AN ENERGY AWARE RESOURCE ALLOCATION MODEL FOR CLOUD COMPUTING

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

MASTER OF TECHNOLOGY

IN

COMPUTER SCIENCE AND TECHNOLOGY

SHAHADAT HUSSAIN

ENROLLMENT NO. 13/10/MT/024

SCHOOL OF COMPUTER & SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI-110067
INDIA
2015

SCHOOL OF COMPUTER & SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI, 110067 (INDIA)

CERTIFICATE

This is to certify that the dissertation entitled **"An Energy Aware Resource Allocation Model for Cloud Computing"** is being submitted by **Mr. Shahadat Hussain** to **School of Computer and Systems Sciences, Jawaharlal Nehru University New Delhi-110067, India** in the partial fulfillment of the requirements for the award of the degree of **Master of Technology in Computer Science and Technology.** This work has been carried out by him in the School of Computer and Systems Sciences under the supervision of **Dr. Zahid Raza.** The matter personified in the dissertation has not been submitted for the award of any other degree or diploma.

24.7.15

DR. ZAHID RAZA

(SUPERVISOR)

27.7.15

DEAN, SC&SS

JNU, NEW DELHI

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

DECLARATION

I hereby declare that the dissertation work entitled **"An Energy Aware Resource Allocation Model for Cloud Computing"** in partial fulfillment for the requirements for the degree of **"Master of Technology in Computer Science and Technology"** and submitted to School of Computer & Systems Sciences, Jawaharlal Nehru University, New Delhi-110067, India, is the authentic record of my own work carried out during the time of Master of Technology under the supervision of Dr. Zahid Raza. This dissertation comprises only my original work. This dissertation is less than 100,000 words in length, exclusive tables, figures and bibliographies.

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

Shahadat Hussain
Enrollment No. 13/10/MT/024
M. Tech (2013-15)
SC&SS, JNU
New Delhi India -110067

ACKNOWLEDGEMENT

First and foremost, I would like to thank and praise Allah almighty, the only One God, for the opportunities and support I was given along the way. This work would not have been possible otherwise.

I am very glad to express my sincere gratitude and thanks to my supervisor **Dr. Zahid Raza** for his guidance. I would like to express special thanks to Dr. Zahid Raza for many helpful discussions and his intellectual input to make dissertation work worthy. His extensive and invaluable research experiences were very helpful in my dissertation and the most important thing was the helping nature of him that contributes an important share in fulfillment of this work. The methodology, philosophy and problem solving methods learned by him have been very beneficial in this work and would be afterward.

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

I would like to accord my sincere thanks to Dr. Mohammad Shahid, Mr. Raza Abbas Haidri, Ms Mehnaz Perveen, Mr. Mohammad Sajid, Mr. Taj Alam, Mr. Krishan Veer Singh, Mr. Sumit Kumar, Mr. Dinesh Kumar, Mr. Rahul Singh, Mr. Vipin Kumar and Mr. Waliullah for their valuable suggestions for my dissertation work.

My parents and family members have been my strength through the research. I especially thank my father and mother for their patience, unconditional love and economical as well as moral support for completing this dissertation. Finally I would like to express thanks to each person & thing which is directly or indirectly related to my dissertation work.

Shahadat Hussain

Dedicated to

My parents Mr. Azimul Haque, Mrs. Shamsun Nesa
and great grandmother Mrs. Zainul Nesa
for their love, inspiration, support and heartfelt prayers.

Table of Contents

Abstract

Cloud computing is a simple notion that has appeared from heterogeneous distributed computing where all computing infrastructure is provided by a service provider. The end-users simply make use of the services accessible through the cloud computing paradigm and pay in support of the used services. The cloud paradigm offers conceivable form of services, such as computational resources for high performance computing applications, web services, social networking, and telecommunications services. These services are broadly classified as: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure as-a-service (IaaS). SaaS is a kind of services where software hosted by the service provider can be used by many users with only payment for the time it being used. PaaS is a computing set-up that allows creation of application easily and quickly underneath it without the complexity of buying and maintaining the required software. To run any software including applications and operating systems, IaaS provides capabilities to users which comprise processing powers, storage, network and other computing resources.

The pricing models for different types of services differ from provider to provider. Various criteria determine the quality and production cost of the service. The duration of this service (makespan) and the consumed energy are two such important criteria. The idea is to provide end-users with a more flexible service that takes into account the duration of the service and the consumed energy. Applications, services and resources in cloud environment belong to different organizations with different objectives. Task scheduling is a key process in cloud computing infrastructures which can have much impressed on system performance. In general scheduling of tasks problem is shown to be an NP-Complete problem. Because of its key importance this problem has been extensively studied and several algorithms have been proposed in literature for both homogeneous and heterogeneous systems. Precedence-constrained parallel applications are one of the most typical application models used in scientific and engineering fields. Such applications can be deployed on homogeneous or heterogeneous systems (HCSs) like cloud computing infrastructures. Most of the works reported in the literature propose algorithms to minimize the completion time (makespan) with less focus on energy

consumption. Since the problem is NP-complete various approaches with various constraints have been proposed. This work proposes a scheduling model with an algorithm takes into account even energy consumption along with makespan of the task submitted as DAG. The algorithm first considers the variation of energy consumption of processors then selects the job module based on the highest upward rank and schedules it to the processor which dissipates minimum computational energy in an insertion-based approach. The method is based on Dynamic Voltage Scaling (DVS) enabled processors that work in two modes i.e. static when doing no execution along with processing while doing computation for the task and dynamically adjusts the voltage supply level to reduce power consumption. However, this reduction is achieved at the expense of sacrificing clock frequencies. Algorithm is evaluated on a variety of workloads and results shows that it not only reduces the energy consumption, but also maintains a good quality of scheduling too which is very important for the cloud paradigm.

List of Acronyms

ALAP	As-Late-As-Possible
AWS	Amazon Web Services
APN	Arbitrary Processor Network
APIs	Application Programming Interfaces
BNP	Bounded Number of Processors
CMOS	Complementary Metal Oxide Semiconductor
CP	Critical Path
CRM	Customer Relationships Management
DAG	Directed Acyclic Graph
ECC	Expected Computation Cost
ECE	Expected Computational Energy
EFT	Earliest Finish Time
EST	Earliest Start Time
ETF	Earliest Time First
HLF	Highest Level First
HLFET	Highest Level First with Estimated Times
HLFNET	Highest Levels First with No Estimated Times
LP	Longest Path
LPT	Longest Processing Time
IaaS	Infrastructure as a Service
NaaS	Network as a Service
NIST	National Institute of Standard and Technology
P2P	Peer-to-Peer
PaaS	Platform as a Service
QoS	Quality of Service
RMS	Resource Management System
RT	Processor Ready Time
SaaS	Software as a Service
SCFET	Smallest Co-levels First with Estimated Times

SCFNET	Smallest Co-levels First with No Estimated Times
SLA	Service-Level Agreements
TAT	Turnaround Time
TDB	Task-Duplication Based
TSP	Traveling Salesman Problem
UaaS	Users as a Service
UNC	Unbounded Number of Clusters
VMM	Virtual Machine Monitor
VMs	Virtual Machines

List of Figures

List of Tables

Chapter 1

Introduction

The complexity and size of computational world is increasing with a significant pace and requires a computing model that can supports processing of large volumetric data with the use of clusters of commodity computers. Organizations invest time, effort and budgets to meet the ever changing business needs and scales up their IT infrastructure such as software, hardware and other services. However, scaling up process are slow with an on-premises IT infrastructure and optimal utilization of resources are unaccomplished. A paradigm that provides computing over the internet is termed as cloud computing where services are available on demand as pay-per-use basis. Cloud Computing provides computing paradigms where computing occurs without much human interaction even possible at remote locations. It comprises of both the system software and hardware in the data centers that provide the services and delivery of applications as services over the Internet. It facilitates computing resources for any need at a low cost just after a simple connection to the cloud and gives the flexibility to the clients as when client demands or business needs changes companies can easily and cost-efficiently scale up and scale down the amount of resources needed. This helps the saving of capital expenditure for additional on-premises infrastructure. Highly automated information technology services like big data analysis, customer relationship management, self-serviced storage, self-serviced networking and self-serviced computing etc. are provided by cloud. It an enabling revolutionary business models with hardware and software capability getting delivered virtually through devices which is dramatically changing the nature of commerce and society in an effective, efficient and in a much cheaper way. Basically three kinds of service are drawn by the clouds: Infrastructure as a Service (IaaS), Software as a Service (SaaS) and Platform as a Service (PaaS) [1, 2]. With the uses of cloud computing system interface software the cloud can be accessed simply as web based services that host all the files and applications needed for job execution. Cloud computing are deployed in different modes viz. Private cloud, Public cloud, Community cloud and Hybrid cloud. The process of matching operation requirement to the offer of a

cloud service is relatively straight forward; a greater challenge arises when determining which deployment mode to use.

1.1 Emergence of Cloud Computing

Roots of clouds computing can be tracked by observing the several technological advancements in hardware (multi-core chips, virtualization technologies), system management, distributed computing and internet technologies. The maturity of technological advancements itself leads to the cloud computing. Some of those advancements viz. peer to peer computing, cluster computing and grid computing are discussed in the following sections [1].

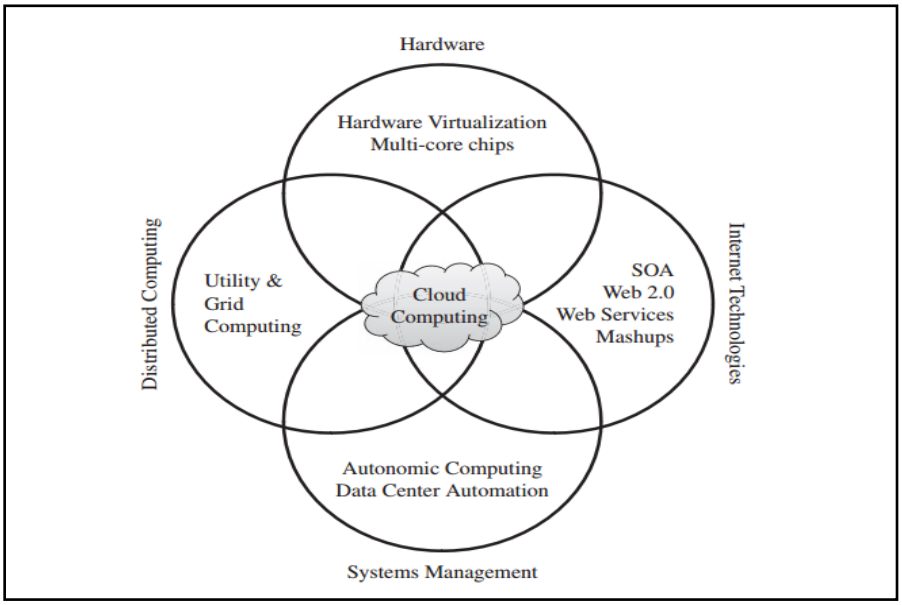

Figure 1.1 Convergence of Various Advances Leading to the Advent of Cloud Computing [1].

1.1.1 Peer-to-Peer Network Computers

Peer-to-Peer (P2P) computing nodes agrees for peer nodes (computers) to share content directly with one or more in a decentralized approach. In pure P2P computing, there is no concept of clients or servers since all peer nodes are equal and concurrently be both clients and servers. The objectives of P2P computing consist of cost sharing or

decrement, resource aggregation and interoperability, enhanced scalability and reliability, enlarged autonomy, anonymity or isolation, dynamism, and ad-hoc communication and collaboration [3]. Peer-to-peer (P2P) network families are decentralized model where each node is referred to as peer have equivalent capabilities and responsibility. At a single instance of time, peers can play the role of server and client i.e. the peer can serve the incoming request from other peers and at the same time initiate the requests on the network. This differs from traditional client/server architectures, in which a client can only send a request to a server which respond after some time. With an increase of numbers of peers added to the networks, the performance of network improves. For the completion of task at hand peers organizes themselves into ad-hoc groups where they share bandwidth, collaborate and communicate with each other. Each peer can leave the group at any time while new ones are joining the group and can perform uploading and downloading simultaneously. Another capability of P2P network is in terms of fault-tolerance where with the use of other peer a P2P application will continue when a peer fails [3]. Examples include Napster, Skype and Gnutella etc.

1.1.2 Clusters Computers

Very often applications need more computing power than a sequential computer can provide. One way of overcoming this limitation is to improve the operating speed of processors and other components so that they can offer the power required by computationally intensive applications.

There are many applications like modeling, simulation and analysis of complex systems such as climate, galaxies, molecular structure, scientific and engineering applications etc. that require high performance computing which is satisfied with the help of cluster computing. In cluster, over a small geographical area various computers with similar kind of operating system, software and hardware are linked together that provides alternative to symmetric multiprocessing with high performance and availability. Characteristics of applications which runs over the cluster includes applications having large run time, real time constraints, large memory usages, high input/output usages and fault tolerance. Unlike other computing paradigms cluster facilitates techniques where similar tasks that are controlled and managed by software are performed by set of computing nodes.

Installations of clusters are based on factors like less maintenance frequency routines, resource consolidation and centralized management, easy scalability and fault tolerance. The same concept as of cluster is followed by super computers that are usually expensive and requires huge operational energy, except the fact that they are not locally inter connected and are already merged into one box [4]. Cluster started taking off in 90's as cluster of IBM, Sun, DEC workstations connected by 10Mb Ethernet LAN, HP clusters, etc.

1.1.3 The Grid

Alternative to the traditional large parallel and distributed system a deployment model where computers from various domains combine together to provide remote access to IT assets while aggregating processing capability is called the grid system. The resource of grid includes all elements of computing including hardware, software, applications and networking devices. The pools of resources are managed by the Resource Management System (RMS) which is central to the system and tackles the issues like scheduling of processors, allocation of network bandwidth and disk storage management. CPU-scavenging, cycle-scavenging cycle stealing or shared computing creates a grid from the unused resources in a network of participants whether worldwide or internal to an organization. Grids generally categorizes in two groups data grids and compute grids. Data grid focuses on data location, data transfer, data access and critical aspects of security on the other hand compute grid provide users with computational power for solving tasks [5]. With grid, teams can come together to get better answers to difficult questions, daunting problems yields immediate results, gain in accuracy and speed with no significant cost increases translating into real competitive advantages in the marketplace. Example includes NorduGrid, OurGrid, Sun Grid, Techila and Xgrid etc.

1.2 The Cloud

Cloud computing has been created as an umbrella term to explain a category of sophisticated on-demand computing services initially offered by commercial providers, such as Amazon, Google and Microsoft. It denotes a model on which a computing infrastructure is viewed as a "cloud," from which businesses and individuals access

applications from anywhere in the world on demand [3]. The main principle behind this model is offering computing, storage and software "as a service."

NIST defines cloud computing as: "*Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"*[6].

Many practitioners in the commercial and academic spheres have attempted to define exactly what "cloud computing" is and what unique characteristics it presents. A paper published by Professors Rajkumar Buyya [3] have defined it as follows: "Cloud is a parallel and distributed computing system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements (SLA) established through negotiation between the service provider and consumers

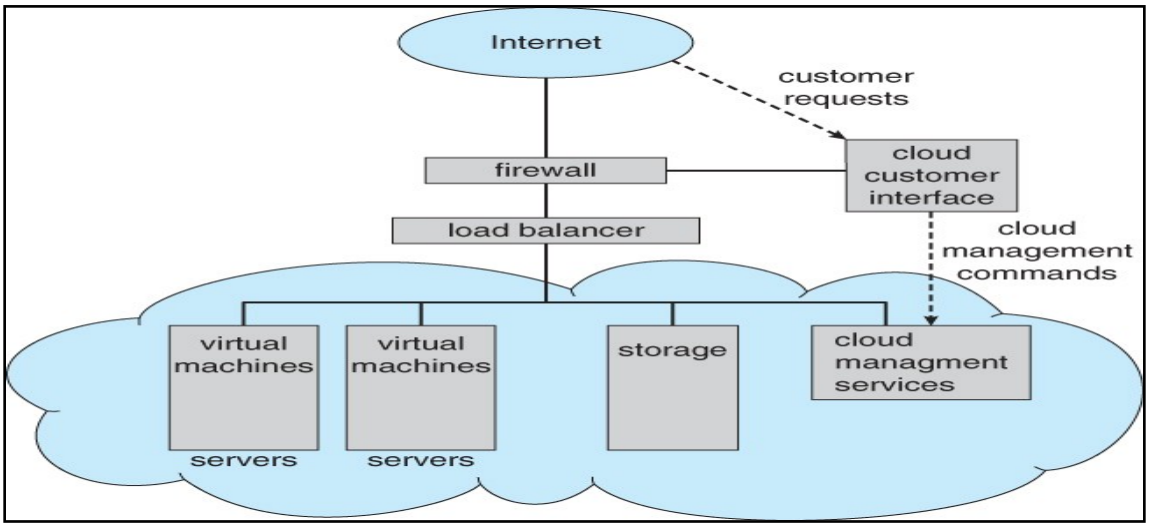

Figure 1.2 Cloud Computing Paradigm [7]

While there are countless other definitions, there seems to be common characteristics between the most notable ones listed above, which a cloud should have: (i) pay-per-use (no ongoing commitment, utility prices); (ii) elastic capacity and the illusion of infinite resources; (iii) self-service interface and (iv) resources that are abstracted or virtualized.

1.2.1 Architecture

Architecture of cloud computing is defined by many researchers and organization. It is defined by NIST with description of five essential characteristics, three cloud service models and four cloud deployment models [6].

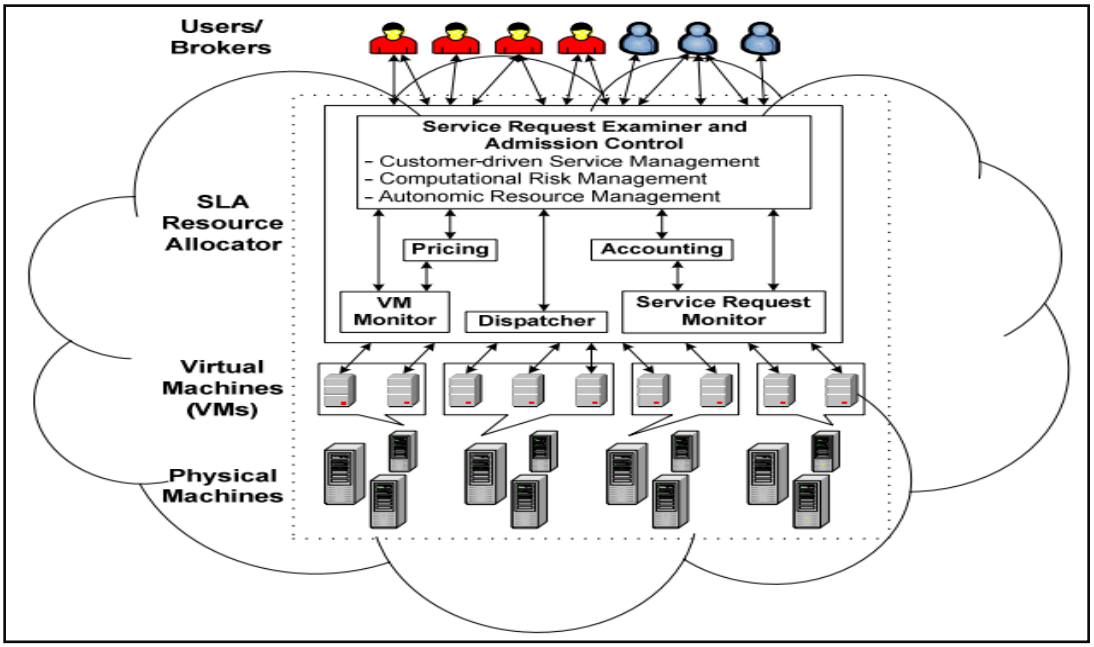

Figure 1.3 The Reference Architecture of Cloud Computing [3].

Basically the core stack and management comprises the whole system. There are three layers in the core stack viz. Resources, Platform and Application Physical hardware, virtualized computing resources, networking resources and storage composed of infrastructure layer or resource layer. Platform layer could be divided into many sub layers and is the most complex part of the cloud system where task scheduling and/or transaction dispatches are managed by a computing framework. Catching capability and unlimited storage are provided by storage sub layer. Figure 1.3 presents the complete overview of cloud computing architecture.

1.2.2 Components

A key aspect of cloud computing in the web application space is that the user is abstracted away from the elastic resources and distributed communication which occurs

within the cloud. This abstraction allows companies to dynamically expand, contract and migrate their computation and storage tasks between various distributed nodes, without the user experiencing any disruption. There are basically four main components involved [8]. A brief description of these components is presented in the following sections.

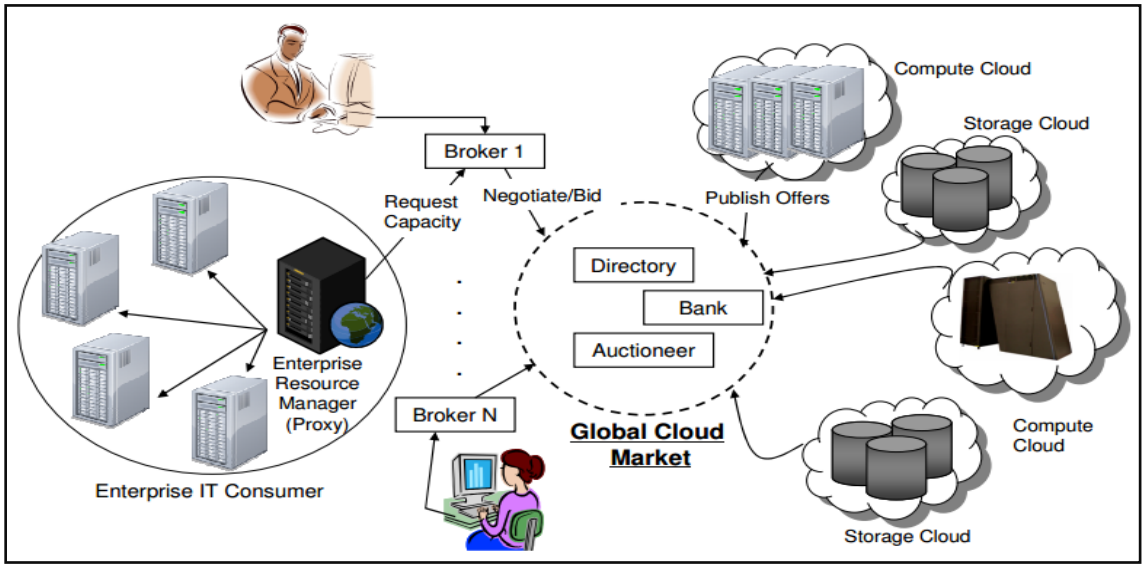

Figure 1.4 Global Cloud Exchange and Market Infrastructure for Trading Services [3]

1.2.2.1 End user

End users or brokers acting on their behalf submit service requests from anywhere in the world to the data center and cloud to be processed. End users are mainly devices that interact with the system. Three categories of clients are there; thick clients e.g. a regular computer that connects to the cloud using a web browser, thin clients e.g. computers having display and with no internal hard drives and mobile device e.g. PDAs or smart phones. In contrast to the traditional in-house infrastructure, where a company had full control over end user devices, the new cloud computing era puts the choice of the device into the hands of the user.

1.2.2.2 Resource Allocator

The Resource Allocator acts as the interface between the cloud service provider and external user/brokers. It requires the interaction of the following mechanisms to support

Service Level Agreement (SLA)-oriented resource management.

- Admission Control and Service Request Examiner: When a service request is first submitted, the Admission Control Service and Request Examiner mechanism deduce the submitted request for QoS necessities before deciding whether to accept or reject the request. Thus, it guarantees that there is no overloading of resources whereby many service requests cannot be fulfilled successfully due to limited resources available.
- Pricing: The Pricing mechanism decides how service requests are charged. For instance, requests can be charged based on submission time (peak/off-peak), pricing rates (fixed/changing) or availability of resources (supply/demand).
- Accounting: The Accounting mechanism maintains the actual usage of resources by requests so that the final cost can be computed and charged to the users. In addition, the maintained historical usage information can be utilized by the Service Request Examiner and Admission Control mechanism to improve resource allocation decisions.
- VM Monitor: The VM Monitor mechanism keeps track of the availability of VMs and their resource entitlements.
- Dispatcher: The Dispatcher mechanism starts the execution of accepted service requests on allocated VMs
- Service Request Monitor: The Service Request Monitor mechanism keeps track of the execution progress of service requests

1.2.2.3 The Platform

Platform in cloud computing is an application over which other applications are launched. It usually comprises of programming languages such as Ajax (Asynchronous JavaScript and XML). In cloud computing, the application is launched to another application called the platform. The platform usually comes as the programming language such as Ruby on Rails or Ajax. Cloud service users should opt the system with

the set of programming languages that runs as the platform which they don't have compatibility issues.

1.2.2.4 Physical Machines

The data center consists of multiple computing servers that provide resources to meet service demands. It comprises of storage devices, servers, network bandwidth, deployment software, platform virtualization and cloud management software.

1.2.3 Service Models

Cloud service models comprise all the cloud service offerings and can be categorize into one or more. Generally three fundamental models are used; Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) as the top levels of taxonomy [6].

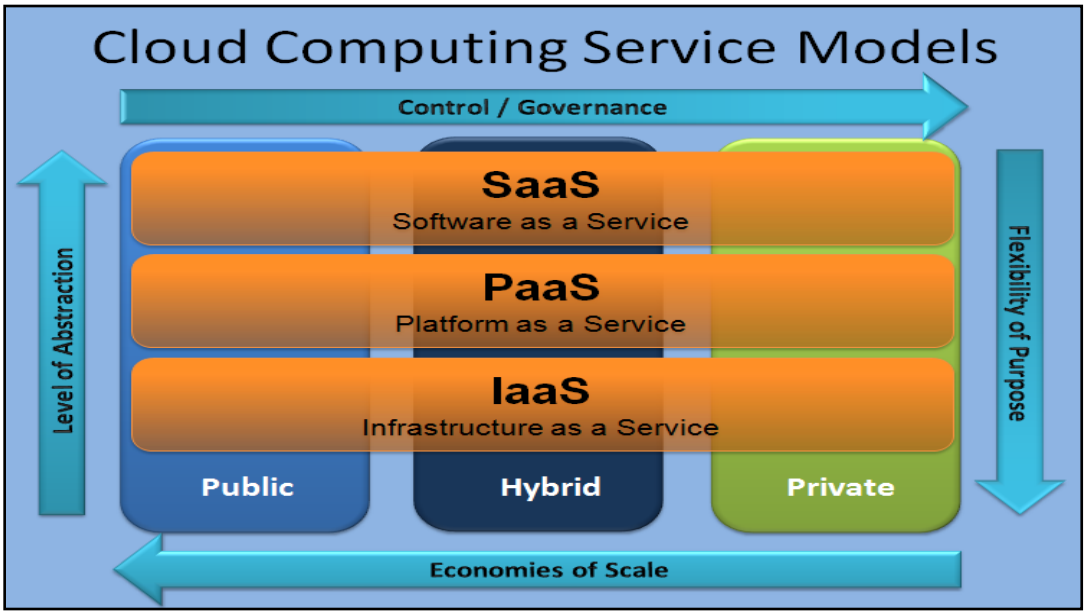

Figure 1.5 Cloud Service Models [9]

Other than these, other service modules like network as a service (NaaS) or even users as a service (UaaS) are used to name a few. The access ability of applications or services that are hosted over the cloud can purchase by consumers on pay-per-use basis. Through various client devices such as thin client interface like web browser or devices having

programming interface cloud application and services are accessible. Migration to the public cloud is expected to accelerate. While other deployment models – such as managed hosting, hybrid hosting, and private clouds – are expected to grow as well, public clouds will undoubtedly be the area of the most substantial growth. Organizations that are better positioned to provide public cloud services are likely to benefit the most; therefore, most new cloud service offerings target public clouds. While these models open up new opportunities with potential benefits, they also present novel challenges that should be considered when deciding upon a solution. Some common concerns in the field are related to security, privacy, and availability of data. These service models are introduce here describes some of the benefits and challenges, and examine relevant case studies that highlight real-world scenarios, challenges and lessons learned. Figure 1.4 presents a formal view of cloud computing service model architecture with their functional components.

1.2.3.1 Software as a Service (SaaS)

SaaS is a kind of services where software hosted by the service provider can be used by many users with only payment for time it's being used. Some example service providers are Google Apps, Customer Relationships Management (CRM) systems and Salesforce. Some major characteristics of SaaS includes: Application Programming Interfaces (APIs) which allows integration between different pieces of software, web access capability to commercial software, centralized software management software, software delivered in "one to many" model etc. Some example service providers are Google Apps, Customer Relationships Management (CRM) system and Salesforce.

1.2.3.2 Platform as a Service (PaaS)

To design, build, test, deploy and update online custom applications a high-level integrated environment is provided by PaaS. Capabilities like access to the platforms are purchased by consumers to deploy their acquired or self developed applications over the platform. PaaS is a computing set-up that allows creation of application easily and quickly underneath it without the complexity of buying and maintaining the required software. PaaS differs with SaaS as instead of software delivered over the cloud PaaS

provides a platform for creation of software. Key examples include GAE, Microsoft Azure etc.

1.2.3.3 Infrastructure as a Service (IaaS)

To run any software including applications and operating systems, IaaS provides capabilities to users which comprise processing powers, storage, network and other computing resources. Rather than purchasing clients instead buy resources as fully outsourced service on demand. Some of the IaaS providers are, Flexiscale, GoGrid, Open Stack, Eucalyptus and AWS.

1.2.4 Deployment Models

Organizations use cloud computing as per their requirements and accordingly they own control over the environment. The way the cloud services are used varies from one organization to the other. There is an aggressive growth in business for cloud adoption in order to cut capital expenditure and efforts. On the other hand, cloud services brings challenges for IT Management and security risks that can be more exorbitant for the organization even considering the cost saving gained after shifting to cloud systems. Therefore, before opting for various deployment models it is very important for businesses to understand their requirements. Four cloud deployment models are recommended by the National Institute of Standard and Technology [6].

1.2.4.1 Private Cloud

The cloud infrastructure is operated exclusive for a single organization and is very customizable. Physical location of servers includes data center on-premises and can be managed in any ways like a third party hosting, the organization itself or some combination of the two. Example of private cloud providers include Nettricity, which allows for the expansion of servers and storage space.

1.2.4.2 Community Cloud

The cloud infrastructure where services are provisioned exclusively for the use of specific community or an organization having consumers with shared concerns. The infrastructure

may be owned and operated by a particular community, a third party or one or more community of an organization or some combination of them.

1.2.4.3 Public Cloud

That service provider is responsible for the management and administration of the systems and services. The client is only responsible for the software and application that are installed on the end-user system. Connections to cloud are usually made through the Internet. Public cloud are managed and operated by third parties giving each individual client an attractive pay-as-you-go model. A popular example of the same could be Amazon Web Services (AWS).

1.2.4.4 Hybrid Cloud

The hybrid cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds). Hybrid cloud is complex and expensive to implement but offers the freedom to implement necessary organizational needs. With the maturity of computing era hybrid cloud implementation is likely become common.

1.2.5 Technological Support

The ability to host different users over the same physical resources is called multitenancy which takes cloud computing apart from earlier computing paradigms. In cloud, the complexity of underlying hardware or software is hidden with the creation of an intelligent abstraction layer supported by virtualization.

1.2.5.1 Virtualization

Virtualization is the key to cloud computing which enables sharing of different operating system over the same hardware and increases the efficiency of resources. Virtualization facilitates increased availability of resources, faster deployment of work load, resource scaling and live migration of virtual machines over different servers [10]. Virtualization technology can be leverage in many ways for example; User State Virtualization where

the end user machine virtualization which stores business critical data, Application Virtualization where line-of-business applications are deployed, managed or maintained and OS Virtualization where a complete desktop experience to entire server virtualization environment can be delivered.

1.2.5.2 Virtual Machines

An operating environment which represents real machine and can host or run operating system using software is termed as virtual machines. In cloud with, background consumers and dynamism, rent and pay for virtual machines enables cloud infrastructure which is positioned in data centers to be shared among numerous users.

1.2.5.3 Hypervisor

A hypervisor or Virtual Machine Monitor (VMM) is a part of computer hardware, firmware or software which runs and creates virtual machines. A computer over which hypervisor creates one or more virtual machines is called host machine. Each virtual machine is termed as a guest machine. Virtual operating platform for guest operating system is facilitated by hypervisor which manages the execution of guest operating system. There are two types of hypervisor in general [10]; Figure 1.5 presents the architecture of different types of hypervisors.

Type 1 Hypervisor: Sits on the bare metal computer hardware like the CPU, memory, etc. All the guest operating systems are a layer above the hypervisor. That means hypervisor is the first layer over the hardware. Example Xen, Denali.

Type 2 Hypervisor: They run over a host operating system, not over the bare metal hardware. The hypervisor is second layer over the hardware and the guest operating systems run a layer over the hypervisor. Example includes VMware.

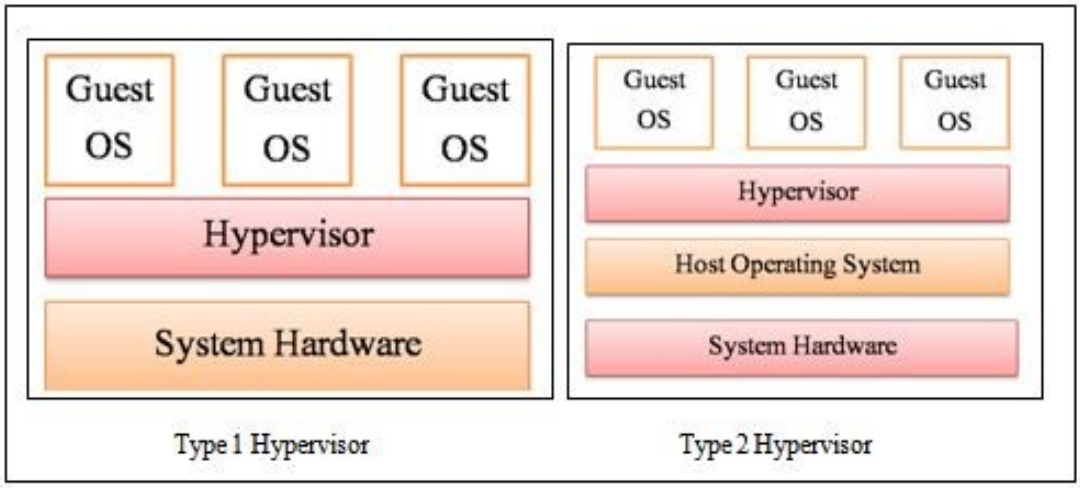

Figure 1.6 Types of Hypervisors with Their Functional Characteristics

1.3 Essential Characteristics of Cloud

The popularity of the cloud is due to it being beneficial to a lot of services providers and organization. Just being a Web based applications do not qualify it to be a cloud application. The service around the application and application itself must demonstrate certain individuality before they can be considered the right cloud implementation. The NIST explanation of cloud computing outlines five key cloud characteristics: on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service. In order for the offering to be considered as true cloud offering all five of these characteristics must be satisfied [1, 6, 11].

1.3.1 On-demand Self-service

This offers advantages for both the consumer and to provider of the services. There is an automated process for resource request and fulfillment of the request. Without help of any support staff or administrator, the customers can request and receive access to the services manually. Implementing user self-services allows consumers to speedily procure and access the services they wish for. Users have the control to change cloud capabilities, such as network storage space and server time as needed. Usually, users are billed for pay-for-what-you-use only.

1.3.2 Broad Network Access

Accessing of business management solutions over cloud requires devices like office computers, laptops, tablets and smart phones. Hence cloud services should be implemented in such a way that they can be easily accessed. To connect to the application or service, consumers should only be required to have a basic network connection. To use the services a large amount of bandwidth must not be required. Mobility is particularly attractive for businesses so that during on off-times or business hours, employees whether they are at home or in the office can stay on projects, contracts and customers.

1.3.3 Resource Pooling

The computing resources by providers are pooled together to serve several customers with a multi-tenant model, where active assignment of dissimilar physical and virtual resources are taking place according to the users requirement. Location liberty is a sensed here as the customer normally has no power or knowledge over the precise location of the provided resources that contain network bandwidth, memory, processing and storage.

1.3.4 Rapid Elasticity

The ability of cloud environment to easily grow and satisfy user demand describes rapid elasticity cloud of environment. Automated and on-demand based resources are provisioned and released. An increased capacity of computing that is needed by cloud environment for a short period of time called burst capacity is handled by cloud efficiently with the help of rapid resource provisioning.

1.3.5 Measured Service

Cloud environment must have the ability to measure usage. Various matrix such as data used, bandwidth used and time used can be the parameter to quantify the usage. Realistic nature of the cloud allows customers only pay for what they use. Based on utilization resource usage are measured, and reported (billed) in a transparent manner. For a particular day when service is not used the consumer should not be charged.

1.4 Major Challenges of Cloud Computing

Amid various possibilities with cloud systems a range of challenges are also associated with it since users for its effective realization. Security issues posses serious threat to the organizations data and application. Moreover, in cloud system a tradeoff between communication, computation and integration must be considered. Business value brought after transition to the cloud by organization are lucrative causing the organizations to stepping into the cloud however thorough knowledge of challenges as well as advantages must be acquired before moving to the environment [1, 2, 3, 11].

1.4.1 Data Security and Confidentiality

Security of data and application is the mainly debated issue in the field of cloud computing which restricts the expansion of cloud to greater extent. Data can be compromised at numerous data-lifecycle stages: during transfer to public cloud from the internal company network, at public cloud during data is stored and during data backup and restore processes. Moreover, in cloud computing the pooled computing resources and the multi-tenancy model has introduced new security challenges that require novel training to deals with.

1.4.2 Energy Consumption

The implementation of cloud computing enables penetrative applications from various domain and provides computing as utility. On the other hand, this implementation brings tremendous energy consumption along with associated concerns and costs. Idle power wastage by servers when it runs at low utilization is one of the major causes of energy inefficiency in data centers. A task scheduling algorithm is needed for minimizing energy consumption in cloud servers, with the help of the task scheduling algorithm, the tasks can be scheduled to a minimum number of servers by maintaining the task response time constraints.

1.4.3 Performance

Service level agreements and delivering the desired QoS defines any cloud providers services. Most cloud provider covers infrastructure availability only and mentions nothing about the performance. If performance-related necessities are required by organization's applications, these requirements should be discussed with cloud providers and ensured to be supported.

1.4.4 Reliability and Availability

Service reliability characterizes the ability of a system to provide acceptable service, which means correct or accurate service delivered within an acceptable time. Service reliability is essentially the portion of service requests that are successfully served (i.e., are not defective) within the maximum acceptable service latency. Cloud reliability is broadly a function of four individual components: the reliability of software and hardware, personnel reliability of provider, connection reliability to the subscribed services and the consumer's own personnel reliability. Also disruption in services is a major worry as consumers might need all their deposited information at anytime. More risk of failure are there as compared to conventional services as there are more weak points in the chain of elements required to access the cloud.

1.4.5 Scalability and Elasticity of Resources

The great advantage of scaling up or down resources to meet workload can lead to service breakdown if it is not implemented properly. Since the cloud does not scale up resources quickly enough a web application developer which hosts its service over the cloud can see how the response time gradually increases with the increase in usage of the applications. Also scaling must be limited by some threshold which stops the continuous increase in the resources allocation that prevents the service denial by the cloud providers when consumer's application malfunctions.

1.4.6 Resource Management and Scheduling

Resource management and scheduling is the critical function of any man-made system that influences the three fundamental criteria for the assessment of a system viz.

functionality, performance and cost. Based on their service-level agreements a cloud system which comprises a set of interconnected and virtualized computers are dynamically provisioned and presented as one or more unified computing resources. Dynamic and efficient allocation of resources are challenging in the cloud computing environment where many alternative computers with varying capacities and capabilities are available.

1.4.7 Interoperability and Portability

One of the major adoption barriers in cloud computing are interoperability and portability. Cloud users must have the flexibility of *migration* in and out of the system and switching to other clouds whenever they wish for without any vendor lock-in period. Migration of applications among clouds is a tough task because of poor *portability* and limited interoperability due to lack of availability of standardized APIs.

Chapter 2

Task Scheduling in Cloud Computing

Scheduling a task on the cloud resources refers to the mapping of these tasks on the computational resources of the cloud in order to meet the scheduling objectives. With regards to the autonomy, scalability and performance of the system, the architecture of a scheduling infrastructure is very significant. Distributed, centralized and decentralized are the three main categories of the scheduling architecture. In distributed scheduling, there are multiple lower-level entities and a central manager which is responsible for assignment of individual task to low-level providers and managing the complete execution of task. More over in centralized scheduling architecture, all scheduling decisions for the tasks are made by a central controller. Here, the track of all available resources in the system and maintaining all the information about the tasks are kept by the scheduler [12-14]. In contrast, the limitations of distributed or centralized structures with respect to autonomy, scalability, fault-tolerance and most importantly the adequacy of resources in the cloud computing environment are wiped out by the decentralized scheduler. It assumes the autonomy of each entity and has as its own control over the resources that derives it to take policy based scheduling decisions [14-15]. Cloud computing service providers' one of the goals is to use the resources efficiently and gain maximum profit. This leads to task scheduling as a core and challenging issue in cloud computing. This chapter presents different scheduling strategies and algorithms in cloud computing.

2.1 Scheduling Objective

Based on some particular objectives, schedulers generate the mapping of tasks to resources. To get optimization of a specific outcome scheduler makes use of a function which takes into account the needed objectives. The normally used scheduling objectives in a cloud computing environment are associated with the resource utilization and tasks

completion time [14]. In order to satisfy the requirement of consumers, the schedulers use a particular strategy for mapping the tasks to appropriate cloud resources.

2.2 Entities Coordination

Entities in the cloud are analyzed as autonomous units that have their own will in sharing their capabilities and are able to perform some computations. The management of interdependencies between the independent computational units that have no global control is a big challenge with cloud system. Synchronization between different entities controls the effectiveness of managing the interdependencies of entities in the cloud environment. Consequences of lack in coordination bring overhead in communication which decreases the system performance [13-14]. The process of coordination in the cloud environment with respect to the resource management and scheduling of application or services entails dynamic information exchange among a range of entities.

2.2.1 Coordination Mechanism

Problems linked with interdependencies are cut and resolved by coordination mechanism which includes interaction protocols and a set of decision points for coordinated-control that are bounded for to dealing with the interdependency problems. Interaction protocols are the methods by which an entity cooperates with another entity through various communication protocols. Efficient coordination between entities in the cloud involves a plenty of negotiation policies and coordination mechanisms. The coordination mechanisms in cloud computing environment are generally Market-based which views the cloud environment like a virtual Marketplace where buying along with selling of services, storage resources and computation are done by cost-effective entities that work together with each other. Efficient resource allocation is facilitated by such coordination mechanism where the resource provider that export its confined resources to contractor works as a supervisor and decisions about admission control based on negotiated Service Level Agreements are taken care by resource brokers.

2.2.2 Coordination Structure

The pattern of communication and decision making which is required while resolving the complexities related with interdependencies among entities is termed as coordination structure. With the utilization of some specific communication devices interaction with entities are coordinated. These devices are classified into two categories: One-to-one and One-to-many. One-to-one interaction with the consumers and resource providers as per Service Level Agreement is facilitated by these devices. On the other hand One-to-many communication is simple but expensive as it broadcasts the number of messages over the network that uses bandwidth of the network.

2.3 Scheduling –an NP Complete Problem

Finding an optimal schedule for a set of jobs is an NP-complete problem even in the following two restricted cases: (1) only one time unit is required by all the jobs and (2) only two processors resolving there and all jobs require one or two time units [16]. As an effect the preemptive scheduling problem of all-purpose is also NP-complete. These results are practically the same for proving that the scheduling problems stated are intractable.

2.3.1 Class of Problems: P and NP

There are two families of problems P and NP that are very significant in field of computer science. These families comprise the vastness of our practical computational problems and have been essential to the theory of computation for years. Turing machine, introduced by Alan Turing in 1936 is the standard computer model in computability theory which describes the problems precisely. The classification of problems with assumptions in categorization is presented in Figure 2.1.

2.3.2 P Class of Problems

Introduction to Algorithms by Thomas H. Cormen describes the classes as "The *class P consists of those decision problems that are solvable in polynomial time. More specifically, they are problems that can be solved in time O* (n^k) *for some constant k,*

where n is the size of the input to the problem." [17]. So P is just the set of decision problems for which polynomial-time algorithms exists and is also called a set of tractable decision problems. Example includes testing whether a number is prime or not.

2.3.3 NP Class of Problems

The second class of decision problem where the notation NP stands for "*non deterministic polynomial time*" has been described as "*The class NP consists of those problems that are* ***verifiable*** *in polynomial time. It means that if we were somehow given a solution of the problem, the same can be verified in time polynomial in the size of the input to the problem."* The definition of NP entails the idea of a non-deterministic algorithm [17].

2.3.3.1 NP – Hard Problems

The decision problem P_i is NP-hard if every problem in NP is polynomial time reducible to P_i. In very much relaxed way, it means that P_i is 'as hard as' all the problems in NP. If P_i can be solved in polynomial-time, then so can all problems in NP. Equivalently, if any problem in NP is ever proved intractable, then P_i must also be intractable [17-18]. Example include traveling salesman problem (TSP).

2.3.3.2 NP - Complete Problems

A decision problem P_i is NP-complete if: it is NP-hard and it is also in the class NP itself. The informal and relaxed description states that P_i is one of the hardest problems in NP. So the NP-complete problems form a set of problems that may or may not be intractable but, whether intractable or not, are all, in some sense, of equivalent complexity. Example problem of finding an optimal schedule for a set of jobs onto a set of processors is *NP*-complete [16].

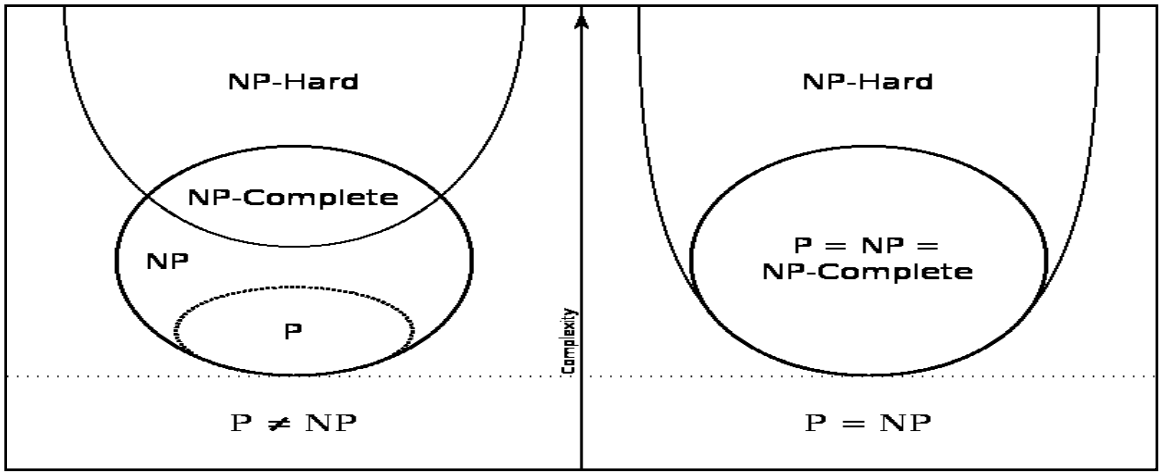

Figure 2.1 Euler Diagram for P, NP, NP-Complete and NP-Hard Set of Problems [19]

2.3.4 Dealing with NP Complete Problems

To address the NP-complete set of problems various approaches can be used. Some of these are listed as follows.

2.3.4.1 Brute Force

Since when the concept of NP-completeness was first developed computers have now acquired very much speed. Nowadays computers are much faster than long years back. These days various moderate size problems can be solved without difficulty by the use of some clever algorithms. For some small problem instances brute force search through all possibilities is now possible. The NP-complete traveling salesperson problem asks for the smallest distance tour through a set of specified cities. Using extensions of the cutting-plane method we can now solve, in practice, traveling salespeople problems with more than ten thousand cities [20].

2.3.4.2 Approximation

Although exact solution of NP-complete optimization problems cannot be obtained however a good approximate answer often can be achieved. For example- Arora [21] gives an efficient algorithm that digs in very close to the best possible path while considering the traveling salesman problem which is an NP-complete problem.

2.3.4.3 Heuristics and Average-Case Complexity

The center of attention in analysis of NP-completeness is how algorithms execute over worst possible inputs. Though, the exact problems that arise into practice can be probably much easier to solve. Various heuristics comes up from specific problems in their fields employed by many computer scientists to solve NP-complete problems [22]. Role of average-case complexity in areas of computer science is also significant in this case like greedy algorithm for Independent Set which uses the strategy to always pick the vertex which has fewest neighbors. This vertex is now in the independent set and all of it neighbors can now never be in the independent set.

2.4 Scheduling Problem in Cloud

Cloud system as explained in earlier section is a decentralized computing environment where distributed entities in the environment are intended to take the scheduling decisions. The job is to be executed by the cloud services provider for the consumers. Therefore, the controls and knowledge for scheduling criteria are distributed between two major entities of cloud environments: Consumers and Providers [14].

2.4.1 Overview

Representation of requests within the cloud is viewed as the series of activities that are necessary to complete a task called as workflow where tasks are piled according to data flow, computational dependencies and service dependencies. Workflow is categorized as data intensive when data centric necessities such as retrieval of large volume of data and hefty storage space are high. Similarly, a workflow is categories as computation intensive when task requires high amount of computation. Also various categories of workflow are there based on different optimization parameters. Scheduling a workflow is a procedure to discover the mapping of tasks within workflow to the appropriate resources so that the execution can be completed with the fulfillment of objective functions, such as execution energy minimization and execution time minimization. Present workflow scheduling practices are non-coordinated and work independently while performing scheduling related activities. This leads to mostly under-utilization of some valuable resources or

over-utilization and a bottleneck on various others. Further, load sharing and utilization problems of cloud system resources got worse because of lacking in coordination mechanism during brokering process. Considering the dynamic resource behavior in the cloud an optimized workflow execution can be obtained by cooperative decision making while scheduling in an open environment.

2.4.2 Scheduling Problem and Structure

The entities (consumers and providers) in the cloud environment are accountable for their own decision making and are mostly autonomous. In that scenario the scheduling problems gains an additional attribute from the characteristics of the environment. That means the whole knowledge about the problem is not widely spread and no entity in the environment have the global vision about the problem. This problem in general termed as distribution of knowledge problem. Two following definitions are introduced accordingly [3, 14].

Definition 1: "A Distributed Scheduling Problem is characterized by the knowledge of the problem is distributed among entities and no entity has a global view of the problem." Further, the decision making capabilities and autonomy of entities are also the required characteristics in cloud environment. It means that entities are not controlled by any other entity and are driven by its own objectives.

Definition 2: "A Decentralized Scheduling Problem is a Distributed Scheduling Problem consisting of self-interested entities and is autonomous in their decision making." Distribution of control is an essential characteristic of decentralized scheduling problems that means outside body such as other entities in the environment should not be involved in entities policy making process. To achieve the respective objectives of entities self-interested entities need to be cooperative with each other and cooperation must not enforced by binding agreements from a third parties.

2.4.3 Task Scheduling Over Cloud

Cloud systems accept application requests generally in two ways that depends on to the architecture of systems. In one scenario tasks are submitted by consumers dynamically to the scheduler. These tasks are generally added in a queue and the schedulers take the

tasks from this queue for scheduling to computing nodes which process the tasks after completion of the current running task [14]. On the other hand in scheduling with broker mechanism, clients put a message on the task queue to initiate a task. Each message corresponds to a particular task. Task can contain additional parameters like the queue name, countdown timer and much more. These arguments are parsed by the celery and saved appropriately for each task. Tasks are assigned to worker nodes, as per the scheduling algorithm followed. The broker then delivers the message to a worker. Worker nodes listening to the queue, on receiving the task, adds the task to its own queue, and executes it when its turn arrives. On task execution, results are published to the configured result backend. And thus the customer should be in a position to retrieve results from the result backend as and when required [3].

2.4.4 Task Scheduling versus VM Scheduling

Task scheduling is based on the requirement of a user who has a set of jobs to compute and is a vital research topic in cloud computing. The consumers submit set of jobs to the job scheduler which allocate them over to the resources available in the cloud.

Virtual machines (VM) provisioning on the other hand is the procedure of creating VM instances over the host machine of cloud provider that matches the configurations (software environment), critical characteristics (storage, memory) and requirements (availability zone) of the provider [3, 10].

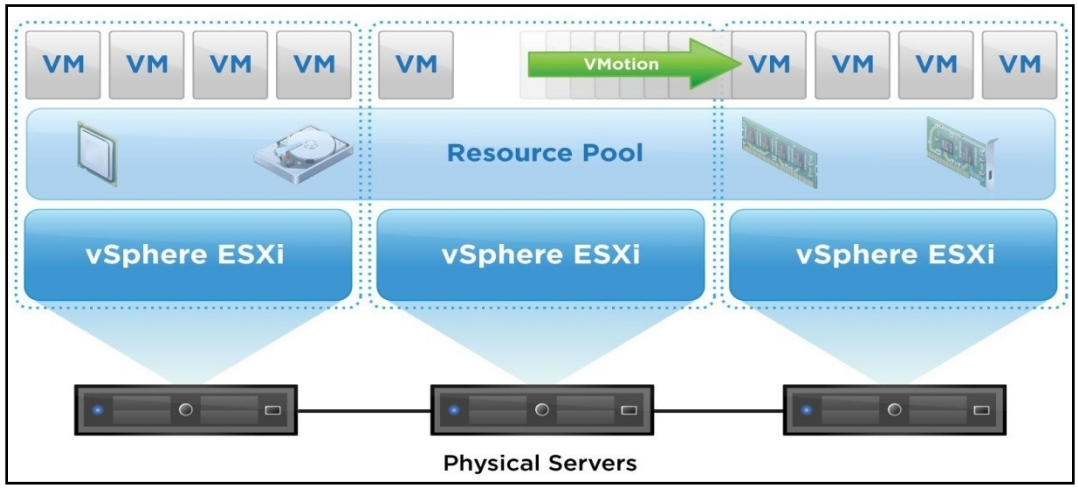

Figure 2.2 Virtual Machine Provisioning on Physical Servers [23]

2.5 Classification of DAG Scheduling Algorithms

A parallel program can be represented by a node and edge weighted directed acyclic graph (DAG), in which the node weights represent task computation costs and the edge weights represent data dependencies as well as the communication costs between tasks. At the highest level scheduling problem are divided into two classes, depending upon the structure of the task graph whether is of special structure like tree or an arbitrary structure. Majority of the algorithms presume the graph to be a kind of special structure such as a forks-join, tree etc. Moreover parallel programs come in diversity with their structures usually, and as such a lot of current algorithms are intended to deal with the arbitrary graphs. These algorithms can be divided further into two classes based on uniformity and randomness in computation cost of the tasks group [13, 15, 24].

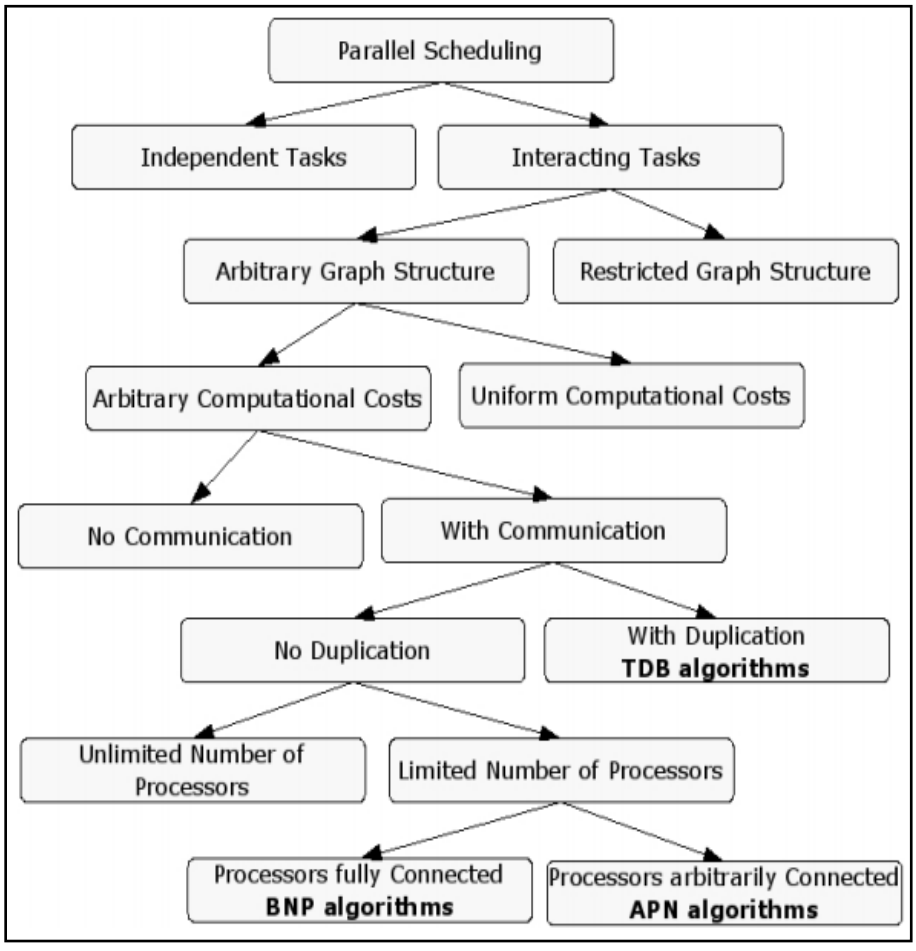

Figure 2.3 A Partial Taxonomy of the Multiprocessor Scheduling Problems [24].

A typical taxonomy of the task scheduling algorithms on multiprocessor environment is presented in Figure 2.2. Scheduling algorithms which regards the inter-task communication assumes the accessibility of an unrestricted number of processors and are called the UNC (unbounded number of clusters) scheduling algorithms, whereas other algorithms that assumes fixed number of processors and are called BNP (bounded number of processors) scheduling algorithms. The processors are supposed to be totally linked in both classes of algorithms and require no attention to routing strategies or in contention linking used for communication. UNC algorithms employ technique which is also known as clustering where every module is considered as a cluster at the start of scheduling process. In the later steps, two clusters are combined if the combining results decrease in the time of completion and combining process goes on until the clusters allows it. Conversely, the BNP algorithms do not require any such post processing step. With the most general mode in consideration, there has been a design of few algorithms like APN (arbitrary processor network) scheduling algorithms which in addition to scheduling the modules also schedules messages over the link for network communication. The TDB (Task-Duplication Based) scheduling algorithms schedule tasks over processors with replication which decreases the scheduling lengths and also assumes the accessibility of boundless number of processors. The motivation behind the TDB scheduling algorithms is to diminish the communication overhead through redundant allocation of various to a number of processors [24].

2.5.1 DAG Scheduling Preliminaries

The majority of scheduling algorithms are relying upon the so-called list scheduling methods [24]. The fundamentals of the list scheduling is to make a scheduling sequence of modules (an ordered list of modules for scheduling) and then execute the following two steps repeatedly until all the modules in the graph are scheduled: (1) Take out first module from the scheduling sequence and (2) Assign the module to a processor that posses earliest start-time. Priorities of modules can be determined by various methods like, as HLF (Highest Level First) Coffman [25]; LP (Longest Path) Coffman [25]; LPT (Longest Processing Time) Friesen [26]; and CP (Critical Path) Graham [27].

In recent times numerous scheduling algorithms based over dynamic list scheduling approach have been recommended [24] where after each allocation of module priorities of unscheduled modules are recomputed using the algorithms. Potentially better schedules are generated using this approach however the time complexity of the scheduling algorithms are increase by dynamic approach. For assignment of priorities of job modules the two commonly used traits are the b-level (bottom level) and t-level (top level). In addition with these some of the algorithms employ an attribute called As-Late-As-Possible (ALAP) start-time. A brief description of these algorithms is presented in the following sections [24].

2.5.1.1 Computing a t-level

The length of longest path from an entry module to a module m_i (excluding m_i) is termed as *t-level* (top level) of the module m_i. There may exist more than one longest path that constitutes a single *t-level*. Sum of all modules and edge weights down the path comprises the length of a path.

2.5.1.2 Computing a b-level

A key structure in directed acyclic graph (DAG) called critical path (CP) which is sequence of tasks where each task dependent on the next and prevents an earlier finish. The *b-level* (bottom level) of a module is restricted from above by the length of a CP. In the scheduling process, the t-level of a node varies while the b-level is usually a constant, until the node has been scheduled.

2.5.1.3 Computing ALAP

An attribute described as *As-Late-As-Possible* (ALAP) start time is utilized by many DAG [24]. Without any further increase in schedule length ALAP start-time quantifies by how far the start time of module can be postponed without increasing the schedule length.

2.5.2 Brief Survey over DAG Scheduling Algorithms

Starting with description of some former scheduling algorithms that works over restricted DAG models to the various algorithms that get rid of all simplifying assumptions a brief

surveys over DAG scheduling algorithms that are reported in literature are presented here. The discussion comprises description of various classes of known algorithms.

Scheduling DAGs with Restricted Structures: Early scheduling algorithm designs were based over simplifying postulation regarding the DAG as well as processor network model [28]. For example, the modules in the DAG were presumed to be of unit computation cost with no communication overhead. In this regard a polynomial-time algorithm for in-tree structured DAGs was proposed by Hu [29] which creates best possible schedules with unit computations and with no communication however the numerals of the processors $\boldsymbol{p}$ was assumed to be restricted. The critical step in the algorithm is a module labeling procedure that partitions the task graph into many levels and during scheduling practices tasks of same level is allocate to the available processors.

Scheduling Arbitrary DAG without Communication: This includes random structured DAGs having arbitrary computational costs and zero communication overhead. Adam [24] carried out a broad simulation examination over the performance of various level-based list scheduling heuristics which strongly suggest the proof that the CP (critical path) based algorithms have near-optimal solution. The examined heuristics includes HLFET (Highest Level First with Estimated Times); HLFNET (Highest Levels First with No Estimated Times); Random (nodes in the DAG are assigned priorities randomly); SCFET (Smallest Co-levels First with Estimated Times) and SCFNET (Smallest Co-levels First with No Estimated Times). One more study carried out by Kohler [30] over the above heuristics implies the concept that with the increase in the number of processors the performance of the critical path based algorithms improves.

Pappadimitrou and Yannakakis [31] proposes a linear time algorithm which computes most favorable solution for scheduling a unit computational interval-ordered DAG to various processors. In an interval-ordered DAG, two modules are precedence-related if and only if the modules can be mapped to non-overlapping distance over the real line. Using the property a list can be constructed by taking the number of successors of a module as priority. The best possible schedule can be obtained in O (v +e) where v and e are number of vertices and edges respectively of the DAG. However the removal of constraints for example unit computational tasks makes the problem NP-complete.

The foremost clustering algorithm that is not a list scheduling algorithms is the Sarkar's algorithm [32]. Primarily it arranges the edges of the DAG in decreasing order of their weights. Afterward at every clustering step the edges are visited in the ordered list as they appear in the list and zeroed if there is no increment in the parallel time. The computation of the parallel time has to be done at all clustering step topologically passing through the scheduled DAG. The complexity of algorithm is O (e (v + e)) as there are e such traversals.

The HLFET (Highest Level First with Estimated Times) is again a list scheduling algorithm that initially calculates the static *b-level* for every module and after that builds a ready list in non ascending order of static *b-level*. Then repetitively it schedules the entry module in the ready list to a processor which facilitates the earliest start time and updates the list with the new ready modules. The algorithm works with O (v^2) time complexity.

ETF (Earliest Time First) focuses over keeping the processors busy and can be considered as a model closed load balancing scheme. Priorities of unmapped tasks ready for execution are computed at each scheduling step; with the help of tentative mapping of given task to all the processors the task priority which is simply the earliest start time of tasks are determined. The task with the lowest priority is selected and mapped to the processor assign corresponding to this priority. Statically, computed priorities can be used for breaking the ties. The algorithm works with the time complexity of is O (v^2p) where v and p are number of modules and processors respectively. Since the heuristics is not based over critical path the most important ready task is not mapped first always. As a result the reduction of partial schedule length at every scheduling step may not be achieved. Since start-time of a node over all the processors is used exhaustively for the calculation of earliest start time of a node it is also a BNP algorithm [24].

2.6 Scheduling Properties

In the cloud environment resources are put up for sale in the marketplace and customers are required to have capability to portray their preference [3, 10, 14]. Based over the characteristic of the cloud infrastructure various properties are identified to facilitate a plenty of bidding language design. The sufficiency in bidding language is acknowledged

according to the properties as well as necessities of the cloud entities (consumers and providers) and the kind of the scheduling solution.

2.6.1 Time-based Requirements and Availability

Clients have time-based necessities for execution of tasks. Execution of tasks after the time limit is not preferred and may have no significance. Therefore the bidding language is to allow the consumers to convey their choice over time related constraint for their tasks. *Start time and End time* are the two variables that generally describe the time range of prerequisites.

2.6.2 Support for Requirements

When the request is being executed the specific requirements of consumers towards computational resources (operating system, memory and speed etc.), services and storage resources (storing capacity) must be satisfied. The bidding language should encourage consumers to convey the choice as per the necessities as well as its support for providers to portray the reserve values on the resources capability.

2.6.3 Support for Allocation Constraints

Workflow of tasks supplied by consumers in cloud environment requires an accurate execution of sequence that means the logic of the workflow must be represented by the execution which obeys the allocation constraints.

2.6.4 Multiple Consumers and Multiple Goods Expressiveness

The cloud environment supports several consumers that demands to make use of multiple utilities that are offered by various providers. In cloud environment multiple service providers make public their resources through a middleware which consists a global directory and is also provide supports to multiple consumers request while discovering the resources. A market system is to consume those services as well as set up a market structure which allows various buyers to consume a range of utilities owned by various providers. A bidding language is needed to facilitate the expressiveness for such market structure.

2.6.5 Trade of Resources

In cloud environment role played by entities are not similar always. A computation provider turns into a consumer while it needs use of some definite service from a different provider. Also, for the execution of their services a service provider may require additional computational resource owned by some other individuals or group. Trades among entities allow flexibility for entities to make full utilization of their resources and at the same time get hold over the necessary resources within a definite budget. A bidding language should facilitate the expressiveness of consumers and providers of the resources to do business.

Chapter 3

The Proposed Model

The model under consideration is a task scheduling strategy which supports execution of computationally intensive dependent modules over a diverse set of interconnected processors. Hereafter, we use the terms task and module interchangeably in the chapter. The algorithm considers energy consumption differences by various algorithms and introduces an energy efficient strategy for allocation of job modules to processors. The pseudo code of the proposed energy-aware algorithm (alloc_energy) is presented in section 3.7. The proposed algorithm takes two inputs viz. a computationally intensive job which comprises set of dependent modules, a heterogeneous interconnected computing system and returns schedule with turnaround time with energy consumption as output. The overall functioning of the model comprises two phases: *phase I* computes rank of the modules in order to produce priority list of modules whereas *phase II* selects the appropriate processor based on expected computational energy (ECE) metric. The model works as a static scheduler where information regarding all the computing resources as well as all the tasks in the job is assumed to be available by the time the job is scheduled. Jobs are submitted and the scheduler evaluates the suitability of job modules which results in an improvement in the energy consumption by the processors. The chapter starts with presentation of the scheduler, the notation, equations and data structures used, job characteristics for the design of scheduling model. Later, the energy aware scheduling algorithm is presented followed by the illustrative example, simulation results and their analysis.

3.1 The Scheduler

The scheduler system comprises an application, a targeted computing environment and performance criteria for scheduling. The application or job is represented by a *directed acyclic graph (DAG)* or task graph as $\boldsymbol{G = (V, E)}$, where $\boldsymbol{V}$ is set of $\boldsymbol{v}$ job modules ($\boldsymbol{m_1, m_2, m_3}$, …, $\boldsymbol{m_n}$) that are the logical unit comprising of instructions and $\boldsymbol{E}$ is set of $\boldsymbol{e}$ edges

between the modules. These modules can be independent or dependent from other modules of the same job. Independent modules can be executed without any delay but dependent modules have to wait for their parents to initialize their execution. Each node of the DAG is labeled by the module number which is used as an identification number module and each edge is labeled by the communication cost (expected communication cost) between modules. For a given task graph a module without any parent is termed as entry module m_{entry} and module without child is called as exit module m_{exit}. The primary goal of a scheduler is to run the submitted job and this aim must be achieve at the end. Modules runs over various processors facilitated by computing environment that are the nodes actually executing the application or job. The target computing environment is assumed as it consist of a set of p heterogeneous interconnected processors where all the inter processor communication are assumed to be performed without contention. When a job is submitted to the scheduler it creates a list of modules from the given task graph according to their priorities in non increasing order. Following that it selects the ready module with highest priority from the list and assigns it to the suitable processor that minimizes the processing energy of the module. The processing energy of the job is calculated as the sum of individual computational energy of job modules. The type of scheduling algorithm under the study is recursive and static as the scheduler has all the information about task precedence, parallelism in modules and architecture of processors inter-connectivity a priory. The proposed scheduler is designed to minimize the execution energy of the job submitted for execution and generates a schedule based on the algorithm. The priority mechanism and parameters for evaluation of processing energy of modules are explained in the upcoming sections of this chapter.

3.1.1 Notation Used

The following notations are used in the model:

- G **:** Directed Acyclic Graph (DAG) or job
- V **:** Set of Vertices of in the Directed Acyclic Graph
- n **:** Number of vertices in the Directed Acyclic Graph
- E **:** Set of Edges in the Directed Acyclic Graph
- m_i **:** i^{th} Module of the Job G

- e_{ij} **:** Communication Messages and Precedence Constraints Between Modules m_i and m_j.
- m_{entry} **:** First Module of the Job
- m_{exit} **:** Exit Module of the Job
- $\mathbf{E_g}$**:** Number of Edges in the Directed Acyclic Graph
- C_{ij} **:** Communication Cost Between Module m_i and Module m_j
- p **:** Total Number of Processors
- p_k **:** A Processor From the Set of Processors p Whose Identification Number is k
- f_x **:** Operating Frequency of Processor p_x
- $\mathbf{rank_u}$**:** Upward Rank of the Job Module
- v_x **:** Voltage Required to Sustain the Frequency of Processor p_x
- c_x **: C**apacitance of the Processor p_x.
- ρ_i **:** Power Dissipated by the Processor p_i
- W_{ij} **:** Expected Computation Cost to Compute Module m_i on the Processor p_j
- $\bar{w}i$ **:** Average Computation Cost of Module m_i
- E_{static} **:** Ii is the Energy Dissipated While Processor is Sitting Idle
- $E_{processing}$**:** It is the amount of energy consumed when the processor is specifically processing the modules
- E_{total}**:** Total Computational Energy for the Submitted Job

3.2 Heterogeneous Computing System

The proposed algorithm assumes the target computing environment is a set p of interconnected heterogeneous processors and is represented by $p = \{p_x : 1 \leq x \leq N\}$. Each processor is represented by a vector (f_x, v_x, c_x) where f_x represents the operating frequency of the processor p_x, v_x is voltage required to sustain the frequency of processor p_x and c_x is the capacitance of the processor p_x which is decided at the processor manufacturing time. Heterogeneity of processors means the processors have different speeds, operating voltage and processing capabilities. Each processor is composed of a local memory unit and do not share memory of other processors so the communication relies solely on message-passing. The algorithm assumes the processors in consideration

are operating in two states. That means either they are processing a job module or sitting idle. Depending upon the state of operation they draw two different voltages from the power supply and correspondingly operate with two clock frequencies. Power consumed by the processors depends on their state of operations details of which is described in detail in section 3.6 of this chapter.

3.3 Job Characteristics

A job is assume to be in the form of *directed acyclic graph (DAG)* $G= (V, E)$, where V is set of v nodes and E is a set of e directed edges. A node in the DAG corresponds to a module ($m_1, m_2, m_3, \ldots, m_n$) which a set of instructions that must be executed sequentially without preemption over a single processor. The weight of a module m_i is called the computation cost over a particular processor and is denoted by w_i . *ECC* is a $V \times p$ *Expected Computation Cost* matrix in which the entry $ECC(i, j)$ denotes the estimated execution time to compute module m_i on processor p_j. We presume every module of a parallel program can execute at any processor although the completion times on different processors may be different. The edges in the DAG which is denoted by e_{ij} correspond to the communication messages and precedence constraints between the modules m_i and m_j. The weight of an edge between two modules m_i and m_j is called the communication cost of the edge and is denoted by C_{ij}. The source module of an edge is called the parent while the sink module is called the child. A module with no parent is called an entry module and a module with no child is called an exit module. The precedence constraints of a DAG indicate that a module cannot start execution before it gathers all of the messages from its parent modules. The communication cost between two modules assigned to the same processor is assumed to be zero while over different processor it is non zero. A sample job with the above considerations is presented in Figure 3.1

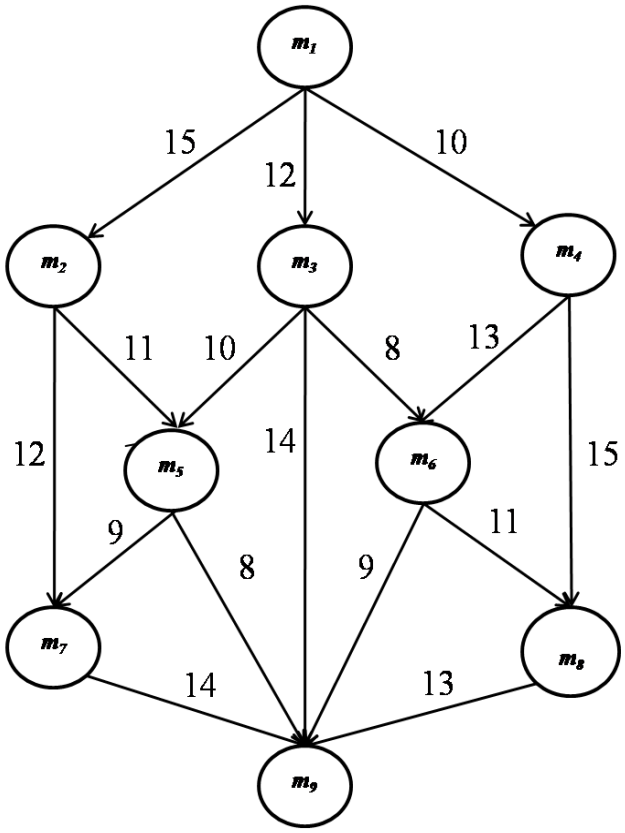

Figure 3.1: A Sample Job Representation by Directed Acyclic Graph (DAG).

3.3.1 Data Structures and Parameters Used

The proposed scheduler considers the following data structure:

- **Graph[*n*][*n*]:** A matrix of size ***n*×*n*** where ***n*** is number of modules in the job. Graph[i][j] represents communication cost from module m_i to module m_j which is equivalent to C_{ij}. Since a DAG has no self loop or cycle therefore the value of Graph[i][i] will always be equal to zero.
- **Module Node:** The Module Node contains information about a Module number and Level of module.
- **ECC**[*n*][*p*]**:** A matrix of size ***n*×*p*** where ***n*** represents the number of modules in the job and ***p*** represents the total number of processors. The entry ECC[i][j] represents the expected computation cost of module m_i over the processor p_j.

- **$rank_u[m_i]$:** It is an one-dimensional array where entry rank[m_i] represents a numeric value for module m_i that corresponds to the upper rank for the module.
- **ECE[n][p]:** A matrix of size $n \times p$ where n represents the number of modules in the job and p represents the total number of processors. The entry ECE[i][j] represents the expected computation energy of module m_i over the processor p_j.
- **Succ(m_i):** An array containing list of all immediate successor modules of module m_i.

In order to achieve efficient schedule of a job *G* on heterogeneous processors, it is required to define the following basic parameters:

- ***Processor Ready Time*** (RT_i)**:** It is the time duration for which processor p_i is available for execution.
- ***Earliest Start Time*** (EST_i): It is defined as starting time of module m_i on the processor where it can be started as early as possible. It can be written as

  $EST_i = 0$, If the number of immediate parents are zero.

  $$EST_i = \max_{m_1 < j < (m_k)} \{ EFT_j + c_{ji} \},$$ If number of parents are k. - - - **(1)**

  Where k (>0) represents number of immediate parents [34].
- ***Earliest Finish Time*** (EFT_i)**:** Earliest finish time of the scheduled module m_i over the processor p_j is defined as the sum of ***EST*** and Computation Time W_{ij} of the module. It can be written as [33].

  $$EFT_i = EST_i + W_{ij} \quad \text{----------(2)}$$
- ***Turnaround Time*** (TAT_j)**:** It is defined as the total time taken between the submission of the job for execution and return of the completion output to the user which is calculated as [33]

  $$TAT_i = max(EFT_{m_{exit}}) \quad \text{----------(3)}$$

3.4 Prioritization of Job Modules

Job modules are sequenced in the algorithms by their scheduling priorities that are based on upward rank which is computed recursively by traversing the directed acyclic graph

(DAG) upward beginning from the exit module. Before scheduling each module m_i is labeled with the average computation cost $\bar{w}i$ which is defined as:

$$\bar{w}i = \sum_{j=1}^{p} \frac{w_{ij}}{p} \quad \text{----------} \quad (4)$$

For the exit module m_{exit} the upward rank value is equal to:

$$\text{rank}_u(m_{exit}) = \bar{w}_{exit} \quad \text{----------} \quad (5)$$

The upward rank of a module (m_i) other than exit modules is recursively defined as:

$$\text{rank}_u(m_i) = \bar{w}_i + \max_{m_j \in succ(m_i)} \left(c_{ij} + \text{rank}_u(m_j)\right) \quad \text{--------} \quad (6)$$

Where c_{ij} is the average communication cost of edge (i, j), $succ(m_i)$ is the set of immediate successors of task m_i and $\bar{w}_i$ is the average computation cost of module m_i. Modules are sorted in non decreasing order based on their rank_u value. Tie-breaking is done randomly. The non increasing order of rank_u values presents a topological order of modules which is a linear order that preserves precedence constraints.

3.5 QoS Parameters Addressed

In the proposed model a new algorithm called ***alloc_energy*** is designed for job scheduling in distributed environments such as cloud computing infrastructures.

3.5.1 Energy

The algorithm exploits the heterogeneity in of cloud resources in order to find a solution which reduces the computational energy consumption for a job. Furthermore, it suggests the two states working of processors as static and processing states where least amount of energy dissipates in static while processing state dissipates considerably higher amount of energy. The model utilizes the two states working of processors which results in saving of computational energy for the job.

3.5.2 Turnaround Time

Since the job under consideration are dependent set of modules, execution starts and end with two specific modules called as entry module and exit module respectively. The

algorithm evaluates the turnaround time of the submitted job which is the total time taken between the submission of the job and return of the completion output.

3.6 Energy Consumption Model

The energy consumed by processor while executing a job denoted by E_{total} is composed of static energy E_{static} that is energy dissipated while processor is sitting idle and processing energy $E_{processing}$ which is the amount of energy consumed when the processor is specifically processing the job modules. Proposed model assumes the static energy consumption is of least concern since the power dissipated while processing the job module is the most significant factor for the energy consumption [34]. A typical scenario of consumption of processing energy and static idle energy has been presented in Figure 3.2. The power dissipation while processing a job module $\rho_{processing}$ with a CMOS (Complementary Metal Oxide Semiconductor) based microprocessor is computed as

$$\rho_{processing(i)} = a_i * c_i * v_i^2 * f_i \quad \text{----------} \quad (7)$$

For a processor p_i which is executing the job the parameters a_i, c_i, v_i and f_i *are* the number of switches per clock cycle, the total capacitance load, the voltage supplied during execution and the operating frequency respectively. The parameters a_i and c_i are device related constants and depend upon device capacity. Thus, the energy consumed by a processor p_i while processing module m_k is computed as:

$$E_{processing} = \rho_{processing} * W_{ik} \quad \text{----------} \quad (8)$$

Here W_{ik} is the computation cost of module m_i on the processor p_k. In the job execution period, we assume that the processors operate at the highest level of supply voltage and at highest frequency. However they scale down their voltage and frequency to lowest supply voltage and lowest operational frequency during idle period. The power consumed while the processor is not executing any job module and is sitting idle is called the static power ρ_{static} which is fixed for a processor. Thus the total energy consumed in the idle period by a processor E_{static} can be computed as:

$$E_{static} = \sum_{i=1}^{p} \rho_{static}(i) * t_i \quad \text{----------} \quad (9)$$

Where t_i and $\rho_{static}(i)$ are the idle time and power dissipation respectively of processor p_i while sitting idle.

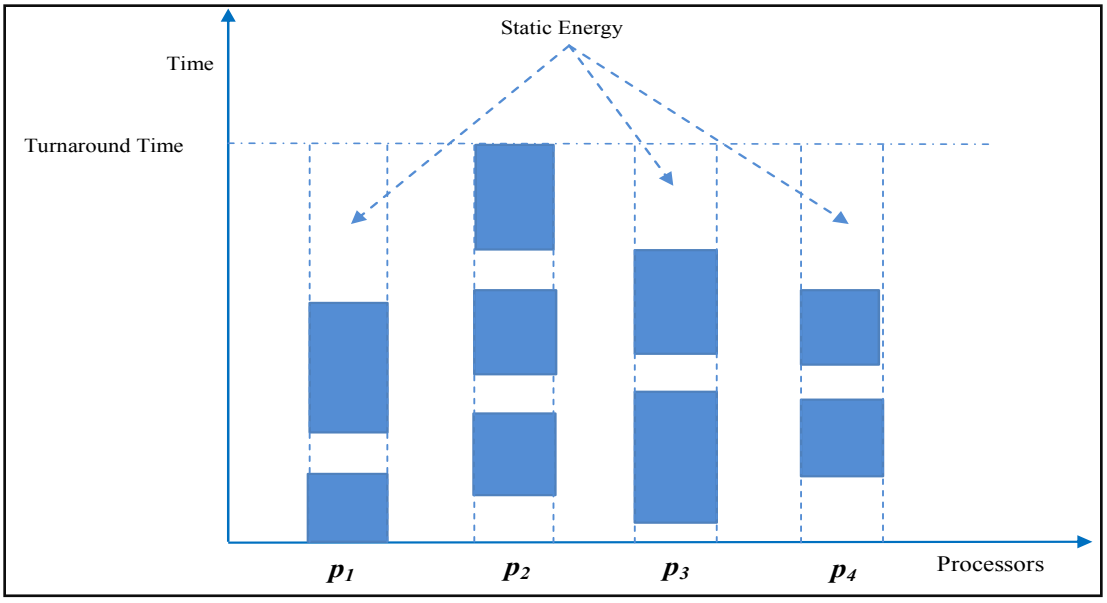

Figure 3.2 Processing and Idle Energy Dissipation

The total energy consumed during processing the job module is computed as

$$\mathbf{E_{processing\_total}} = \sum_{i=1}^{p}(\mathbf{RT_i} - \boldsymbol{t_i}) * \boldsymbol{\rho_{processing_i}} \quad \text{----------} \quad \mathbf{(10)}$$

Where (RT_i **-** t_i) is the total time during which processor p_i was executing the job modules with the utilization of $\rho_{processing_i}$ power of processor.

Therefore, the total of computation for the entire job is given as

$$E_{total} = E_{static} + E_{processing} \quad \text{----------} \quad \mathbf{(11)}$$

3.7 Proposed Algorithm

The proposed model aims to minimize the computational energy and evaluates turnaround time for the job bounded for the execution. To realize it the job is submitted in the desired format as explained in section 3.3. Thereafter, the scheduler works according to the following mechanism which is described below in the following steps

Pseudo code for the proposed algorithm

alloc_energy ( )
{
Evaluate the mean value of computation cost $\overline{w}_i$ of all the job modules
Construct a list of modules in reversed topological order i.e. RevTopoList
{
for each module m_i in RevTopoList do
if module is exit module
rank_u of exit module = $\overline{w}_i$
endif
max = 0
for each child m_y of module m_i do
if C_{iy} + rank_u (m_y) > *max* then
max = C_{iy} + rank_u (m_y)
endif
endfor
rank_u (m_i) = $\overline{w}_i$ + *max*
endfor
}
Sort the modules of scheduling list in non increasing order of their rank_u value
{
while there is unscheduled module in the list do
Select the first module m_i from the list for scheduling
for each processor p_k in the *processor- set* ($p_k \in p$) do
Calculate *expected computation energy matrix* ECE (m_i, p_k)
Assign module m_i to the processor p_k that has minimum value in the ECE matrix for module m_i
endfor
endwhile
}
Turnaround time of the job is calculated as per equation (3)
{
for each processor $p_x \in p$
Static energy is evaluated using equation (9)
Processing energy is evaluated using equation (10)
endfor
}
Total energy for the entire job is calculated as sum of static and processing energy using equation (11)
}

- In the first step modules of the entire job are leveled with average computation cost $\bar{w}_i$ which is calculated based on the equation (4) and list of modules in the reverse topological order are generated.
- In second step $rank_u$ value for all modules are evaluated by taking in account the mean computation cost of module and communication cost from each child of the module. The detailed description for the upward rank evaluation has been provided in section 3.4.
- In third step the modules are sorted with their rank value in non increasing order and a list is generated.
- Following step starts with first modules m_i in the list. The algorithm searches for an appropriate idle time slot for module m_i onto the processor p_j which results in offering minimum computational energy, at the time when all input data of m_i sent by m_i's immediate predecessors modules have arrived at processor p_j. The search continues until finding th e first idle time slot that is capable of holding the computational cost of module m_i. The module is removed from the list and the process is repeated until all the modules in the list get scheduled. The expected computational energy of modules for all processors is evaluated using equation (10) and is provided in ECE matrix.
- Later when all modules get allocated on to the processors, turnaround time is evaluated and presented as one of the outputs of the algorithm which is equal to total time taken between the submission of the job and earliest finish time of exit module.
- In the last step static and processing energy for all the processors are computed seperately by using equations (9) and (10). Total computational energy for entire job computed as summation of all static and processing energy dissipated using equation (11) and presented as output of the algorithm.

3.8 Illustrative Example

To better understand the model, an example is illustrated in this section so that the basic working of the model can be understood. The parameters have been scaled down for a

better illustration of the working of the model but the model is competent to work with any data values. The example considers a scenario where four heterogeneous processors are fully interconnected and are available for execution. The schedule for each module allocated on a processor can be represented as p_n [EST_i, m_i, EFT_i] using ***EST*** and ***EFT*** values of that module. Initially, all processors are considered to be idle. Since the job bounded for computation is represented in the form of DAG, let us assume the job presented in Figure 3.1 as a sample job for execution. The job comprises of nine modules m_1, m_2, m_3, … m_9 that are represented by a circle having an entry for module number. The edges connecting various modules represent the Communication Cost (C_{ij}) between modules. The expected computation cost matrix (ECC) for the job is given in the Table 3.1 of which each entry ECC(i,j) denotes expected computation cost of module m_i over processor p_j (in milliseconds).

Table 3.1

Expected Computation Cost Matrix

Modules	p_1	p_2	p_3	p_4
m_1	9	12	16	11
m_2	15	11	13	9
m_3	14	10	9	12
m_4	12	14	7	16
m_5	13	9	10	14
m_6	14	11	16	8
m_7	8	16	11	13
m_8	10	8	12	14
m_9	16	12	9	14

As per described in algorithm in section 3.4, we first evaluate the mean value of the expected computation cost $\overline{w}_i$ for each module by the help of equation (4). Modules with their $\overline{w}_i$ value are listed in the Table 3.2.

Table 3.2

Modules	$\bar{w}_i$
m_1	12.00
m_2	12.00
m_3	11.25
m_4	12.25
m_5	11.50
m_6	12.25
m_7	12.00
m_8	11.00
m_9	12.75

With the utilization of $\bar{w}_i$ and C_{ij} values in equation (5) and (6) the upward rank values for each module calculated and is presented as Table 3.3

Table 3.3

Upward Rank of the Job Modules

Modules	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9
$rank_u$	*109.25*	*82.25*	*80.50*	*85.25*	*59.25*	*60.00*	*38.75*	*36.75*	*12.75*

After the calculation of $rank_u$ values, priorities are assigned to each module. The highest priority is given to the module which has highest value for $rank_u$. The modules are ordered as per their $rank_u$ value in non increasing order. Hence, the final priority list of modules is presented as

m_1 m_4 m_2 m_3 m_6 m_5 m_7 m_8 m_9

Let us assume there are four heterogeneous hypothetical processors that are fully interconnected and all inter processor communications between them are assumed to be performed without contention. The proposed algorithms assume computation and

communications are overlapped with each other. Since processors are under consideration are available with two state of operation the power dissipated by them differs accordingly. Depends on their state of operations, typical parameters of four processors under consideration are described in Table 3.4

Table 3.4

Parameters of Heterogeneous Processors

Processors	ρ_{static}	c(×10^{-10} Farad)	v(Volts)	f(GHz)	$\rho_{processing}$(Watt)
p_1	65.2	254	1.44	2.58	135.88
p_2	37.2	278	1.30	2.28	107.11
p_3	50.0	266	1.52	2.64	162.24
p_4	77.0	292	1.36	2.32	125.30

The algorithm considers the power dissipated by processors while sitting idle is ρ_{static} that is fixed and the corresponding value for each processor is given in the Table 3.4. The power dissipated by the processors during the execution of job modules is termed as $\rho_{processing}$ which is calculated by the use of equation (7).

Table 3.5

Expected Computational Energy Matrix

Modules	p_1	p_2	p_3	p_4
m_1	1.2229	1.2853	2.5958	1.3783
m_2	2.0382	1.1782	2.1091	1.1277
m_3	1.9023	1.0711	1.4601	1.5036
m_4	1.6305	1.4995	1.1356	2.0048
m_5	1.7664	0.9639	1.6224	1.7542
m_6	1.9023	1.1782	2.5958	1.0024
m_7	1.0870	1.7137	1.7846	1.6289

m_8	1.3588	0.8568	1.9468	1.7542
m_9	2.1740	1.2853	1.4601	1.7542

The Expected Computational Energy Matrix **ECE**[n][p] has been presented in Table 3.4. Entries **ECE(**i, j**)** of matrix represents the energy consumed by processors while executing the module the value of which are provided by calculation based on equation (8). The final schedule for modules is generated according to the entries of Table 3.5. The search for an appropriate idle time slot of a module m_i on a processor p_j starts at time equal to EST_i of m_i on p_j. That means the time when all input data of m_i sent by m_i's immediate predecessor modules have arrived on processor p_j. The search continues until finding the first idle time slot that is capable of holding the computation cost of module m_i. The final schedule for the job considered in our example is given in Figure 3.3

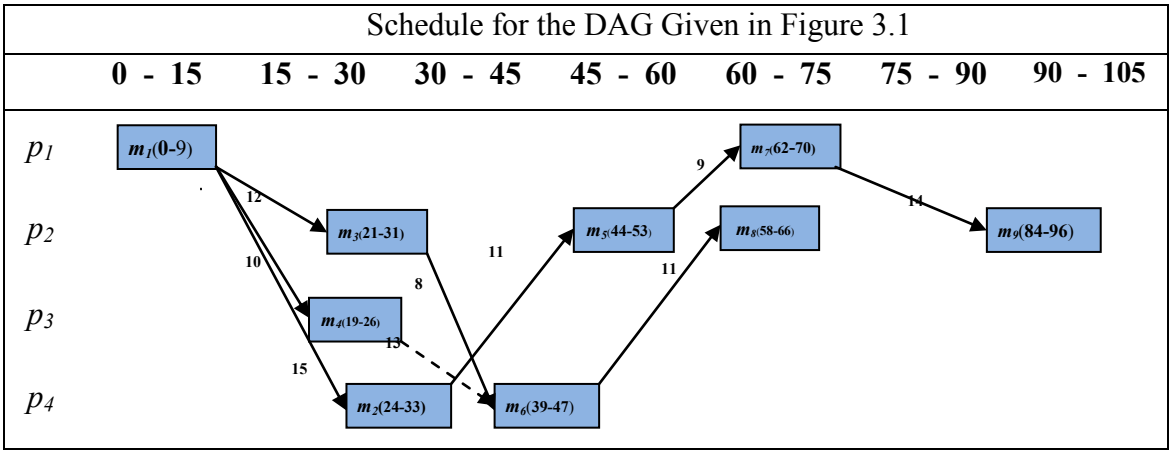

Figure 3.3 Schedule fort The Job Considered In Illustrative Example

After all the modules of the job is scheduled the turnaround time of the job is calculated using the equation (3). In this algorithm the turnaround time of job is simply *EFT* (Earliest Finish Time) of the exit module m_{exit}. Therefore the turnaround time for the entire job is given as

$$\mathbf{TAT_j} = \mathbf{96.00} \text{ milliseconds.}$$

The idle time and processing time for each processor during the job execution are summarized as

Processors	p_1	p_2	p_3	p_4
Idle Time (t_i)	79	57	89	79
Processing Time (TAT-t_i)	17	39	7	17

The total energy consumption $\boldsymbol{E_{total}}$ accounts for both the energy required to execute the assigned modules, and the energy that each processor consumes while sitting idle which is calculated by th help of equations (9), (10) and(11). The split of static and processing energy by each processor are shown in tabular form given below as

Processors	p_1	p_2	p_3	p_4
Static Energy (Joule)	5.1508	2.1204	4.4500	6.0830
Processing Energy (Joule)	2.3099	4.1771	1.1356	2.1301

The summation of static energy dissipated by all the processors while executing the job is found to be

$$\boldsymbol{E_{statiic}} = \mathbf{17.8042} \text{ Joule.}$$

Similarly, the energy consumed by all the processors while processong the job is calculated which is equal to

$$\boldsymbol{E_{processing}} = \mathbf{09.7527} \text{ Joule.}$$

The total energy dissipated while execution of the entire job is summation of static and processing energy consumed by the processors which is equal to

$$\boldsymbol{E_{total}} = \boldsymbol{E_{static}} + \boldsymbol{E_{processing}} = \mathbf{27.5569} \text{ Joule.}$$

3.9 Simulation Results

To evaluate the performance of the proposed model in terms of energy consumption and schedule length rigorous simulation runs were conducted. Simulation runs were carried out on Intel Core i7-3770, 3.4 GHz, 6.0 GB RAM machine running under Windows 7 Professional 64-bit Operating System. The programming language of choice was MATLAB R2013a (Version 8.1). The experiments use large number of random DAG with characteristics described in section 3.3. The study started with analyzing the model's performance by randomly varying the job attributes like the number of modules in the job

and their computation and communication requirements. While observing total computational energy (E_{total}) and turnaround time for the job (TAT_j) the model was simulated with different consideration taking into account for example running a job with different number of modules over fixed number of processors and running a single job over different number of processors. The same job is then simulated using the scheduling strategy of round-robin and the E_{total} observed is then compared with the E_{total} obtained with the proposed model. The results are summarized as Figure 3.4. It can be observed that the scheduling strategy based on proposed algorithm always results in less E_{total}.

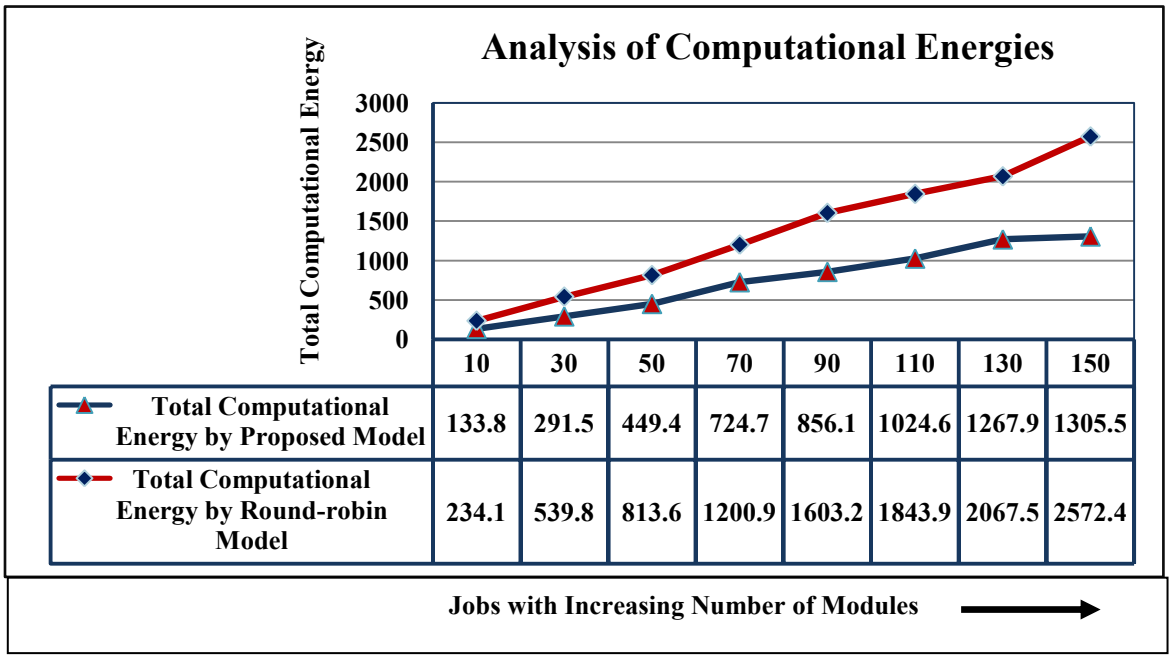

Figure 3.4 Analysis of Computational Energy Consumptions

For the above experiments, the TAT_j generated for each job has also been determined and plotted against increase in the job modules as shown in Figure 3.5

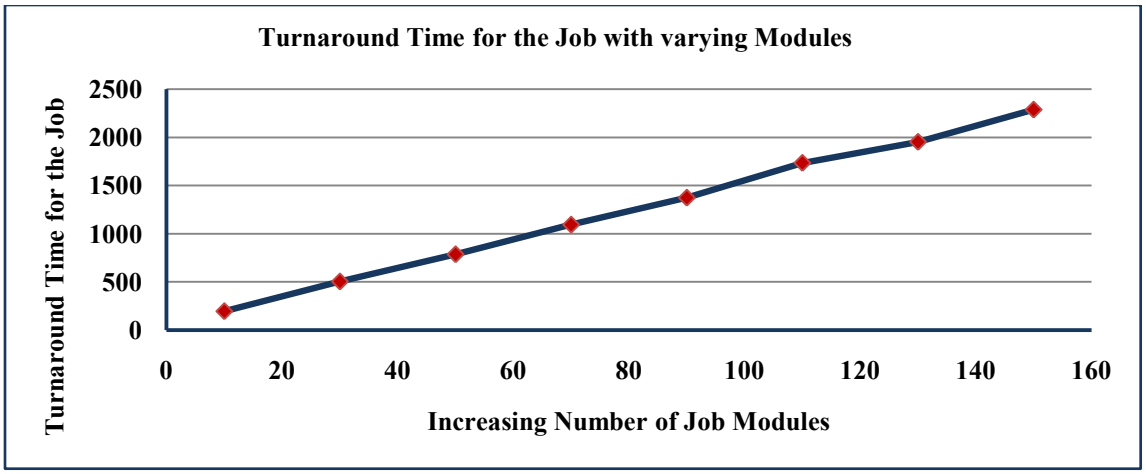

Figure 3.5 Turnaround Time of Job Modules with Increase in Numbers of Modules in Job

The next set of experiments intended to observe the effect of the change in E_{total} for the same job with varying number of processors. In this case, keeping the number of modules and their computation cost same. Since the increase in number of processors creates more idle time slots which leads to more static energy consumption in addition with computational energy thus larger value of E_{total}. The results obtained have been presented in Figure 3.6

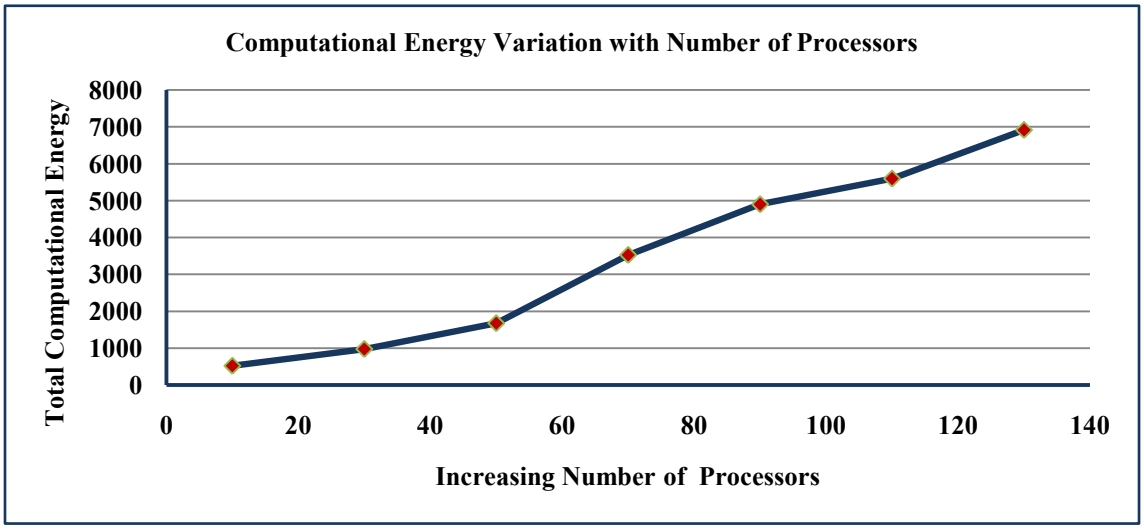

Figure 3.6 Variation of Computational Energy with Increase in Number of Processors

Observations

- The allocation of modules gets affected by the two factors considered for allocation viz. computational energy of module on to a processor, time to finish execution of the previous workload on to a processor.
- The proposed model allocates modules on to the processor which consume minimum computational energy among all while taking care the dependency of module.
- For a fixed number of processors, total computational energy increases with increase in modules of the job as processors have to execute large number of modules.
- Processors may have two working states i.e. idle and processing states. The energy consumption rate is different under different modes.
- The total computational energy increases with the increase in number of processors for the same job becouse idle time of processors increases hence static energy.
- Execution of a job on the distributed systems incurs a substantial communication cost which results in increase in turnaround time with increase in number of modules in the job.

Many experiments were conducted to study the behavior of the model but the results obtained were observed to be along the same line as reported in this work.

Chapter 4

Conclusion and Future Research Direction

Scheduling mechanism is an important issue in case of computing and is very much necessary to improve the server and resource utilization also to increases the performance of the computer. In this work, issue of allocating dependent job modules onto heterogeneous processors is addressed with an objective of minimization of computational energy. Here the allocation scheme considers both energy consumption and communication overhead between the processors. To facilitate the presentation of the algorithm appropriate mathematical modeling was presented to describe a computational system framework, parallel job modules with precedence constraints and energy consumption. The simulation experiments have been conducted to study the effectiveness of the model in terms of energy consumption and turnaround time using a large set of randomly generated DAGs. The experiments results show that the model based on proposed algorithm always results in saving significant computational energy.

Efficient scheduling of job modules represented by DAG is vital to maximize the benefits of execution. The DAG structure frequently occurs in many regular and irregular applications. However finding an optimal schedule without violating precedence constraints among job modules is known to be an NP-Complete problem. Consequently, various category of heuristic algorithm such as list-scheduling, clustering and task duplication-based algorithms have been developed in literature. The list-scheduling algorithm provides better scheduling with minimum overhead while clustering and task duplication-based algorithms reduce the communication cost thereby minimizing the schedule length. Further genetic approach is also applied to job scheduling in order to generate better schedule than heuristic algorithm.

In this work, the model is list-based heuristic algorithm which has been developed for completely connected heterogeneous processors. Priority of the job modules is assigned based on a parameter called upper rank which is evaluated by taking into account the size of data received from predecessor modules and average computation cost of modules.

The algorithm schedule job modules based on their priority while adopting insertion based policy. Modules are scheduled on suitable processor which dissipates minimum computational energy thus results in reducing the total computational energy for the job.

The algorithm developed in this work can be further extended in many aspects as follows:

- A comparative study of all proposed model with established peers.
- The developed algorithm assumes much idealized model of the target system where processors are fully connected and all communications are performed concurrently without contention. Hence the developed algorithm can be extended to arbitrarily connected processors by considering the communication while scheduling the task.
- The algorithm can be extended to include energy consumption of components such as communication channels.
- The algorithm developed in this work can be extended to dynamic scheduling environment so that limitations of static scheduling can be overcome.
- As the number of processors increases, the likelihood of processors and link failure also increases. Hence scheduling with fault tolerance becomes an important issue. The developed algorithm may extend to include fault tolerance feature.

References

1 Buyya, Broberg, Goscinski “Cloud Computing Principles and paradigms”, John Wiley & Sons, 2011.

2 Hwang, Kai, Fox, Geoffary C. and Dongarra, Jack J, “ Distributed and Cloud Computing: From Parallel Processing to the Internet of Things”, Morgan Kauffman, 2013.

3 Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I (2009) “Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the fifth utility.“ Futur Gener Comput Syst 25(6):599-616.

4 Buyya, R. (1999), editor. “High Performance Cluster Computing: Architectures and Systems.” Prentice Hall - PTR, NJ, USA, 1999.

5 Parashar, M. and Lee, C. (2005). “Grid Computing - Introduction and Overview.” Proceedings of the IEEE, Special Issue on Grid Computing. IEEE Press, 93 (3) March.

6 Mell P, Grance T: “The NIST definition of Cloud Computing.” Gaithersburg, MD: NIST, Special Publication 800–145; 2011.

7 *http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/1_Introduction.html*

8 Mark Wallis, Frans Henskens , Michael Hannaford, “Expanding the Cloud: A Component-Based Architecture to Application Deployment on the Internet,” Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, p.569-570, May 17-20, 2010.

9 *https://toyinogunmefun.wordpress.com/2011/06/16/effective-data-protection-for-cloud-computing-and-its-relevance-in-the-nigeria-economy.*

10 A. J. Younge, R. Henschel, J. T. Brown, G. von Laszewski, J. Qiu, and G. C. Fox, "Analysis of virtualization technologies for high performance computing environments," International Conference on Cloud Computing, 2011.

11 Marinescu, Dan C., “Cloud Computing: Theory and Practice”, Morgan Kauffman, 2013.

12 Banino C, Beaumont O, Carter L, Ferrante J, Legrand A, Robert Y. “Scheduling

strategies for master–lave tasking on heterogeneous processor platforms." *IEEE Transactions on Parallel and Distributed Systems* 2004; **15**(4):319–330.

13 M. Shahid, Z Raza, M Sajid. "Level based Batch scheduling strategy with idle slot reduction under DAG constraints for computational grid," Journal of Systems and Software, 2015

14 S. K. Panda and P. K. Jana, "An Efficient Task Scheduling Algorithm for Heterogeneous Multi-cloud Environment", 3rd IEEE International Conference on Advances in Computing, Communications and Informatics, pp. 1204-1209, 2014.

15 M. Shahid and Z. Raza, "Level based batch Scheduling Strategies for Computational Grid," Int. J. Grid and Utility Computing, Inderscience publisher,Vol. 5, No. 2, pp 135-148, 2014.

16 Ullman JD (1975) "NP-complete scheduling problems." J Comput Syst Sci 10(3):384–393.

17 T. Cormen, C. Leiserson, R. Rivest, and C. Stein, "Introduction to Algorithms," 2nd edition, McGraw Hill, New York, 2001.

18 J.K. Lenstra, A.H.G. Rinnooy Kan, P. Brucker. "Complexity of machine scheduling problems," Ann Discrete Math, 1 (1977), pp. 343–362.

19 *https://en.wikipedia.org/wiki/NP-complete#/media/File:P_np_np-complete _np-hard.svg*

20 D. Applegate, R. Bixby, V. Chvatal, W. "Cook On the solution of traveling salesman problems," Documenta Math. ICM (1998), pp. 645–656.

21 Sanjeev Arora, "Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems," Journal of the ACM (JACM), v.45 n.5, p.753-782, Sept. 1998.

22 Lance Fortnow, "The status of the P versus NP problem," Communications of the ACM, v.52 n.9, September 2009.

23 *https://www.vmware.com/products/vsphere/features/drs-dpm*

24 Yu-Kwong Kwok , Ishfaq Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," ACM Computing Surveys (CSUR), v.31 n.4, p.406-471, Dec. 1999.

25 J. Bruno, E.G. Coffman, and R. Sethi, “Scheduling Independent Tasks to Reduce Mean Finishing Time,” Communications of the ACM, vol. 17, no. 7, July 1974, pp. 382-387.

26 D.K. Friesen, “Tighter Bounds for LPT Scheduling on Uniform Processors,” SIAM Journal on Computing, vol. 16, no. 3, June 1987, pp. 554-560.

27 R.L. Graham, “Bounds for Certain Multiprocessing Anomalies,” Bell System Technical Journal, 45, 1966, pp. 1563-1581.

28 H. Gabow, “An Almost Linear Algorithm for Two-Processor Scheduling,” Journal of the ACM, 29, 3, 1982, pp. 766-780.

29 T.C. Hu, “Parallel Sequencing and Assembly Line Problems,” Operations Research, vol. 19, no. 6, Nov. 1961, pp. 841-848.

30 W.H. Kohler, “A Preliminary Evaluation of the Critical Path Method for Scheduling Tasks on Multiprocessor Systems,” IEEE Transactions on Computers, Dec. 1975, pp. 1235- 1238.

31 D.S. Johnson, C.H. Papadimitriou and M. Yannakakis, “How Easy Is Local Search?,” Journal of Computer and System Sciences, vol. 37, no. 1, Aug. 1988, pp. 79-100.

32 V. Sarkar, “Partitioning and Scheduling Parallel Programs for Multiprocessors,” MIT Press, Cambridge, MA, 1989.

33 Topcuoglu H, Hariri S, Wu M-Y. “Performance-effective and low-complexity task scheduling for heterogeneous computing.” *IEEE Transactions on Parallel and Distributed Systems* 2002; **13**(3):260–274.

34 A. Chandrakasan et al, "Low power CMOS digital design," IEEE Journal of Solid State Circuits, pp. 473-484, April 1992.