# Energy-Efficient, Green Job scheduling for Geographically Distributed Data Centers


By
**Farrukh Mahmood**
**2011-NUST-MS-CS-016**

Supervisor
**Dr.Zahid Anwar**
**Department of Computing**

A thesis submitted in partial fulfillment of the requirements for the degree of MSCS

In
School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(August 2015)

# Approval

It is certified that the contents and form of the thesis entitled "**Energy-Efficient, Green Job scheduling for Geographically Distributed Data Centers**" submitted by **Farrukh Mahmood** have been found satisfactory for the requirement of the degree.

Advisor: **Dr.Zahid Anwar**
Signature: _____

Date: _____

Committee Member 1: **Dr. Peter Charles Bloodsworth**

Signature: _____
Date: _____

Committee Member 2: **Dr. Hamid Mukhtar**

Signature: _____
Date: _____

Committee Member 3: **Dr. Asad Waqar Malik**

Signature: _____
Date: _____

# Abstract

Geographically distributed data centers are used as backbone infrastructure by big IT companies to meet rapidly increasing IT services demands and as the use of internet and information technology is increasing across the globe. High energy consumption,increased operational expenditures and high carbon footprint are becoming points of great concern related to data centers. In this work we present green aware, network aware, energy efficient job scheduling mechanism for geographically distributed data centers.It considers availability of green energy at each data center to maximize utilization of green energy, it considers the amount of under utilized computation resource at individual data centers while assigning job to a data center, which helps to achieve better server consolidation that results in better energy efficiency and also considers network load at each data center which helps avoiding hotspots in data center networks. This approach to data center selection mechanism helps to create better balance between server consolidation and network utilization at individual data centers. In this work we also present a framework for simulating infrastructure of geographically distributed data centers based on OMNET++, an open source object-oriented modular discrete event network simulation framework. It also contains power consumption model for calculation of energy consumption and a hierarchical statistics collection model for performance evaluation. We have implemented and evaluated the performance of our job scheduling mechanism on this framework to prove efficiency of our job scheduling mechanism.

# Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **Farrukh Mahmood**
Signature: _____

# Acknowledgment

Thanks to Almighty Allah, by whom all faculties are blessed to me.
I offer my sincere gratitude to my supervisor Dr. Zahid Anwar, who has supported and guided me throughout my thesis work with his patience and knowledge. This thesis would not have been completed without his guidelines and encouragement.
I wish to take this opportunity to express my heartiest thanks and deep- est sense of gratitude to Dr. Peter Charles Bloodsworth, Dr. Hamid Mukhtar and Dr. Dr. Asad Waqar Malik who gave me opportunity to conduct this study.
I would also like to thank all my friends and family members for their continuous encouragement and moral support.

*I dedicate this thesis to my deceased mother whose death was the greatest and irreparable loss that I met during this endeavor.*

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Introduction

In recent years, cloud computing has evolved to provide cost effective IT services. Fast penetration of IT services in human society across all continents has made service providers to build large data centers in geographically distributed locations to cater these scaled up and distributed service demands. This distribution of infrastructure has also enabled them to achieve reliability, reduction in operational cost and better utilization of green energy sources. Today, large companies like Google, Yahoo and Microsoft have adopted this model to serve millions of users worldwide.

Cloud computing relies on large data centers at the back end to provide services. Data centers consume lot of energy not only in account of large amount of IT equipment but also for supporting cooling and power distribution infrastructure. Energy consumption of data centers is expected to double every five years [1].Increasing energy consumption in data centers in not only becoming critical issue for service providers as it is becoming important factor in operational expenditures, also it is catching attention of environment watchdogs[2][3]. Energy cost is projected to rise to half of operational expenditures in few years from current 10% value [4]. Out of total energy consumed by a data center, a handsome share of 40% is consumed by IT equipment, which includes both computing servers and network hardware while cooling system and power distribution system use 45% and 15% respectively [5]. Two thirds of this 40% share is utilized by computing servers while remaining one third is utilized by network hardware [6]. Figure 1.1 depicts the operational expenditure distribution of a typical data center.

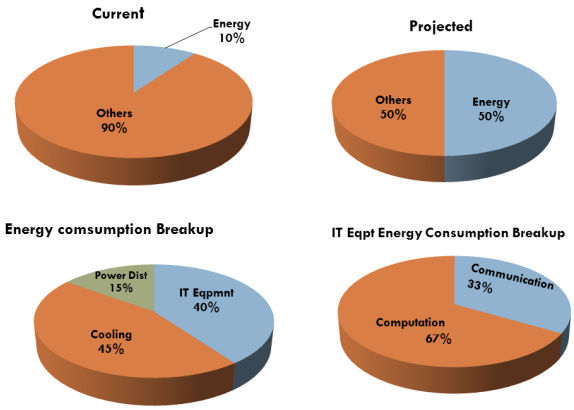


Figure 1.1: Data Center Operational Expenditure Breakup

Rapid increase in energy footprint of data centers has resulted in bigger carbon foot print which has led to serious environmental concerns. At one level efforts have been made to reduce the consumption of traditional brown energy to reduce environmental impact. Techniques like DVFS (Dynamic Voltage and Frequency Scaling) and DPM(Dynamic Power Management) were developed but this approach has very limited capacity to solve the problem. As result of this,cloud service providers have started focusing on techniques which incorporate partial utilization of green energy sources in data centers [7].Consequently; interest of research community has increased in developing techniques for integrating green energy into data centers ([8],[9], [10], [11], [12]).

This wish to integrate green energy as power source along with conventional brown energy adds complexity to the job scheduling problem in data centers. This complexity arises from the mismatch between inherent intermittence of green energy sources and fluctuating workloads of data centers.Green energy generation variates between its minimum (approaching zero) and maximum capacity over time. Work load also has peak hours and down hours during different times of the day.If job scheduling is not done smartly, this mismatch increases the chances for increased utilization of brown energy, hence undermining the objective of green energy integration.

In existing works, efforts have been made to address this challenge of mismatch between green energy availability and data center workloads but these have not effectively optimized the utilization of green or brown energy in ge-

ographically distributed data centers. Moreover, none of the work has considered network load at individual data centers while assigning jobs to data centers, which is critical factor for the jobs which involve data transfers.

In this work we present green aware, network aware, energy efficient job scheduling mechanism for geographically distributed data centers named DDENS(Distributed Data center Energy-Efficient,Network-Aware scheduling). This work is inspired by [13], which propose scheduling mechanism for single data center. It tries to achieve balance between conflicting requirements of server consolidation and network load balancing. It considers workload profile of servers to achieve server consolidation for maximizing energy conservation and network loads profile to avoid hotspots in the network. Our work extends this approach to job scheduling mechanism of geographically distributed data centers. Along with availability of green energy, it considers work load profile in form of amount of under utilized computation resource at individual data centers while assigning job to a data center, which helps to achieve better server consolidation that results in better energy efficiency.

It also considers network load profile at each data center which helps to achieve the objective of avoiding hotspots in data center networks. In short, extension of this approach to the data center selection mechanism helps achieving objective of balanced server consolidation and network performance at individual data centers.

One other big challenge faced by researchers in this domain is lack access to infrastructure. Researchers can not afford to own such big infrastructure and companies owning it do not allow access to researchers. Solution to this problem is simulation. Simulation frameworks have been developed to enable researchers to conduct research in this domain.

In this work, we have also developed a simulation framework for geographically distributed data centers named GreenFrame. It contains power consumption model for different active components of the system which lets to calculate energy consumption of components as well as whole system. It also contains hierarchical statistics collection mechanism, which allows collecting different statistics about components, subsystem and system level. These statistics can be used in job scheduling mechanism as well as performance evaluation. Due to its modular structure, different global and local scheduling mechanisms can be easily implemented. This framework is based on OMNET++ [14], an open source object-oriented modular discrete event network simulation framework and associated INET [15] framework, which is an

open source model library for developing models of communication networks

## 1.2 Thesis Aims and Objectives

This research aims to;

**I.** Development Energy Efficient, Network Aware, Green Job Scheduling Algorithm for Geographically Distributed Data Centers.

**II.** Develop simulation Framework for Geographically Distributed Centers based on OMNET++ Platform with Power consumption feature.

## 1.3 Thesis Organization

Rest of the thesis is organized as follow:

Chapter 2 at first discusses works already done related to both single data centers and geographically distributed data centers and provide critical analysis of these efforts. Then ,it describes existing simulation platforms and their comparison critical analysis.At the end, it draws conclusion from these analysis.

Chapter 3 describes system architecture and working of proposed system.It depicts and elaborates working of the data center architecture followed in this work, hierarchical scheduling mechanism and hierarchical statistics collection mechanism.At the end,it models the network traffic simulation mechanism with in data center and describe it working.

Chapter 4 talks about implementation details. It describes structure of OMNET++, elaborates class diagram, and explains implementation of simple and compound modules.

Chapter 5 describes evaluation of the system. It presents comparison study with comparison baselines on basis of energy efficiency, green energy utilization and network performance.I also concludes thesis work and highlights future research directions.

# Chapter 2

# Related Research

This chapter at first describes and discusses efforts already made related to achieving energy efficiency and green energy integration in single data centers and provides critical analysis of existing solutions. Afterward, it discusses previously done efforts related to achieving same objective in geographically distributed data centers and provide critical analysis of these efforts. Then ,it describes three well known existing simulation platforms, analyzes their features and provides comparison and critical analysis.At the end, it draws conclusion from these analysis and establishes a case for our work.

## 2.1 Studies Related Single Data Center

Numbers of efforts have been made to use mixture of green energy with traditional energy sources in order to reduce the carbon footprint of data centers.

Work in [13], proposes a distinctive approach of job scheduling in single data center by taking into account the requirement of network congestion avoidance. This requirements in conflicting with one important requirement of energy efficiency which is achieved through server consolidation. This work achieves balance between these conflicting requirements by making scheduling decision based on a metric which considers work load of servers, racks and modules along with communication potential of access switches and aggregate switches.This is very novel and elegant approach for job scheduling but it does not take green energy integration into account.

Work in [16], uses a predictive green energy aware scheduling mechanism for both service and batch jobs which utilizes short term prediction of solar and wind energy production. It uses both solar and wind energy as green

energy source. Green energy prediction is made in 30 minutes intervals. At the beginning of every batch job allocation interval, predictor estimates the average green energy available in next interval and provides this data to scheduler. Scheduler then computes number of green job slots available in next interval and based on this data it schedules the jobs.

Work in[9], proposes GreenSlot, a solar and grid energy powered parallel batch job scheduler for a data center.It uses green energy prediction, brown-energy-cost-awareness, and least slack time first (LSTF) job ordering. It tries to improve consumption of green energy, by predicting the amount of green energy available in near future.It also ensures that job deadlines are met. GreenSlot predicts the amount of solar energy that will be available in the near future, and schedules the workload to improve the green energy consumption while meeting the jobs deadlines. If green energy falls short,it utilizes brown energy to meet deadlines.

In[10], proposes that for work loads with slack time, if we adjust the execution of work loads by playing with available slack times,we can converge the difference between green energy availability and work loads, hence utilization of green energy can be maximized.Scheduling algorithm uses follow the energy approach,In this approach,if sufficient green energy supply is not currently available,either due to low production or high consumption, jobs with slack time are deferred to the time when either green energy production increases or consumption decreases.This work considers scientific batch jobs,where sufficient slack time is available. It considers wind as green energy source.

One drawback in these approaches is that they only work for single data center and are inherently unable to exploit the advantage of complimenting green energy supplies in different regions. Secondly, besides [13], none of them while scheduling jobs, consider possibility of development of network hotspots, which may hinder the timely execution of communication intensive jobs.Thirdly, they strive to improve share of green energy in over energy consumption of data center,but ignore factor of efficiency of green energy utilization itself.

## 2.2 Studies Related Multiple Data Center

Work in [17], proposes a frame work to intelligently exploit the difference of brown energy costs during different day times in same region and across

different regions to reduce cost of energy consumption. It also exploits availability of green energy to minimize environmental impact.Framework applies optimization based request distribution and enable service providers to manage energy consumption and cost by taking advantage of previously mentioned factors using one of two optimization policies.After every hour,service provider front ends compute fractions of user requests using optimization based policy and then distribute incoming requests during next hour according to this computation.Second policy uses heuristics to first exploit data center with best energy efficiency and then exploits cheap energy.

Work in [12], besides exploitation of different brown energy costs, investigates the possibility for exploitation of geographical distribution of internet scale systems for environmental gains. This research shows that remarkable reduction in consumption of green energy can be achieved by adopting "Follow the renewable" approach in such distributed systems. It contributes i three dimensions. First, a distributed algorithms for geographical load balancing wile assuring optimality. Second, algorithms to investigate impacts of using geographical load balancing on demand response.Third,investigation of the scenario in which data center is almost wholly powered by green energy.

Work in [18], provides a holistic job scheduling mechanism for geographically distributed data centers. It promises to decrease the consumption of brown energy by replacing it with green energy. It caters the work loads with sufficient slack. It considers amount of green energy available, outside temperature,energy consumption of cooling systems related to data centers. Factors related to workload like fluctuation,deadline and job structure are also considered.

None of these studies consider network profile of a data center while scheduling jobs to individual data centers. As a consequence its highly likely that situation may regularly arise in data centers when jobs executing face severe competition for network resources.This may result in network hot spots within data centers. Moreover,they ignore the factor of efficiency of green energy utilization itself.

## 2.3 Simulation Frameworks

In this section, we have reviewed three of well known simulation frameworks in this domain.Cloudsim [19] is extensible simulation framework for modeling, simulation and experimentation of cloud infrastructures.Greencloud [20] is packet-level simulator for energy-aware cloud computing data centers. It provides facility for modeling of data center components energy consumption.iCancloud[21] is simulation framework which provides ability to predict trade-off between cost and performance in cloud infrastructure. Table 2.1 provides summary comparison between he three.

| | **Feature** | **Cloudsim** | **GreenCloud** | **iCanCloud** |
|---|---|---|---|---|
| 1 | Platform | Any | NS2 | OMNET++ |
| 2 | Programming Language | Java | C++/OTcl | C++ |
| 3 | Availability | OpenSource | OpenSource | OpenSource |
| 4 | Power Consumption | Yes | Yes | No |

Table 2.1: Comparison of Simulation Frameworks

Simulation frameworks exhibit some specific feature based on the platform they are built on.Cloudsim is based on Java platform,so it can run on any machine which is installed with JVM. Greencloud is a packet level simulator, which is developed as an extension to NS2 simulation platform. iCancloud is built on OMNET++ platform.

Every platform has associated language,in which further systems can be developed using that platform.Cloudsim is associated with Java. Greencloud supports C++ and OTcl. OTcl is comparatively difficult to work with. iCanCloud supports C++ as is the case with OMNET++. All three are available as open source.

Energy efficiency is one of important objective of data center job scheduling algorithms. Any simulation platform in this domain must provide feature to compute energy consumption of data center components to evaluate scheduling algorithm's energy efficiency. Power consumption model enables a platform to provide this feature. Cloudsim and Greencloud both possess power consumption model and provide energy consumption calculation feature, but iCancloud does not contain power consumption model.

OMNET++ is a discrete event simulation infrastructure, which is gaining huge popularity in research community due to its areas of application and flexible architecture.It has generalized structure and allows to build discrete event simulation platforms, which monitor and account the interaction of objects,as a result are time efficient and scalable.One of its strength is its modular architecture which provides ability to develop reusable components called modules.As shown in Table:2.1, iCanCloud,an existing well known simulator based on OMNET++ does not support power consumption model in order to evaluate energy consumption.Our proposed framework provides this feature which allows to compute energy consumed by the data center components. It includes energy model for both servers and network hardware.

## 2.4 Conclusion

In our literature study, we have noted that existing works in this domain, try to take advantage of complimenting distributed green energy resources to decrease the utilization of green energy, try to achieve maximum possible energy efficiency by server consolidation and also try to exploit other environmental and data center features. These work do not consider and balance the conflicting requirement of network congestion avoidance at data centers, which involves distribution of network traffic. Our work does take this conflicting factor into account and creates a balance between energy efficiency by server consolidation, Network Load and individual job performance.In purview of absence of cloud simulation framework based on OMNET++ with power consumption model,Our work also presents simulation framework for cloud which contains power consumption model for servers and network hardware.

# Chapter 3

# Design and Methodology

This chapter describes system architecture and working of proposed system's major components.At first, it elaborates the data center architecture followed in this work.Then it depicts major components of hierarchical scheduling mechanism proposed in this work, models interaction among these components and describe their working.Afterward, it elaborates major components of hierarchical statistics collection mechanism,their internal coordination and interaction with scheduling mechanism.At the end,it models the network traffic simulation mechanism with in data center and describe it working.

## 3.1 Architecture

### 3.1.1 Data Center Architecture

Although because of modular design of our simulation framework,any data center topology can be designed, in this work we have considered well known and widely used three tier architecture for individual data center. Successors of three tier architecture have been developed but it is still most widely used architecture.It consists of three tiers.

**Core Tier** Provides the high-speed packet switching back plane for all flows going in and out of the data center. The core layer provides connectivity to multiple aggregation modules.

**Aggregation Tier** Provide important functions, such as service module integration, Layer 2 domain definitions, spanning tree processing, and default gateway redundancy.

**Access Tier** This is where servers actually reside. Servers are arranged into racks and connected to Top of Rack (ToR) switch. ToR's connect to aggregation tier switches. Four aggregation switches along with connected racks

form modules.
we have assumed that we have homogenous computation and network hardware in our data centers.

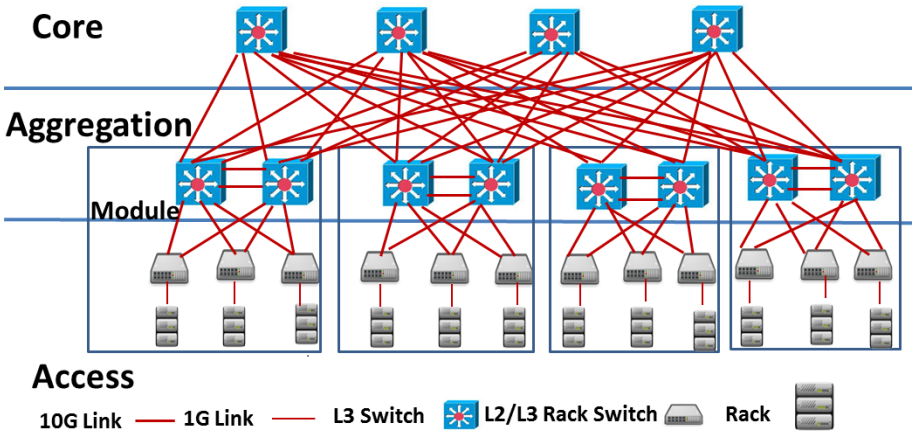


Figure 3.1: Three Tier Architecture of Data center

### 3.1.2 Scheduling Mechanism Architecture

In this work we have designed a hierarchical scheduling mechanism as shown in Figure:3.2.It consists of a main scheduler which receives jobs from users and after selecting appropriate data canter,forwards jobs to local scheduler of that data center. Local data center using local scheduling mechanism selects appropriate server and sends job to that server, where it is finally executed.

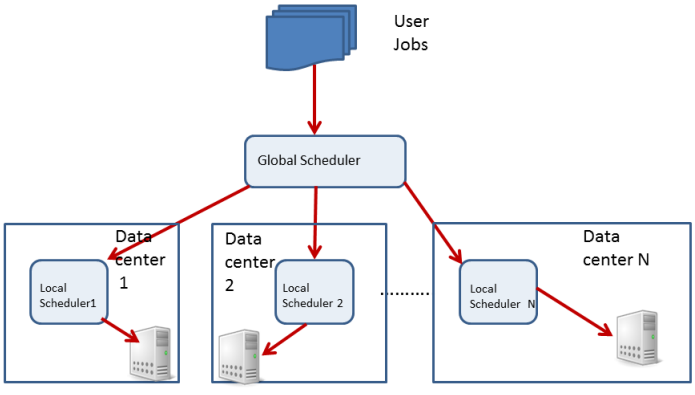


Figure 3.2: Hierarchical Scheduling Mechanism

### 3.1.3 Statistics Collection Mechanism Architecture

Every scheduling mechanism needs statistics to make scheduling decision. In this work we have designed a hierarchical statistics collection mechanism as shown in Fig: 3.3.

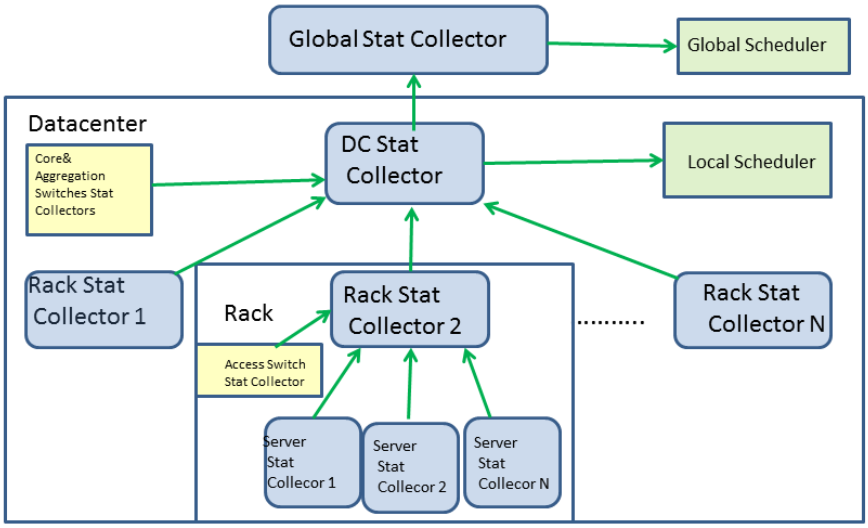


Figure 3.3: Hierarchical Scheduling Mechanism

Every server has a statistics collector module.It collects information about different parameters of the server. Statistical data from all servers in a rack are accessed by statistics collector of that rack. Rack statistics collector also collects statistics from ToR switch of that rack. All this information goes to data center statistics collector, which also collects statistics from statistics collectors of all switches from core-tier and aggregation tier. These statistics collected by data center collector are used by local scheduler to make scheduling decision for assigning incoming jobs to servers. Data center computation load profile and network load profile is also sent by data center statistics collector to Global statistics collector. Global statistics collector provides statistics to main scheduler, which makes data center selection decision based on these statistics.

### 3.1.4 Traffic Simulation Mechanism Architecture

In our work network traffic profile is important scheduling factor and it's work load model consists of workloads having balanced processing and communication needs. Network traffic simulation mechanism is shown in Fig: 3.4.

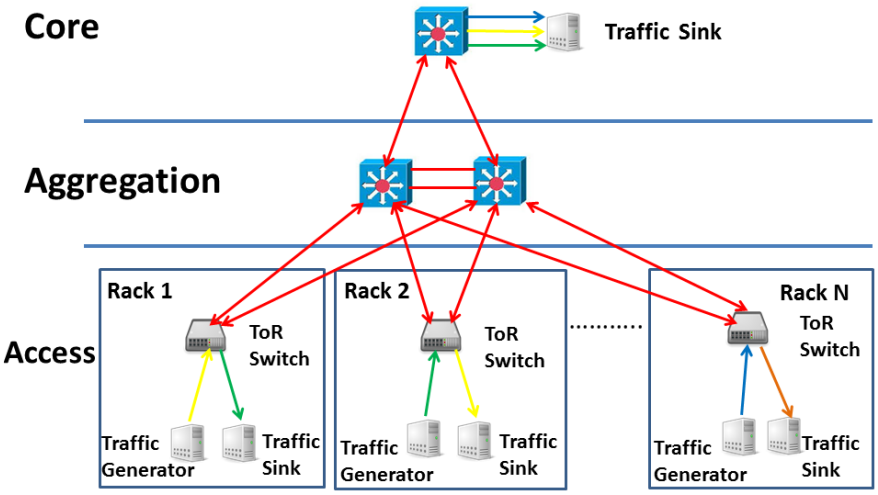


Figure 3.4: Traffic Simulation Mechanism

At access tier every rack consists of a traffic generator node and a traffic sink node. At core tier a traffic sink node is connected to core network. To simulate intra-data center traffic, traffic generator nodes at each rack generates 5KB*N data in form of UDP packets every second, where N is number of jobs running in rack. This traffic is destined to traffic sink nodes of randomly selected racks within data center. To simulate data center outbound traffic, each traffic generator in racks generate 200Kbps*N bit stream destined to sink node connected to core routers in core tier.

## 3.2 Energy Consumption Model

Most part of energy in server is utilized by processor, disk storage and network. Energy consumption equation of a server is given as under[22].

$$P_{CPU} = 14.5 + 0.2U_{CPU} + (4.5e^{-8})U_{mem} + 0.003U_{disk} + (3.1e^{-8})U_{net} \quad (1)$$

where
$U_{CPU}$=Utilization of CPU
$U_{mem}$= Utilization of Memory.
$U_{disk}$= Utilization of Hard Disk
$U_{net}$= Utilization of network interface.
Energy consumption model for network switch as under[23].

$$P_{switch} = P_{chassis} + n_{linecards}\ P_{linecard} + \sum_1^n n_{ports}\ P_r \quad (2)$$

where
$P_{chassis}$ = Power consumed by the switch hardware
$P_{linecard}$= Power consumed by any active network line card.
$P_r$= Power consumed by a port running at the rate r.

An idle server utilizes 2/3 of peak energy and 1/3 energy is variably used by processor with changing load.Two well known methods for energy saving are DVFS(Dynamic Voltage Frequency Scaling) and DPM (Dynamic Power Management). DVFS dynamically scales processor frequency up and down with changing load. It's impact is limited to 33% of energy utilized by processor. DPM shuts down the idle hardware.So, provides much high energy efficiency when properly applied. Due to high potential for power saving we have used DPM as power saving method.

## 3.3 Workload Model

Cloud work loads are represented by jobs,which are defined by parameters like, computation requirement, deadline,communication requirement and slack time. Cloud jobs are classified into three categories.
**Computationally Intensive:** These are the jobs like scientific computation,which have high computational requirements but very low communication requirements. These jobs don't create problem of network congestion. Good strategy is to schedule these jobs on minimum numbers of servers,hence putting maximum number of servers and Network components to shutdown/sleep.
**Communication Intensive:** These are the jobs like video streaming, which have low computational resources demand but very high network resources requirements.These jobs have high potential of creating network hotspots in data centers. Its good policy to distribute these jobs such a way that network load is distributed.It causes more energy consumption by network

components.
**Balanced:** These type of jobs require both computation and network resources proportionally. One needs to consider both server load and network load while scheduling these kind of jobs.

In this work we have considered balanced workloads with reasonable slack.As our work involves scheduling decision making based workload and network load. Balanced work loads compliment our methodology. Reason able slack times provide better opportunity to exploit differing availability of green energy sources in different regions.

# Chapter 4

# Implementation

This chapter explains implementation of the proposed solution. It starts with comprehensive intro of Tools and Libraries used in which it explains architecture and working of OMNET++ and features provided by INET framework. It is followed by NED implementation of compound modules and then C++ implementation of simple modules.

## 4.1 Tools and Libraries Used

In this work we have used OMNET++, a discrete event simulation development environment along with its INET model library which provides model for wired and wireless networks. This section provides overview of OMNET++ environment and utility of INET library.

### 4.1.1 OMNET++

OMNET++ is not a simulator itself,but provides an environment to develop simulation frameworks. It has component based architecture and simulation models can be developed from highly reusable components called modules.

#### 4.1.1.1 Types of Modules

There are two types of modules in OMNET++. Simple Modules,which are active modules and implement some kind of functionality in C++. Simple modules group together in unlimited hierarchy to make Compound Modules as shown in Figure: 4.1. In OMNET++ models are referred to as Networks which depict system module. Model structures are defined in OMNET++ NED language.

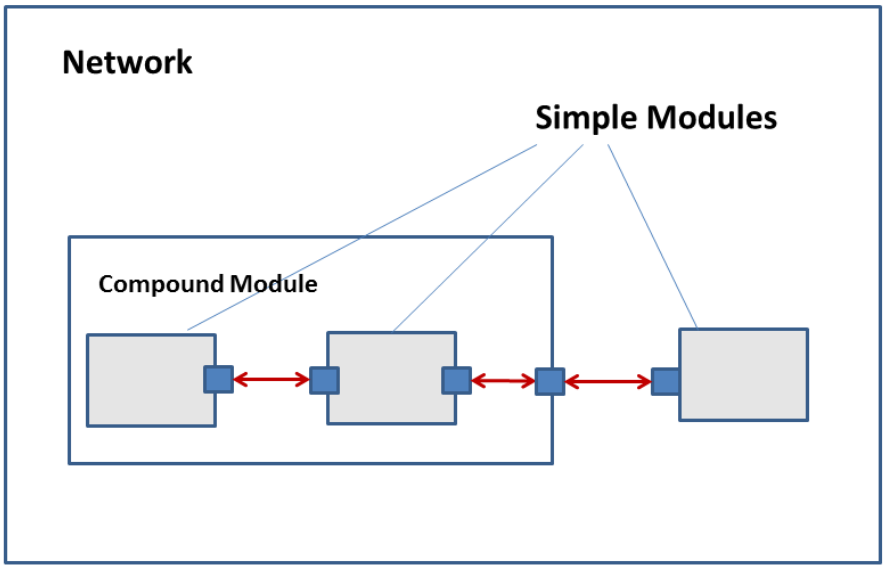


Figure 4.1: Modules Hierarchy

#### 4.1.1.2 Module Parameters

Parameters are used to define simple module's behavior or model's topology. Parameters can be defined as numeric, string, boolean and XML data trees. Parameters are assigned either in NED file or in omnetpp.ini file.

#### 4.1.1.3 Messages,Gates,Links

Modules communicate by exchanging messages. A message can depict a packet in data network, a job or any other mobile entity. Messages are received and sent by modules through in gates and out gates via links which are labeled as connections.

#### 4.1.1.4 Module Definition

A simple module is defined in NED file as shown in Listing.4.1.

Listing 4.1: Simple Module NED Defination

```
Simple xmodule
  {
       parameters:
       int capacity;
      gates:
      input in;
      output out;

       }
```

Every simple module implements functionality using C++ in .cpp file as shown in Listing 4.2.

Listing 4.2: Simple Module .cpp Implementation

```
class xmodule : public cSimpleModule
{
  protected:
    // The following redefined virtual function holds the algorithm.
    virtual void initialize ();
    virtual void handleMessage(cMessage *msg);
};

void xmodule:: initialize ()
{
    // Initialize is called at the beginning of the simulation.

}

void xmodule::handleMessage(cMessage *msg)
{
    // The handleMessage() method is called whenever a message
        arrives
}
```

Method initialize() is called at start of simulation and sets any initialization parameters. Method handlemessage() actually implements the algorithm,it executes whenever it receives message from other module or internally generated (to implement timers).

A compound module is defined in NED file as shown in Listing 4.3.

Listing 4.3: Compound Module NED Defination

```
1  module Host
2      {
3       parameters:
4              ......
5       gates:
6              .....
7       submodules:
8              .....
9       connections:
10             .....
11
12           }
```

A compound module may have simple modules or compound modules as sub modules.

### 4.1.2 INET Framework

It is an open-source model library for the OMNeT++ simulation environment which provides facility to develop simulations of wired and wireless networks. INET provides models for TCP/IP stack, Ethernet,802.11, Point to Point protocol,server application models and many other components for simulation of data networks.
In our work we have reused components provided by INET to avoid reinventing the wheel and have focused our attention to build the framework to support simulation of job scheduling mechanism, implementation of power model and integration of green energy.

## 4.2 Implementation of Proposed Framework

### 4.2.1 Class Diagram

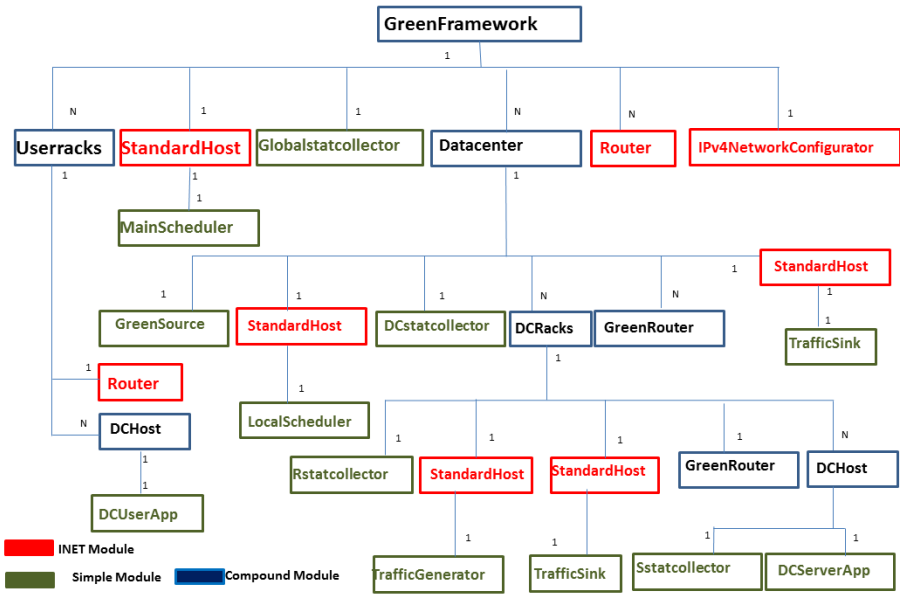


Figure 4.2: Class Diagram

### 4.2.2 Implementation of Compound Modules

In this section we describe the NED implementation of compound modules in this framework.

#### 4.2.2.1 GreenFramework Network Module

This is system Module, highest in hierarchy and denoted as Network as per syntax of OMNET++. Its NED Implementation is shown in Listing 4.4.

Listing 4.4: GreenFramework

```
network GreenFramework
{
    submodules:
     // DC1,DC2,  depictdata centers.
          DC1: Datacenter {};
          DC2: Datacenter {};

    //   R1,R2,   .depict Public Internet Routers.
          R1: Router {};
          R2: Router {};
                    .
          GlobalScheduler: StandardHost {};
          Gstatcollector: Globalstatcollector {};

  // user racks to generate jobs.
        RedUsers: UserRacks {};
        WhiteUsers: UserRacks {};

 //   configurator to configure IPV4 configurations.
       configurator: IPv4NetworkConfigurator {};

         Connections:

}// end of Greenframe
```

It is composed of constituent data centers, Public Internet Routers,StandardHost module named GlobalScheduler on which MainScheduler simple module is running that implements scheduling algorithm for data center selection,a global statistics collector module to compute system level statistics and provide to global scheduler for decision making.

#### 4.2.2.2 Datacenter Module

This is compound Module that implements single data center following three tier architecture as shown in Figure 3.1.Its NED Implementation is shown in Listing 4.5.

Listing 4.5: DataCenter Module

```
1  module Datacenter
2   {
3    parameters:
4
5
6    gates:
7                    .
8    submodules:
9          //Aggregation Tier switches
10         AS1[aggregateswitch]: GreenRouter {};
11        //Core Tier switches
12         CS1[corerouter]: GreenRouter {};
13       //Server racks
14         Rks[accessswitch]: DCRacks {};
15     //Local Scheduler
16         LocalScheduler: StandardHost {};
17     //datacenter statistics collector
18         DCcollector: Dcstatcollector{};
19     //Green energy Source
20         Gsource:GreenSource{};
21
22      connections :
23                       .
24
25 }// end of Datacenter
```

It is composed of core tier switches,aggregation tier switches, Racks of servers,data center statistics collector which collects data center level statistics ,green energy source and StandardHost module named LocalScheduler on which LocalScheduler simple module is running that implements scheduling algorithm for server selection.

#### 4.2.2.3 UserRack Module

This is compound Module that implements the racks of user machines, which are generating work load.Its NED Implementation is shown in listing 4.6.

Listing 4.6: User Rack Module

```
1  module UserRack
2   {
3       parameters:
4                       ..
5       gates:
6
7       submodules:
8
9           computingServer[N]: DCHost {};
10          accessRouter: Router {};
11
12      connections:
13                          .
14  }
```

It is composed of an top of rack switch, number of DCHost machines,each of them running a instance of simple module DCUserApp, which implements work load generation mechanism.

#### 4.2.2.4 DCRack Module

This is compound Module that implements the racks of data center servers, which execute user jobs.Its NED Implementation is shown in Listing 4.7.

Listing 4.7: Datacenter Rack Module

```
module DCRacks
{
    parameters:

    gates:

    submodules:
        //commputing servers
         computingServer[N]: DCHost {};
        //ToR Switches
         accessswitch: GreenRouter { };
         // Rack statistics collector
          statcollector : Rstatcollector{};
        // Network traffic generator
          trafficgen : StandardHost{};
        //Network traffic sink
          trafficsink : StandardHost{};
    connections:
                        ..
}
```

It is composed of an top of rack switch,number of DCHost machines,each of them running a instance of simple module DCServerApp which implements job execution mechanism,rack statistics collector, traffic generation and traffic sink modules.

#### 4.2.2.5 GreenRouter and DCHost

GreenRouter and DCHost extend NodebaseRouter and NodeBase classes of INET framework and inherit most of functionality from there.

### 4.2.3 Implementation of Simple Modules

In this section we describe C++ implementation of simple modules in this framework.

**Algorithm 4.1** Main Scheduler Algorithm

```
1: Timer ← 0
2: epoch ← tminutes
3: for job J from Job Queue Q with shortest slack do
4:     t ← Deadline
5:     P_r ← ComputationRequirement
6:     R_c ← CommunicationRequirement
7: end for
8: DC=SelectDC()
9: if DC ≠ NULL then
10:    Send(J, DC) //send Job J to Datacenter DC
11: else
12:    Add(J, Q) //add Job J to Job queue Q
13: end if
14: function SelectDC
15:    g ← NULL
16:    D' is empty Set
17:    for all dϵD do //D is set of all Data Centers
18:        if U_pc > P_r and U_nc > R_b then
19:            D'+ {d}
20:        end if
21:    end for
22:    for all d'ϵD' do
23:        if G_c of d' > g then //Gc is green energy capacity
24:            g ← d'
25:        end if
26:    end for
27:    return g
28: end function
29: if Timer == epoch then
30:    ComputeProfile()
31: end if
32: procedure ComputeProfile
33:    for all dϵD do //D is set of all Data Centers
34:        G_c ← duration(g) − current_time //duration(g) gives duration of
    green energy availability
35:        U_pc ← (2000 ∗ n1 + 4000 ∗ n2 + 6000 ∗ n3 + 8000 ∗ n4 + 10000 ∗
    n5)/(n1 + n2 + n3 + n4 + n5)
36:        U_nc ← (b1 + b2 + .... + bn)/n
37:        Timer ← 0
38:    end for
39: end procedure
```

#### 4.2.3.1 Main Scheduler

This simple module implements mechanism for selection of data center for scheduling incoming jobs.Its working is described in Algorithm 4.1.In lines 1 2,it sets timer to zero and value of epoch(time duration) after which main scheduler computes profiles of data centers.In lines 3-7 it assesses the requirements of user job like deadline, processing requirement and communication requirement. In line-8 calls procedure SelectDC, which is defined from line 14 to line 28. It first selects set of data centers whose work load profile $U_{pc}$ and network load profile $U_{nc}$ satisfy the computation and communication requirements of the Job.Out of this set of data centers it selects and returns the data center with highest value of green energy available.

If SelectDC does return a Data Center job is sent to it otherwise job is sent to job queue.After expiry of epoch,scheduler reassesses the profile of data centers by calling procedure ComputeProfile.For each data center ComputeProfile does following; (1) In line 34, computes green energy availability duration.(2) In lines 35 computes work load profile by calculating waited average of under utilized computational resources. It multiplies weight of each of five load slabs computed in GlobalStatcollector with number of machines in that slab and divide by sum of number of machines in all slabs. (3) in line 36,it computes network load profile by dividing sum of free bandwidth in all active switches in data center by total number of active switches.

#### 4.2.3.2 Local Scheduler

This simple module implements mechanism for selection of server for scheduling incoming jobs.Its working is described in algorithm 4.2. From line 1 to line 4,this algorithm initializes the variables like weighted coefficients $\alpha$,$\beta$,$\gamma$, proportional coefficient $\phi$ and bandwidth over provision factor $\delta_r$. Then it calls function ServerSelect() which is described in lines 7-19.Job is scheduled to server returned by this function. ServerSelect() selects server on basis of metric M described below

$$M = \alpha.\ fs + \beta.\ fr + \gamma.\ fm \quad (3)$$

where,

$\alpha$= weighted co-efficient for servers, high value favours overloaded servers in underutilized racks.

$\beta$= weighted coefficient for racks, high value favour computationally loaded racks with low network activity.

$\gamma$= weighted coefficient for modules, higher value favors loaded modules.

**Algorithm 4.2** Local Scheduler Algorithm

```
1: α ← 0.7
2: β ← 0.2
3: α ← 0.1
4: Setφandδ_r
5:  S=SelectSever()
6: Schedule(J,S) //Schedule Job J to Server S
7: function SELECTDC
8:     machine ← 0
9:     for all sϵS do //S is set of all Servers
10:        fs ← Ls ∗ Qr(q)
11:        fr ← 1/n ∗ Σi = 1..n(Ls).Qm(q)
12:        fm ← 1/k ∗ Σj = 1..k(Lr(l))
13:        M ← α.fs + β.fr + γ.fm
14:        if M > machine then
15:            machine ← s
16:        end if
17:    end for
18:    return machine
19: end function
```

$$\alpha + \beta + \gamma = 1 \tag{4}$$

where,

$$\text{fs} = \text{Ls Qr (q)}^{\phi} / \delta_r \tag{5}$$

where,

Ls is Load factor of Server,$\delta_r$ is bandwidth over provisioning factor at rack switch,Qr(q)= Communication potential at access Switch Queue.$\phi$ is a coefficient defining the proportion between Ls and Qr (q) in the metric. Given that both Ls and Qr (q) must be within the range [0, 1] higher $\phi$ values will decrease the importance of the traffic-related component Qr (q).

$$\text{fr} = 1/\text{n} * \Sigma\text{i}=1..\text{n (Ls).Q}_m\text{(q)}^{\phi} / \delta_r \tag{6}$$

where, Qm (q) =Communication potential of module aggregation Switches Queue.

$$\text{fm} = 1/\text{k} * \Sigma\text{j}=1..\text{k(Lr)} \tag{7}$$

where

k = number of racks in module.

Lr = Load factor of rack=1/n * Σi=1..n (Ls).

#### 4.2.3.3 Globalstatcollector

This simple module implements mechanism for collection of statistics from data center statistics collectors which are used by main scheduler for decision making. Class Globalstatcollector is described in Listing 4.8.

Listing 4.8: Global Statistics Collector

```
1  class globalstatcollector : public cSimpleModule
2      {
3        private:
4       // datastructures to hold collected statistics
5       protected:
6       // to initialize datastructures and implement timer.
7           virtual void initialize ();
8      //at expiry of each timer, collects statistics from DC statistics
            collectors .
9           virtual void handleMessage(cMessage *msg);
10      public
11     // Methods used by main scheduler to get statistics
12          virtual double GetPloadmatrix(int);
13          virtual double GetNloadmatrix(int);
14 };
```

Initialize() method sets initial values of attributes and also implements timer. handleMessage() method implements statistics collection logic and does so at each expiry of timer.GetPloadmatrix() and GetNloadmatrix() public methods are used by main scheduler to access computational and network profile data respectively.

#### 4.2.3.4 DCstatcollector

This simple module implements mechanism for collection of statistics from data center racks statistics collectors and statistics of network switches which are used by local scheduler for decision making. It also generates data center processing load and network load profile,which is passed on to Global Statistics Collector. Class DCstatcollector is given in Listing 4.9.

Listing 4.9: Datacenter Statistics Collector

```
1  class DCstatcollector : public cSimpleModule
2      {
3       private:
4       // datastructures to hold collected statistics
5       protected:
6       // to initialize datastructures and implement timer.
7          virtual void initialize ();
8       //at expiry of each timer, collects statistics from rack
            statistics collectors .
9          virtual void handleMessage(cMessage *msg);
10      public
11      // Getter,Setter Methods for DC energy,processing profile and
            network profile.
12         virtual void SetEnergy(double);
13         virtual double GetEnergy();
14         virtual void SetPload(double,int);
15         virtual double GetPload(int);
16         virtual  void SetNload(double,int);
17         virtual double GetNload(int);
18     };
```

Initialize() method sets initial values of attributes and also implements timer. handleMessage() method implements statistics collection logic and does so at each expiry of timer. SetEnergy() and GetEnergy() methods are used to access data center energy consumption data. SetPload() and GetPload() are used to access data center work load profile data. SetNload() and GetNload() are used to access data center network load profile data.

#### 4.2.3.5 Rstatcollector

This simple module implements mechanism for collection of statistics from statistics collectors of individual servers in a rack. These statistics are further passed on to data center Statistics Collector. Class Rstatcollector is given in Listing 4.10.

Listing 4.10: Rack Statistics Collector

```
1  class Rstatcollector : public cSimpleModule
2      {
3       private:
4       // datastructures to hold collected statistics
5       protected:
6       // to initialize datastructures and implement timer.
7          virtual void initialize ();
8       //at expiry of each timer, collects statistics from server
              statistics collectors .
9          virtual void handleMessage(cMessage *msg);
10      public
11      // Getter,Setter Methods for rack energy,server processing load,
             rack processing load profile and network profile.
12         virtual void SetEnergy(double);
13         virtual double GetEnergy();
14                              virtual void SetProcRate(double,int);
15         virtual double GetProcrate(int);
16         virtual void SetPload(double,int);
17         virtual double GetPload(int);
18         virtual  void SetNload(double,int);
19         virtual double GetNload(int);
20     };
```

Initialize() method sets initial values of attributes and also implements timer. handleMessage() method implements statistics collection logic and does so at each expiry of timer. SetEnergy() and GetEnergy() methods are to access rack's energy consumption data. SetProcRate() and GetProcRate() are used to access processing load of individual servers in the rack. SetPload() and GetPload() are used to access rack's work load profile. SetNload() and GetNload() are used to access rack's network load profile.

#### 4.2.3.6 Sstatcollector

This simple module implements lowest level of statistics collection mechanism. It records statistics of individual server machine in which it resides.These statistics are further passed on to rack Statistics Collector of the rack in which this server machine resides. Class Sstatcollector is given in Listing 4.11.

Listing 4.11: Server Statistics Collector

```
1  class Sstatcollector : public cSimpleModule
2      {
3       private:
4       // datastructures to hold collected statistics
5       protected:
6       // to initialize datastructures and implement timer.
7           virtual void initialize ();
8      //at expiry of each timer, collects statistics from DCServerApp.
9          virtual void handleMessage(cMessage *msg);
10     public
11    // Getter,Setter Methods for server energy, and server processing
          load.
12         virtual void SetEnergy(double);
13         virtual double GetEnergy();
14         virtual void SetProcRate(double);
15         virtual double GetProcRate(int);
16
17 };
```

Initialize() method sets initial values of attributes and also implements timer. handleMessage() method implements statistics collection logic and does so at each expiry of timer. SetEnergy() and GetEnergy() methods are used to access server's energy consumption. SetProcRate() and GetProcRate() are used to access processing load of the server.

#### 4.2.3.7 TrafficGenerator

This simple module implements traffic generation module that simulates network load in data center. It generates amount of traffic proportional to work load of rack in which it resides. Most of traffic is destined to randomly selected rack's traffic sink module to simulate intra data center network load but small part is destined to traffic sink module in core tier of data center to simulate data center outbound traffic. Class TrafficGenerator is given in Listing 4.12.

Listing 4.12: Traffic Generator

```
1  class TrafficGenerator : public cSimpleModule
2    {
3     private:
4     // variables
5     protected:
6     // to initialize variables and implement timer.
7         virtual void initialize ();
8    //at expiry of each timer, generates traffic .
9        virtual void handleMessage(cMessage *msg);
10       virtual IPvXAddress chooseDestAddr();
11       virtual void setSocketOptions();
12       virtual cPacket *createPacket();
13       virtual void sendPacket();
14
15 };
```

Initialize() method sets initial values of attributes and also implements timer. handleMessage() method implements logic to generate traffic. chooseDestAddr() method chooses destination address for generated traffic. setSocketOptions() sets socket options for out going packets. createPacket() creates traffic packets and sets payload proportional to rack's work load. sendPacket() method sends traffic packet created.

#### 4.2.3.8 TrafficSink

This simple module implements traffic disposing mechanism for traffic generated by Traffic Generator module to simulates network load in data center. Class TrafficSink is given in Listing 4.13.

Listing 4.13: Traffic Sink

```
1  class TrafficSink : public cSimpleModule
2      {
3       private:
4       // variables
5       protected:
6       // to initialize variables.
7       virtual void initialize();
8       //at arrival of traffic, disposes it off .
9       virtual void handleMessage(cMessage *msg);
10      virtual void processPacket(cPacket *msg);
11 };
```

Initialize() method sets initial values of variables. handleMessage() method implements logic to dispose every incoming traffic packet using method processPacket().

#### 4.2.3.9 DCServerApp

This simple module implements functionality of data center server.It executes incoming jobs and compute energy consumed. Class DCServerApp is given in Listing 4.14.

Listing 4.14: Data Center Server Application

```
1  class DCServerApp: public cSimpleModule
2      {
3      private:
4     // variables
5      protected:
6          virtual void  initialize (int stage);
7          virtual void handleMessage(cMessage *msg);
8          virtual void updateTskListQ;
9          void updateTskComputingRatesQ;
10         void setComputingRate();
11         void eUpdateQ;
12         void setCurrentConsumption();
13         double getMostUrgentTaskRate();
14         double getCurrentLoad();
15         void  sched(double delay, cMessage *msg);
16
17           };
```

Initialize() method sets initial values of variables. handleMessage() method implements logic to execute Incoming jobs and energy computation using following methods. updateTskList() updates task list to see which tasks are completed. updateTskComputingRates() updates computing rate of every task. setComputingRate() sets computing rate of server. eUpdate() updates the total energy consumed.setCurrentConsumption() sets current energy consumption rate of server. getMostUrgentTaskRate() Compute highest MIPS/deadline ratio in task list. getCurrentLoad() gives current load of server. sched() schedules next job.

#### 4.2.3.10 DCUserApp

This simple module implements functionality of data center server.It executes incoming jobs and compute energy consumed. Class DCServerApp is given in Listing 4.15.

Listing 4.15: Data Center User Application

```
1  class DCUserApp: public cSirnpleModule
2                   {
3  private:
4  // variables
5  protected:
6  virtual void  initialize (int stage);
7  virtual void handleMessage(cMessage *msg);
8  virtual I PvXAddresschooseDestAddr();
9  virtual CloudTask *createCloudTaskPacket();
10 virtual cPacket sattachTaskParameters(cPacket *pk);
11 virtual void sendPacket();
12 virtual void setSocketOptions();
13
14 };
```

Initialize() method sets initial values of attributes and also implements timer. handleMessage() method implements logic to generate jobs. chooseDestAddr() method chooses destination address which is address of Main Scheduler. createCloudTaskPacket() creates job structure packet. attachTaskParameters() attaches job parameter to created job structure packet. setSocketOptions() sets socket options. sendPacket() method sends job packet created.

#### 4.2.3.11 GreenSource

This simple module implements functionality of data center green energy source.It simulates intermittency and variability of green energy source over different regions. Class GreenSource is given in Listing 4.16.

Listing 4.16: Green Source

```
1  class GreenSource: public cSimpleModule
2      {
3        private:
4        // variables
5        protected:
6        virtual void  initialize (int stage);
7        virtual void handleMessage(cMessage *msg);
8        virtual double getEnergy()
9     };
```

Initialize() method sets initial values of attributes and also implements timer. handleMessage() method implements logic to simulates intermittency and variability of green energy source. getEnergy() method is used to access value of available green energy.

# Chapter 5

# Evaluation

This chapter details the evaluation of Distributed DENS(DDENS), proposed job scheduling algorithm for geographically distributed data centers. This evaluation is done using GreenFrame, framework proposed in our work. It provides features for evaluation of energy consumption by computation machines, network components, overall system, network performance and integration of green energy sources into data centers.Evaluation shows that DDENS not only provides very good energy efficiency,better green energy utilization but also provides much better network performance.

## 5.1 Comparison Baseline

We have used three algorithms as comparison baselines to evaluate our work.

**I.** Round Robin Algorithm(RR): In this case both global scheduler and all local schedulers are running RR algorithm. It distributes incoming workload equally without any parameter consideration.It is a good base case for evaluation of a scheduling algorithm.

**II.** MinBrown Algorithm(MB): tries to minimize brown energy usage in data centers.

**III.** RR+DENS: In this case we are running RR at global scheduler,which distributes jobs equally to data centers.At local schedulers,we are running DENS.

Factors considered in comparison are energy efficiency, utilization of green energy and network congestion avoidance.

## 5.2 Evaluation Scenario

In this work we have considered four data centers DC1, DC2, DC3 and DC4. Each located in different time zone for the purpose of simulating different green energy availability levels

### 5.2.1 Datacenter Setup

We have used three tier data center topology as shown in figure. 3.1. We have assumed that our data center consists of homogeneous computation and network hardware. Each server consists of single core of 10,000MIPS (Million Instructions per Second) computation power. Energy consumed by a server at peak load is taken 225 watts same as class 2006 volume server [24]. Data center topology information is given in table below.

| Sr. | Parameter | Value |
|---|---|---|
| 1 | Servers | 1536 |
| 2 | Racks | 32 |
| 3 | Servers per Rack | 48 |
| 4 | Access Switches | 32 |
| 4 | Aggregation Switches | 8 |
| 4 | Core Switches | 4 |
| 4 | Server-Access Switch link | 1 G |
| 4 | Fat Tree Network links | 10 G |

Table 5.1: Datacenter Parameters

Network Load Network traffic is simulated using mechanism shown in Figure.3.4. Traffic generator module generates 200kbps *N bit stream destined to core tier sink to simulate data center outbound traffic where, N is number of jobs running in that rack. It also generates constant UDP packets of size 5KB*N every second where N is number of jobs in the rack, which are destined to randomly selected racks traffic sink to simulate intra data center traffic.

### 5.2.2 Work Load

Dead lined Jobs are generated by DCUserApp module in user racks. Rate of job generation is defined by exponential function of time exponential(t).

These jobs come with balanced processing and communication demands with 15 % to 50% slack time.

### 5.2.3 Green Energy

Maximum green energy available is kept to 50% of maximum energy demand of data center. Green energy availability of a data center varies between zero and maximum. At any instance of time one data center is in 0% bracket of maximum green energy capacity, one is in 33% bracket, one is in 66% bracket and one is in 100%. It is simulated in a way that, over simulated period, each data center provides equal supply of green energy, so as a result, all data centers receive approximately same amount of work load, maximum load of each data center goes up to 50% of its capacity and average load of a data center remains approximately 30% of capacity.

## 5.3 Results

This section elaborates the results of comparison evaluation.It describes the results for overall energy efficiency, utilization of green energy and network performance in terms of congestion avoidance.

### 5.3.1 Energy Efficiency

Table 5.2 provides comparison of energy consumed by Servers, Network and whole data centers while using RR(Round Robin),MB(MinBrown),RR+DENS(Round Robin plus DENS) and DDENS(Distributed DENS).

### 5.3.2 Green Energy Utilization

Table 5.3 shows comparison of green energy utilization as %age of total energy consumption. It shows that MinBrown achieves 40% improvement over Round Robin in utilization of green energy and DDENS achieves 52% improvement. DDENS out performs both baselines on this important scale.

| Parameter | | Power Consumption kWh | | | |
|---|---|---|---|---|---|
| **Datacenter** | **Entity** | **RR** | **MB** | **RR+DENS** | **DDENS** |
| DC1 | Datacenter | 852.6k | 417.4k(49%) | 485.9(57%) | 443.9k(52%) |
| | Server | 686.9k | 309.5k(45%) | 364.1(53%) | 331.7k(48%) |
| | Network | 165.7k | 107.9k(65%) | 119.5 (72%) | 112.2k(67%) |
| DC2 | Datacenter | 839.1k | 401.8k(48%) | 478.2(57%) | 433.6k(52%) |
| | Server | 682.3k | 299.3k(44%) | 380.6(55%) | 327.4k(48%) |
| | Network | 156.8k | 102.5k(65%) | 114.6(73%) | 106.2k(68%) |
| DC3 | Datacenter | 857.9k | 410.0k(48%) | 489.0(57%) | 455.9k(53%) |
| | Server | 691.7k | 304.7k(44%) | 366.7(53%) | 340.1k(49%) |
| | Network | 166.2k | 105.3k(63%) | 117.8(71%) | 115.8k(70%) |
| DC4 | Datacenter | 845.3k | 416.4K(49%) | 481.6(57%) | 447.5k(53%) |
| | Server | 685.1k | 310.3k(45%) | 369.9(54%) | 332.9k(49%) |
| | Network | 160.5k | 106.1k(67%) | 116.8(73%) | 114.6k(71%) |

Table 5.2: Energy Efficiency Comparison

| **Algorithm** | **Type** | **DC1** | **DC2** | **DC3** | **DC4** |
|---|---|---|---|---|---|
| Round Robin | Green | 24.8 % | 22.3% | 26.9% | 23.1% |
| | Brown | 75.2% | 77.7% | 73.1% | 76.9% |
| MinBrown | Green | 34.7% | 30.2% | 37.6% | 34.5% |
| | Brown | 65.3% | 69.8% | 62.4% | 65.5% |
| RR+DENS | Green | 28.2% | 25.6% | 30.1% | 27.8% |
| | Brown | 71.8% | 74.4% | 69.9% | 72.2% |
| DDENS | Green | 42.7% | 38.9% | 45.9% | 39.1% |
| | Brown | 57.3% | 61.1% | 54.1% | 60.9% |

Table 5.3: Green Energy Consumption %age of Total Energy

Figure 5.1. shows comparison of normalized average value of consumption of green energy, brown energy and total energy at all four data centers against all four techniques.these values are normalized against average of total energy consumed by RR algorithm.

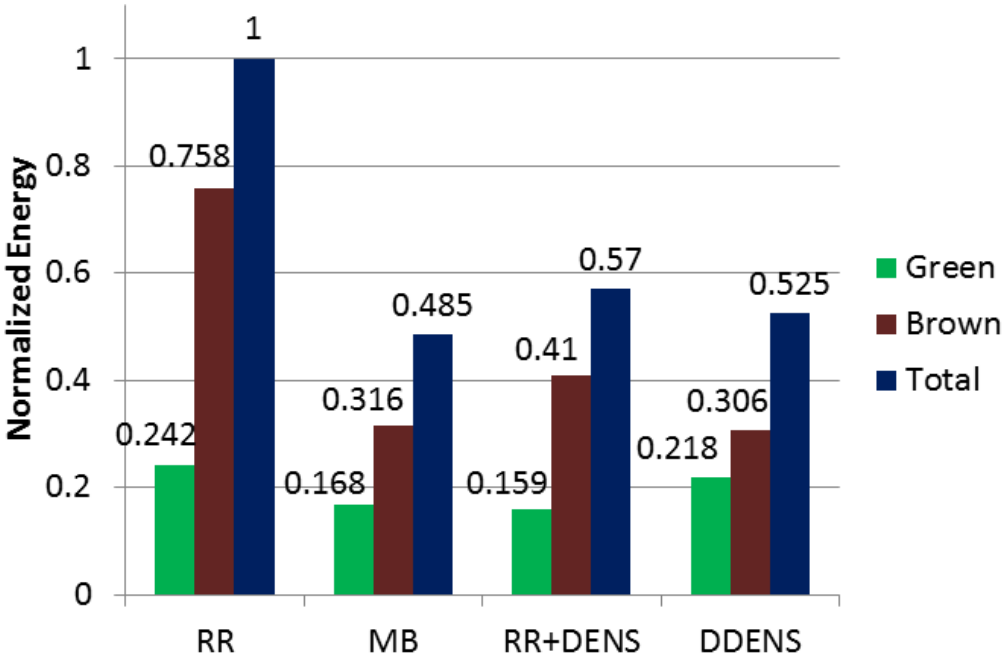


Figure 5.1: Normalized Green and Brown Energy Consumption

### 5.3.3 Network Performance

Figure 5.2 depicts the network performance of all network switches in all data centers in terms of avoidance of network congestion. It shows that in case of Round Robin all switches are minimally burdened , so there is no chance of congestion.It happens because Round Robin distributes jobs equally across the data center so network load is also equally distributed, but the cost is very poor energy efficiency and green energy utilization.On the other hand, MinBrown tries to schedule jobs on minimum portion of data center,as a result all network demand is put to small portion of network, which causes some switches to face congestion. DDENS also tries to schedule jobs on minimum number of servers,while avoiding network congestion.Small cost of this gain is that it uses few more network switches and servers as compared to MinBrown.RR+DENS ovoids network congestion as DENS is running at each data center's local scheduler,but it's performance is compromised against DDENS because it is running RR at global scheduler,which does not support local schedulers in achieving best balance of network load distribution and server consolidation

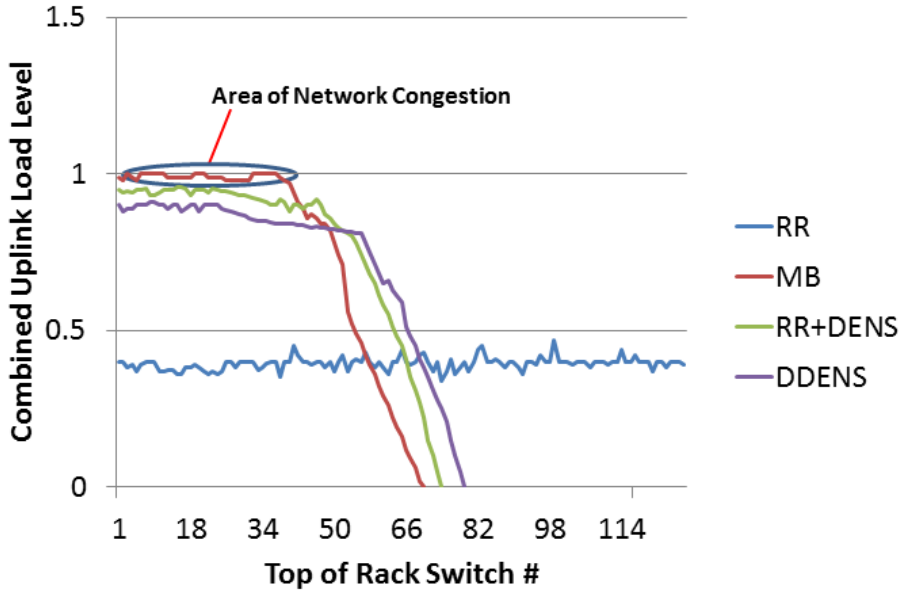


Figure 5.2: Network Performance Comparison

## 5.4 Discussion

Table. 5.2 shows that MinBrown on average utilizes 48.5% of energy consumed by base case of Round Robin algorithm. For RR+DENS and DDENS this value is 57% and 52% respectively.It shows that all three achieve very good energy efficiency, as compared to base case of RR.It is when we we compare them with each other, we get insight about their working and performance.

If we compare MB and DDENS, we see that DENS uses 3.5% more energy on average than MB. This fact along with,facts provided by network performance analysis in Figure.5.2 conform to very design rationale of DDENS and highlights its advantage over MB. Design rationale of DDENS focuses on achieving balance between network load distribution and energy consumption. MB does not take into account any effort to avoid network congestion and schedules jobs on minimum possible numbers of servers. As is result congestion occurs at some of switches as shown in network performance comparison.In effort to avoid network hotspots, DDENS utilizes small number of extra servers,which leads to this small fraction of extra energy consumption.Gain achieved against this small cost in very valuable as shown in Figure.5.2. that no network switch enters in state of congestion in this case.

If we compare RR+DENS with others, we see that it consumes 8.5% more energy as compared to MB and 4.5% more energy as compared with DDENS.Its over and above difference of energy consumption with DDENS proves and highlights the advantage and usability of our concept of extending DENS approach at distributed level. It verifies that global scheduler in our work compliments the objective of achieving energy efficiency at data centers locally.

When discussing utilization of green energy,Table.5.3 and Figure. 5.1 provides us with following insight. On average,DDENS better utilizes green energy by 7%, which proves worth of our work on this scale too. While comparing RR+DENS , we see that its green energy utilization is poor than MB and DDENS and more trending towards performance of RR. This is because like RR,it is also not making any effort to match the work load with green energy availability.

# Chapter 6

# Conclusion and Future Directions

In this chapter we present a conclusion on the basis of evaluation and comparison results depicted in chapter 5. We also state future directions for work, in future work section of this chapter.

## 6.1 Conclusion

In the light of discussion in previous chapter we reach the conclusion that,DDENS not only provides us very good energy efficiency and better green energy utilization,but also provides additional safeguard function against network congestion. It balances the tradeoff between energy efficiency through server consolidation and network requirement distribution. Hence it is very promising job scheduling mechanism for geographically distributed data centers with green energy integration, specially where jobs involve communication along with computation.

## 6.2 Future Work

Our future includes following.

**I.** Adding job migration facility, so that under process jobs with reasonable work remaining can be migrated to other data center, when green energy supply is downgraded at data center where it is currently running.

**II.** Adding sleep state functionality to Servers, so that instead of shutting down idle servers , they are put to sleep. It will help reducing hardware wear & tear caused by repeated shutdown and power up.

# Bibliography


1. J. Koomey, “Worldwide electricity used in data centers”, *Environmental Research Letters, pages 3:1–3:8, 2008.*
2. Fan, X., Weber, W.-D., Barroso, L.A., “Power provisioning for a warehouse-sized computer”, in *Proceedings of the ACM International Symposium on Computer Architecture, San Diego, CA, June 2007.*
3. Raghavendra, R., Ranganathan, P., Talwar, V., Wang, Z., Zhu, X.,”No “power” struggles: Coordinated multi-level power management for the data center”, in*: APLOS (2008).*
4. Gartner Group, 2011. [Online]. Available: *http://www.gartner.com/*
5. Brown, R.,”Report to congress on server and data center energy efficiency”, *public law 109-431, Lawrence Berkeley National Laboratory, Berkeley, 2008.*
6. Shang, L., Peh, L.-S., Jha, K.N.,” Dynamic voltage scaling with links for power optimization of interconnection networks”, in *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, (2003).
7. Data Center knowledge, “How googles data centers use renewable energy,” 2011. [Online]. Available*:http://www.datacenterknowledge.com/archives/2011/09/08/howgoogles-data-centers-use-renewable-energy/*
8. I. Goiri, et al. GreenHadoop,”Leveraging green energy in data-processing frameworks”, in *Proceedings of EuroSys’12, pages 57–70, 2012.*
9. I. Goiri, “GreenSlot Scheduling energy consumption in green datacenters”, in *Proc. of SC’11, pages 20:1–20:11,2011.*
10. A. Krioukov,” Integrating renewable energy using data analytic systems: Challenges and opportunities”, *IEEE Data Engineering Bulletin, pages 1–9, 2011.*
11. Y. Zhang,”GreenWare, Greening cloud-scale data centers to maximize the use of renewable energy”, *in Proceedings of Middleware’11, pages 143–164, 2011.*
12. Z. Liu, ”Greening geographical load balancing”, in *Proc. of SIGMETRICS’11, pages 233–244, 2011.*
13. D Kliazovich, P Bouvry, SU Khan,”DENS: data center energy-efficient, network-aware scheduling”, *Cluster computing 16 (1), 65-75.2013.*
14. OMNET++.[Online]. Available: *https://omnetpp.org/.*
15. INET.[Online]. Available: *https://inet.omnetpp.org/.*
16. B. Aksanli,”Utilizing green energy prediction to schedule mixed batch and service jobs in data centers”, *in Proceedings of HotPower’11, pages 5:1 5:5, 2011.*
17. K. Le,” Cost- and energy-aware load distribution across data centers”, in *Proceedings of HotPower’09, pages 1–5, 2009.*
18. R. Chen, X. Weng, B. He, M. Yang, B. Choi, and X. Li,” Improving large graph processing on partitioned graphs in the cloud”, *in Proceedings of ACM SoCC’12, 2012.*
19. CloudSim. [Online]. Available: *http://www.cloudbus.org/cloudsim/*
20. GreenCloud . [Online]. Available: *http://greencloud.gforge.uni.lu/*
21. iCanCloud. [Online]. Available: *http://www.arcos.inf.uc3m.es/~icancloud/Home.html*

22. Amazon EC2. [Online]. Available: *http://aws.amazon.com/ec2/*
23. Mahadevan, P., Sharma, P., Banerjee, S., Ranganathan, P.," Energy aware network operations", *in IEEE INFOCOM Workshops, pp. 1–6 (2009)*
24. JG Koomey," Estimating total power consumption by servers in the US and the world", *Oakland, CA: Analytics Press. 2007.*