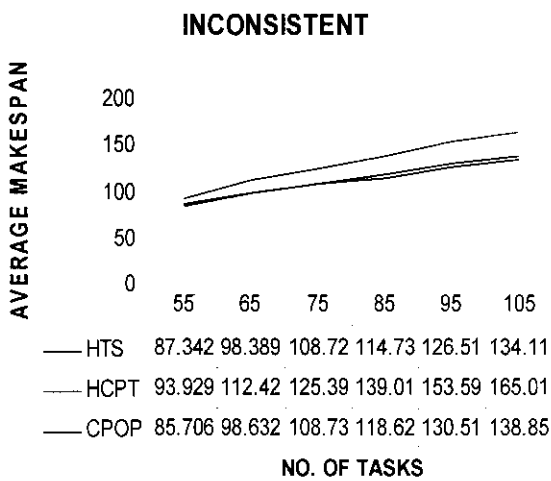


Figure 6.4

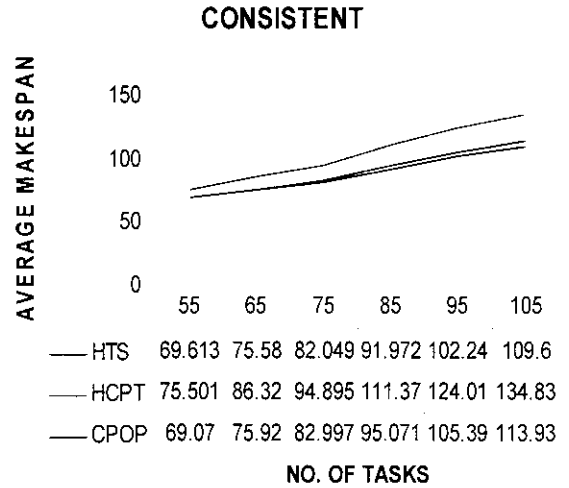


Figure 6.5

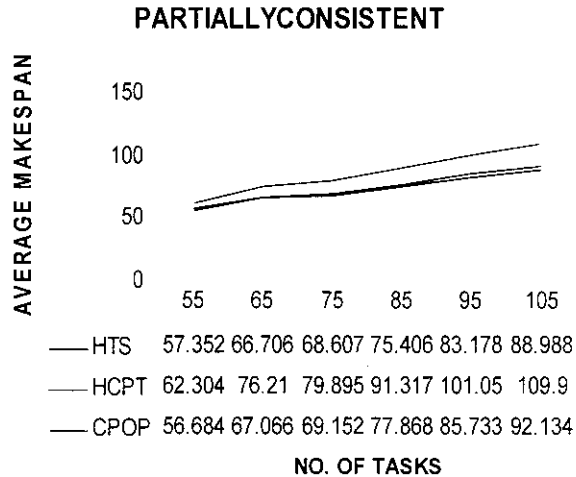


Figure 6.6

6.1.3 High Task Heterogeneity Low Machine Heterogeneity

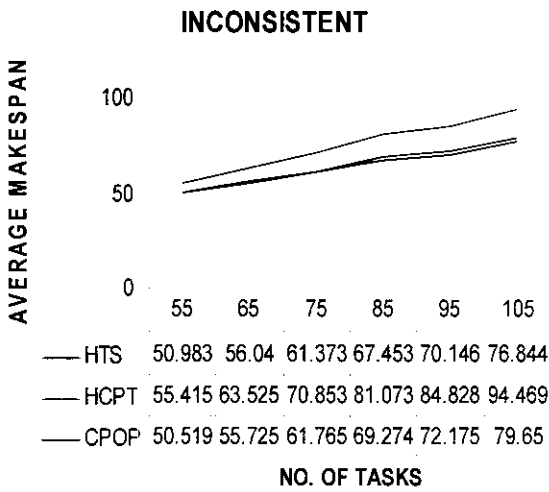


Figure 6.7

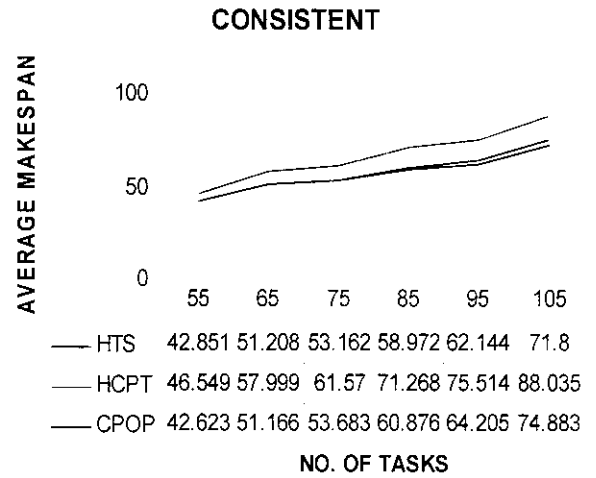


Figure 6.8

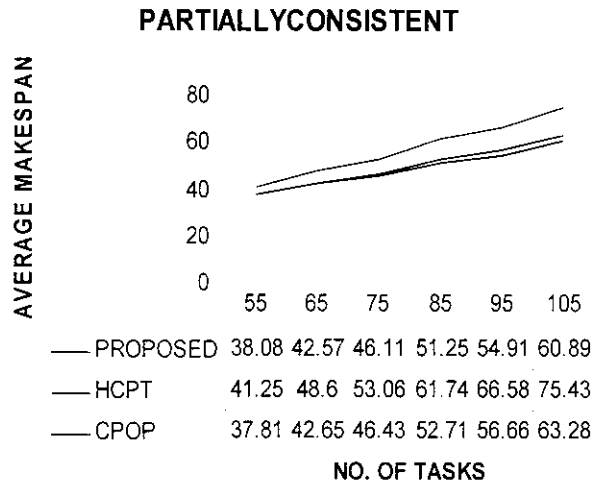


Figure 6.9

6.1.4 High Task Heterogeneity High Machine Heterogeneity

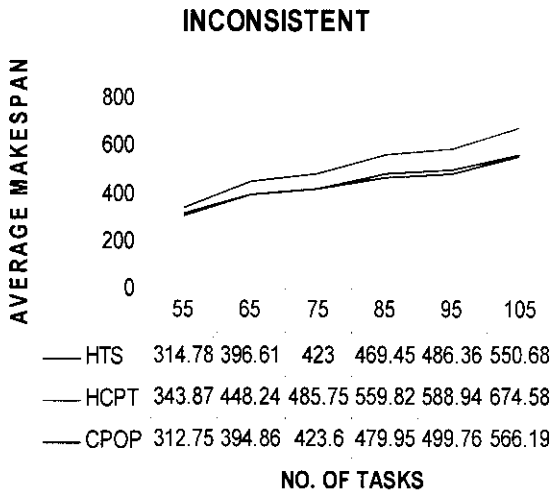


Figure 6.10

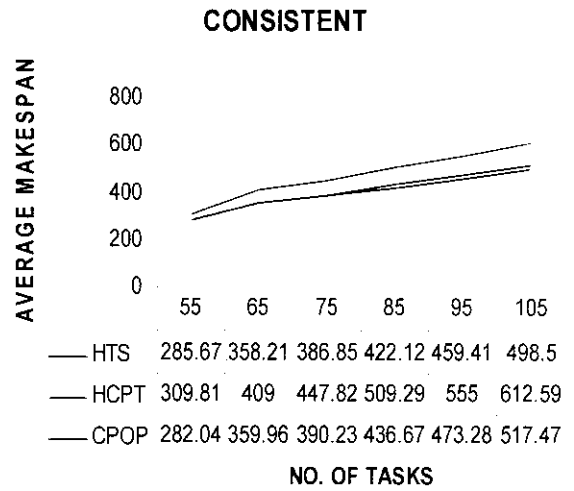


Figure 6.11

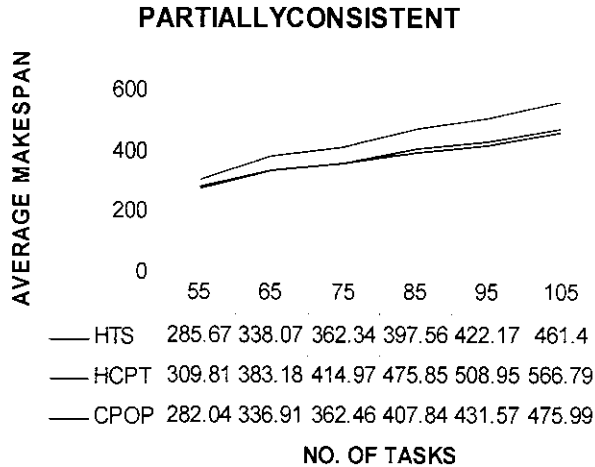


Figure 6.12

6.2 Speedup Comparison

6.2.1 Low Task Heterogeneity Low Machine Heterogeneity

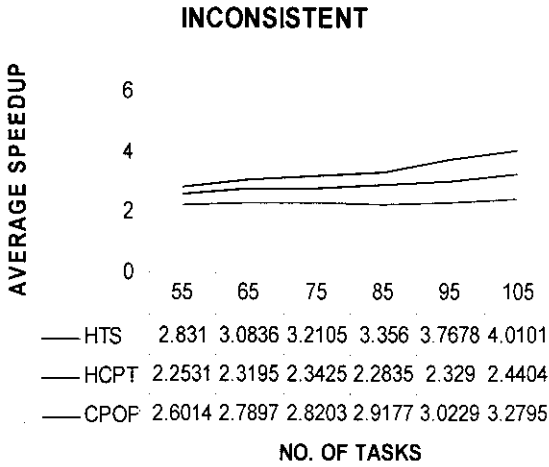


Figure 6.13

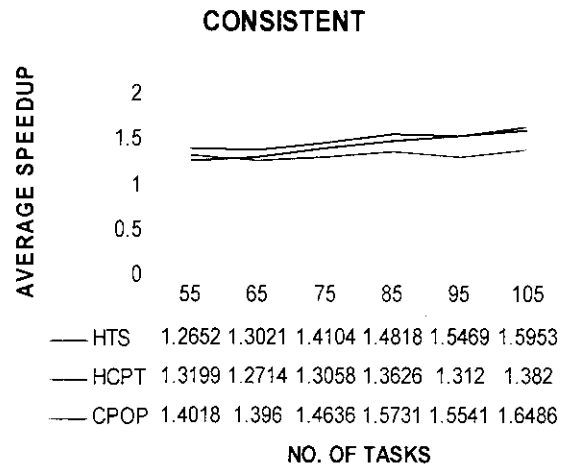


Figure 6.14

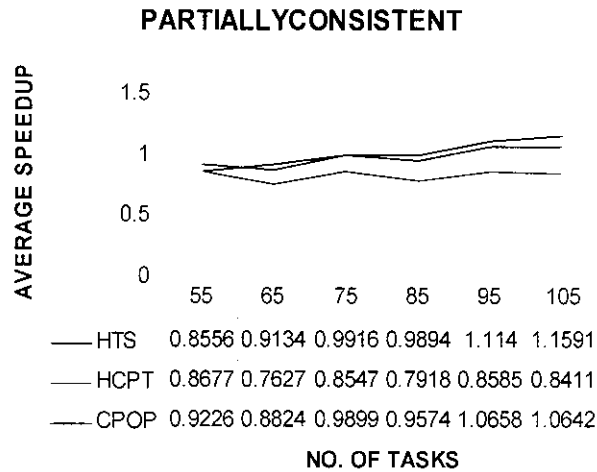


Figure 6.15

6.2.2 Low Task Heterogeneity High Machine Heterogeneity

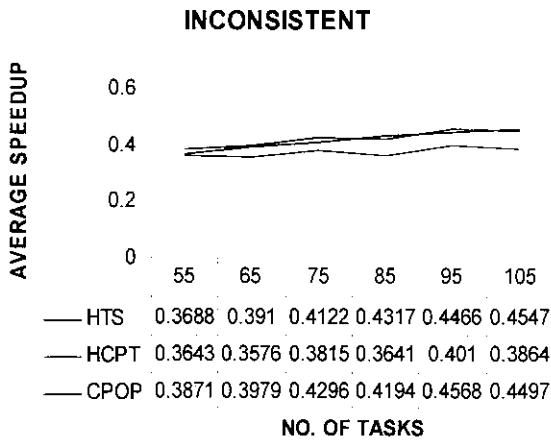


Figure 6.16

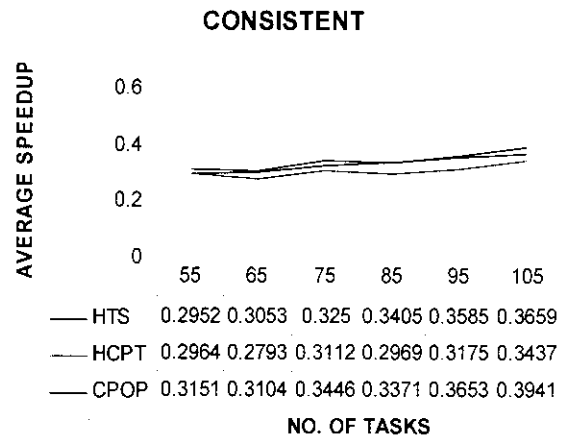


Figure 6.17

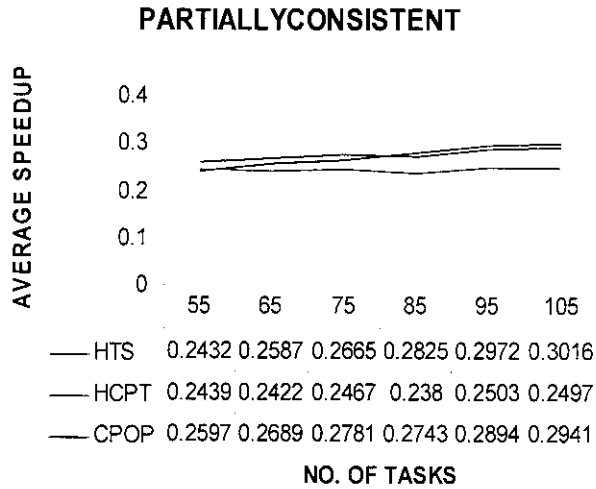


Figure 6.18

6.2.3 High Task Heterogeneity Low Machine Heterogeneity

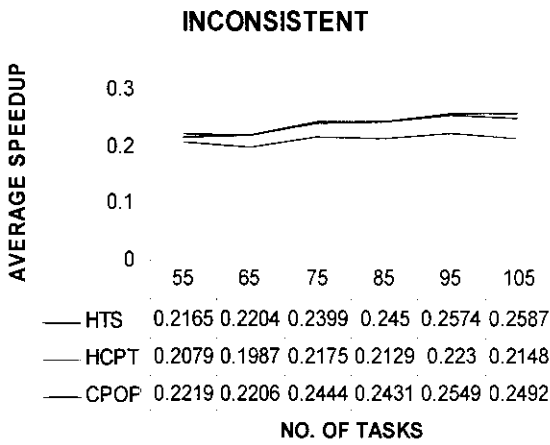


Figure 6.19

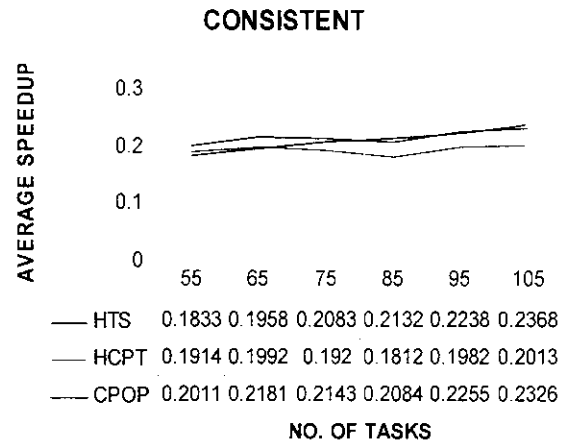


Figure 6.20

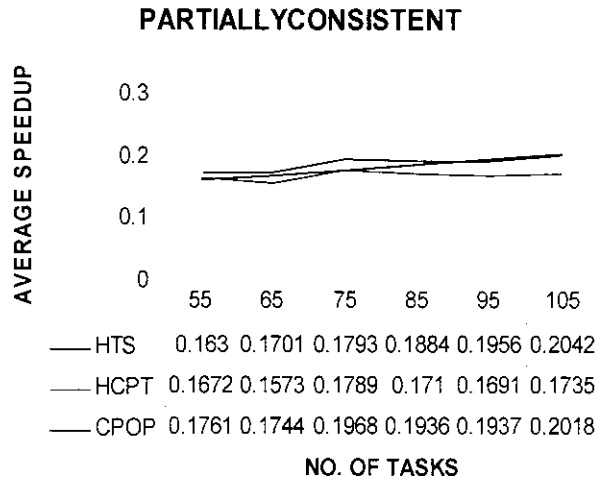


Figure 6.21

6.2.4 High Task Heterogeneity High Machine Heterogeneity

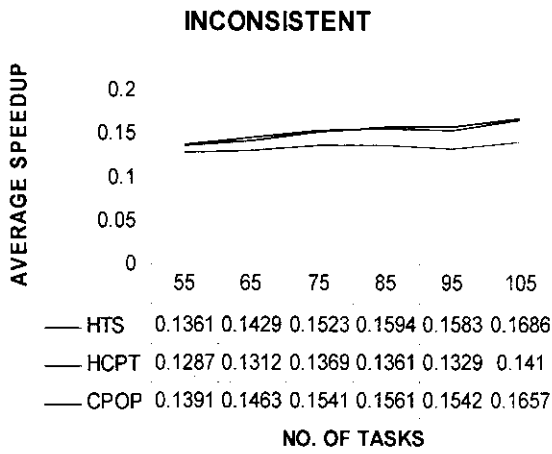


Figure 6.22

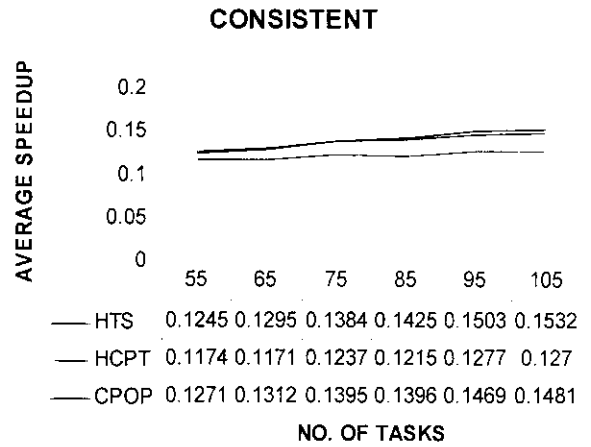


Figure 6.23

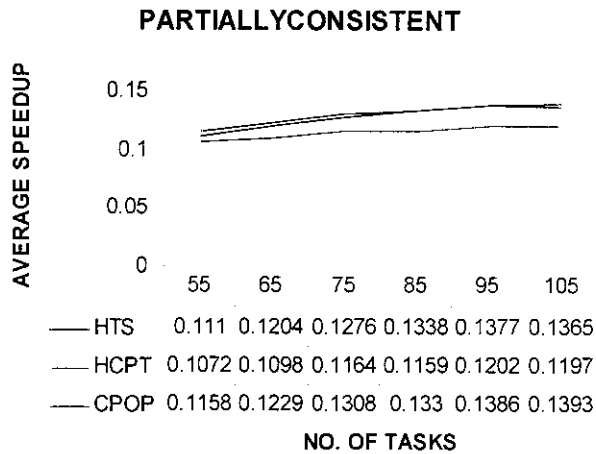


Figure 6.24

6.3 Number of Favorable Cases for 1000 Trials

6.3.1 Low Task Heterogeneity Low Machine Heterogeneity

6.3.1.1 Inconsistent

NO OF NODES	METHODS		
	PROPOSED	HCPT	CPOP
55	490	46	294
65	573	20	256
75	646	15	223
85	742	7	151
95	786	6	104
105	844	2	83

Table 6.1

6.3.1.2 Consistent

NO OF NODES	METHODS		
	PROPOSED	HCPT	CPOP
55	376	66	354
65	468	40	308
75	524	30	274
85	661	15	166
95	739	10	149
105	806	8	89

Table 6.2

6.3.1.3 Partially Consistent

NO OF NODES	METHODS		
	PROPOSED	HCPT	CPOP
55	410	78	321
65	571	26	259
75	552	25	271
85	722	11	164
95	756	13	130
105	833	10	79

Table 6.3

6.3.2 Low Task Heterogeneity High Machine Heterogeneity

6.3.2.1 Inconsistent

NO OF NODES	METHODS		
	PROPOSED	HCPT	CPOP
55	363	106	485
65	496	36	436
75	453	29	482
85	672	15	282
95	667	23	278
105	735	13	232

Table 6.4

6.3.2.2 Consistent

NO OF NODES	METHODS		
	PROPOSED	HCPT	CPOP
55	389	117	457
65	477	48	444
75	504	50	415
85	663	24	284
95	682	20	272
105	750	14	213

Table 6.5

6.3.2.3 Partially Consistent

NO OF NODES	METHODS		
	PROPOSED	HCPT	CPOP
55	379	95	472
65	479	47	433
75	530	34	400
85	661	27	290
95	675	19	286
105	727	9	245

Table 6.6

6.3.3 High Task Heterogeneity Low Machine Heterogeneity

6.3.3.1 Inconsistent

NO OF NODES	METHODS		
	PROPOSED	HCPT	CPOP
55	406	86	473
65	448	40	480
75	464	31	465
85	646	20	312
95	645	22	295
105	711	24	248

Table 6.7

6.3.3.2 Consistent

NO OF NODES	METHODS		
	PROPOSED	HCPT	CPOP
55	396	112	448
65	458	49	451
75	525	40	405
85	676	16	288
95	690	28	251
105	758	18	198

Table 6.8

6.3.3.3 Partially Consistent

NO OF NODES	METHODS		
	PROPOSED	HCPT	CPOP
55	393	137	418
65	457	49	457
75	496	56	412
85	659	22	287
95	686	24	270
105	764	18	194

Table 6.9

6.3.4 High Task Heterogeneity High Machine Heterogeneity

6.3.4.1 Inconsistent

NO OF NODES	METHODS		
	PROPOSED	HCPT	CPOP
55	360	108	502
65	451	37	466
75	470	42	465
85	622	29	330
95	659	29	302
105	659	29	302

Table 6.10

6.3.4.2 Consistent

NO OF NODES	METHODS		
	PROPOSED	HCPT	CPOP
55	421	104	452
65	490	47	438
75	530	44	411
85	672	29	276
95	691	32	267
105	749	19	215

Table 6.11

6.3.4.3 Partially Consistent

NO OF NODES	METHODS		
	PROPOSED	HCPT	CPOP
55	414	112	448
65	454	43	471
75	490	50	439
85	656	18	306
95	615	33	325
105	711	16	249

Table 6.12

6.4 Conclusion

In this project, we presented the **Heterogeneous Task Scheduling (HTS)** algorithm for scheduling tasks onto any number of heterogeneous machines. Based on the experimental study using a large set (60K) of randomly generated application graphs with various characteristics, the HTS outperformed the other algorithms in terms of performance, complexity and cost metrics includes speedup, frequency of best results and average makespan. Because of its robust performance, low running time, and the ability to give stable performance over a wide range of graph structures, the HTS algorithm is a viable solution for the DAG scheduling problem with **higher number of nodes**, on heterogeneous systems. Based on our performance evaluation study, we also observed that the HTS algorithm has given either better performance and better running time results than existing algorithms or comparable results with them.

6.5 References

[1] Tarek Hagra, Jan Janeček, "A Simple Scheduling Heuristic for Heterogeneous Computing Environments," *Proceedings of the Second International Symposium on Parallel and Distributed Computing (ISPDC,03)*,2003.

[2] Haluk Topcuoglu, Salim Hariri, Min-You Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. on Parallel And Distributed Systems*, Vol. 13, No. 3, pp.260-274, March 2002.

[3] T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, and R. Freund. "A comparison study of static mapping heuristics for a class of meta tasks on heterogeneous computing systems". In *8th IEEE Heterogeneous Computing Workshop (HCW'99)*, pages 15-29, Apr. 1999.

[4] G.Sih, and E.Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures," *IEEE Trans. In Parallel and Distributed Systems*, Vol.4, pp.75-87, 1993.

[5] A.Radulescu, and A.van Gemund, "Fast and Effective Task Scheduling in Heterogeneous Systems," *9th Heterogeneous Computing Workshop*, pp.229-238, 2000.