Optimizing the Energy Consumption of Servers and Networks in Cloud Data Centers

by

Jun Liu

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Information Systems Engineering) in the University of Michigan at Dearborn 2015

I would like to lovingly dedicate this dissertation to my Mother and my daughter.

A C K N O W L E D G M E N T S

This dissertation would not have been successfully completed without the contributions of many people. First of all, I would like to thank my thesis advisor, Dr. Jinhua Guo, who has given me the opportunity to undertake a PhD and provided me with invaluable guidance and advice throughout my PhD training. I would like to thank Dr. Jinhua Guo for supporting me during these years. I am also grateful to Dr. Di Ma for discussions and suggestions on related topics that helped me improve my knowledge in the area. I would like to express my gratitude to all my PhD committee members, Dr. Kiumi Akingbehin, Dr. Di Ma, and Dr. Weidong Xiang, for their constructive comments and suggestions on improving my work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Data center is a cost-effective infrastructure for storing large volumes of data and hosting large-scale service applications. Cloud computing service providers are rapidly deploying data centers across the world. with huge number of servers and switches. These data centers consume significant amounts of energy, contributing to high operational costs. Thus, optimizing the energy consumption of servers and networks in data centers can reduce operational costs.

In a data center, power consumption is mainly due to servers, networking devices, and cooling systems, an effective energy saving strategy is to consolidate the computation and communication into smaller number of servers and network devices and then power off as many unneeded servers and network devices as possible.

In this thesis, we propose several novel methods to reduce the energy consumption of computer systems and networks in data centers, while satisfying Quality of Service (QoS) requirements specified by cloud tenants.

First, we study energy efficient scheduling of periodic real-time tasks on multi-core processors with voltage islands, in which cores are partitioned into multiple blocks (termed voltage islands). We propose a Voltage Island Largest Capacity First (VILCF) algorithm for energy efficient scheduling of periodic real-time tasks on multi-core processors. It achieves better energy efficiency by fully utilizing the remaining capacity of an island before turning on more islands or increasing the voltage level of the current active islands. We provide detailed theoretical analysis of the approximation ratio of the proposed VILCF algorithm in terms of energy efficiency.

Second, we study the resource allocation problem for virtual networks in data centers. A cloud tenant expresses computation requirement for each virtual machine (VM) and bandwidth requirement for each pair of VMs. A cloud provider places the VMs and routes the traffic among the VMs in a way that minimizes the total number of servers and switches used while providing both computation and bandwidth guarantees. The unneeded servers and switches are powered off to conserve energy. We formulate a special Multi-Capacity Bin Packing problem that consolidates VMs into the fewest number of servers. We present a weighted graph cut algorithm to map the consolidated virtual network into a data center network that minimizes the number of links and switches used.

CHAPTER 1

Introduction

Data centers are cost-effective infrastructures for storing large volumes of data and hosting large-scale service applications. Data centers contain hundreds of thousands of servers, interconnected via switches, routers and high-speed links. Today, large companies such as Amazon, Google, Facebook, and Yahoo! routinely use data centers for storage, web search, and large-scale computations [1]. With the rise of cloud computing, service hosting in data centers has become a multi-billion dollar business that plays a crucial role in the future Information Technology industry.

However, a large-scale computing infrastructure consumes enormous amounts of electrical power leading to very high operational costs that will exceed the cost of the infrastructure in a few years. In 2013, U.S. data centers consumed an estimated 91 billion kilowatt-hours of electricity, equivalent to the annual output of 34 large (500-megawatt) coal-fired power plants. The annual electricity consumption of data centers is projected to increase to roughly 140 billion kilowatt-hours by 2020, the equivalent annual output of 50 power plants, costing American businesses $13 billion annually in electricity bills and emitting nearly 100 million metric tons of carbon pollution per year [2].

In a data center, power consumption is mainly due to servers, networking devices, and cooling systems. There are two main approaches for reducing the energy consumption of data centers: (a) shutting down devices or (b) scaling down performance. The former, commonly referred as Dynamic Power Management (DPM), results in the greatest savings, since the average workload often remains below 30% of its capacity in cloud computing systems [3]. The latter corresponds to Dynamic Voltage and Frequency Scaling (DVFS) technology, which can adjust the performance of the hardware and power consumption to match the corresponding characteristics of the workload.

Virtualization represents a key technology for efficient operation of cloud data centers. Data center resources are often underutilized since the average load is about 30% of its capacity [3]. Energy consumption in virtualized data centers can be reduced by appropriate decision on which physical server a virtual machine (VM) should be placed. Virtual machine consolidation strategies try to use the fewest possible number of physical machines to host a certain number of virtual machines. According to Open Compute project report [4], 93% of the energy consumption in a data center depends upon efficient utilization of computing resources at data centers.

In this thesis, we propose several novel methods to reduce the energy consumption of computer systems and networks in data centers, while satisfying Quality of Service (QoS) requirements specified by cloud tenants.

1.1 Real-Time Task Scheduling Problem on Multi-core Processors with Voltage Islands

As the rate of the clock speed improvement of a single processor slowed, manufacturers shifted to add more independent processors (known as cores) into a single chip, which is called Multi-core Processor. Multi-core processors can achieve higher throughput with the same clock frequency. In addition, the proximity of multiple cores on the same chip allows the cache coherency circuitry to operate at a much higher frequency. However, power management becomes more challenging for real-time systems using multi-core processors.

To reduce overall power consumption in multi-core processors, the use of multiple supply voltages has been widely adopted. Often, this approach is realized through the use of core based voltage-islands [5]. A voltage island is a group of on-chip cores powered by the same voltage source, independently from the chip-level voltage supply. More voltage islands and fine-grained control of shutdown mechanism lead to lower power consumption.

Power consumption of chips generally breaks down into two sources [6], dynamic power and leakage power. The dynamic power of a core is a function of working frequency of the core. The dynamic power increases as the core frequency increases. This function is a convex function. There is a certain frequency where the energy efficiency of a core is optimal. The dynamic power of a core can be reduced by executing at the low frequency using Dynamic Voltage and Frequency Scaling (DVFS) technique [7,8]. In general, if the workloads across all cores are balanced, it would allow each core work at relatively low frequency and thus consume less dynamic power.

Leakage power comes from the circuit leakage current. As the frequency of a core is regulated to be slow, the dynamic power decreases, but the proportion of leakage power increases and leakage power becomes the dominant factor of the power consumption of a core. To reduce leakage power of a multi-core processor, when all cores in an island are idle, the island can be powered off using dynamic power management (DPM) [9] .

As the demand for concurrent processing and increased energy efficiency grows, it is expected that multi-core processors will become widely used in real-time systems. Energy-efficient task scheduling with various deadline constraints has received a lot of attention. Real-time tasks are partitioned into subsets, and these real-time subsets are assigned into cores respectively. Because the dynamic power is a convex function of core working speed, the workloads across all cores must be balanced so that each core work at relatively low frequency and thus consume less dynamic power. However, it is very challenging to sche real-time tasks to cores so that the workload of each core is balanced.

1.2 Virtual Network Mapping Problem in Data Centers

Since average server utilization in data centers is only 20%-30%, one method to improve the utilization of resources and reduce energy consumption is to dynamically consolidate Virtual Machines (VMs) into smaller number of physical machines using the virtualization technology. Virtualization partitions available resources and share them among different tenants. Server virtualization allows cloud providers to create multiple VM instances on a single physical server, thus improving the utilization of servers. Server virtualization also allows VMs to migrate between servers to consolidate workloads and reduce the number of active servers in data centers. Network virtualization aims at creating multiple virtual networks to improve network utilization. With such virtualization, the resources can be scheduled with fine-granularity. The energy consumption can be reduced by powering off idle serves and switches, thus eliminating the leakage power consumption.

In addition to energy efficiency, network performance in data centers is another important concern. Cloud providers do not offer guaranteed network resources to tenants. The bandwidth achieved by traffic flows between VMs of a tenant depends on a variety of factors outside the control of the tenant, such as the network load and placement of VMs.

Currently, most cloud data centers offer only guarantees on computation (CPU, memory, and storage) requirement for each VM, but networks are shared between tenants in a best-effort manner. Consequently, tenants experience variable and unpredictable network performance [10]. But a wide range of applications such as user-facing web applications , transaction processing web applications and MapReduce-like data intensive applications need predictable application performance. Without guaranteed network performance, cloud is unable to effectively support various classes of applications that rely on predictable performance.

To guarantee the network performance for tenants, Hetish proposed to abstract tenant requirements as a virtual network(VN). An effective network abstraction model [11] serves two purposes. One purpose is for tenants to specify their network requirements in a simple and intuitive yet accurate manner. The other purpose is to facilitate easy translation of these requirements to an efficient deployment on the low level infrastructure components.

There are two popular virtual network abstractions: hose model and pipe model. In the hose model abstraction, such as Oktopus [10] and ElasticSwitch [12], all VMs are connected to a central (virtual) switch by a dedicated link (hose) having a minimum bandwidth guarantee. This model mimics dedicated clusters with compute nodes connected through Ethernet switches. However, it does not accurately express the requirements for applications with complex traffic interactions. In the pipe model abstraction, such as SecondNet [13] and CloudNaaS [14], it specifies bandwidth guarantees between pairs of VMs as virtual pipes. When the traffic is predicable and relative stable, this model can precisely capture the application traffic needs. In this thesis, we are using the pipe model to describe the traffic needs of tenants.

Mapping multiple virtual networks into a data center network is very challenging. The data center must provide all resources that each virtual network requires, including the CPU and memory requirement for each VM and bandwidth requirement for each pair of VMs. Virtual network mapping must analyze tenant VNs and identify VM patterns with high traffic flows and consolidate VM patterns to physical machines in close proximity to reduce the number of active servers and switches.

Since Today's data center networks have been designed with redundant links and switches, Virtual Network Mapping must also optimize the flow routing cross data center networks to minimize the number of active switches. VN mapping can save energy by shutting down some unused physical machines and switches. Jointly optimizing server energy consumption and network energy consumption is a challenging problem in VN mapping. Better mapping strategy can minimize the number of active physical machines and switches, increase the resource utilization, and thus save energy.

1.3 Research Problems and Objectives

This thesis tackles the research challenges in the aforementioned two problems: Real-Time Task Scheduling Problem on Multi-Core Processors with Voltage Islands and Virtual Network Mapping Problem in Data Centers.

In the first problem, this thesis studies energy efficient scheduling of periodic real-time tasks on multi-core processors with voltage islands, in which cores are homogeneous and partitioned into multiple blocks (termed voltage islands) and each block has its own voltage supply. Cores in the same block always operate at the same voltage level, but can be adjusted by using Dynamic Voltage and Frequency Scaling (DVFS). In particular, the following research problems are investigated:

- How to schedule real-time tasks into multi-core processor with voltage island model?
- How to decide the core speed according to the workload?
- What is the lower bound of the scheduling problem?
- What is the upper bound of the scheduling algorithm?
- How to derive the approximation ratio of a scheduling algorithm?

In the second problem, this thesis investigates the energy efficient resource allocation problem for virtual networks in cloud data centers with a three-tier fat-tree topology. A tenant's request can be abstracted to a virtual network with a set of virtual machines (VMs) and the links between each pair of VMs. A cloud provider places the VMs and route the traffic among the VMs in a way that minimizes the total number of servers and switches used while providing both computation guarantee for each VM and bandwidth guarantee for each pair of VMs. The unneeded servers and switches are powered off to conserve energy. In particular, the following research problems are investigated:

- What is the bandwidth constraint in a three-tier fat-tree network?
- How to formalize the Virtual Network Mapping problem under computation and communication constraints?
- How to consolidate VMs into a smaller number of physical servers in a way that minimizes the number of servers used?
- How to identify traffic patterns in virtual networks?
- What is the time complexity of VM packing?
- How to map VM clusters into physical machines in a data center?

1.4 Contributions

The contributions of this thesis can be broadly divided into three categories: survey and analysis of the power saving strategies, competitive analysis of periodic real-time task scheduling problem on multi-core processors with voltage islands, analysis of virtual machine placement and flow routing for communication-intensive service applications in data centers. The **main contributions** are:

- A review of the state-of-the-art in energy-efficient computing.
- Competitive analysis of periodic real-time task scheduling problem on multi-core processors with voltage islands.
  - Formal definitions of the Voltage Island Energy-Efficient Real-Time Task Scheduling (VIEES) problems.
  - Proposed a Voltage Island Largest Capacity First (VILCF) algorithm for energy-efficient scheduling of periodic real-time tasks on multi-core processors.
  - Analysis of the lower bound of the energy consumption of the optimal solutions for Voltage Island Energy Efficient Scheduling (VIEES) problem.
  - Theoretical analysis of the approximation ratio of the proposed VILCF algorithm, in terms of energy efficiency, against the lower bound of the optimal solution.
- Virtual machine placement and flow routing for communication-intensive service applications in data centers.
  - Formal definition of the Virtual Network Mapping problem in the fat-tree topology data centers.
  - Prove Virtual Network Mapping is a NP-hard problem.
  - Proposed a two-phase VM placement and network routing algorithm that significantly reduces both the number of active servers and switches needed to provide QoS guarantee.

* Virtual Machine Packing: formulated a special Multi-Capacity Bin Packing problem that consolidates VMs into the fewest number of servers.
* Virtual Network Mapping: presented a weighted graph cut algorithm to map the consolidated virtual network into a data center network that minimizes the number of links and switches used.

The results from this work have been published in two journal papers [15] [16] and four conference papers [17] [18] [19] [20].

1.5 Thesis Organization

In Chapter 2, we present a survey of energy-efficient computing systems, as well as the scope of this thesis and its positioning in the area. We review the energy efficiency techniques in hardware design including Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management (DPM). Then, we present the recent development on multi-core processors with voltage islands and energy efficient real-time task scheduling algorithms on multi-core processors. Furthermore, we describe several new data center network architectures and two popular virtual network models.

In Chapter 3, we present the preliminary work on powering saving design for servers with response time constraint. We introduce a smart power saving scheme, PowerSleep, which aims at power saving for a single server. PowerSleep can minimize power consumption of a single server under the mean job response time constraint while adopting the extended M/G/1/PS queuing model for job arrival and job processing. PowerSleep adjusts the server working frequency during running time by DVFS, and puts the server into the sleep mode once the queue is empty, and activates the server to work once a new job arrives. To overcome the transition overhead, PowerSleep adds procrastination sleep period when a new job arrives while the severs still in the sleep mode. The server keeps sleeping during the procrastination sleep period to collect more jobs into the queue. After the procrastination sleeping time, the sever wakes up to process jobs. This approach reduces the mode transition overhead, but it increases the job response time.

In Chapter 4, we investigate energy efficient scheduling of periodic real-time tasks on multi-core processors with voltage islands. We propose a Voltage Island Largest Capacity First (VILCF) algorithm for energy efficient scheduling of periodic real-time tasks on multi-core processors. It achieves better energy efficiency by fully utilizing the remaining capacity of an island before turning on more islands or increasing the voltage level of the current active islands. We provide detailed theoretical analysis of the approximation ratio of the proposed VILCF algorithm in terms of energy efficiency.

In Chapter 5, we explore the resource allocation problem for virtual networks in data centers. We formulate a special Multi-Capacity Bin Packing problem that consolidates VMs into the fewest number of servers. A server is represented by a bin with multiple capacities corresponding to the computation resource (CPU, memory, and storage) and communication bandwidth resource available. Unlike bin packing, the cumulative bandwidth demand of all VMs hosted in a server can be smaller than the sum of bandwidth demand of individual VMs due to the co-location. Furthermore, we present a weighted graph cut algorithm to map the consolidated virtual network into a data center network that minimizes the number of links and switches used.

CHAPTER 2

Review of the State of the Art

2.1 Energy-Efficient hardware

2.1.1 Dynamic Voltage and Frequency Scaling

Dynamic Voltage and Frequency Scaling (DVFS), a voltage reduction technique for battery-operated systems scaling, was introduced in the 90s [21], which dramatically reduces power consumption in large digital systems by adapting both voltage and frequency of the system with respect to changing workloads. Equipped with DVFS, a regulated system can adjust the supply voltage of a digital circuit at the functional boundary for the speed requirements, the temperature, and the technology parameters.

DVFS describes the use of two power saving techniques (dynamic frequency scaling and dynamic voltage scaling) used to save power in embedded systems including cell phones.

The power consumption of a digital circuit is given by the well-known formula [21]:

$$Power = f * C * V^2 + V * I_{DC} \tag{2.1}$$

where f is the operating frequency, C is the equivalent capacitance of the circuit, and I_{DC} is the static current. V is the supply voltage of the digit circuit. The first term is the dynamic power.

Power consumption of digital circuit generally breaks down into two sources, dynamic power and leakage power. The dynamic power consumption of a digital circuit P_{dyn} is : $f * C * V^2$

The followed formula shows the dependency of operating frequency on supply voltage [22]:

$$f \propto \frac{(V - V_{th})^{\alpha}}{V} \tag{2.2}$$

where V_{th} is the threshold or switching voltage, and the exponent α is an experimentally derived constant that, for current technology, is approximately 1.3 to 2. Its V^2 factor suggests reducing supply voltage as the most effective way to decrease power consumption.

The static power lost due to leakage current is $V * I_{DC}$. Leakage current, the source of static power consumption, is a combination of sub-threshold and gate-oxide leakage: $I_{leak} = I_{sub} + I_{ox}$. The power of the two leakage currents are $P_{sub} \propto 1 - e^V$ and $P_{gate} \propto V^2$ [22]. Leakage power can be reduced by voltage scaling and sleep transistors since Both P_{sub} and P_{gate} are a function of the supply voltage. leakage power continues to become a dominant contribution to power consumption for future silicon technologies.

Dynamic voltage and frequency scaling (DVFS) can be applied to different levels of granularity [23]. Per-chip DVFS uses the same power delivery network to reach every core, and consequently, binds each core to the same DVFS schedule. Per-core DVFS uses a separate voltage regulator for each core and therefore allows every core to have an independent DVFS schedule. Cluster-level DVFS uses multiple of on-chip regulators drive a set of DVFS domains, or clusters, so that one or more cores are associated with each cluster. The smaller the granularity, the more complex the design and the larger the overhead. The trend towards multi-processor architectures makes scaling on individual processors an attractive approach. Many applications tend to map well on parallel processing architectures, especially digital signal processing applications.

2.1.2 Multiple Supply Voltage

As dynamic power is proportional to the square of supply voltage, reducing supply voltage can significantly reduce active power consumption. Multi-supply voltage (MSV) [24] is introduced to provide finer-grain power reduction and performance trade-off.

MSV can make a chip to work as slowly as possible with the lowest possible supply voltage. The voltage scaling technique allows modules on the critical paths to use the highest voltage level thus meeting the required timing constraints while allowing modules on noncritical paths to use lower voltages thus reducing the energy consumption. This scheme tends to result in smaller area overhead compared to parallel architectures. Consequently two or more supply voltages were employed in the chip. Reducing the supply voltage however increases the circuit delay.

There are two types of MSV. In row-based type, there are interleaving high and low supply voltage standard cell placement rows. In region-based type, circuits are partitioned into voltage islands where each voltage island occupies a contiguous physical space and operates at a supply voltage that meets the performance requirement.

Power reduction of MSV can be considered in two aspects, multiple voltage scheduling and MSV design. The multiples voltage scheduling problem is to assign a supply voltage level selected from a finite and known number of supply voltage levels to each operation in a data flow graph (DFG) and schedule various operations so as to minimize the energy consumption under given timing constraints. JuiMing Chang [25] presented a dynamic programming technique for solving the multiple supply voltage scheduling problem.

In MSV design, how to divide the regions in the circuit is a problem, especially for the circuit with region based type MSV, Designers partition circuits into a few groups based on their performance requirement and the connectivity between modules. Each group is then specified with a supply voltage. Logic boundaries are largely used group partitioning. However, these natural boundaries in a design are almost always non-optimal boundaries for supply voltages.

Another problem of using MSV is routing of multiple supply voltage lines. In region based MSV, each voltage island has power line to support power, but it is high cost to implement fragmented power networks as implementing such complex power network will take a lot of precious routing resources from design. That is there is a trade-off between lower energy dissipation and higher routing cost. Huaizhi Wu in [26] investigated Voltage Island boundary design in a circuit for optimal power versus design cost trade-off under timing requirement.

2.1.3 Dynamic Power Management

Dynamic power management refers to power management schemes implemented in electronic systems while programs are running [9].

Electronic systems can be viewed as collections of components which may be heterogeneous in nature. Such components may be active at different times, and correspondingly consume different fractions of the power budget. Electronic systems are designed to be able to deliver peak performance when requested. Nevertheless, peak performance is required only during some time intervals., system components are not always required to be in the active state [27] .

DPM can enable and disable components, as well as tune their performance to the workload to achieve energy-efficiency. DPM is a design methodology for dynamically reconfiguring systems to provide the requested services and performance levels with a minimum number of active components or a minimum load on such components. There is a dynamic power managers in DPM, a power manager implements a control procedure based on some observations and/or assumptions on the workload.

DPM is a design methodology for dynamically reconfiguring systems to provide the requested services and performance levels with a minimum number of active components or a minimum load on such components.

The power manager implements a control procedure or user-defined and/or application-specific power management strategies [28] to control component power state based on some observations and/or assumptions on the workload.

A component can be modeled by a finite-state representation called power state machine (PSM). States are the various modes of operation. DPM controls power consumption of a component by transition its power modes. State transitions have a power and delay cost. In general, low-power states have lower performance and larger transition latency than states with higher power.

DMP can be implemented on several levels: component, system and network cover and relate different approaches to system-level DPM.

Vassos Soteriou and Li-Shiuan Peh [29] applied DPM to interconnection networks to save power consumption. The links which interconnect network node routers are a major consumer of power. They studied graph connectivity of a 2D mesh topology and power balance in the graph. and proposed a dynamic power management policy where network links are turned off and switched back on depending on network utilization in a distributed fashion.

2.2 Energy-Efficient Multi-Core Processor with Voltage Island Model

2.2.1 Multi-Core processor

As semiconductor technology of CMOS marches forward in accordance with Moores Law, more and more transistors can be packed into a single chip, this has caused chip speeds to rise. However, transistors can not shrink forever. Manufacturers can not continue making single processor cores more powerful limited by current transistor technology. One reason is that as a transistor gets smaller, gates are unable to block the flow of electrons, transistors tend to consume more power; another reason is that increasing clock speeds causes transistors to switch faster and thus generate more heat and consume more power. They shift to add more independent and actual processors(known as cores) into a single chip using so many transistors, this chip is called multi-core processor [30]. In Multi-core processors, different cores execute different Instructions, operating on different Data. All cores share the same memory

The multi-core chips do not necessarily run as fast as the highest performing single-core models, but they can achieve higher throughput with the same clock frequency by handling more work in parallel. There are two kinds of parallelism. One is Instruction-level parallelism, this kind of parallelism enables processor to reorder, pipeline instructions, split them into microinstructions, do aggressive branch prediction. Another is thread-level parallelism. Many new applications are multi-threaded. For multiple tasks that all have to run at the same time, thread-level parallelism enable the chips to use a separate core for each task.

For applications that cannot be parallelized statically, To take advantage of multi-core chips, programmers must find good places to break up the applications, divide the work into roughly equal pieces that can run at the same time.

Krishnan, et al. [31] presented an efficient chip-multiprocessor (CMP) architecture for exploiting thread-level parallelism. This architecture speculatively executes sequential binaries without the need for source recompilation. It uses software support to identify threads from a sequential binary. It includes memory disambiguation hardware to detect inter-thread memory dependence violations.

2.2.2 Voltage Island

Voltage Islands is a system architecture and chip implementation methodology, that can be used to dramatically reduce active and static power consumption for System-on-Chip (SoC) designs [32].

As the scale of process technologies steadily shrinks, more and more devices can be implemented on a single chip. This enables various applications to be realized as System-on-a-Chip (SoC) designs. SoCs consist of programmable processors and peripheral cores that are connected to standard bus-based architectures.

The dynamic power of a core in a chip is a function of operation speed, it increases as operation frequency increases. However, frequency of operation has increased at a faster rate than the scaling of the silicon process technology. This has led to an increase in power density of a SoC. In addition to active power, there are components of leakage power, the most dominant of which is the subthreshold current of the transistors in the circuit which drives significant increases in leakage power. The combination of increasing active power density and increasing leakage currents has created a power management problem in the semiconductor industry [5].

The core-base design using voltage islands is a new technique which helps reducing both switching and standby components of power dissipation. Simply speaking, a voltage island is a group of on-chip cores powered by the same voltage source, independently from the chip-level voltage supply. The use of voltage islands permits operating different portions of the design at different voltage levels in order to optimize the overall chip power consumption. Voltage island is a type of MSV.

Introducing voltage islands makes the chip design process even more complicated with respect to static timing, power routing, floor-planning,

Floor-planning is to create voltage island on the chip. Hu et al. [33] studied the floor-planning process that is assumed to happen after the die size and package have been chosen. The task of island partition creation and level assignment are implemented under the physical constraints involved in the design process. They proposed an method to minimize power consumption through architecting voltage islands in core-based designs and presenting an algorithm for simultaneous voltage island partitioning, voltage level assignment, and physical-level floor planning.

Wan-Ping Lee et al. [34] handled voltage island partitioning by dynamic programming (DP). Given a netlist without reconvergent fanouts, the DP can guarantee an optimal solution for the voltage assignment in linear time while meeting timing constraint.

However, the chip reliability which is an immunity to soft errors, is becoming another important issue since large amount of transistors are packed in a chip. Soft errors are circuit errors caused due to excess charge carriers induced primarily by external radiations. Reducing Supply voltage is a most effective way to reduce system power consumption but it makes cores more sensitive to soft errors. Yang et al. [35] proposed a reliability based voltage island partitioning and floor-planning for System-On-a-Chip (SOC) design.

2.2.3 Energy-Efficient Task Scheduling in Multi-core processor

Power-aware computing and energy efficient scheduling of real-time tasks have been well studied in the past decade. Due to the convexity of power consumption function, dynamic voltage and frequency scaling (DVFS) that slows down the processing speed of processors is a commonly used technique for energy savings [7] [8]. Based on the DVFS technique, many real-time task scheduling schemes have been developed for uniprocessor and multiprocessors, e.g., [36], [37].

There have been studies on scheduling real-time tasks on multi-core processors. Most of them assume per-core DVFS, where each core has its own power supply and can change its voltage and frequency level independently from other cores, e.g., [38], [39], [40], [41] [42]. Chen et al. [38] investigated energy-efficient scheduling of periodic real time tasks over multiple DVFS processors with the leakage power consideration and proposed a polynomial time algorithm, Largest Task First (LTF), to derive task mappings that try to execute at a critical frequency. Vinay Devadas et al. [42] suggested a real-time scheduling algorithm which keeps the performance demands of each core balanced by migrating tasks between cores. In addition, they proposed Algorithm Dynamic Core Scaling that changes the number of active cores to reduce leakage power consumption in the case of low load.

Some recent works have explored the voltage islands model. However, they focused on either configurations of voltage islands or different applications. Qi and Zhu [43] studied symmetric and asymmetric block configurations for multi-core processors, which contain the same and different number of cores on each block, respectively. Each block has a DVFS-enabled power supply. They investigated the block-partitioned core configurations for multi-core processors and evaluated the energy efficiency for different block configurations.

Ozturk et el. [44] observed that at the compile time, the loads across of cores in a chip are imbalanced. They developed algorithms to map the application codes to voltage islands and assign different voltages to different processors to provide energy saving. Santiago and Chen in [45] studied the periodic real-time task scheduling and proposed a Single Frequency Approximation (SFA) scheme that uses a single voltage and frequency for executing, particularly, the lowest voltage and frequency that satisfies the timing constraints in voltage island based multi-core processors. However, they used the original Largest Task First (LTF). Devadas et al. [46] investigated energy management in multi-core processor with one voltage island for a periodic real-time workload that is partitioned to processing cores by taking into account both static and dynamic power. In their solution, they first select a subset of cores upon which the workload can be executed without dissipating excessive static power, then assign core speed by DVFS to reduce dynamic power.

Kong et el. [47] presented a real-time task scheduling approach used to determine the proper number of active voltage islands, task partition, and frequency assignment for voltage island model. They used Algorithm Last Task Fist (LTF) to partition tasks into subset and assign the subsets into cores. In their model, cores in the same block can work at different power mode. Some cores can work at a certain speed to process tasks while others can be set idle with no load assigned into.

Sheshadri et el. [48] employed DVFS technique in session-less power constrained test scheduling of a system-on-chip (SOC). Scaling voltage and frequency using DVFS can alter the test time and power of a core test, but in SOC test scheduling, it is restricted by the maximum frequency limit of individual cores and the power limit of an SOC. They proposed heuristic approaches to minimize the overall test time of an SoC by scaling the voltage and frequency of SOC under its power limit and the maximum frequency limit of individual cores.

Recently, researchers have started exploring energy-efficient scheduling with the considerations of the non-negligible power consumption of leakage current for current and future circuit manufacture process [38] [46] [42]. To save energy consumption, cores or voltage islands might be turned off whenever needed. Chen [38] calculated the approximation ratio of their algorithm to be 1.283 under considering leakage power.

2.3 Virtualization

2.3.1 Virtualization

Virtualization is a method used to improve energy efficiency of data center which allows the sharing of one physical server among multiple virtual machines (VM). Virtualization is implemented in both the server and switch domain but with different objectives. Server domain virtualization usually achieves energy efficiency by sharing limited resources among different applications.

Virtualization in the Data Center Network domain, on the other hand, aims to implement logically different addressing and forwarding mechanisms on the same physical infrastructure. Data Center Network virtualization separates logical networks from the underlying physical network, letting each virtual network (VN) can implement customized network protocols and management policies [49]. Also, since VNs are logically separated from one another, implementing performance isolation and application QoS is facilitated. Management procedures of VNs will be more flexible because every VN has its own control and management system. Furthermore, isolation offered in network virtualization environments can also minimize the impact of security threats.

Virtualization enables services to be moved between servers, and virtualization has multiple VMs which can serve different applications multiplexed to share one server. Data center resources are underutilized since the average traffic load accounts for about 30% of its resources [3], data centers can migrate virtual machines to consolidate workloads on a set of servers and then shut down underutilized servers to save a great power. The migration of VMs is optimized by selecting the VMs to be relocated on the basis of heuristics related to utilization thresholds.

Stage et al. [50] discuss the impact of VM live migration on the network resources. A migration scheduler determines the optimal schedule for themigrations, based on the knowledge of their duration, starting time and deadline. The optimal scheduler schedules the live migrations in such a way that the network is not congested by the VM live migration load. The live migrations are also fulfilled in time. Beloglazov et al. [51] have proposed that live migration of VMs can be used to concentrate the jobs on a few physical nodes so that the rest of the nodes can be put in a power saving mode. The allocation, of new requests for VMs, is done by sorting all the VMs in a Modified Best First Decreasing (MBFD) order with respect to the current utilization.

In [52], they presented a green energy-efficient scheduling algorithm which makes the use of priority job scheduling for Cloud computing. The priority job scheduling is used to select VMs for executing jobs. The VMs are selected according to the weight computed by resource and the SLA level required by users. Their method can satisfy the minimum resource requirement of a job and prevent the excess use of resources. The DVFS technique is used to control the supply voltage and frequency for servers in Cloud computing. This technique can reduce the energy consumption of a server when it is in the idle mode or the light workload.

In [53] Wang et al. studied the problem of achieving energy efficiency in data center networks using traffic engineering. They proposed an time-aware virtual machine placement problem. Based on the unique features of data center networks and the switch power model, they proposed three main principles for virtual machine assignment in order to achieve energy efficiency of the network in data centers. Then, they analyzed the relation between the power consumption and routing and propose a two-phase energy-efficient routing algorithm.

2.3.2 Bandwidth Guarantee

Cloud data centers depend on high-performance networks to connect servers within the data center and to the rest of the world. Many cloud customers do want to be able to rely on network performance guarantees.

Ballani et al. [10] summarize several measurement studies showing huge variations in intra-cloud bandwidth, and describe how performance variability leads to poor and unpredictable application performance. Some of these studies have also shown that no-guarantee cloud networks also suffer from high and highly variable latency, and high loss rates. In Oktopus [10], they proposed a recursive VM allocation approach for homogeneous bandwidth demand.

Guo et al. [13] proposed the Virtual Data Center (VDC) abstraction for their SecondNet architecture, with three service models: type-0 service guarantees bandwidth between pairs of VMs3; type-1 service provides ingress/egress guarantees for a specific endpoints; other traffic is treated as best-effort. An endpoint can be an entire VM, or a TCP/UDP port on a VM.

Wang et al. [54] proposed an online VM packing algorithm while ensure dynamic bandwidth Demand in data centers. Unlike the traditional VM packing schemes that characterize the network bandwidth demands of VMs by a fixed value, they capture VM bandwidth demand by random variables following probabilistic distributions. they pack VMs into servers such that the number of serer used is minimal and the chance for VM bandwidth demand violating network constraint is below a threshold.

However, Oktopus [10] and SecondNet [13]are not work conserving. Work conservation can fully utilizes the spare bandwidth from unreserved capacity increase VM bandwidth. Popa [12] applied work conservation to ElasticSwitch. ElasticSwitch provides minimum bandwidth guarantees with hose model abstractions. The hose model offers the abstraction that all VMs of one tenant appear to be connected to a single virtual switch through dedicated links. Each hose bandwidth guarantee is transformed into pairwise VM-to-VM rate-limits, and work conservation is achieved by dynamically increasing the rate-limits when the network is not congested.

Popa et al in [55] identified three main requirements that a desirable solution for sharing cloud networks should meet: min bandwidth guarantee, proportionality (ranging from the network level to the link level) and high utilization, and a set of properties to guide the design of allocation policies in the trade-off space. They showed that one cannot simultaneously provide both bandwidth guarantees and network proportionality. The paper proposes mechanisms that can achieve different subsets of the desired properties: link-level proportionality, restricted forms of network proportionality, and minimum bandwidth guarantees over tree-structured networks.

Rodrigues et al. [56] described Gatekeeper, which focuses on providing predictable performance. Gatekeeper attempts to provide each tenant with the illusion of a single, nonblocking switch connecting all of its VMs. Each VM is given guaranteed bandwidth, specified per-VM, into and out of this switch.4 Optionally, a VMs maximum bandwidth can be set larger than its guarantee, to allow use of otherwise underutilized bandwidth. This allows the provider to trade off between efficiency and predictability, by adjusting either or both of the minimum and maximum bandwidths.

2.3.3 Pipe Model

There are four granularities of bandwidth guarantees:Tenant aggregate, per-VM hose model (see Figure (2.1)), per-VM Pipe model (see Figure (2.2)), and per-flow QoS model. These models are listed in increasing granularity and flexibility of expressing a tenant's needs.

Jiang et al [57] investigated VM placement of multiple tenants in data center. They formulated a problem jointly optimizing VM placement and traffic routing in data center. The VM traffics of a tenant is represented by a traffic matrix which is a pipe model. They proposed heuristic algorithms using Markov Chian to bring a solution.

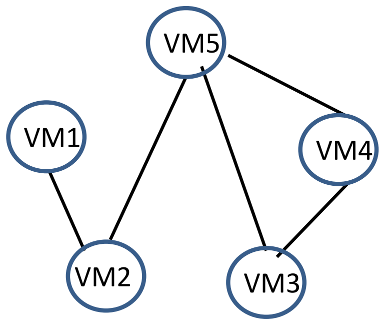

Figure 2.1: Pipe Modle

Guo et al. in [13] proposed an algorithm to map a set of VMs to a data center with arbitrary topology, their algorithm is divided into two steps, first is to decide weather the allocation is available, another is how to allocate VMs. They extract a set of physical servers in cluster structures of different size from data center, and map the VMs to these clusters. These clusters looks like container to hold the VMs and their traffics. However, it is hard to know the proper size of a cluster before making allocation, and how to efficiently extract clusters from a data center is another issue.

Wang et al. in [58] investigated power model of switches and proposed an algorithm to allocate VMs to fat-tree structure data center. In their algorithm, they first keep building ”super” VMs by merging VMs pairs with the largest traffic flow load and assigning them into a server until exhausting all server resource, then they regroup these ”super” VMs by k-means clustering and put these clusters into pods. Their algorithm can be further refined.

Meng et al. in [59] researched the optimization of VM placement in data centers to improve scalability; VMs with large mutual bandwidth usage are assigned to host machines in close proximity in data center. The NP-hard VM placement problem is approximated using a two-tier algorithm that takes the traffic matrix between the VMs and cost matrix between the hosts as input. The algorithm partitions the servers into clusters based on the cost between clusters. The VMs are then partitioned into VM clusters in such a manner that minimizes the inter-cluster traffic.

Daniel et al. in [60] proposed an traffic aware algorithm for VM placement in heterogeneous bandwidth model. Their approach partitions VM sets by removing the edge with minimum traffic load in the request virtual network until all partitioned parts can be assigned into the servers of data center, extract VM patterns by traffic matrix, But, they did not consider network resource allocation and bandwidth guarantee for VM placement.

Fang et al. in [61] investigated data center network cost optimization based on VM placement. They use VM traffic matrix to allocate VMs to data centers. They use Gomory Hu Tree to assign VMs into physical servers, and then use tabu search approach to map the physical servers into data center racks.

Li [62] studied the VM placement problem for cost minimization. The cost is caused by PMs which hold VMs and network cost which is mainly determined by inter-PM traffic. They observed that it is hard to minimize both PM cost and network cost. There is a trade off between the two costs. They define network cost by various functions, according to different communication models and proposed an approximation solution to jointly optimize PM cost and network cost.

2.3.4 Hose Model

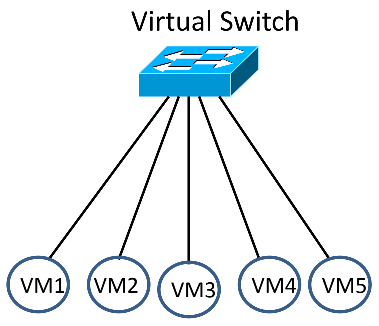

Figure 2.2: Hose Model

Hose model is initially proposed [63] for Virtual Private Networks (VPNs), it can be applied to data center as well. Hose model aggregates the demands for multiple different communications, all VMs are connected to a central (virtual) switch by a dedicated link (hose). having a minimum bandwidth guarantee.

Ballani in Oktopus [10] propose two virtual network abstraction models that cater to tenant requirements. The first is virtual cluster which abstracts a tenant as a virtual network in such a topology: all VMs are connected to a single, non-oversubscribed (virtual) switch resulting in a one-level tree topology. virtual cluster is a homogeneous hose model, each VM requires the same amount of compute resource and link bandwidth connecting to virtual switch. virtual cluster is suitable to all applications, but the tenant cost is not low and provider flexibility is not high. The second model is Virtual Oversubscribed Cluster (VOC), VOC interconnects clusters with switch-to-VM bandwidth B with a fixed over-subscription factor O. Comparing with virtual cluster model, VOC is not suitable to all applications , but it is more flexible to providers to allocate and tenant cost is lower.

Zhu et al. [64] generalized virtual cluster which is a homogeneous hose model to a heterogeneous hose model where each VM can have a heterogeneous bandwidth guarantee. This model can more accurately express the requirements for applications composed of multiple tiers with complex traffic interactions than virtual cluster, the bandwidth between VMs of the same tenant can vary significantly over time, depending on the network load and usage from other tenants.

With the hose model, each VM gets a minimum guarantee for all its traffic, irrespective of whether the traffic is destined to the same tenant or not. However, in multi-tenant data center, tenants can access other tenants or services, inter-tenant communication should be considered. Ballani et al. [65] proposed a hierarchical hose model with communication dependency for inter-tenant communication abstraction. In this model, each VM of a tenant gets a minimum bandwidth guarantee irrespective of intra- or inter-tenant traffic and the whole beyond this, the hierarchical hose model introduces an aggregate abstraction for a tenant's inter-tenant traffic. The tenant also gets a minimum bandwidth guarantee for its aggregate inter-tenant traffic.

In [66], the authors first profiled the traffic patterns of several popular cloud applications, and find that they generate substantial traffic during only $30\% - 60\%$ of the entire execution, suggesting existing simple VC models waste precious networking resources. We then propose a fine-grained virtual network abstraction, Time- Interleaved Virtual Clusters (TIVC), that models the time-varying nature of the networking requirement of cloud applications.

Base on hierarchy hose model, Shen [67] proposed Dual-Hose Model for bandwidth guarantees in multi-tenant cloud networks over-provisioning in accommodating tenant requests. This model decouples the guarantee for each VMs inter-tenant traffic from the one for its intra-tenant traffic. Each VM is provided with a minimum bandwidth guarantee for its traffic to other VMs of the same tenant and a minimum bandwidth guarantee Be for its traffic to other tenants, while in hierarchy hose model, VM only ensures minimum bandwidth guarantee for the aggregated inter and intra tenant traffics.

Lee [11] studied interactive applications such as web and OLTP hosted in todays cloud environments. These applications have complex and tiered structures. Lee proposed a TAG model to abstract these multi-tier application bandwidth requirements. TAG is a graph, where each vertex represents an application tier where a set of VMs performing the same function. bandwidth guarantees from one tier to another tier is labeled as directed edges between the corresponding vertices in the TAG model. VMs in a tier, if they have traffics among them, these VMs form a hose model. For tier A to tier B , VMs in tier a and B form a directional hose model.

2.4 Data Center Architecture

2.4.1 Data Center Architecture

Data centers can be categorized mainly in two classes, the switch-centric and the server-centric. In switch-centric data center, switches are the dominant components for interconnection, while in server-center data center, servers with multiple Network Interface Cards (NIC) take part in routing and packet forwarding decisions. Switch-centric data centers include VL2 [68], Portland [69], Fat-Tree [70] .Switch-centric data centers include Dcell [71], Bcube [72].

The fat-tree topology, depicted in Figure (2.3), consists of k pods, each of which consisting of $\frac{k}{2}$ edge switches and $\frac{k}{2}$ aggregation switches. Edge and aggregation switches connected as a clos topology and form a complete bipartite in each pod. Also each pod is connected to all core switches forming another bipartite graph. Fat-Tree built with k-port identical switches in all layers of the topology and each of which supports $\frac{k^3}{4}$ hosts. fat-tree IP addresses are in the form 10:pod:subnet:hosted. In fat-tree, the address lookup is implemented by a two table lookup approach instead of the longest prefix matching. Address lookups are done in two steps; first the lookup engine does a lookup on the first level table to find the longest matching prefix. Then the matched address is used to index the second level table which holds the information of the IP address and output port to reach the intended destination.

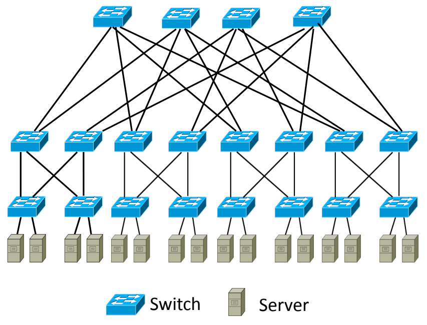

Figure 2.3: Fat-tree topology

VL2 was proposed in [68] and considered as a solution to overcome some of the critical issues in conventional data centers such as over-subscription, agility and fault tolerance.VL2 supports VM migration from server to server without breaking the TCP connection and keeping the same address. VL2 implements a clos topology between core and aggregation layers to provide multi-path and rich connectivity between the two top tiers. VL2 employs Valiant Load Balancing (VLB) to evenly load balance traffic flows over the paths using Equal Cost Multi Path (ECMP).

Portland was proposed in [69] whose DCN topology is based on a fat-tree network topology. Portland consists of three layers: edge, aggregation and core. It is built out of low cost commodity switches. Portland and fat-tree both differ in the addressing scheme for packet routing but both at the end aim at providing agility among services running on multiple machines. Both reduce broadcast by intercepting Address Resolution Protocol (ARP) requests and employ a unicast query through a centralized lookup service. Portland implements hierarchical Pseudo MAC (PMAC) addresses for layer 2 routing and forwarding protocol. PortLand assigns a unique PMAC address

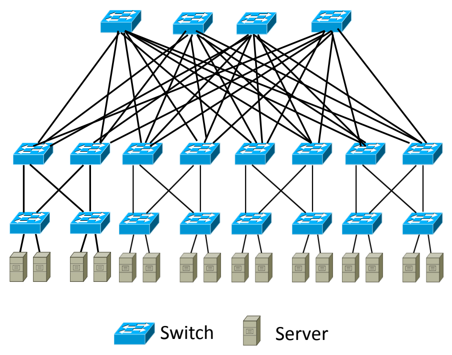

Figure 2.4: VL2 topology

to each end host.

Unlike switch centric designs, server centric designs appeared to use servers to act as relay nodes to each others and participate in the traffic forwarding. Server centric schemes such as Bcube [72], Dcell [7] [71] can provide low diameter compared to switch centric schemes, can provide high capacity and support all types of traffic, especially for the intensive computing applications with very low delays.

2.4.2 Energy-Efficient Data Center

The DCN architectures are over-provisioned for peak loads and fault tolerance. On average, the DCNs remain highly underutilized with an average load of around 5%-25% . Network over-provisioning and underutilization can be exploited for energy efficiency. Heller et al. [73] proposed ElasticTree, to consolidate the workload on a subset of network resources to save energy. The authors estimated a feasibility of around 50% energy savings using simulation and hardware prototype.

Shang [74] proposed an energy-aware routing algorithm for data centers. The objective of energy aware routing is to save power consumption via putting idle devices on sleep or shutting them down and using few network devices to provide routing with no sacrifice on network performance. The algorithm first computes the network throughput through basic routing. Then, it gradually removes switches until the network throughput approaches the predefined performance threshold.

Finally, it powers off or puts on sleep mode the switches that are not involved in the final routing.

Fat-tree interconnection networks are one of the most popular topologies. The particular characteristics of this topology is that it is designed with redundancy to provide multiple alternative paths for each source/destination pair. Alonso et al. [75] presented a DPM mechanism that dynamically switches on and off network links as a function of traffic and this mechanism is designed to guarantee network connectivity.

In [76], the authors proposed data center energy-efficient network aware scheduling (DENS) whose main objective is to balance the energy consumption of a data center with performance, QoS and traffic demands. DENS achieves this objective via the implementation of feedback channels between network switches for workloads consolidation distribution amendments to avoid any congestion or hot spots occurrences within the network which can definitely affect the overall performance. Congestion notification signal by overloaded switches can prevent congestion which may lead to packet losses and sustain the high data center network utilization.

CHAPTER 3

Preliminary Work: Power Saving Design for a Single Server

3.1 Introduction

Power-aware design for servers is a prominent design issue, since servers consume a major part of the power in a data center. Power consumption of a server can be reduced by Dynamic Voltage Frequency Scaling (DVFS). However, DVFS is not efficient in modern computer systems where the static power consumption plays a major role in the total server power consumption. As shown in [77], the static power dissipation when a server is idle could reach up to 60% of the peak power, and is worsened if the power waste in power delivery and cooling sub-systems is counted, which could increase the power consumption by $50\text{-}100\%$. One way to reduce the static power consumption is to power off a server when it has no job to execute. Jobs arrive at a server randomly. When a server is in the sleep mode and a job arrives, the server must wake up to execute the job. This makes a server to turn on/off frequently. Dynamic Power Management (DPM) can be used to switch servers between the sleep and idle modes.

DVFS and DPM can be jointly used for the power management in a single server. However, slowing server working speed and frequent mode transition between the active mode and the sleep mode introduce significant overheads in terms of time and energy. This can further delay the task response time.

Since clients are very sensitive to the server performance. Delayed response to users will have negative effects for a hosting company including client frustration and revenue loss. Therefore the job response time must be considered. Because the power saving and the performance cannot be ensured at the same time, there is a trade off between the power saving of a server and the job response time. Reducing the power consumption while maintaining the response time constraint has been an important problem in the server system design.

In this chapter, PowerSleep, a smart power saving scheme, is introduced. PowerSleep can minimize the power consumption of a single server under the mean job response time constraint. PowerSleep adopts the extended M/G/1/PS queuing model for job arrival and job processing. It uses both DVFS and DPM techniques to reduce the power consumption of a server.

Since mode transitions between the running mode and the sleep mode introduce a net timing overhead for a server, they degrade the performance in terms of the mean response time of jobs. When the server utilization is low, the mode transition incurs little overhead. The higher the server utilization is, the bigger the mode transition overhead is. Therefore, it is necessary in the design of the sleep mode to reduce the mode transition overhead as much as possible.

To overcome the mode transition overhead, PowerSleep adds a procrastination sleep period when a new job arrives while the sever is still in the sleep mode. The server does not start to work immediately. Instead, it keeps sleeping during the procrastination sleep period to collect more jobs into the queue. After the procrastination sleeping time, the sever wakes up to process jobs. This approach can decrease the mode transition frequency and thus reduce the mode transition overhead, but it increases the job response time.

3.2 System Model

Only a single server is considered in this system model. r is defined as the working speed ratio of the server to its maximum speed, and r_l is defined as the lowest speed ratio when no job is executed, $r_l \leq r \leq 1$. When r is greater than r_l, the server is then in the running model. When no job is executed in the sever, $r = r_l$, the server is in the idle mode. To save power consumed, DMP can put a server into the sleep mode. If a job arrives, the server is switched to the running mode to process the job. A server is in transition mode during the mode transitioning period.

The power consumption of a server in each mode is as follows:

- *Idle mode*: The server consumes the static power P_I.
- *Running mode*: The power consumption $P_R(r)$ by a server at a speed ratio r is

$$P_R(r) = \alpha(r - r_l)^\gamma + P_I \tag{3.1}$$

  where $\gamma \geq 1$.
- *Sleep mode*: The power consumption by a server is P_S, while $P_S \ll P_I$.
- *Transition mode:* The server consumes the transition power P_T, which is assumed to be equal to the one in the running mode.

The system in this work is based on the M/G/1/PS server model. A job arrives at the server in a Poisson distribution with an arrival rate λ. A job service time follows a generalized distribution with a given mean value $\mathbb{E}(S)$ when executing at the maximum speed. All jobs in the queue are scheduled into a processor by the Processor Sharing (PS) scheduling algorithm. μ is denoted as $\frac{1}{\mathbb{E}(S)}$, $\mu = \frac{1}{\mathbb{E}(S)}$. ρ is denoted as $\frac{\lambda}{\mu}$, $\rho = \frac{\lambda}{\mu}$.

3.3 The Design of the PowerSleep Scheme

The job response time is an important concern in the design of the PowerSleep scheme. Both DVFS and DPM may prolong the job service time. From the time perspective, a server is either in the working state when it processes jobs or in the non-working state when it is set in the idle, sleep, or transition mode. These two states alternate in cycles. A server working cycle is defined from the time when a server changes form the non-working state to the working state to the next time when it changes from the non-working state to the working state.

In the design of PowerSleep, the following constant parameters of time periods are introduced to overcome the above raised concern by utilizing both DVFS and DPM:

- Idle period threshold δ_h is the minimum length of the idle-queue duration before the server is put into the sleep power mode.
- Sleep period threshold δ_e is the maximum length of the period during which the server can stay in the sleep power mode continuously.
- Procrastination sleep period δ_x, if the job arrives earlier before the expiration of δ_e, the server will be procrastinated in the sleep mode for δ_x period so that jobs can be batched together to reduce the short idle periods of the job queue.
- Mode transition period δ_s and δ_w are the transition time from the idle mode to the sleep mode and from the sleep mode to the idle mode respectively.

Time periods δ_h, δ_e, and δ_x are defined constant parameters in PowerSleep.

With the above pre-determined constant parameters of time periods $(\delta_h, \delta_e, \delta_x)$, and of speed ratio r, PowerSleep can be described as the following steps:

1. Once the queue is empty, the server intends to hold on in the idle power mode for δ_h time unit;
2. If a new job arrives before the expiration of the δ_h time unit, the server will immediately serve the new job. Otherwise, the server will be set to the sleep power mode and then it will be enforced to stay in the sleep power mode for δ_e time unit;
3. If a new job arrives before the expiration of the δ_e time unit, the server will remain in the sleep power mode for a procrastination sleep period δ_x time unit (counted from the arrival of the new job). In other words, the job will wait for δ_e time units. Otherwise, the server will be put in the idle power mode again until a new job arrives;

4. Once the server is in the running power mode, the server runs at a constant speed ratio r in the running power mode serving jobs in the queue until the queue become empty;

5. Once the queue is empty, repeat Step 1.

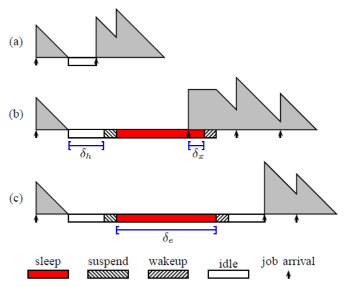

Figure 3.1: Scenarios for PowerSleep

δ_i is defined as the length of a preceding idle-queue duration waiting for a job arrival. Figure 3.1 illustrates the change of the power mode with PowerSleep under different scenarios, where the X-axis is the time line and the Y-axis is the workload in the queue. (a) If $\delta_i \leq \delta_h$, when a new job arrives, the server will immediately serve the new job; (b) If $\delta_h \leq \delta_i \leq \delta_h + \delta_s + \delta_e$, when a new job arrives, the server will remain in the sleep power mode for extra δ_x time units, and then wake up to serve; (c) If $\delta_i \geq \delta_h + \delta_s + \delta_e$, when the server is waken up, the server will stay in the idle power mode until a new job arrives.

3.4 Power Consumption and Response Time Analysis

Under PowerSleep, when a job arrives at an empty queue, it cannot be served immediately; rather the server requires an additional amount of time δ_x (called a starter) to start from non-working state to working state to serve the new first job. Jobs which arrive to a server in working will join the queue and be served in turn as in a simple queuing system. Starter under PowerSleep includes the wake-up transition plus the procrastination sleep period and may also include the remaining portion of a suspend transition. *Queue with Starter* model [78] is adopted to here used to analyze PowerSleep model with a starter.

The server has different power consumptions in different modes, the probabilities of these modes at which server works can be obtained by the Queue with Starter model. In an M/G/1 server under PowerSleep with Starter T_X, a job arrival rate λ, and a generalized service time distribution with a given mean value $\mathbb{E}(S)$, the probabilities of these modes at which server works are :

$$\pi_R = \lambda\mathbb{E}(S) \tag{3.2}$$

$$\pi_T = (1 - \lambda\mathbb{E}(S))\frac{\lambda(\delta_s + \delta_w)e^{-\lambda\delta_h}}{1 + \lambda\mathbb{E}(T_x)} \tag{3.3}$$

$$\pi_S = (1 - \lambda\mathbb{E}(S))\frac{e^{-\lambda\delta_h}\lambda(\lambda\delta_x + e^{-\lambda\delta_s}(1 - e^{-\lambda\delta_e}))}{1 + \lambda\mathbb{E}(T_x)} \tag{3.4}$$

$$\pi_I = 1 - \pi_R - \pi_T - \pi_S \tag{3.5}$$

where π_R, π_T, π_S, and π_I are defined as probabilities of running mode, transition mode, sleep mode and idle mode respectively.

With the probabilities of the power modes, in an M/G/1 server under PowerSleep, the mean power consumption of a server is:

$$\mathbb{E}(P) = P_R(r)\pi_R + P_T\pi_T + P_S\pi_S + P_I\pi_I \tag{3.6}$$

The response time under PowerSleep must be considered. It is shown in [78] that the additional delay in a queue introduced by a starter is independent of the response time in the system without starters. Using this independence property, it is then easy to calculate the total response time in the system with starters: it is simply the sum of the response time in the queue without starters plus the additional delay R_X introduced by starter. By the traditional queue theory, the mean response time of a job in a server without starters is $\frac{\mathbb{E}(S)}{1-\lambda\mathbb{E}(S)}$. By [78], in a system with a job arrival rate λ

and starter, the mean additional delay introduced by Starter is

$$\mathbb{E}(R_X) = \frac{\mathbb{E}(T_X) + \frac{1}{2}\lambda \mathbb{E}(T_X^2)}{1 + \lambda \mathbb{E}(T_X)} \tag{3.7}$$

In an M/G/1/PS server under PowerSleep with Starter T_X, a job arrival rate λ, and a generalized service time distribution with a given mean value $\mathbb{E}(S)$, the mean response time of a job is

$$\mathbb{E}(R) = \frac{\mathbb{E}(S)}{1 - \lambda \mathbb{E}(S)} + \mathbb{E}(R_X) \tag{3.8}$$

In order to evaluate the performance of power consumption and response time under PowerSleep, $\mathbb{E}(T_X)$ and $\mathbb{E}({T_X}^2)$ must be obtained.

By the definition of a starter in *Queue with Starter*, Starter T_x under PowerSleep includes the wake-up transition plus the procrastination sleep period and may also include the remaining portion of a suspend transition, which depends on the preceding idle-queue period δ_i before a new job arrival. Based three scenarios classified by δ_i, the starter T_X is summarized as:

$$T_X = \begin{cases} \delta_h + \delta - \delta_i \\ \delta_x + \delta_w \\ \delta_h + \delta + \delta_e - \delta_i \\ 0 \end{cases} \tag{3.9}$$

where δ is $\delta = \delta_s + \delta_x + \delta_w$.

The preceding idle-queue period λ_i is the same as the idle period defined in an ordinary M/G/1 model, which follows the exponential distribution with a mean value $\frac{1}{\lambda}$. Therefore, for T_X defined in (3.9), its mean value and variance can be obtained as:

$$\begin{aligned}\mathbb{E}(T_X) &= \int_{\delta_h}^{\delta_h+\delta_s} (\delta_h + \delta - t)e^{-\lambda t}dt + \int_{\delta_h+\delta_s}^{\delta_h+\delta_s+\delta_e} (\delta_x + \delta_w)e^{-\lambda t}dt \\ &+ \int_{\delta_h+\delta_s+\delta_e}^{\delta_h+\delta+\delta_e} (\delta_h + \delta + \delta_e - t)e^{-\lambda t}dt \\ &= e^{-\lambda\delta_h}(\delta - \frac{1}{\lambda} + \frac{1}{\lambda}(e^{-\lambda\delta_s}(1 - e^{-\lambda\delta_e}) + e^{-\lambda(\delta+\delta_e)})) \qquad (3.10)\end{aligned}$$

and

$$\begin{aligned}\mathbb{E}(T_X{}^2) &= \int_{\delta_h}^{\delta_h+\delta_s} (\delta_h + \delta - t)^2 e^{-\lambda t}dt + \int_{\delta_h+\delta_s}^{\delta_h+\delta_s+\delta_e} (\delta_x + \delta_w)^2 e^{-\lambda t}dt \\ &+ \int_{\delta_h+\delta_s+\delta_e}^{\delta_h+\delta+\delta_e} (\delta_h + \delta + \delta_e - t)^2 e^{-\lambda t}dt \\ &= e^{-\lambda\delta_h}((\delta - \frac{1}{\lambda})^2 + \frac{1}{\lambda^2}(1 - 2e^{-\lambda(\delta+\delta_e)}) + \frac{2}{\lambda}(\delta_x + \delta_w - \frac{1}{\lambda})e^{-\lambda\delta_s}(1 - e^{-\lambda\delta_e}))) \qquad (3.11)\end{aligned}$$

Let $\sigma = e^{-\lambda\delta_h} + \lambda\delta - 1 + e^{-\lambda\delta_s}(1 - e^{-\lambda\delta_e}) + e^{-\lambda(\delta+\delta_e)}$, the probabilities defined in equations (3.2) to (3.5) can be written as

$$\pi_R = \frac{\rho}{r} \qquad (3.12)$$

$$\pi_T = (1 - \frac{\rho}{r})\frac{\lambda(\delta_s + \delta_w)}{\sigma} \qquad (3.13)$$

$$\pi_S = (1 - \frac{\rho}{r})\frac{\lambda\delta_x + e^{-\lambda\delta_s}(1 - e^{-\lambda\delta_e})}{\sigma} \qquad (3.14)$$

$$\pi_I = 1 - \pi_R - \pi_T - \pi_S \qquad (3.15)$$

In an M/G/1/PS server under PowerSleep, the mean power consumption is

$$\begin{aligned}\mathbb{E}(P) =&\alpha(r - r_l)^\gamma(\frac{\rho}{r} + (1 - \frac{\rho}{r})\frac{\lambda(\delta_s + \delta_w)}{\sigma}) + P_I \\ &+(P_I - P_S)(1 - \frac{\rho}{r})\frac{\lambda\delta_x + e^{-\lambda\delta_s}(1 - e^{-\lambda\delta_e})}{\sigma} \qquad (3.16)\end{aligned}$$

and the mean response time of a job is

$$\mathbb{E}(R) = \frac{1}{\mu(r-\rho)} + \frac{\frac{\lambda}{2}\delta^2 + (\delta_x + \delta_w)e^{-\lambda\delta_s}(1 - e^{-\lambda\delta_e}))}{\sigma} \tag{3.17}$$

$\hat{R}$ is denoted as mean response time threshold. The optimization problem of minimizing the mean power consumption under a given mean response time constraint can easily be formulated as follows:

$$\text{minimize } \mathbb{E}[P] \tag{3.18a}$$

$$\text{subject to } \mathbb{E}[R] \le \hat{R}, \tag{3.18b}$$

$$\max\{r_l, \rho\} \le r \le 1. \tag{3.18c}$$

The optimization problem defined in (3.18) can in general be solved with a Lagrangian function

$$L = \mathbb{E}[P] + \chi\mathbb{E}[R] \tag{3.19}$$

To resolve this optimization problem, $\frac{\partial L}{\partial r} = 0$, $\frac{\partial L}{\partial \delta_h} = 0$, $\frac{\partial L}{\partial \delta_x} = 0$, $\frac{\partial L}{\partial \delta_e} = 0$ are set. Variables r^*, δ_h^*, δ_x^*, δ_e^* are denoted as the corresponding optimal values.

In an M/G/1/PS server under PowerSleep, the minimal power consumption $\mathbb{E}(P^*)$ under the mean response time threshold $\hat{R}$ can be achieved with an optimal configuration of r^*, δ_h^*, δ_x^*, and δ_e^*.

3.5 Conclusion

This chapter explores how to minimize the mean power consumption in a server under the mean response time constraint for reducing the power cost. PowerSleep, a smart power-saving schemes proposed in this chapter, applies both DVFS and DPM to put the server to a low-power sleep mode. By adopting the extended M/G/1/PS queuing model for job arrival and execution, PowerSleep presents how to jointly decide the execution speed for jobs and the sleep period such that the mean response time constraint is satisfied and the mean power consumption is minimized.

CHAPTER 4

Voltage Island Aware Energy Efficient Scheduling of Real-Time Tasks on Multi-Core Processors

4.1 Introduction

This chapter studies energy efficient scheduling of periodic real-time tasks on multi-core processors with voltage islands, in which cores are partitioned into multiple blocks (termed voltage islands) and each block has its own power source to supply voltage. Cores in the same block always operate at the same voltage level, but can be adjusted by using Dynamic Voltage and Frequency Scaling (DVFS). Algorithm Voltage Island Largest Capacity First (VILCF) is proposed for energy efficient scheduling of periodic real-time tasks on multi-core processors. It achieves better energy efficiency by fully utilizing the remaining capacity of an island before turning on more islands or increasing the voltage level of the current active islands. A detailed theoretical analysis of the approximation ratio of the proposed VILCF algorithm in terms of energy efficiency is provided. In addition, the experimental results show that VILCF significantly outperforms the existing algorithms when there are multiple cores in a voltage island.

4.2 System Models

4.2.1 Power Model

In this study, we adopt a practical and widely accepted power model [33] [34] [43]. Power consumption of a core is composed of two parts, leakage power and dynamic power. We denote dynamic power P_d, which can be adjusted by DVFS. P_d increases as working speed of core increases, we denote the working speed of cores s, the function of dynamic power to s is:

$$P_d(s) = C_{ef} V_{dd}^2 s \tag{4.1}$$

where V_{dd} denotes working voltage of cores, C_{ef} denotes effective switch capacitance, $s = \kappa \frac{(V_{dd}-V_t)^2}{V_{dd}}$, V_t denotes threshold voltage, and κ denotes hardware-design-specific constant, ($V_{dd} \geq V_t \geq 0, \kappa > 0$, and $C_{ef} > 0$).

We assume leakage power is a constant. The total power consumption per core is the sum of leakage power and dynamic power: $P_d(s) + \beta_2$. We denote leakage power β_2. We normalize the power consumption function as :

$$P(s) = s^3 + \beta \tag{4.2}$$

where s^3 represents dynamic power, and β represents leakage power, see [38].

From equation (4.2), we know that $P(s)$ is a convex and non-decreasing function of the core speed s. We draw the curve of this function in figure (4.1), s_{min} and s_{max} denote the lower bound and upper bound of s respectively. From the power to speed curve, we find that there exists a critical speed s_0 at which power consumption is optimal, (in this study, we assume $s_{min} \leq s_0 \leq s_{max}$), when cores run at critical speed, $\frac{P_d(s)}{s}$ is minimum. By solving $\frac{dP(s)}{ds} = 0$, we derive $s_0 = \sqrt[3]{\frac{\beta}{2}}$, see figure (4.1).

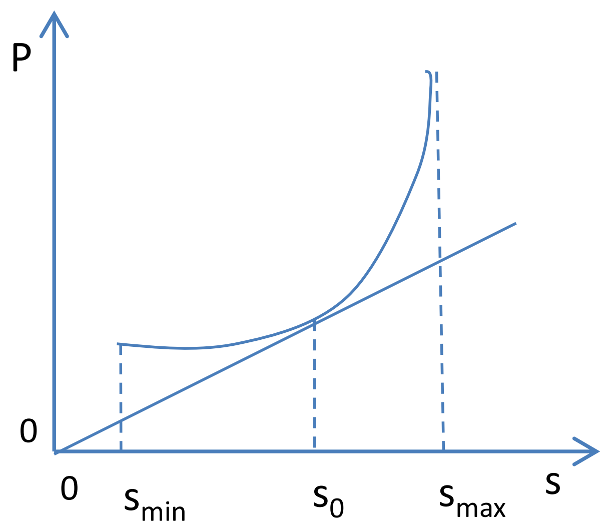

Figure 4.1: Power consumption to speed.

4.2.2 Periodic Real-Time Task Model

We focus on a set of periodic independent real time tasks denoted by $T = \{\tau_i(\phi_i, p_i, c_i), i = 1...n\}$. Each task τ_i contains a sequence of task instances refereed as jobs which arrive periodically. For task τ_i, ϕ_i is the phase which is the release time of the first task instance, p_i is the period, and c_i is computation time. We define the ***size*** of task τ_i as $\frac{c_i}{p_i}$. For two tasks τ_i and τ_j, we say task τ_i is greater than τ_j when we have $\frac{c_i}{p_i} > \frac{c_j}{p_j}$, and task τ_i is less than τ_j when we have $\frac{c_i}{p_i} < \frac{c_j}{p_j}$. We assume the deadline of each task is the same as the period p_i. Further, we assume $\phi_i = 0$, and all tasks arrive at time 0.

Given a set T of tasks, the ***hyper-period*** of T, denoted as L, is defined as the least common multiple (LCM) of the periods of tasks in T. For a given core i with a set of n assigned periodic tasks, the processor utilization factor U is the fraction of processor time spent in the execution of the task set $\frac{c_i}{p_i}$, utilization factor for n tasks at this core is given by:

$$U = \sum_i \frac{c_i}{p_i} \tag{4.3}$$

If no task is assigned to a core, we say the core is empty.

For uniprocessor independent real time tasks scheduling, the earliest-deadline first (EDF) scheduling algorithm is an optimal scheduling algorithm on preemptive uniprocessors. To meet time constraints of all tasks at one core, the working speed of the core needs to be faster than U, $U \leq s$.

4.2.3 Multi-Core Processors with Voltage Islands

As to Chip Multi-core Processors of voltage island model, we assume all cores are homogeneous. Cores are partitioned into M blocks, ($M \geq 2$), and each block contains the same number of cores, denoted as K ($K \geq 1$). Each block has only one supply voltage line. Cores of the same block are set to the same voltage V_{dd}, and consume the same power. We can adjust voltage on the block level dynamically using DVFS, but different blocks can work under different voltage. We also apply DPM technique to turn on/off blocks. If no task is assigned to a block, the block can be turned off and put into dormant mode. When new task comes, the block can be reactivated. Activating block from dormant mode to active mode needs additional energy E_{sw} and time overhead. In this study, to simplify our model, we do not consider this overhead and let E_{sw} be equal to 0. DVFS + DPM can reduce the power consumption of the cores and make them work energy-efficiently.

We denote (m, k) with $0 \leq m \leq M - 1$ and $0 \leq k \leq K - 1)$ as core k of block m. We let ***index*** of core (m, k) to be $m * K + k$. The index of core $(0, 0)$ is the smallest of all cores, while the index of core $(M - 1, K - 1)$ is the largest. We denote $T_{m,k}$ as a set of tasks assigned to core (m, k), and denote $l_{m,k}$ as the ***load*** of core (m, k) where $l_{m,k}$ is equal to $\sum_{\tau_i \in T_{m,k}} \frac{c_i}{p_i}$. We let l_m^{max} be $\max_k(l_{m,k})$ of block m, and denote l_m^{max} as ***limit*** of block m. We define ***critical core*** of block m as the core of this block whose load is l_m^{max}. The working speed and the voltage of block m are determined by the load of the critical core of that block. We denote $t_{m,k}$ as ***capacity*** of core (m, k) while $t_{m,k}$ is equal to $l_m^{max} - l_{m,k}$, the capacities of all critical cores are 0.

4.3 Problem Definition and An Approximation Algorithm

We are interested in finding an algorithm to schedule a set of periodic and independent real time tasks to a CMP with voltage islands while making the power consumption minimum under time constraint.

4.3.1 Voltage Island Energy-Efficiency Scheduling (VIEES)

Definition 1. *Given a set T of real time tasks over CMP with M blocks and K identical cores in each block, the power function of each core is $P(s) = s^3 + \beta$, where $\beta > 0$, and all tasks are ready at time 0. Each periodic task $\tau_i \in T$ is associated with a computation requirement in c_i CPU-cycles and a period p_i, where the relative deadline of τ_i is p_i. The cores can work on any speed in $[S_{min}, S_{max}]$. The problem is to minimize the energy consumption in the hyper-period L of tasks in T in the scheduling of tasks in T without missing the timing constraint, where each task is executed entirely on a core.*

Lemma 1. *VIEES is an NP hard problem.*

Proof. VIEES is an optimization problem. Since decision is no harder than optimization, we prove VIEES is NP hard if its corresponding decision problem is NP hard. A corresponding decision problem of VIEES is to decide whether a set of tasks can be assigned to cores such that the sum of task sizes of each core is equal.

The NP-hard of the decision problem is proved by a reduction from the 3-PARTITION problem, which is NP-complete [79, 80]. 3-PARTITION problem is expressed like this: Given n numbers $a_1, ..., a_n \in S$, it is to decide if there are sets S_1, S_2, S_3 with $S_1, S_2, S_3 \subset S$ and $S_1 \cap S_2 = \phi, S_1 \cap S_3 = \phi, S_2 \cap S_3 = \phi$ and $S_1 \cup S_2 \cup S_3 = S$ such that $\sum_{a_i \in S_1} a_i = \sum_{a_j \in S_2} a_j = \sum_{a_k \in S_3} a_k$. Given a 3-Partition instance, an instance is created for VIEES problem by setting task size $\frac{c_i}{p_i} = \frac{a_i}{\sum_{j=1}^{n} a_j} \in (0, 1]$. The workload of each core can be perfectly balanced (sum of task sizes of each core is equal) if and only if the set S can be partitioned into three disjoint sets of equal sums.

□

4.3.2 Energy Consumption

Given a set of real time independent tasks T, we partition them into task subsets $T_{0,0}, T_{0,1}, ..., T_{M-1,K-1}$, and assign $T_{m,k}$ to core (m, k) for all $0 \leq m \leq M-1, 0 \leq k \leq K-1$, these task subsets are disjoint, $T_{m,k} \cap T_{i,j} = \emptyset$ where $i \neq m$ or $j \neq k$, then we assign working speeds to blocks, we call this assigned task subsets and working speeds of blocks a ***Schedule*** SC. A schedule is feasible if all core speeds assigned for its time intervals are valid, no task misses its timing constraint, and each task is executed entirely on a core. The energy consumption of a schedule SC is denoted as $\Phi(SC)$. A schedule is optimal if it is feasible, and its energy consumption is minimal of all feasible schedules.

Due to convexity of power consumption function, for every task $\tau_i \in T_{m,k}$, the minimum energy consumption schedule would execute all of the tasks at s_0 if $l_{m,k} \leq s_0$, or at $\sum_{\tau_i \in T_{m,k}} \frac{c_i}{p_i}$ if $l_{m,k} \geq s_0$, see [38]. A core would turn into the dormant mode when it completes all jobs and becomes idle.

Considering the minimum energy consumption of tasks $\tau_i \in T_{m,k}$ executed on core (m, k) in the hyper-period L, if $l_{m,k} \leq s_0$, the minimum energy consumption is $\frac{L}{s_0} \cdot l_{m,k} \cdot P(s_0)$, otherwise, the minimum energy consumption is $L \cdot P(l_{m,k})$. We define energy consumption function $\psi(l)$ for tasks in $T_{m,k}$ completed on the core (m, k) during time L with load $l = l_{m,k}$ as:

$$\psi(l) = \begin{cases} L(l^3 + \beta), & \text{if} \quad l > s_0 \\ L\frac{l}{s_0}({s_0}^3 + \beta), & otherwise \end{cases} \tag{4.4}$$

For schedule SC, $\Phi(SC)$ is equal to $\sum_{m=0}^{M-1} K * \psi(l_{m,0})$ because in Voltage Island model, $\psi(l_{m,k})$ is equal to $\psi(l_{m,0})$ for $1 \leq k \leq K-1$.

The following Lemmas resulting from the convexity of the power consumption function will be widely used for algorithmic analysis in this study [38].

Lemma 2. $\psi(\gamma x+(1-\gamma)y) \leq \gamma\psi(x)+(1-\gamma)\psi(y)$, *for any non-negative reals x, y and* $0 \leq \gamma \leq 1$.

Lemma 3. *Suppose that* $l_m + l_n = l'_m + l'_n$ *, and* $l_m \leq l'_m, l'_n \leq l_n$*, then* $\psi(l'_m) + \psi(l'_n) \leq$

$\psi(l_m) + \psi(l_n)$.

4.3.3 Voltage Island Largest Capacity First (VILCF) Algorithm

Because VIEES is NP hard, we explore heuristic scheduling algorithm. In this study, we propose a task scheduling algorithm named Voltage Island Largest Capacity First (VILCF) for the VIEES problem.

Algorithm 1 VILCF

```
Input: $T, M, K$
Output: $T_{0,0}, T_{0,1}, ..., T_{M-1,K-1}$
1: sort all tasks in $T$ in non-increasing order $\frac{c_i}{p_i}$ for $\tau_i \in T$
2: set $l_{0,0}, l_{0,1}, ..., l_{M-1,K-1}$ to 0
3: set $t_{0,0}, t_{0,1}, ..., t_{M-1,K-1}$ to 0
4: set $T_{0,0}, T_{0,1}, ..., T_{M-1,K-1}$to $\phi$
5: for $i = 1$ to $|T|$ do
6:     find the largest $t_{m,k}$;(break tie by index of core)
7:     if $\frac{c_i}{p_i} \leq t_{m,k}$
8:         $T_{m,k} \leftarrow T_{m,k} \cup \tau_i$ and $l_{m,k} \leftarrow l_{m,k} + \frac{c_i}{p_i}$
9:     else find the smallest $l_{m,k}$; (break tie by index of core)
10:        $T_{m,k} \leftarrow T_{m,k} \cup \tau_i$ and $l_{m,k} \leftarrow l_{m,k} + \frac{c_i}{p_i}$
11:   sort $T_{m,k}$ in non-increasing order by $l_{m,k}$,map the task subsets to cores by index, and $t_{m,k} \leftarrow l_m^{max} - l_{m,k}$
12: end for
```

With VILCF, initially the capacities and loads of all cores are set to 0, and task subsets assigned to the cores are set to ϕ, $(t_{m,k} = 0, l_{m,k} = 0, T_{m,k} = \phi, m = 0, ..., M-1; k = 0, ..., K-1)$, and tasks in T are sorted in non-increasing order by their size $\frac{c_i}{p_i}$. We get the largest task, and compare it to the largest capacity of all cores, if the task is no greater than the largest capacity, then this task is assigned to the core with the largest capacity; otherwise, the task is assigned to the core with the minimum load. After this task scheduling, we sort the assigned task subsets by their loads in non-increasing order and map them to the cores by index; then recompute the capacities of all cores, and continue scheduling next task. We observe that core $(m, 0)$ is the critical core of block m for $0 \leq m \leq M-1$, and $l_{m,0}$ is the limit of block m, ($l_m^{max} = l_{m,0}$), because the index of core $(m, 0)$ is the smallest of block m. The time complexity of VILCF is $O(|T| \log |T|)$.

For example, given a chip with 8 cores partitioned into two blocks and four cores in each block, we schedule a set of tasks T, $T = [6, 5, 5, 4, 3, 3, 2, 2, 2, 2]$ into these cores. Initially, $l_{m,k} = 0, t_{m,k} = 0, T_{m,k} = \phi, m = [0, 1], k = [0, 1, 2, 3]$. task $\frac{c_0}{p_0} = 6$ is assigned to core $(0, 0)$ because the task is greater than all capacities, and the index and load of core $(0, 0)$ are both the smallest; after this scheduling, core $(0, 0)$ becomes the critical core of block 0, and the limit of block 0 l_0^{max} is 6, the capacities of block 0 are updated to $t_{0,0} = 0, t_{0,1} = t_{0,2} = t_{0,3} = 6$; then task $\frac{c_1}{p_1}$ is assigned to core $(0, 1)$, because its capacity is 6 and index is smallest, $t_{0,1}$ is changed to 1; and tasks $\frac{c_2}{p_2} = 5$ and $\frac{c_3}{p_3} = 4$ are assigned to core $(0, 2)$ and $(0, 3)$ respectively. As to task $\frac{c_4}{p_4} = 3$, it is greater than all capacities, so we select core $(1, 0)$ with minimum load and smallest index; task $\frac{c_5}{p_5} = 3$ is assigned to core $(1, 1)$. The capacity of core $(0, 3)$ is 2 which is the largest, so we schedule task $\frac{c_6}{p_6} = 2$ into core $(0, 3)$; task $\frac{c_7}{p_7} = 2$, $\frac{c_8}{p_8} = 2$ are assigned to core $(1, 2)$ and $(1, 3)$; for task $\frac{c_9}{p_9} = 2$, it is greater than capacities of all cores, so this task is assigned to core $(1, 3)$ whose load is minimum. After finishing task scheduling, we sort the loads of task subsets and map the task subsets to the cores by index, so we obtain $l_{0,0} = 6, l_{0,1} = 6, l_{0,2} = 5, l_{0,3} = 5, l_{1,0} = 4, l_{1,1} = 3, l_{1,2} = 3, l_{1,3} = 2$.

4.4 Lower Bound of Energy Consumption of the VIEES Problem

In this section, we derive the lower bound of the energy consumption of VIEES problem which is used to show the approximation ratio of Algorithm VILCF. The approximation ratio can be obtained by comparing $\Phi(SC_{VILCF})$ to the lower bound of optimal solutions for all input instances.

4.4.1 Semi-VIEES Problem

We assume the tasks in set T to be sorted in a non-increasing order by task sizes, $\frac{c_i}{p_i} \geq \frac{c_{i+1}}{p_{i+1}}$. Let k^+ be the largest index satisfying that after scheduling this task, there is no core which is empty, let T^+ be the set of the first k^+ tasks of T.

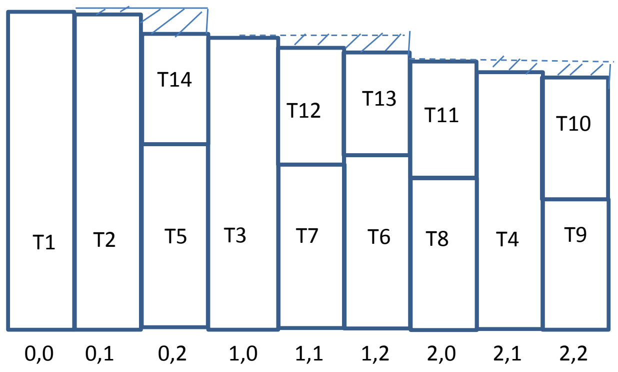

Figure 4.2: Optimal Schedule of the Semi-VIEES problem if $\sum_{m=0}^{M-1}\sum_{k=0}^{K-1}(l_m^{max} - l'_{m,k}) \leq \sum_{\tau_i \in T \setminus T^f} \frac{c_i}{p_i}$, $M = 3, K = 3$.

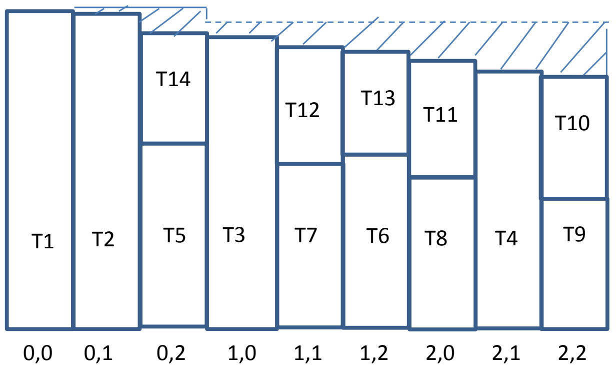

Figure 4.3: Optimal Schedule of the Semi-VIEES problem if $\sum_{m=0}^{M-1}\sum_{k=0}^{K-1}(l_m^{max} - l'_{m,k}) > \sum_{\tau_i \in T \setminus T^f} \frac{c_i}{p_i}$, $M = 3, K = 3$.

We define a task as a ***completing task*** when we schedule this task, it is no greater than the largest capacity and the core which it is assigned to is not empty.

Lemma 4. *Scheduling a completing task does not change the task subsets assigned to the critical cores by Algorithm VILCF.*

Proof. Suppose task τ_i is assigned to core (m, k) as a completing task, we know that $\frac{c_i}{p_i} \leq l_m^{max} - l_{m,k}$ we obtain $\frac{c_i}{p_i} + l_{m,k} \leq l_{m,0}$ while $\frac{c_i}{p_i} + l_{m,k}$ is the load of new changed task subsets, since it does not exceed the load of the critical core of the same block, scheduling of task τ_i does not change task subset assigned to critical cores. □

Lemma 5. *Scheduling tasks in T^+ makes $|T_{m,0}|$ to be equal to 1,(m=0,1,...,M-1).*

Proof. There always exists an empty core before task with index k^+ is assigned by Algorithm VILCF. When task τ_i is assigned to block m with no task, this task would be assigned to core $(m, 0)$ which becomes critical core of block m after this scheduling, the capacity of core $(m, 0)$ is 0, the capacities of all other cores of block m are updated to be greater than 0. If a new task comes, it would be assigned to one core with largest capacity, if the new task is a completing task, by Lemma 4 no task subset of critical cores is changed; otherwise, the new coming task is assigned to an empty core because its load is 0 which is the minimum, so no more task would be assigned to the critical cores which already have one task assigned. □

Let k^* be the largest index satisfying that $\frac{c^*}{p^*}$ is not a completing task, and is assigned to a core which has only one task τ' with $\frac{c^*}{p^*} \geq \frac{1}{2}\frac{c'}{p'}$. Let T^f be the set of the first k^* tasks of T, denote $T \backslash T^f$ the set of remained tasks in T after subtracting T^f. We observe k^+ should be no larger than k^* ($k^+ \leq k^*$), denote $T^f \backslash T^+$ the set of remained tasks in T^f after subtracting T^+.

Lemma 6. *Scheduling tasks in T^f is an optimal solution by Algorithm VILCF.*

Proof. By Lemma 5, critical cores has only one task assigned after scheduling tasks in T^+. We continue to schedule a task $\tau_i \in T^f \backslash T^+$ into a core which contains only one task τ_j, making the new assigned task subset to have two tasks.

We find the load of the new assigned task subset is no greater than that of any core which already has completing tasks assigned to. Suppose core (m, k) has completing tasks, there is one task $\tau_k \in T_{m,k}$ satisfying $\frac{c_k}{p_k} \geq \frac{c_j}{p_j}$ because of the largest task first scheduling strategy of Algorithm VILCF. τ_i is no greater than any other tasks in $T_{m,k}$ since τ_i is the smallest of all assigned tasks, and core (m, k) has at least two tasks, so load $\frac{c_i}{p_i} + \frac{c_j}{p_j}$ is no greater than that of core (m, k).

From above description, scheduling a task in $\tau_i \in T^f \backslash T^+$ does not change the index of task subset which already has completing tasks, and it does not change the task subset of critical core which is in the same block, either, so no task subset which has completing tasks is assigned to a critical core after sorting step in Algorithm VILCF. Hence, after scheduling tasks in $T^f \backslash T^+$, the critical cores contain at most two tasks, and for any critical cores which has two tasks, one task must come from $T^f \backslash T^+$ which is greater than half of the other.

We can not swap tasks between critical core $(m, 0)$ and non-critical core (m', k'), $(k' \neq 0, 0 \leq m, m' \leq M - 1)$. Assume core $(m, 0)$ has two tasks τ_i, τ_j with $\frac{c_i}{p_i} \geq \frac{c_j}{p_j}$, if we swap τ_i for any task in core (m', k'), this swapping increases $l_{m',k'}$ to be greater than $l_{m',0}$ since τ_i is not a completing task; if we swap τ_j, this swapping increases $l_{m,0}$ since τ_j is less than any task in core (m', k').

Only critical cores are examined here. For any three tasks τ_i, τ_j, τ_k assigned to critical cores which have two tasks, this equation holds.

$$\frac{c_i}{p_i} \leq \frac{c_j}{p_j} + \frac{c_k}{p_k} \tag{4.5}$$

We could transform a schedule SC into another schedule which assigns at most two tasks in T^f on a critical core without increasing energy consumption. Suppose in a schedule SC, three tasks τ_i, τ_j, τ_k are assigned to core $(m, 0)$, we can have a core $(m', 0)$ who is assigned to two tasks by Algorithm VILCF has only one task τ_l, for all these tasks, Equation (4.5) holds. We can move one task τ_i from core $(m, 0)$ into core $(m', 0)$, to create a new schedule SC', let $T^{SC'}_{m,0}$ to be $T^{SC}_{m,0} \backslash \{\tau_i\}$, and $T^{SC'}_{m',0}$ to be $T^{SC}_{m',0} \bigcup \{\tau_i\}$, by Equation (4.5), we know $l^{SC}_{m',0} < l^{SC'}_{m',0}$ and $l^{SC}_{m,0} > l^{SC'}_{m,0}$, then we get $\psi(l^{SC}_{m',0}) + \psi(l^{SC}_{m,0}) > \psi(l^{SC'}_{m',0}) + \psi(l^{SC'}_{m,0})$ by Lemma 3. $\square$

We first schedule tasks in T^f by applying Algorithm VILCF, let $l'_{m,k}$ denote load of core (m, k) under this scheduling. Note that if $T \backslash T^f \neq \emptyset$. We relax the constraint of the VIEES problem by allowing task migration and simultaneous execution on multiple cores for the tasks in $T \backslash T^f$, such that tasks could be executed on more than one cores simultaneously. We define such a relaxed problem as ***SEMI-VIEES*** problem.

4.4.2 Optimal Solution of Semi-VIEES Problem

In voltage Island model, we know that, for core(m, k) with $t_{m,k} > 0$, $(0 \leq m \leq M-1$, $0 \leq k \leq K-1)$, we can add load δ to it without increasing energy consumption if $l_{m,k} + \delta \leq l_m^{max}$. The total load which can be added to the cores of bock m is $\sum_{k=0}^{K-1}(l_m^{max} - l'_{m,k})$, see figure (4.2), the shaded area in block m in figure (4.2) represents $\sum_{k=0}^{K-1}(l_m^{max} - l'_{m,k})$.

Here let us show an optimal schedule SC^* of the SEMI-VIEES problem. Let $\Gamma = \sum_{\tau_i \in T \backslash T^f} \frac{c_i}{p_i}$, and let λ and λ_m for $0 \leq m \leq M-1$ be the positive values that satisfies:

$$\text{Minimize } \lambda \tag{4.6a}$$

$$\text{Maximize } \lambda_m \tag{4.6b}$$

$$\text{Subject to } \sum_{m=0}^{M-1} \sum_K \max\{\lambda - l'_{m,k}, \max\{\lambda_m - l'_{m,k}, 0\}\} = \Gamma \tag{4.6c}$$

$$\lambda_m \leq l'_{m,0}, m = 0, ..., M-1 \tag{4.6d}$$

In SEMI-VIEES problem, since task migration and simultaneous execution of a task on multiple cores are allowed for the tasks in $T \backslash T^f$, we can distribute the computation of these tasks among the cores in two steps. In the first step, for core (m, k) in the $m - th$ block with $t_{m,k} \geq 0$ and $0 \leq m \leq M-1$, if $\lambda_m \geq l'_{m,k}$, we distribute load $\lambda_m - l'_{m,k}$ of the tasks in $T \backslash T^f$ to core (m, k), and the resulting $l'_{m,k}$ is equal to λ_m. $l'_{m,k}$ is either greater than or equal to λ_m, we know this load distribution does not increase energy consumption if $\lambda_m \leq l'_{m,0}$, we hope to distribute load of tasks in $T \backslash T^f$ as much as possible to cores in this step, but by the constraint of $\lambda_m \leq l'_{m,0}$, it is possible that there is still remained load of tasks in $T \backslash T^f$ after subtracting the load distribution of the first step. In the second step, we distribute remained load of tasks in $T \backslash T^f$ to core (m, k) while $0 \leq m \leq M-1$, $0 \leq k \leq K-1$ and, if $\lambda > l'_{m,k}$, we distribute load $\lambda - l'_{m,k}$ of the tasks in $T \backslash T^f$ to core (m, k), and the resulting $l'_{m,k}$ is equal to λ, for core (m, k), $l'_{m,k}$ is either greater than or equal to λ.

Let SC^* denote the resulting schedule, where $\Phi(SC^*) = \sum_{m=0}^{M-1} K * \psi(l'_{m,0})$. We now prove the optimality of SC^* and the relation of the loads in SC^* and SC_{VILCF}.

We see two cases of optimal schedule of the Semi-VIEES problem with $M = 3, K = 3$, in case 1 (see figure 2), $\sum_{m=0}^{M-1} \sum_{k=0}^{K-1} (l_m^{max} - l'_{m,k})$ is no greater than $\sum_{\tau_i \in T \backslash T^f} \frac{c_i}{p_i}$; in case2 (see figure 3), $\sum_{m=0}^{M-1} \sum_{k=0}^{K-1} (l_m^{max} - l'_{m,k})$ is greater than $\sum_{\tau_i \in T \backslash T^f} \frac{c_i}{p_i}$.

Lemma 7. *SC^* is optimal schedule for the Semi-VIEES problem.*

Proof. Let SC be any feasible schedule of the Semi-VIEES problem, consider two critical cores core $(m, 0)$, and $(m', 0)$, let $T_{m,0}^{SC}$, $l_{m,0}^{SC}$ be set of task of T^f and load assigned on core $(m, 0)$ in SC respectively. Assume $\sum_{\tau_i \in T_{m,0}^{SC}} \frac{c_i}{p_i} > \sum_{\tau_i \in T_{m',0}^{SC}} \frac{c_i}{p_i}$ and $T_{m,0}^{SC} \cup T_{m',0}^{SC} = T_{m,0}^{SC^*} \cup T_{m',0}^{SC^*}$, we find $\sum_{\tau_i \in T_{m,0}^{SC^*}} \frac{c_i}{p_i}$ and $\sum_{\tau_i \in T_{m',0}^{SC^*}} \frac{c_i}{p_i}$ are between $\sum_{\tau_i \in T_{m,0}^{SC}} \frac{c_i}{p_i}$ and $\sum_{\tau_i \in T_{m',0}^{SC}} \frac{c_i}{p_i}$ since scheduling first k^* tasks is an optimal schedule in SC^*.

Let $l_{m+m'} = l_{m,0}^{SC} + l_{m',0}^{SC} - \sum_{\tau_i \in T_{m,0}^{SC} \cup T_{m',0}^{SC}} \frac{c_i}{p_i}$, let $\lambda'_{m+m'}$ be the value satisfies $(\max(\lambda'_{m+m'} - \sum_{\tau_i \in T_{m,0}^{SC^*}} \frac{c_i}{p_i}, 0) + \max(\lambda'_{m+m'} - \sum_{\tau_i \in T_{m',0}^{SC^*}} \frac{c_i}{p_i}, 0)) = l_{m+m'}$, $l_{m,0}^{SC^*}$ and $l_{m',0}^{SC^*}$ are $\max(\lambda'_{m+m'}, \sum_{\tau_i \in T_{m,0}^{SC^*}} \frac{c_i}{p_i})$ and $\max(\lambda'_{m+m'}, \sum_{\tau_i \in T_{m',0}^{SC^*}} \frac{c_i}{p_i})$ respectively. Both $l_{m,0}^{SC^*}$ and $l_{m',0}^{SC^*}$ are between $l_{m,0}^{SC}$ and $l_{m',0}^{SC}$, by Lemma 3 and equation (5), $\psi(l_{m,0}^{SC^*}) + \psi(l_{m,0}^{SC^*}) \leq \psi(l_{m,0}^{SC}) + \psi(l_{m,0}^{SC})$, we transform SC into SC^* without increasing energy consumption. $\square$

For critical cores in both schedules SC_{VILCF} and SC^*, it could be likelihood that $l_{m,0}$ is equal to $l'_{m,0}$ for $0 \leq m \leq M-1$, if it is not, let $\hat{M}$ be the smallest index of block satisfying $l'_{\hat{M},0} \neq l_{\hat{M},0}$.

Lemma 8. *$l_{\hat{M},0}$ is at most $\frac{3}{2} l_{min}$ where l_{min} is the minimum load of the $M * K$ cores derived from Algorithm VILCF.*

Proof. Consider core $(\hat{M},0)$, since $l_{\hat{M},0} \neq l'_{\hat{M},0}$, $T_{\hat{M},0}$ assigned to critical core $(\hat{M},0)$ changes after scheduling task in $T\backslash T^f$ by Algorithm VILCF, so there is at least one non-completing task in $T\backslash T^f$ assigned to a core whose index is no larger than $\hat{M}$ by Lemma 4, the load of this core is no less than $l_{\hat{M},0}$.

Consider two cores $(M-1,K-2)$ and $(M-1,K-1)$ with the least loads l_{min} and l'_{min}, $(l'_{min} \leq l_{min})$. A non-completing task $\tau', \tau' \in T\backslash T^f$ is assigned to core $(M-1,K-1)$ by Algorithm VILCF, the load of core $(M-1,K-1)$ is $l'_{min}+\frac{c'}{p'}$, after sorting step in Algorithm VILCF, the index of this task subset is no larger than $\hat{M}$. We know $\frac{c'}{p'} \leq \frac{1}{2}l'_{min}$ since τ' is not in T^f, we obtain $\frac{l'_{min}+\frac{c'}{p'}}{l_{min}} \leq 1+\frac{\frac{c'}{p'}}{l_{min}} \leq \frac{3}{2}$. Since $l_{\hat{M},0}$ is no greater than $l'_{min}+\frac{c'}{p'}$, so we obtain $l_{\hat{M},0} \leq \frac{3}{2}l_{min}$. □

4.5 Approximation Ratio of Algorithm VILCF

We have derived the lower bound of the minimum energy consumption. We now show the approximation ratio of Algorithm VILCF. The approximation ratio can be obtained by comparing $\Phi(SC_{VILCF})$ to the lower bound of optimal solutions for all input instances. First we figure out the worst energy consumption by algorithm VILCF.

Figure 4.4: Upper bound of energy consumption of the VIEES problem, $M=3, K=3$.

Lemma 9. *In a schedule of upper bound of energy consumption of the VIEES problem, $l_{m,k}$ is equal to $l_{m+1,0}$ for $0 \leq m \leq M-2; 1 \leq k \leq K-1$.*

Proof. Given two cores $(m,k),(m+1,0), k \neq 0$, we have $l_{m,i} > l_{m+1,0}$, note that the two cores are in different blocks. Let $\delta = l_{m,k} - l_{m+1,0}$, load δ does not increase energy consumption in block m while more load is processed, so energy efficiency is better when $l_{m,k}$ is greater than $l_{m+1,0}$ than when they are equal, see figure (4.4). □

Lemma 10. *Suppose $f(y) = \frac{(\mu*\psi(2y)+(\hat{M}-\mu)\psi(3y))}{\hat{M}*\psi(\frac{\Lambda}{K*\hat{M}})}$ for positive numbers $\hat{M} \geq 3$ and Λ, and a non-negative number μ, where $0 \leq y, 0 \leq \mu \leq \hat{M}$ and $\Lambda = K*(\mu*2y+(\hat{M}-\mu)*3y)-(K-1)*y$,* *then* $f(y) \leq \frac{54\hat{M}^3-18\hat{M}(2\delta*\epsilon-3M\delta)+2(9\hat{M}^2+2\delta*\epsilon)^{\frac{3}{2}}}{27\hat{M}\delta^2}$ *where* $\delta = 2\hat{M}-1$, *and* $\epsilon = 3\hat{M}-1$.

Proof. We have to consider three cases: (1) $3y < s_0$, (2) $2y \leq s_0 \leq 3y$, and (3) $s_0 < 2y$. For the first case, the numerator and the denominator of $f(y)$ are the same, hence, $f(y) = 1$.

For the second case, we have two sub cases $s_0 \leq \frac{\Lambda}{K*\hat{M}}$ and $s_0 > \frac{\Lambda}{K*\hat{M}}$. By definition $\mu = \frac{3y\hat{M}-\frac{\Lambda+(K-1)y}{K}}{y}$, and then $\mu \geq \frac{(3\hat{M}-1)*y-\frac{\Lambda}{K}}{y}$.

For the first sub case, we have $f(y) \leq \frac{K*(\frac{\frac{\Lambda}{K}+(1-2\hat{M})y}{y}*((3y)^3+\beta)+(\frac{(3\hat{M}-1)y-\frac{\Lambda}{K}}{y})\frac{2y}{s_0}s_0{}^3)}{K*\hat{M}*((\frac{\Lambda}{K*\hat{M}})^3+\beta)}$.

By rephrasing $\hat{y} = \frac{y}{\Lambda/(K*\hat{M})}$ and $\hat{s_0} = \frac{s_0}{\Lambda/(K*\hat{M})}$, we obtain:
$f(y) \leq \frac{\frac{\hat{M}+(1-2\hat{M})\hat{y}}{\hat{y}}*(3\hat{y})^3+(\frac{(3\hat{M}-1)\hat{y}-\hat{M}}{\hat{y}})\frac{2\hat{y}}{\hat{s_0}}\hat{s_0}{}^3}{\hat{M}} = g(\hat{y})$.

By solving $g(\hat{y})' = 0$, given $2\hat{y} \leq 1 \leq 3\hat{y}$, we derive:

$$\hat{y} = \frac{3\hat{M}+\sqrt{9\hat{M}^2+2(2\hat{M}-1)(3\hat{M}-1)\hat{s_0}^2}}{9(2\hat{M}-1)} \tag{4.7}$$

We set $\delta = 2\hat{M}-1$, and set $\epsilon = 3\hat{M}-1$, as a result:
$f(\hat{y}) \leq \frac{54\hat{M}^3-18\hat{M}(2\delta*\epsilon-3M\delta)\hat{s_0}^2+2(9\hat{M}^2+2\delta*\epsilon*\hat{s_0}^2)^{\frac{3}{2}}}{27\hat{M}\delta^2}$, the last inequality comes from $\hat{s_0}$ = 1, we obtain:

$$f(\hat{y}) = \frac{54\hat{M}^3-18\hat{M}(2\delta*\epsilon-3M\delta)+2(9\hat{M}^2+2\delta*\epsilon)^{\frac{3}{2}}}{27\hat{M}\delta^2} \tag{4.8}$$

Similarly, we could have $f(\hat{y}) \leq (\frac{54\hat{M}^3-18\hat{M}(2\delta*\epsilon-3M\delta)\hat{s_0}^2+2(9\hat{M}^2+2\delta*\epsilon*\hat{s_0}^2)^{\frac{3}{2}}}{27\hat{M}\delta^2})/\hat{s_0}^2$, for the second sub-case, where $1 \leq \hat{s_0} \leq 1.5$ and the last inequality comes from $\hat{s_0}$ = 1.

For third case $s_0 \leq 2y$, we have $f(y) \leq \frac{K*(\frac{\frac{\Lambda}{K}+(1-2\hat{M})y}{y}*((3y)^3+\beta)+(\frac{(3\hat{M}-1)y-\frac{\Lambda}{K}}{y})((2y)^3+\beta))}{K*\hat{M}*((\frac{\Lambda}{K*\hat{M}})^3+\beta)}$.

By solving $f'(\hat{y}) = 0$, we get $\hat{y} = \frac{38\hat{M}}{3*(30\hat{M}-19)}$.

We obtain:

$$f(y) \leq \frac{4*19^3*\hat{M}^2}{27*(30\hat{M}-19)^2} \tag{4.9}$$

From equation (4.8) and (4.9), we find that value of function (4.9) is no greater than function (4.8) when $\hat{M} \geq 3$, and in the second case $f(\hat{y}) = 1.283$ while in the third case $f(\hat{y}) = 1.13$ if $\hat{M} = \infty$. $\square$

Theorem 1. *For a chip multi-core processor of voltage island with $\hat{M}$ blocks, the approximation of algorithm VILCF to the VIEES problem with $E_{sw} = 0$ is* $\frac{54\hat{M}^3 - 18\hat{M}(2\delta*\epsilon - 3M\delta) + 2(9\hat{M}^2 + 2\delta*\epsilon)^{\frac{3}{2}}}{27\hat{M}\delta^2}$ *where $\hat{M} \geq 3, \delta = 2\hat{M} - 1$, and $\epsilon = 3\hat{M} - 1$.*

Proof. $\Phi(SC_{VILCF})$ is equal to $\sum_{m=0}^{M-1} K*\psi(l_{m,0})$ and $\Phi(SC^*)$ is equal to $\sum_{m=0}^{M-1} K*\psi(l'_{m,0})$, SC^* is a lower bound of the optimal solution of the VIEES problem, we know that the approximation ratio A is $\sum_{m=0}^{M-1} K * \psi(l_{m,0}) / \sum_{m=0}^{M-1} K * \psi(l'_{m,0})$. If for all $0 \leq m \leq M-1$, $l_{m,0}$ is equal to $l'_{m,0}$, then $A = 1$; otherwise, let $\hat{M}$ be the smallest index satisfying that $l'_{\hat{M},0} \neq l_{\hat{M},0}$, we have $\Phi(SC_{VILCF}) = \sum_{m=0}^{M-1} K * \psi(l_{m,0})\delta^{m<\hat{M}} + \sum_{m=0}^{M-1} K * \psi(l_{m,0})\delta^{m\geq\hat{M}}$ and $\Phi(SC^*) = \sum_{m=0}^{M-1} K * \psi(l'_{m,0})\delta^{m<\hat{M}} + \sum_{m=0}^{M-1} K * \psi(l'_{m,0})\delta^{m\geq\hat{M}}_{l'_{m,0}>\lambda} + \sum_{m=0}^{M-1} K * \psi(l'_{m,0})\delta^{m\geq\hat{M}}_{l'_{m,0}=\lambda}$, where δ^a_b is 1 if b is true and a is true; otherwise, it is 0. We know $\sum_{m=0}^{M-1} K * \psi(l_{m,0})\delta^{m<\hat{M}} = \sum_{m=0}^{M-1} K * \psi(l'_{m,0})\delta^{m<\hat{M}}$. We obtain
$A = \frac{\sum_{m=0}^{M-1} \psi(l_{m,0})\delta^{m<\hat{M}} + \psi(l_{m,0})\delta^{m\geq\hat{M}}}{\sum_{m=0}^{M-1} \psi(l'_{m,0})\delta^{m<\hat{M}} + \psi(l'_{m,0})\delta^{m\geq\hat{M}}_{l'_{m,0}>\lambda} + \psi(l'_{m,0})\delta^{m\geq\hat{M}}_{l'_{m,0}=\lambda}}$.

Considering block m with $m < \hat{M}$, we have
$\sum_{m=0}^{M-1}\sum_{k=0}^{K-1} l_{m,k}\delta^{m<\hat{M}} \leq \sum_{m=0}^{M-1}\sum_{k=0}^{K-1} l'_{m,k}\delta^{m<\hat{M}}$ since $l_{m,k} \leq l'_{m,k}$,
so we obtain $\sum_{m=0}^{M-1}\sum_{k=0}^{K-1} l_{m,k}\delta^{m\geq\hat{M}} \geq \sum_{m=0}^{M-1}\sum_{k=0}^{K-1}(l'_{m,k}\delta^{m\geq\hat{M}}_{l'_{m,k}>\lambda} + l'_{m,k}\delta^{m\geq\hat{M}}_{l'_{m,k}=\lambda})$.

We know $\sum_{m=0}^{M-1}\sum_{k=0}^{K-1} l_{m,k}\delta^{m\geq\hat{M}} - \sum_{m=0}^{M-1}\sum_{k=0}^{K-1}(l'_{m,k}\delta^{m\geq\hat{M}}_{l'_{m,k}>\lambda} + l'_{m,k}\delta^{m\geq\hat{M}}_{l'_{m,k}=\lambda})$ is no greater than $\sum_{m=0}^{M-1}\sum_{k=0}^{K-1}(l^{max}_m - l'_{m,k})\delta^{m<\hat{M}}$ because at most the load of $\sum_{m=0}^{M-1}\sum_{k=0}^{K-1}(l^{max}_m - l'_{m,k})\delta^{m<\hat{M}}$ is distributed from $T \backslash T^f$ into all blocks whose index are smaller than $\hat{M}$ in SC^*.

Let $\Lambda = \sum_{m=0}^{M-1}\sum_{k=0}^{K-1} l_{m,k}\delta^{m\geq\hat{M}}$, and we know $\sum_{m=0}^{M-1}\sum_{k=0}^{K-1}(l^{max}_m - l'_{m,k})\delta^{m\geq\hat{M}}$ is no greater than $(K-1)*(l_{0,0} - l_{\hat{M},0})$.

So we have $\Lambda - (K-1)*(l_{0,0} - l_{\hat{M},0}) \leq \sum_{m=0}^{M-1}\sum_{k=0}^{K-1}(l'_{m,k}\delta^{m\geq\hat{M}}_{l'_{m,k}>\lambda} + l'_{m,k}\delta^{m\geq\hat{M}}_{l'_{m,k}=\lambda})$, we obtain:

$$A \leq \frac{\sum_{m=0}^{M-1} \psi(l_{m,0})\delta^{m<\hat{M}} + \psi(l_{m,0})\delta^{m\geq\hat{M}}}{\sum_{m=0}^{M-1} \psi(l'_{m,0})\delta^{m<\hat{M}} + (M-\hat{M}) * \psi(\frac{\Lambda/K - (l_{0,0} - l_{\hat{M},0})}{M-\hat{M}})} \tag{4.10}$$

Let l_{min} be the minimum load of some cores derived from algorithm VILCF. Considering Λ in a schedule of of upper bound of energy consumption by Lemma 9, we know for $m \geq \hat{M}$, the loads of $l_{\hat{M}+1,0}, ..., l_{M-1,0}$ are assigned to K cores respectively, and there is only one core assigned $l_{\hat{M},0}$, and $K-1$ cores assigned l_{min}. So we obtain in an upper bound schedule $\Lambda = \sum_{m=\hat{M}}^{M-1} K * l_{m,0} - (K-1)(l_{\hat{M},0} - l_{min})$.

With Lemma 8, we know that $l_{min} \leq l_{m,0} \leq \frac{3}{2} l_{min}$ for any $m-th$ block with $m \geq \hat{M}$, so we have $\Lambda \geq \sum_{m=\hat{M}}^{M-1} K * l_{m,0} - (K-1)\frac{1}{2} l_{min}$, Let γ_m be the value satisfying $\gamma_m l_{min} + (1-\gamma_m)\frac{3}{2} l_{min} = l_{m,0}$. For any $m-th$ block with $m \geq \hat{M}$, since $l_{min} \leq l_{m,0} \leq \frac{3}{2} l_{min}$, γ_m be a real number between 0 and 1, and $\psi(l_m) \leq \gamma_m \psi(l_{min}) + (1-\gamma_m)\psi(\frac{3}{2} l_{min})$. By taking $\frac{l_{min}}{2}$ as y, we derive:

$$\Lambda \geq K * (\mu * 2y + (M - \hat{M} - \mu) * 3y) - (K-1) * y \tag{4.11}$$

where $0 \leq \mu \leq \hat{M}$.

By Lemma 2, as a result we know that $\sum_{m=0}^{M-1} \psi(l_{m,0})\delta^{m \geq \hat{M}} \leq \sum_{m=0}^{M-1}(\hat{\gamma_m}\psi(l_{min}) + (1-\hat{\gamma_m})\psi(\frac{3}{2} l_{min}))\delta^{m \geq \hat{M}} \leq \mu\psi(2y) + (M - \hat{M} - \mu)\psi(3y)$, we get
$f(y) \leq \frac{\sum_{m=0}^{M-1} \psi(l_{m,0})\delta^{m<\hat{M}} + \mu\psi(2y) + (M-\hat{M}-\mu)\psi(3y)}{\sum_{m=0}^{M-1} \psi(l'_{m,0})\delta^{m<\hat{M}} + (M-\hat{M}) * \psi(\frac{\Lambda/K - (l_{0,0} - l_{\hat{M},0})}{M-\hat{M}})}$ by equation (4.10) and (4.11).

For $0 \leq m \leq M-2$, $l_{m,0} - l_{m+1,0}$ is less than the minimum non-completing task $\tau_{min} \in T \backslash T^f$. When τ_{min} is assigned to core (m,k), we know $\tau_{min} + l_{m,k} \leq 3y$ and $\tau_{min} \leq \frac{1}{2} l_{m,k}$, we obtain $\tau_{min} \leq y$, hence we derive $l_{m,0} - l_{m+1,0} \leq y$ and $l_{0,0} - l_{\hat{M},0} \leq \hat{M} y$.

Since $\frac{\mu\psi(2y) + (M-\hat{M}-\mu)\psi(3y)}{(M-\hat{M}) * \psi(\frac{\Lambda - (K-1)*(l_{0,0} - l_{\hat{M},0})}{M-\hat{M}})} \geq 1$ and $l_{m,0} \geq 3y$ for $m < \hat{M}$, we get
$\sum_{m=0}^{M-1} \psi(l_{m,0})\delta^{m<\hat{M}} \geq \hat{M}\psi(3y)$, we have $f(y) \leq \frac{\hat{M}\psi(3y) + \mu\psi(2y) + (M-\hat{M}-\mu)\psi(3y)}{\hat{M}\psi(3y) + (M-\hat{M}) * \psi(\frac{\Lambda/K - \hat{M}y}{M-\hat{M}})}$.

Consider the denominator of $f(y)$, we have $\hat{M} * 3y + (M-\hat{M}) * \frac{\Lambda/K - \hat{M}y}{M-\hat{M}} \geq M * \Lambda/(M*K)$, so we derive $\hat{M}\psi(3y) + (M-\hat{M}) * \psi(\frac{\Lambda/K - \hat{M}y}{M-\hat{M}}) \geq M * \psi(\frac{\Lambda}{K*M})$ by Lemma 3, so we get $f(y) \leq \frac{\mu\psi(2y) + (M-\mu)\psi(3y)}{M * \psi(\frac{\Lambda}{K*M})}$, by Lemma 10, the theorem is proved.

□

Corollary 1. *The approximation of Algorithm VILCF of the VIEES problem is* $\frac{4*19^3*\hat{M}^2}{27*(30\hat{M}-19)^2}$ *when* β *and* S_{min} *are both* 0 *with block number is no less than* 3.

4.6 Simulation Results

In this section, we evaluate the energy efficiency of our proposed algorithm VILCF under different block partitions. In addition, we compare the performance between VILCF and Largest Task First (LTF) [38].

The workload parameters and performance metrics are set as follows. The power consumption function is set as $P(s) = s^3 + \beta$, where s is the speed of a core and β is the static power consumption. β is a variable between 1 to 5. The task sizes are integral random variables between 1 and 6. The minimum speed S_{min} is set as 0. The hyper-period L is 1 and the switching overhead E_{sw} is 0. The number of cores is fixed to 128. When the number of blocks (voltage islands) increases from 2^1 to 2^5, the number of cores in each block decreases from 2^6 to 2^2, respectively. Experimental results are obtained with 128 independent runs for each configuration.

The ***normalized energy*** of an algorithm is defined as the ratio of energy consumption of the derived schedule to that of the optimal schedule SC^*. Since VIEES is NP-hard, the normalized energy provides an approximation ratio. The smaller the normalized energy is, the better the approximation. As expected, we see the normalized energy decreases when the number of blocks increases, see figure (4.5).

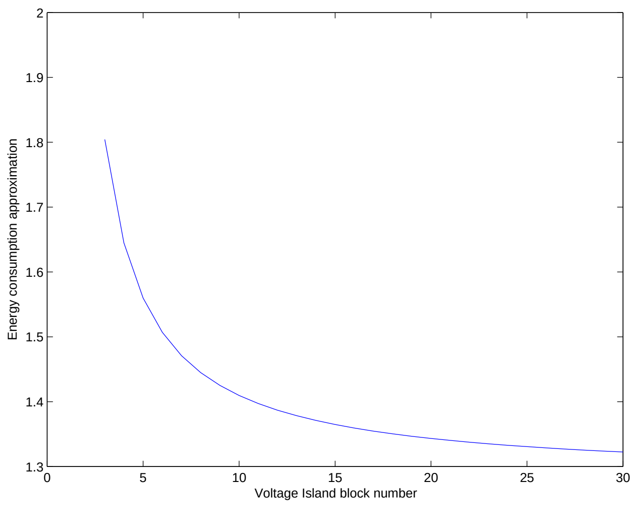

Figure 4.5: The energy consumption approximation to the number of blocks

We compare VILCF with LTF [38]. In both algorithms, the tasks are sorted first, and then the largest task is scheduled first. With LTF, a task is assigned to a core with the minimum load. While with VILCF, a task is assigned to a core with the largest capacity. The other difference is that after each scheduling, VILCF sorts the partitioned task subsets and reassigns them to cores, but LTF does not sort again. In figure 6, the task set size ranges from 150 to 550, and the number of blocks varies from 2^1 to 2^5. The β is 2. VILCF significantly outperforms LTF in term of normalized energy, see figure 6. On average, the normalized energy of VILCF is about 1.06, while that of LTF is 1.17.

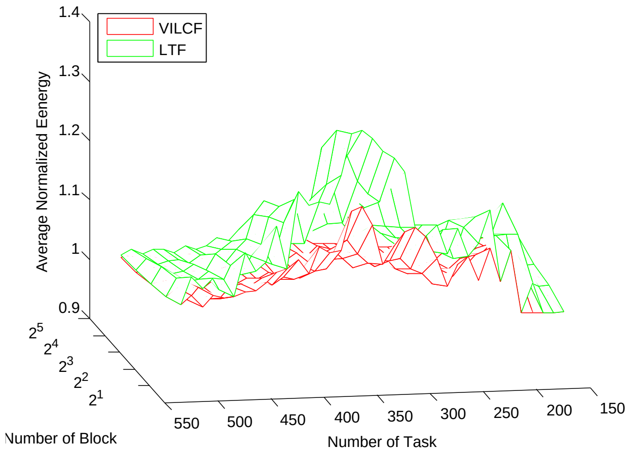

Figure 4.6: Comparison of Normalized Energy between VILCF and LTF when the number of core is 128, the number of blocks ranges in $[2^1, 2^5]$, the task set ranges in $[150, 550]$

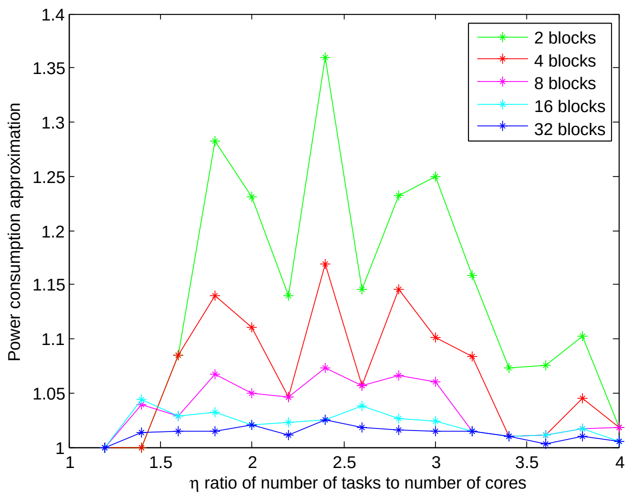

Figure 4.7: Power consumption approximation when η ranges from 1.2 to 4, stepped by 0.2. β is set as 2.

Figure (4.7) shows the average normalized energy of VILCF by varying η, which is the ratio of the number of tasks to the number cores. We vary η from 1.2 to 4, stepped by 0.2. The average normalized energy is worst when η is close to 3. When η is small, most cores are assigned with only one task, and the assignment is nearly the same as the optimal schedule. However, when the ratio η increases, the load of each core increases and the average normalized energy increases. Furthermore, the more number of blocks the chip partitioned, the better the energy-efficiency is, see figure (4.7) and (4.8).

Figure (4.8) shows the average normalized energy of VILCF by varying β. We vary β from 0 to 5, stepped by 0.5. η is set as 3. When β increases, the average normalized energy decreases slightly, even though the proportion of the energy consumption resulting from leakage current to the total energy consumption increases greatly.

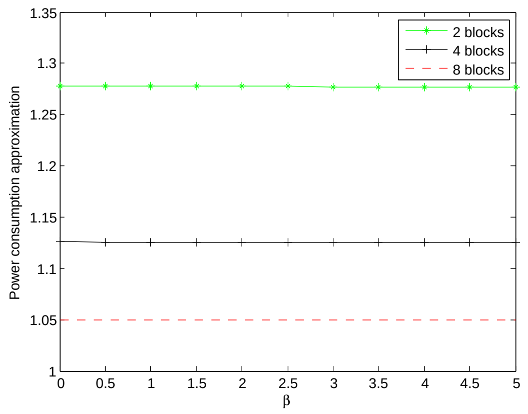

Figure 4.8: Power consumption approximation when β ranges from 0 to 5. η is set as 3.

4.7 Summary

This chapter studies energy efficiency in data centers. It presents a Voltage Island Largest Capacity First (VILCF) algorithm for energy-efficient scheduling of periodic real-time tasks on multi-core processors of voltage island model. It achieves better energy efficiency by fully utilizing the remaining capacity of a voltage island before turning on more voltage islands or increasing the voltage/speed of current active voltage islands. When there is no upper bound on the core speeds and the leakage power is negligible, the approximation ratio of VILCF is $\frac{4*19^3*\hat{M}^2}{27*(30\hat{M}-19)^2}$ where $\hat{M} \geq 3$. For example, the approximation ratio is 1.81 and 1.283, respectively, when $\hat{M}$ is 3 and ∞. Furthermore, when the overhead in turning on/off a processor is negligible, the approximation ratio of VILCF is $\frac{54\hat{M}^3-18\hat{M}(2\delta*\epsilon-3M\delta)+2(9\hat{M}^2+2\delta*\epsilon)^{\frac{3}{2}}}{27\hat{M}\delta^2}$ where $\hat{M} \geq 3, \delta = 2\hat{M} - 1$, and $\epsilon = 3\hat{M} - 1$. In general, the approximation ratio decreases with increasing of the number of blocks.

CHAPTER 5

Traffic and Energy Aware Virtual Network Mapping in Data Centers

5.1 Introduction

This chapter investigates the energy-efficient resource allocation problem for virtual networks in cloud data centers.

Virtualization is an effective approach to save the energy consumption of a tenant placed into a data center. A cloud tenant expresses computation requirement for each virtual machine (VM) and bandwidth requirement for each pair of VMs. Virtualization can consolidates VMs into fewer servers (also termed physical machines (PMs)) to increase utilization of data centers.

However, consolidating VMs of a tenant to fewer servers is restricted by server computation resource such as CPU, memory, and storage. Most tenants can not be placed into a single server, they must be placed into multiple servers. VMs of a tenant placed in different servers always need to communicate to each other. Thus data center need to allocate network resource to the tenant. Recently, more and more applications and services deployed in data centers require link bandwidth guarantees for their communications. So VM placement of a tenant is also restricted by network resource.

Most data center network is designed with redundancy. For example, fat-tree network is designed in a multi-root path topology, the bisection connection of a fat-tree is full connection. Not all switches need be active when a tenant is placed into the data center, unneeded switches can be powered off to save energy consumption.

This chapter studies placing the virtual network abstracted by a tenant into a data center while minimizing the active servers and switches under the constraints of server computation resource and network resource.

5.2 The Model

5.2.1 Network Architectures

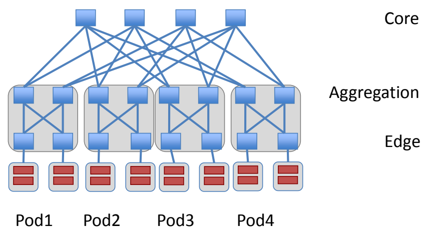

Figure 5.1: Three-tier fat-tree topology

We focus on fat-tree networks, such as Portland [69], since they are widely deployed in data centers. We assume the data center network model of this study is a three-tier homogeneous fat-tree network modeled as graph $G(V, E)$, where V is the set of vertex and $E \in V \times V$ is the set of edges (see Figure (5.1)). The vertex in V are classified into two types: the network switches and the physical machines (PMs) (which are also referred as servers). The sets of switches and PMs are denoted as S and P respectively. Therefore, $V = S \cup P$. The edge $e \in E$ represents a communication link between a PM and a switch, or a pair of switches.

We assume each switch in a fat-tree network has K ports. There are K pods in the network, each of which has K switches. For pod k while $1 \leq k \leq K$, $\frac{K}{2}$ of switches are grouped as aggregation tier ($AGGR_k$) and the other $\frac{K}{2}$ of switches are grouped as edge tier ($EDGE_k$). Switches in edge tier and switches in aggregation tier are connected in a bipartite topology. For switch $i \in EDGE_k$ and $j \in AGGR_k$, we have edge $e_{i,j} = 1$. For edge switch i ($i \in EDGE_k$), it is connected to $\frac{1}{2}K$ PMs which are grouped as $RACK_i$. There are $\frac{1}{4}K^2$ core switches in the core tier (CORE) of data center network, each aggregation switch in every pod is connected to $\frac{1}{2}K$ of core switches. The total number of switches of a three-tier fat-tree network is $\frac{5}{4}K^2$. We denote the bandwidth capacity of a link as C_l, and the computation capacity of a PM as C_s.

A fat-tree network is richly connected to support very high bisection bandwidth. However, its link utilization is often very low when data center networks run well below the capacity. To reduce the energy consumption, traffic flows could be consolidated into smaller number of links/switches and then power off unused links and switches.

5.2.2 VM Communication Bandwidth Allocation

In a multi-tenant data center, when a tenant makes a request to a data center, it simply asks for some amount of computation resource such as CPU, memory, and storage. A tenant's computation instances, virtual machines (VMs), always demand some network resources to support the communication among them. In this study, a tenant is abstracted to a virtual network using pipe model, and this virtual network is modeled as a graph G_v with VMs T as vertex and traffic matrix M as edges. A tenant can specify a diverse set of computation and bandwidth demands for different VMs. Thus, the modeled graph is a weighted graph. The vertex weight represents the VM computation demand r_i, and the edge weight represents the traffic flow size $m_{i,j}$ between VMs i and j.

The VM placement is restricted by two constraints: PM computation capacity C_s and link bandwidth capacity C_l. We define the ***load*** of PM p as $\sum_i \Delta_{i,p} r_i$, for PM p, its load should be no more than its computation capacity C_s, ($\sum_i \Delta_{i,p} r_i \leq C_s$).

We study VM flow routing in three-tier fat-tree networks. We discuss this problem in four cases. In case 1, a pair of VMs i and j ($i, j \in T$) along with flow $m_{i,j}$ are placed into PM p with $\Delta_{i,p} = 1$ and $\Delta_{j,p} = 1$. Their communication is carried out inside PM p. This intra-PM traffic cost is ignored since no network resource is required for their communication.

In case 2, VMs i and j are placed into two different PMs s and t with $s \neq t$ and $\Delta_{i,s} = 1$ and

Table 5.1: Notation

T	Set of VMs
M	Traffic matrix of a tenant virtual network
$m_{i,j}$	Bandwidth demand by the flow between VM $i \in T$ and VM $j \in T$
r_i	computation resources required by VM i
$\Delta_{i,p}$	A binary decision for assigning VM i to PM p
x_s	A binary decision for switch s being active.
y_p	A binary decision for PM p being active.
$\Omega^k_{m_{i,j}}$	A binary decision for traffic $m_{i,j}$ crossing switch k.
$e_{u,v}$	A binary decision for link between switch u and switch v
$EDGE_k$	Set of switches in edge tier of pod k
$AGGR_k$	Set of switches in aggregation tier of pod k
$CORE$	Set of switches in core tier
$RACK_k$	Set of PMs share edge switch k
W	Set of VM-clusters
D	inter VM-cluster traffic matrix
$d_{i,j}$	Bandwidth required by the flow between VM-cluster $i \in W$ and VM-cluster $j \in W$
$\Gamma_{p,s}$	A binary decision for mapping VM-cluster p to a PM in rack s

$\Delta_{j,t} = 1$, and PMs s and t share the same edge switch k ($s, t \in RACK_k$). Flow $m_{i,j}$ crosses edge switch k, so we have $\Omega^k_{m_{i,j}} = 1$.

In case 3, VMs i and j are placed into two different PMs s and t with $s \neq t$ and $\Delta_{i,s} = 1$ and $\Delta_{j,t} = 1$, and PMs s and t connect to two different edge switches u and v ($s \in RACK_u, t \in RACK_v, u \neq v$) , but they are located in the same pod n ($u, v \in EDGE_n$). According to fat-tree topology, edge switches u and v connect all $\frac{K}{2}$ aggregation switches in this pod. One aggregation switch in $AGGR_n$ is selected to route flow $m_{i,j}$, so we obtain $\Omega^u_{m_{i,j}} = 1$ and $\Omega^v_{m_{i,j}} = 1$ and $\sum_{k \in AGGR_n} \Omega^k_{m_{i,j}} = 1$.

In case 4, VMs i and j are placed into two different PMs s and t with $s \neq t$ and $\Delta_{i,s} = 1$ and $\Delta_{j,t} = 1$, and PMs s and t are located in different pods m and n. PMs s and t connect edge switches u and v respectively ($s \in RACK_u, t \in RACK_v, u \in EDGE_m, v \in EDGE_n, m \neq n$), edge switches u and v select an aggregation switch in their own pod respectively, and these two aggregation switches select a common core switch to route flow $m_{i,j}$. So we obtain $\Omega^u_{m_{i,j}} = 1$ and $\Omega^v_{m_{i,j}} = 1$ and $\sum_{k \in AGGR_m} \Omega^k_{m_{i,j}} = 1$ and $\sum_{k \in AGGR_n} \Omega^k_{m_{i,j}} = 1$ and $\sum_{k \in CORE} \Omega^k_{m_{i,j}} = 1$.

The bandwidth resource of a link is allocated to the flows which cross this link. In a fat-tree network, for two switches u and v, if flow $m_{i,j}$ crosses both of them with $\Omega^u_{m_{i,j}} = 1$ and $\Omega^v_{m_{i,j}} = 1$, and if there is a link between switches u,v with $e_{u,v} = 1$, flow $m_{i,j}$ must cross link $e_{u,v}$, so the bandwidth requirement of the flow to the link is $e_{u,v}\Omega^u_{m_{i,j}}\Omega^v_{m_{i,j}}m_{i,j}$. We define the ***load*** of link $e_{u,v}$ as $\sum_{i,j \in T} e_{u,v}\Omega^u_{m_{i,j}}\Omega^v_{m_{i,j}}m_{i,j}$. The load of link $e_{u,v}$ should be no more than C_l ($\sum_{i,j \in T} e_{u,v}\Omega^u_{m_{i,j}}\Omega^v_{m_{i,j}}m_{i,j} \leq C_l$).

From the four cases above, we study flow routing in each tier of a fat-tree network. First, we consider flow $m_{i,j}$ routing in edge tier switches. We assume link $e_{p,s}$ connects PM p to edge switch s. For flow $m_{i,j}$, if VM i or j is hosted in PM p, flow $m_{i,j}$ must cross link $e_{p,s}$:

$$\Omega^s_{m_{i,j}} = \Delta_{i,p}(1 - \Delta_{j,p}) + \Delta_{j,p}(1 - \Delta_{i,p}) \tag{5.1}$$

where $p \in RACK_s$ The bandwidth which link $e_{p,s}$ must allocate to flow $m_{i,j}$ is $m_{i,j} * (\Delta_{i,p}(1 -$

$\Delta_{j,p}) + \Delta_{j,p}(1 - \Delta_{i,p}))$, the load of link $e_{p,s}$ is $\sum_{i,j \in T} m_{i,j} * (\Delta_{i,p}(1 - \Delta_{j,p}) + \Delta_{j,p}(1 - \Delta_{i,p}))$.

Second, We consider flow $m_{i,j}$ routing in aggregation tier switches in pod k while $1 \leq k \leq K$. We let $\delta_m = \sum_{p \in RACK_m} \Delta_{i,p}(1 - \sum_{p \in RACK_m} \Delta_{j,p}) + \sum_{p \in RACK_m} \Delta_{j,p}(1 - \sum_{p \in RACK_m} \Delta_{i,p})$ while switch $m \in EDGE_k$ and $\epsilon_k = \sum_{m \in EDGE_k} \sum_{p \in RACK_m} \Delta_{i,p}(1 - \sum_{m \in EDGE_k} \sum_{p \in RACK_m} \Delta_{j,p}) + \sum_{m \in EDGE_k} \sum_{p \in RACK_m} \Delta_{j,p}(1 - \sum_{m \in EDGE_k} \sum_{p \in RACK_m} \Delta_{i,p})$. $\delta_m = 1$ means VM i or j is hosted in a PM in $RACK_m$, the other is not. $\epsilon_k = 1$ means VM i or j is is hosted in a PM located in pod k, the other is not.

If PMs which host VM i or j are both located in pod k , then we have
$\sum_{m \in EDGE_k} \sum_{p \in RACK_m} \Delta_{i,p} = 1$ and $\sum_{m \in EDGE_k} \sum_{p \in RACK_m} \Delta_{j,p} = 1$ (see case 1,2,3). In case 1 and 2, we have $\sum_{m \in EDGE_k} \delta_m = 0$. In case 3, flow $m_{i,j}$ crosses two edge switches, we have $\sum_{m \in EDGE_k} \delta_m = 2$, $m_{i,j}$ must cross one aggregation switch in pod k, so we have $\sum_{s \in AGGR_k} \Omega^s_{m_{i,j}} = 1$. In case 4, one of PMs which host VM i or j is located in pod k, we know
$(\sum_{m \in EDGE_k} \sum_{p \in RACK_m} \Delta_{i,p})(\sum_{m \in EDGE_k} \sum_{p \in RACK_m} \Delta_{j,p}) = 0$ and $\epsilon_k = 1$. Flow $m_{i,j}$ must cross one aggregation switch in pod k. We obtain:

$$\begin{aligned}\sum_{s \in AGGR_k} \Omega^s_{m_{i,j}} = (\frac{1}{2}(&\sum_{m \in EDGE_k} \sum_{p \in RACK_m} \Delta_{i,p}) \\ &*(\sum_{m \in EDGE_k} \sum_{p \in RACK_m} \Delta_{j,p}) + \epsilon_k) \sum_{m \in EDGE_k} \delta_m \end{aligned} \tag{5.2}$$

Equation (5.2) indicates flow $m_{i,j}$ crosses one aggregation switch in case 3 and 4 in pod where VMs placed.

Last, we consider flow $m_{i,j}$ routing in core tier switches, we have:

$$\sum_{s \in CORE} \Omega^s_{m_{i,j}} = \frac{1}{2} \sum_{k=1}^{K} \epsilon_k \tag{5.3}$$

Equation (5.3) indicates flow $m_{i,j}$ crosses one core switch in case 4.

5.2.3 Problem Definition

Now we define the problem of traffic and power aware virtual network mapping (TPVNM). Given a data center with a three-tier homogeneous fat-tree network $G = (V, E)$, and a set of VMs T with traffic matrix M, we place VMs into PMs and assign traffic flows onto links/switches while minimizing the active PMs and switches under PM computation capacity C_s and link bandwidth capacity C_l while guaranteeing the computation requirement for each VMs and the bandwidth requirement for each pair of VMs.

$$\text{Minimize} \sum_{s\in S} x_s + \sum_{p\in P} y_p \tag{5.4a}$$

$$\text{Subject to} \sum_{i\in T} \Delta_{i,p} r_i \le C_s * y_p, \tag{5.4b}$$

$$\sum_{p\in P} \Delta_{i,p} = 1, \tag{5.4c}$$

$$\begin{aligned}&\sum_{i,j\in T} m_{i,j} * (\Delta_{i,p}(1-\Delta_{j,p}) + \Delta_{j,p}(1-\Delta_{i,p}))\\ &\quad \le \Omega^k_{m_{i,j}} x_k C_l,\\ &\quad (p \in RACK_k),\end{aligned} \tag{5.4d}$$

$$\begin{aligned}&\sum_{i,j\in T} e_{u,v} \Omega^u_{m_{i,j}} \Omega^v_{m_{i,j}} m_{i,j} \le x_u x_v C_l,\\ &\quad (u \in S, v \in S),\end{aligned} \tag{5.4e}$$

$$\sum_{s\in CORE} \Omega^s_{m_{i,j}} = \frac{1}{2}\sum_{k=1}^{K} \epsilon_k, \tag{5.4f}$$

$$\begin{aligned}&\sum_{s\in AGGR_k} \Omega^s_{m_{i,j}} = (\frac{1}{2}(\sum_{m\in EDGE_k} \sum_{p\in RACK_m} \Delta_{i,p})\\ &* (\sum_{m\in EDGE_k} \sum_{p\in RACK_m} \Delta_{j,p}) + \epsilon_k) \sum_{m\in EDGE_k} \delta_m,\\ &\quad (1 \le k \le K).\end{aligned} \tag{5.4g}$$

Constraint (5.4b) models the computation capacity constraint of a PM where VMs are placed. Constraint (4c) means any VM must be placed into one and only one PM. Constraint(5.4d) captures bandwidth constraint of a link between a PM and an edge switch. Constraint(5.4e) captures bandwidth constraint of a link between switches. Constraints (5.4f) ensure flows to cross at most one core switch. Constraints (5.4g) ensure flows to cross at most one aggregation switch in a pod.

Theorem 2. *TQVMP is an NP-hard problem.*

Proof. This can be proved by a reduction from the 3-Partition problem defined as follows: [81] given $n = 3k$ integers $a_1, a_2, ..., a_n$ and a threshold S such that $\frac{S}{4} \leq a_i \leq \frac{S}{2}$ and $\sum_i^n a_i = kS$. The task is to decide if the numbers can be partitioned into triples such that each triple adds up to S. This problem is NP-complete.

The reduction is as follows. given an instance of 3-Partition, we construct a graph G using integers a_i while $1 \leq i \leq n$. For each a_i, we break it up to a set of smaller integers $a_{i,1}, a_{i,2}, ..., a_{i,m}$ such that $\sum_{j=1}^{m} a_{i,j} = a_i$ and $m \geq 1$. We create a fully connected weighted subgraph for integer a_i with m vertex, the weights of these vertex are $a_{i,1}, a_{i,2}, ..., a_{i,m}$ respectively. So an instance of 3-Partition problem is transformed to that of the TQVMP problem in polinomial time. If the 3-Partition instance can be solved, the TQVMP problem in G can be solved without cutting any edge. If the 3-Partition instance cannot be solved, the optimum TQVMP in G will cut at least one edge. □

5.3 The Solution

Since TPVNM is an NP-hard problem, we explore heuristic algorithm for it. We divide our solution into two phases: Virtual Machine (VM) Packing and Virtual Network (VN) Mapping.

Placing VMs into DCN is to cut graph G_v into subgraphs and put them into PMs under two constraints C_s and C_l while minimizing PMs and switches. If the placement is restricted mainly by PM CPU utilization, we say this virtual network is Computation-Intensive. If the placement is restricted mainly by link bandwidth capacity, we say this virtual network is Communication-

Intensive. Computation-Intensive virtual networks request more CPU cycles while Communication-Intensive virtual networks request more link bandwidth for their placement.

The placement of Computation-Intensive virtual network is a bin packing problem since the VM CPU utilization dominates the VM placement. This problem can be resolved by Algorithm First Fit Decrease (FFD). By Algorithm FFD, the approximation of PM number is 1.22 see [82]. We assume in the worst case all switches are active, for a K port switch fat-tree network, the total switches is $\frac{5K^2}{4}$, and total PMs is $\frac{K^3}{4}$. We assume by the optimal solution no switch needs to be active, so the approximation of Algorithm FFD for VM placement is $1.22(1+\frac{5}{K})$. If $K = 48$, the approximation is 1.35.

Now we consider the more complex case Communication-Intensive VM placement. In this case, our problem TQVMP becomes Quadratic Assignment Problem (QAP), which is a known NP-complete problem. In fact, it is one of the most difficult problems in NP-hard class, and it even cannot be approximated efficiently within some constant approximation ratio. The best option is to host heavy communicated network-aware VMs on PMs that located close to each other.

In this study, we use traffic matrix M to identify traffic patterns with heavy communications, and consolidate VMs into a minimum PMs and minimize the switch number, and then turn off unused PMs and switches for power savings. We divide our solution into two phases: VM grouping and PM mapping.

5.3.1 Traffic-Aware VM Packing

In the first phase, we cut graph G_v into subgraphs and pack VMs in each subgraph into a VM-cluster. A VM-cluster is a subset T' of VMs in T with high intra-group traffics, ($T' \subset T$). $T - T'$ is the subset of VMs outside the cluster in T. We let ***inter-cluster traffic*** of T' to be traffics between T' and $T - T'$, which is $\sum_{i \in T', j \in T-T'} m_{i,j}$. Each VM-cluster will be hosted in a single PM. Therefore we can minimize the number of PMs by minimizing the number of VM-clusters under the constraints C_s and C_l, see equation (5.5).

$$\text{Minimize} \sum_{p\in P} y_p \tag{5.5a}$$

$$\text{Subject to} \sum_{i\in T} \Delta_{i,p} r_i \leq C_s * y_p, \tag{5.5b}$$

$$\sum_{p\in P} \Delta_{i,p} = 1, \tag{5.5c}$$

$$\sum_{i,j\in T} m_{i,j} * (\Delta_{i,p}(1-\Delta_{j,p})+ \Delta_{j,p}(1-\Delta_{i,p})) \leq C_l. \tag{5.5d}$$

This falls into the class of Multi-Capacity Bin Packing Problem [83], which is a known NP-complete problem. Unlike bin packing, in traffic-aware VM packing, the cumulative bandwidth of a cluster of VMs hosted in a server can be smaller than the sum of individual VM bandwidth, as shown in Constraint (5.5d). This is because the communications between co-located VMs are through the shared memory and do not require any communication bandwidth from the server adapter.

For communication-intensive virtual network, it is the link bandwidth resource that is the bottleneck to consolidate VMs into PMs. Therefore, to minimize the number of PMs, we must minimize the inter VM-cluster traffics.

We propose Algorithm VM-Packing (see Algorithm 1) to identify VM-clusters by traffic matrix M while minimizing inter VM-cluster traffics. Algorithm VM-Packing cuts G_v into VM-clusters, which can be hold by PMs under the constraints of C_s and C_l. When VM-clusters are mapped into PMs, all inter VM-cluster traffics are carried out by data center network while intra VM-cluster traffics are ignored.

There are two steps in Algorithm VM-Packing, the first step (Line 3-21) is to identify traffic patterns by traffic matrix M, the second step (Line 22-37) is to pack the traffic patterns into VM-clusters which can be hold by PMs under the constraints of C_s and C_l. In the first step, we have two operations : merge and partition. In the merge operation (Line 4-8), we sort the flows $m_{i,j}$ with VMs $i, j \in T$ by their size in non-increasing order. We get the largest flow $m_{u,v} = \max_{i,j \in T} m_{i,j}$ and two VMs u and v adjacent to it, if $r_u + r_v \le C_s$ and $\sum_{i \in T} m_{u,i} + \sum_{i \in T} m_{v,i} - 2m_{u,v} \le C_l$, we merge VMs u and v into a new super VM, and then recompute the traffic matrix M, this newly created super VM is a traffic pattern. We continue this operation until no VM pair can be merged. Each time we recompute traffic matrix M, and update the virtual network graph G_v.

After merge operation, we do partition operation (Line 9-20). We generate Maximum Spanning Tree (MST) for the merged graph G_v, M^{MST} is the traffic matrix of MST. We sort the flows $m_{i,j}^{MST}$ while VMs $i, j \in T$ by their size in non-decreasing order. We remove the smallest edge $e_{u,v} = \min_{i,j \in T} m_{i,j}^{MST}$, and partition MST into two subtrees MST_1 and MST_2 where T_1 and T_2 are subsets of VMs in MST_1 and MST_2 respectively. For each subtree MST_k and T_k, if $\sum_{i \in T_k} r_i \le C_s$ and inter-pattern traffic of T_k $\sum_{i \in T_k, j \in T - T_k} m_{i,j} \le C_l$, we merge VM set T_k into a new super VM; otherwise, partition MST_k by removing $e_{u,v} = \min_{i \in T_k, j \in T_k} m_{i,j}^{MST_k}$ until CPU utilization of each partitioned pattern is no more than C_s and its inter-pattern traffics is no more than C_l. Then we recompute traffic matrix M of the updated G_v. We repeat merge and partition operations until no new super VM is created in the first step. After the first step, we move to the second step to pack the VM patterns into VM-clusters using Packing algorithm First-Fit.

For example, we have a virtual network graph with 17 VMs, as shown in see Figure (5.2). Let's assume the computation capacity $C_s = 12$ and the link bandwidth capacity $C_l = 24$. We let $r_i = 1$ for $1 \le i \le 12$, $r_i = 3$ for $13 \le i \le 17$, and $\sum_{j \in T} m_{i,j} \le C_l$ for $1 \le i \le 17$. In the first round merge operation, VMs 13 and 14 are merged, and VMs 15 and 16 are merged(see Figure (5.3)). Then we generate a MST tree for the graph, (see Figure (5.4)), run the partition operation to get three new super VMs, VM $1, 2, 3$, VM $4, 5, 6$, and VM $7, 8, 9$, see (Figure (5.5)). We continue the next round operations, and get two new super VMs, VM $1, 2, 3, 7, 8, 9, 11$ and VM $4, 5, 6, 10$ by the merge operation, (see Figure (5.6)). Finally, we get a new super VM $1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12$ by the partition operation, (see Figure (5.7)).

Theorem 3. *The complexity of Algorithm VM-Packing is* $O(N^3 \log^2 N)$ *where* $N = |T|$.

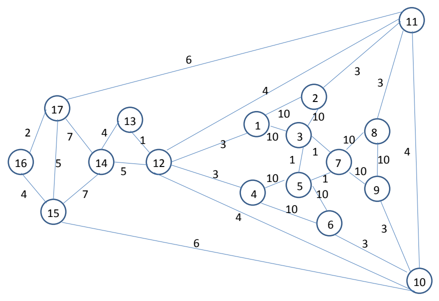
Figure 5.2: Graph of a tenant virtual network with 17 nodes while $C_s = 12$ and $C_l = 24$

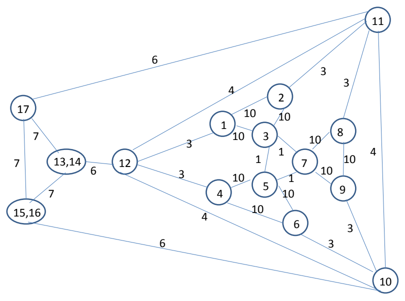
Figure 5.3: Merge VMs 13 and 14, VMs 15 and 16 respectively

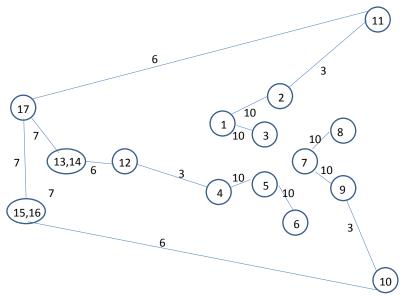
Figure 5.4: Maxium Spanning Tree of the graph

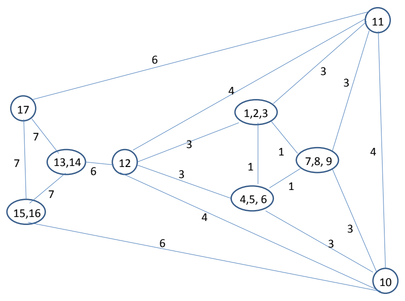
Figure 5.5: partition VMs $1, 2, 3$, VMs $4, 5, 6$, VMs $7, 8, 9$, and then merge them into three super VMs

Algorithm 2 VM-Packing

Input: Graph G_v with VMs of T as vertexes and traffic matrix M as edges
Output: VM-cluster set $W = \{W_1, W_2, ..., W_n\}$, and inter VM-clusters traffic matrix D

```
1:  Set $W_i = \phi$ ,$i = 1, ..., n$
2:  Set $D = 0$
3:  repeat
4:      sort $m_{i,j}$ with $i, j \in T$ in non-increasing order //Start operation merge
5:      while existing VM pairs can be merged under the constraints $C_l$ and $C_s$ do
6:          merge VMs $u$ and $v$ with $m_{u,v} = \max_{i,j \in T} m_{i,j}$ into a super VM
7:          sort $m_{i,j}$ with $i, j \in T$ in non-increasing order
8:      end while// End operation merge
9:      generate Maximum Spanning Tree $MST$ for the graph $G_v$//Start operation partition
10:     enqueue $MST$ to queue $Q$
11:     while $Q$ is not empty do
12:         dequeue the first element $MST'$ from $Q$ where $T'$ is the VM set of $MST'$
13:         if $\sum_{i \in T'} r_i \leq C_s$ and inter-pattern traffic requirement of $T'$ $\sum_{i \in T'.j \in T-T'} m_{i,j} \leq C_l$ then
14:             merge VMs in $T'$ into a super VM
15:         else
16:             partition $MST'$ by removing the edge with minimum weight in $MST'$
17:             put all partitioned trees in to $Q$
18:         end if
19:     end while
20:     recompute the traffic matrix $M$ by putting super VMs back into $G_v$ and recovering the
        removed edges//End operation partition
21: until no super VM is created
    {Packing VMs by Algorithm FF in the following statements}
22: sort VMs in $T$ by their CPU utilization // $T$ is updated by merge and partition operations
23: for each VM $i$ with $i \in T$ do
24:     for each Bin $j$ do
25:         if Bin $j$ can hold VM $i$ under the constraints $C_s$ and $C_l$ then
26:             $W_j \Leftarrow W_j \cup$ VM $i$ // $W_j$ is subset of VMs in Bin $j$
27:         end if
28:     end for
29:     if no Bin can hold VM $i$ then
30:         initialize an empty bin $n$, $W_n \Leftarrow$ VM $i$
31:     end if
32: end for
33: compute traffic matrix $D$ among bins
```

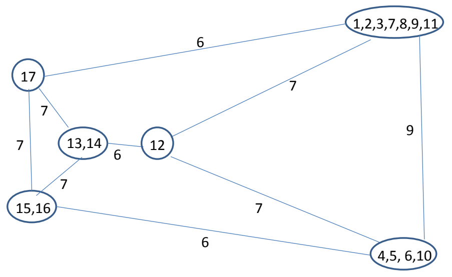

Figure 5.6: Second round merging

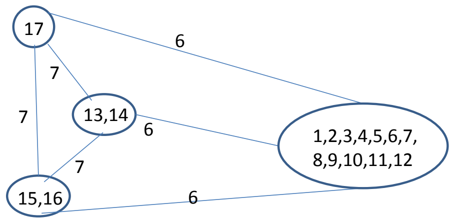

Figure 5.7: Second round partitioning

Proof. In the first step of Algorithm VM-Packing, two operations merge and partition are executed in a loop. Operation merge is also a loop, edges are sorted in non-increasing order, the time cost is $O(N^2 \log N)$, then the largest edge is merged, and traffic matrix M is updated, the time cost is N. The merge loop is is executed at most N times, so the time cost for operation merge is $N^3 \log N$.

For partition operation, the time cost of generating MST is $O(N^2 \log N)$, loop of removing an edge in MST is N times. For each removing, checking subtree is time cost N^2, so the time cost of partition operation is N^3.

The loop of the first step of Algorithm VM-Packing is executed in $\log N$ times. This can be proved in the follow way. We assume the generated MST of graph G_v is composed of such edges: $e_{r,s} \geq e_{s,t} \geq e_{t,u} \geq e_{u,v} \geq e_{v,w} \leq e_{w,x} \leq e_{x,y}$ where VM r is root of MST and edge $e_{v,w}$ is the smallest. In partition operation, we partition MST tree by removing the smallest edge $e_{v,w}$, then MST is divided into two parts MST_i and MST_j where MST_j is composed of $[e_{w,x}, e_{x,y}]$. We assume VMs in MST_j satisfy constraints of C_s and C_l. Edge number in MST_j is either 0 or at least 2 because if one edge in MST_j, this edge must been merged in the merge operation. After this partition, since the edge number is at least two, the newly created super VM w^* must contains over three VMs. The MST is changed into $e_{r,s} \geq e_{s,t} \geq e_{t,u} \geq e_{u,v} \geq e_{v,w^*}$.

In the next round merge operation, if no VM is merged, the MST dose not change, no new super VM can be created, the loop stops. In order to continue the loop, at least two new super VMs v^* and w^* must created by merge operation to update MST. We assume the updated MST is $e_{r,s}, e_{s,t}, e_{t,u}, e_{u,v^*}, e_{v^*,w^*}$. In this updated MST, if we have $e_{r,s} \geq e_{s,t} \geq e_{t,u} \leq e_{u,v^*} \leq e_{v^*,w^*}$, the partition operation can partition the updated MST and create new super VM. For new created super VMs v^* and w^*, each of them are merged by VMs at least one of which must be new super VM created in previous round.

For the $n-th$ round of merge and partition loop, in order to continue the loop, at least two new super VMs must be created by merging at least four VMs and two of them must be super VMs created in previous round, so the VM number is reduced at least 2^n in the $n-th$ round of loop. Therefore, we derive loop is executed in $\log N$ times.

The time cost of Algorithm FF is no more than N^3, so the complexity of Algorithm VM-Packing is $O(N^3 \log^2 N)$.

□

5.3.2 Virtual Network Mapping

After VM packing, we obtain a virtual network $G(W, D)$ with a set of VM-clusters W and inter VM-cluster traffic matrix D. In the second phase and third phase, we minimize the active switches in the data center network which carry out the inter VM-cluster traffics under bandwidth guarantee.

In the second phase, we map the virtual network into data center networks while minimizing the number of active switches. If VM-cluster p is mapped to a PM in $RACK_s$, then Γ_p^s is equal to 1, otherwise, it is equal to 0. If VM i is packed into VM-cluster p, Δ_p^i is equal to 1, otherwise it is equal to 0. If $\Gamma_p^s \Delta_p^i = 1$, VM i is placed into a PM in $RACK_s$. In a DCN with fat-tree topology, for flow $m_{i,j}$ between VMs i and j, if one of VMs i and j is packed into VM-cluster p, the other is not, flow $m_{i,j}$ crosses edge switch s, so we obtain:

$$\Omega^s_{m_{i,j}} = \Gamma^s_p(\Delta^i_p(1-\Delta^j_p) + \Delta^j_p(1-\Delta^i_p)) \tag{5.6}$$

where Δ^i_p, Δ^j_p indicates whether VM i, j is in VM-cluster p respectively.

The problem of minimizing active switches in the data center network under bandwidth guarantee in the second and third phases can be formulated as :

$$\text{Minimize} \sum_{s \in S} x_s \tag{5.7a}$$

$$\text{Subject to } \Omega^s_{m_{i,j}} = x_s\Gamma^s_p(\Delta^i_p(1-\Delta^j_p) + \Delta^j_p(1-\Delta^i_p))$$
$$(p \in W, s \in EDGE_k, 1 \leq k \leq \frac{K}{2}), \tag{5.7b}$$

$$\sum_{i,j \in T} e_{u,v}\Omega^u_{m_{i,j}}\Omega^v_{m_{i,j}} m_{i,j} \leq x_u x_v C_l,$$
$$(u \in S, v \in S), \tag{5.7c}$$

$$\sum_{s \in CORE} \Omega^s_{m_{i,j}} = \frac{1}{2}\sum_{k=1}^{K} \epsilon_k, \tag{5.7d}$$

$$\sum_{s \in AGGR_k} \Omega^s_{m_{i,j}} = (\frac{1}{2}(\sum_{m \in EDGE_k}\sum_{p \in RACK_m} \Delta_{i,p})$$
$$*(\sum_{m \in EDGE_k}\sum_{p \in RACK_m} \Delta_{j,p}) + \epsilon_k)\sum_{m \in EDGE_k} \delta_m,$$
$$(1 \leq k \leq K). \tag{5.7e}$$

The objective of the second phase is to find a VM-cluster mapping from W to PMs in data center to minimize inter-rack traffic and inter-pod traffic. This is QAP problem which is NP-hard.

To minimize inter-rack traffic and inter-pod traffic crossing in data center network, our solution is to map VM-clusters with heavy traffics to PMs located as close as possible in data center so that the flows among them cross minimum switches. Since the data center network is a three-tier fat-tree network, the solution to VM-cluster mapping is divided into two levels. In the first level, VM-clusters in W are mapped to PMs in rack level. The VM-cluster set W and the inter VM-cluster traffic matrix D can be modeled as a graph. The first level mapping is to cut the graph into $\lceil \frac{2|W|}{K} \rceil$ parts termed as rack-level VM-cluster groups W^+ while minimizing inter-part traffics termed as inter rack-level VM-cluster group traffics D^+. Each rack-level VM-cluster group has no more than $\frac{K}{2}$ VM-clusters. The rack-level VM-cluster group size is equal to the rack size. All VM-clusters in a rack-level VM-cluster group are mapped to PMs located in the same rack. This is a Balanced Minimum K-cut Problem (BMKP). After first level mapping, we obtain a set of rack-level VM-cluster groups W^+ and inter rack-level VM-cluster group traffic matrix D^+ which can be modeled as a graph as well, and this graph can be further cut into $\lceil \frac{2|W^+|}{K} \rceil$ parts termed as pod-level VM-cluster groups W^* while minimizing inter-part traffics termed as inter pod-level VM-cluster group traffics D^*. All rack-level VM-cluster groups in a pod-level VM-cluster group are mapped to racks located in the same pod. Because fat-tree is architecture of full bisection bandwidth connection, there is no communication congestion in data center network in this phase. Graph cuts in both levels mapping are similar.

We propose Algorithm VN-Mapping to resolve the BMKP problem described above. We sort flow $d_{p,q}$ while $p, q \in W$ in non-increasing order, we get two VM-clusters or VM-cluster set u and v with $d_{u,v} = \max_{p,q \in W} d_{p,q}$ into a new VM-cluster set. We denote $\hat{W}_k$ to be the set of VM-clusters mapped to $k-th$ rack. If VM-clusters u and v are not mapped into any rack, then map them to rack m which has the largest flow with them ($\sum_{j \in \hat{W}_m} d_{v,j} + d_{u,j} = \max_{1 \leq n \leq \lceil \frac{2|W|}{K} \rceil} \sum_{j \in W_n,} d_{u,j} + d_{v,j}$). If VM-clusters u or v is already mapped to some rack, the other VM-cluster is mapped to the same rack. If rack can not hold the VM-cluster pair, we continue on a VM-cluster pair with the next largest flows. We repeat the combining operations until all VM-clusters are mapped to racks.

After the first level VN mapping, we obtain a set of rack-level VM-cluster groups W^+ and inter-rack traffic matrix D^+. In the second level, we map the rack-level VM-cluster groups to pods while minimizing the number of active core switches. This is similar to the fist level mapping and can be solved by the same VN-Mapping algorithm.

Algorithm 3 VN-Mapping

Input: VM-cluster set W, and Traffic Matrix of VM-clusters D

Output: Rack-level VM-cluster group set $W^+ = \{\hat{W}_1, ..., \hat{W}_n\}$, and Traffic Matrix of rack-level VM-cluster groups D^+

```
 1: calculate needed rack number $n = \lceil \frac{2|W|}{K} \rceil$
 2: sort $d_{p,q}$ while $p, q \in W$ in non-increasing order
 3: **while** existing VM-clusters not mapped to any rack **do**
 4:     find VM-cluster or VM-cluster set pair $u$ and $v$ with the largest traffic between them $d_{u,v} = \max_{p,q \in W} d_{p,q}$
 5:     **if** both VM-clusters $u$ and $v$ are not mapped into any rack **then**
 6:         map the VM-clusters $u, v$ into rack $m$ whose VM-cluster set $\hat{W}_n$ has the largest traffic $\sum_{j \in \hat{W}_m} d_{u,j} + d_{v,j} = \max_{1 \leq n \leq \lceil \frac{2|W|}{K} \rceil} \sum_{j \in W_n} d_{u,j} + d_{v,j}$ to them, $\hat{W}_m \Leftarrow \hat{W}_m \cup$ VM-cluster $u \cup$ VM-cluster $v$
 7:     **else if** one of VM-clusters $u$ and $v$ is already mapped into rack **then**
 8:         map the other VM-cluster to the same rack
 9:     **end if**
10:     Recompute the traffic matrix $D$
11:     Sort $d_{p,q}$ in non-increasing order
12: **end while**
13: compute $D^+$ by $\{\hat{W}_1, ..., \hat{W}_n\}$
```

5.3.3 VM-Pair Flow Routing

After VMs are placed onto rack PMs, it is necessary to choose path for the flows of VM pairs routing in the data center network while minimizing active switches.

Four cases have been discussed in flow routing in the MODEL sector. Case 1 is ignored. A flow of case 2 crosses only one edge switch, so all edge switches which are connected to PMs placed with VMs must be active, the minimum active edge switches is $\rceil \frac{2W}{K} \lceil$. Flows of case 3 and 4 must cross aggregated switches and core switches in data center network, switches in aggregated and core tiers are needed to create paths for these flows.

Since fat-tree network is multi-root topology with network redundancy, not all switches in aggregated tier and core tier are required to work. Network traffic must be moved and aggregated onto a fewer number of paths so that the remaining network elements are put into dormant state for energy conservation.

Selecting routes for the flows of VM pairs placed in data center while minimizing active switches is Multi-Commodity Flow Problem [73] which is NP-hard.

Heller et al [73] proposed a Greedy Bin-Packing algorithm for flow routing. For each flow, the greedy bin-packer evaluates possible paths and chooses the leftmost one in fat-tree (see Figure (5.1)) with sufficient capacity. However, for some traffic matrices, the greedy approach will not find a satisfying assignment for all flows; this is an inherent problem with any greedy flow assignment strategy, even when the network is provisioned for full bisection bandwidth.

We proposed a heuristic algorithm termed Flow-Routing to assign paths for VM-pair flows in a three-tier fat-tree network, Flow-Routing can ensure bandwidth guarantee for all flows while minimizing active switches.

Like Greedy Bin-Packing algorithm proposed by Heller, for each flow, Flow-Routing evaluates possible paths and chooses the leftmost one with sufficient capacity, but under bandwidth guarantee.

We assign index $(1, ..., \frac{K}{2})$ to aggregated switches from left to right respectively in each pod. The $i - th$ aggregated switches of all pods are connected to $\frac{K}{2}$ core switches. So the core switches are divided into $\frac{K}{2}$ groups, we assign index to the groups from the left to right, each group index corresponds to the index of the aggregated switches it is connected to.

Given a set of VMs T along with traffic matrix M, we let M also to represent the set of VM-pair flows because each element $m_{i,j}$ while $i, j \in T$ is a VM-pair flow. We assume flows $m_{i,j}$ and $m_{j,i}$ are the same. We denote M^C is the set of flows of case 4, and denote M_i^A as set of flows of case 3 in pod i while $1 \leq k \leq K$.

Flow-Routing chooses active switches in two steps, Flow-Routing first chooses core switches, and then chooses aggregated switches in each pod.

In the first step. Flow-Routing sorts flows in M^C in non-increasing order. For each core switch from the left to the right in data center network, Flow-Routing selects the largest flow from M^C, Flow-Routing evaluates residual link bandwidth of the switch, if the switch has sufficient bandwidth for the flow, Flow-Routing allocates the link bandwidth of the switch to the flow and uses this switch along with the connected aggregated switches to create a path for the flow; otherwise, Flow-Routing continues next flow until the switch has no sufficient bandwidth for any flow. If existing some flows do not have allocated bandwidth, Flow-Routing selects the second leftmost core switch to allocate bandwidth for the flows as it does on the previous one. This allocation continues until all flows are allocated bandwidth.

Algorithm 4 Flow-Routing

Input: VM set T which have been already mapped into PMs on racks, and Traffic Matrix of VM M

Output: Active core switches and aggregated switches

```
1: select all flows $m_{i,j}$ where VM i and j are placed into different pods into set $M^C$.
2: sort flows in $M^C$ in non increasing order.
3: **for** each core switch $i$ in $CORE$ **do**
4:     **for** I=j to $|M^C|$ **do**
5:         select the largest flow $m_{p,q} = \max M^C$
6:         **if** core switch $i$ has enough link bandwidth for $m_{p,q}$ **then**
7:             allocate link bandwidth of core switch $i$ to $m_{p,q}$
8:             create path using core switch $i$ and aggregated switches in the same pods with VM $i$ $j$
               which core switch $i$ is connected to
9:         **end if**
10:    **end for**
11:    **if** All flows are allocated with link bandwidth **then**
12:        exit for
13:    **end if**
14: **end for**
```

In the second step, Flow-Routing does this work in each pod on flows of M_i^A while $1 \leq i \leq K$ as it does on flows in M^C. Since fat-tree network is in full bisection connection, the bandwidths allocating to the flows are ensured and no congestion occurs in the network. The time complexity of Algorithm Flow-Routing is $O(N^2 \log N)$ where N is the VM set size $|T|$.

5.4 Performance Evaluation

We evaluate the performance of Algorithm VM-Packing and Algorithm VN-Mapping by simulations. The traffic matrix is generated using log-normal distribution with density function shown in Equation (5.8). In all tests of log-normal distribution, the mean μ is set to 0 while the standard deviation σ is set to 1 or 1.5. Increasing σ, we can create a more scattered and higher total traffic between VMs.

$$f(x;\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(lnx-\mu)^2}{2\sigma^2}} \tag{5.8}$$

We test three cases of edge weight distributions, two log-normal distribution where σ is set to 1 and 1.5, and one standard normal distribution where edge weight varies between $[1, 2, 3, 4, 5]$ randomly.

5.4.1 VM Packing

Most existing VM placement solutions ignore the VM packing. They simply assume that each VM has the same size, and each PM supports only one VM [13] or a fixed number of VMs [59, 61]. Algorithm VMP [60] is the only other solution that considers the VM packing together with the VM placement. So, we compare Algorithm VM-Packing with Algorithm VMP. Both algorithms identify patterns by traffic matrix. However, Algorithm VMP does not have the merge step as Algorithm VM-Packing does.

We let the PM computation capacity C_s to be 200, and link bandwidth capacity C_l to be 100. In the graph of a tenant virtual network, the degree of a node varies from 0 to $|T|-1$. We assume for VM i while $i \in T$. We have $r_i \leq C_s$ and $\sum_{j\in T} m_{i,j} \leq C_l$; otherwise the tenant's request for the virtual network may be denied.

Both algorithms output a set of VM-clusters W and a inter VM-cluster traffic matrix D. Since each VM-cluster is hosted by one PM, the number of PMs needed equals to $|W|$ and the inter-PM traffic matrix is the same as that of VM-clusters. We let W' be the set of PMs which host VM-clusters and D' be the inter-PM traffic matrix resulted from Algorithm VMP. We let W be the set of PMs and D be the inter-PM traffic matrix resulted from Algorithm VM-Packing. We define the PM ratio η_s to be $\frac{|W'|}{|W|}$ and the inter-PM traffic ratio η_t to be $\frac{\sum_{i,j\in W'} d'_{i,j}}{\sum_{i,j\in W} d_{i,j}}$.

First, we simulate the number of VMs of a tenant virtual network. Figure (5.8) and (5.9) show

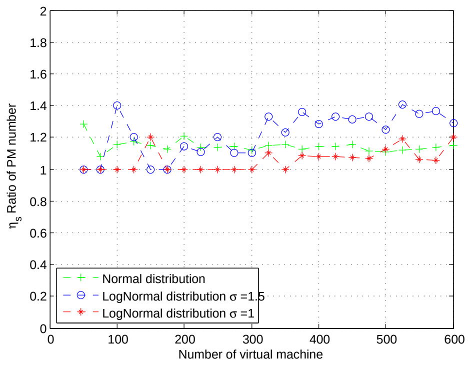

Figure 5.8: The PM ratio when the number of VMs ranged from 50 to 600 stepped by 25, and the mean degree of VMs was 8.

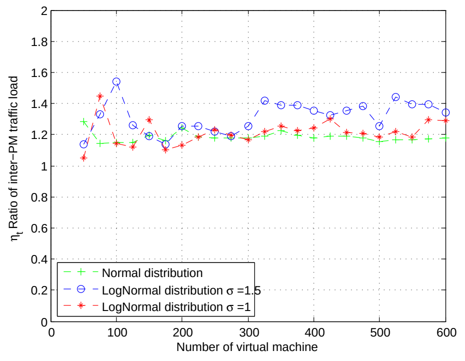

Figure 5.9: The inter-PM traffic ratio when the number of VMs ranged from 50 to 600 stepped by 25, and the mean degree of VMs was 8.

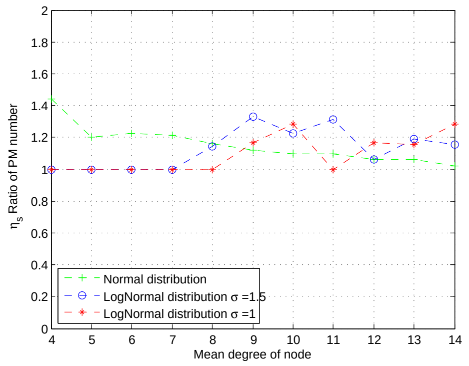

Figure 5.10: The PM ratio when the mean degree of VMs ranged from 4 to 14 stepped by 1, and the number of VMs was 200.

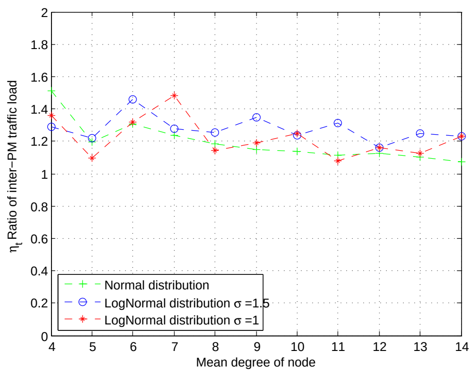

Figure 5.11: The inter-PM traffic ratio when the mean degree of VMs ranged from 4 to 14 stepped by 1, and the number of VMs was 200.

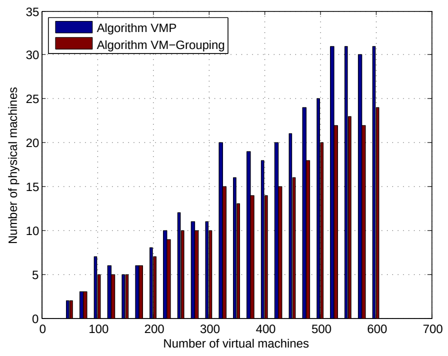

Figure 5.12: The number of PMs when the number of VMs ranged from 50 to 600 stepped by 25, and the mean degree of VMs was 8 in the log-normal distribution with $\sigma = 1.5$.

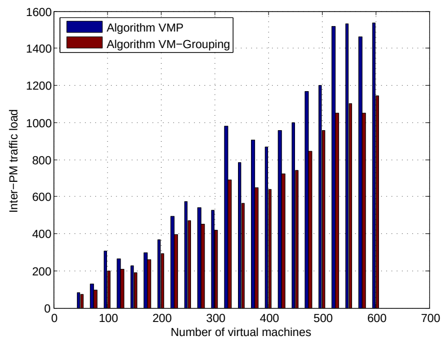

Figure 5.13: The total inter-PM traffic volumes when the number of VMs ranged from 50 to 600 stepped by 25, and the mean degree of VMs was 8 in the log-normal distribution with $\sigma = 1.5$.

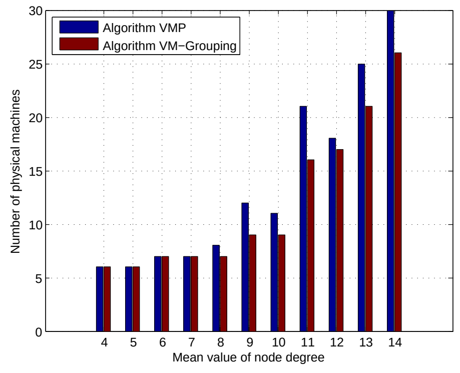

Figure 5.14: The number of PMs when the mean degree of VMs ranged from 4 to 14 stepped by 1 in the log-normal distribution with $\sigma = 1.5$, and the number of VMs was 200.

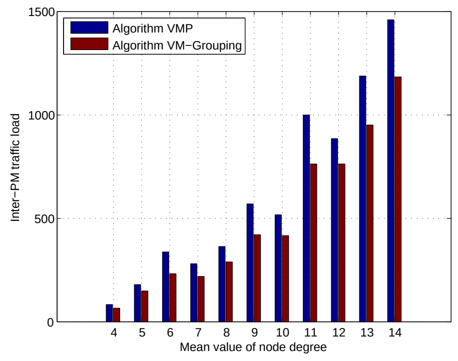

Figure 5.15: The number of PMs when the mean degree of VMs ranged from 4 to 14 stepped by 1 in the log-normal distribution with $\sigma = 1.5$, and the number of VMs was 200.

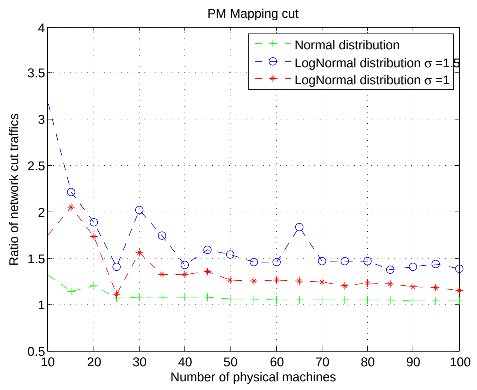

Figure 5.16: Simulation Results when data center network is fat-tree with $K = 16$, cut traffic of Algorithm VN-Mapping to Random

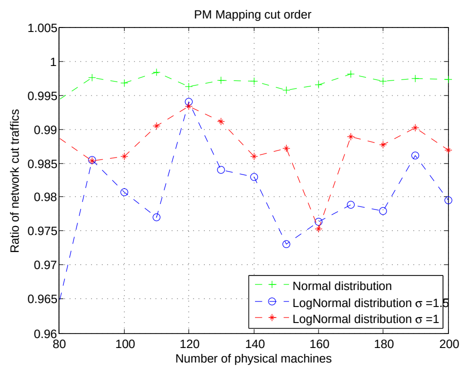

Figure 5.17: Simulation Results when data center network is fat-tree with $K = 16$, cut traffic of Algorithm VN-Mapping to Reverse cut

the PM ratio and inter-PM traffic ratio vary with the number of VMs which ranges from 50 to 600 with step size 25. The mean degree of VMs is 8. Three curves represent three edge distributions. From the figures, we observe the PM ratios η_s and inter-PM traffic ratios η_t are no less than 1 which demonstrates that Algorithm VM-Packing partitions the set of VMs into fewer PMs with fewer inter-PM traffics than Algorithm VMP does in all three cases. Compared to Algorithm VMP, Algorithm VM-Packing reduces the number of PMs by up to 20% when the number of VMs is less than 300 and by 20% to 40% when the number of VMs is more than 300 for the log-normal distribution with $\sigma = 1.5$. Further, it reduces the inter-PM traffics by 10% to 20% when the number of VMs is less than 300 and by 30% to 45% when the number of VMs is more than 300 for the log-normal distribution with $\sigma = 1.5$.

Algorithm VM-Packing is more efficient in log-normal distributions with $\sigma = 1.5$ than $\sigma = 1$, and more efficient in log-normal distributions than the normal distribution when the number of VMs is more than 300. From Figure (5.8) and (5.9), we also observe that when the number of VMs is small, the performances of both algorithms are almost the same in the log-normal distribution with $\sigma = 1$. Occasionally, Algorithm VMP is more efficient than Algorithm VM-Packing, because Algorithm VM-Packing is more suitable in Communication-Intensive cases where the traffic patterns between VMs are more distinct.

Second, we simulate the mean degree of VMs (nodes). Figure (5.10) and (5.11) show the PM ratio and inter-PM traffic ratio vary with the mean degree of nodes which ranges from 4 to 14 step by 1, when the number of VMs is 200. The curves show that Algorithm VM-Packing outperforms Algorithm VMP by 11% to 43% in term of inter-PM traffics. Algorithm VM-Packing performs similar to Algorithm VMP when the mean degree of nodes is less than 8 since the traffic is low. When the node degree increases, Algorithm VM-Packing becomes more efficient than Algorithm VMP.

Figure (5.12) to (5.15) plot test data output by two algorithms in log-normal distribution with $\sigma = 1.5$. Figure (5.12) and (5.13) show the number of PMs and the inter-PM traffics vary with the number of VMs, respectively. Figure (5.14) and (5.15) show the number of PMs and the inter-PM traffics vary with the mean degree of VMs, respectively.

5.4.2 Virtual Network Mapping

After the VM packing, we obtain a consolidated virtual network $G(W, D)$ with a set of VM-clusters W and inter VM-cluster traffic matrix D. Algorithm VN-Mapping maps the virtual network into data center networks while minimizing the number of active switches. We compared Algorithm VN-Mapping with a random algorithm where a VM-cluster has equal probability to be mapped to any PM.

The data center network is a three-tier homogeneous fat-tree topology with $K = 16$. Algorithm VN-Mapping must run two times to implement the VN mapping. We denote W^{+}_{Random} and W^{*}_{Random} the rack-level VM-cluster set and the pod-level VM-cluster set resulted from the random algorithm, respectively. We denote W^{+} and W^{*} the rack-level VM-cluster set and the pod-level VM-cluster set resulted from Algorithm VN-Mapping, respectively. Flows $d^{+}_{i,j}$ and $d^{*}_{i,j}$ represent the traffics carried out in the aggregation tier and the core tier, respectively. We define the ratio of total traffics η_m as $\frac{\sum_{i,j\in W^{+}} d^{+}_{i,j} + \sum_{i,j\in W^{*}} d^{*}_{i,j}}{\sum_{i,j\in W^{+}_{Random}} d^{+}_{i,j} + \sum_{i,j\in W^{*}_{Random}} d^{*}_{i,j}}$ where $\sum_{i,j\in W^{+}_{Random}} d^{+}_{i,j}$ and $\sum_{i,j\in W^{*}_{Random}} d^{*}_{i,j}$ are the sum of inter-rack traffics and the sum of inter-pod traffics obtained by the random algorithm respectively, and $\sum_{i,j\in W^{+}} d^{+}_{i,j}$ and $\sum_{i,j\in W^{*}} d^{*}_{i,j}$ are the sum of inter-rack traffics and the sum of inter-pod traffics obtained by Algorithm VN-Mapping respectively.

We test three distributions of edge weights. As shown in Figure (5.16), η_m varies with the number of PMs and is always greater than 1. Algorithm VN-Mapping outperforms the random algorithm in all cases. It is much more efficient in log-normal distribution with $\sigma = 1.5$ where traffic patterns are the most distinct and identifiable. When the number of PMs is bigger than 40, η_m of log-normal $\sigma = 1.5$ is close to 1.5, η_m of log-normal $\sigma = 1$ is close to 1.15, and η_m of normal distribution is close to 1.

Algorithm VN-Mapping must run two times to implement the virtual network mapping, but two runs could be in different orders. In the normal order, it cuts the VM-cluster set into the rack-level VM-cluster groups first, and then cuts the rack-level groups into the pod-level VM-cluster groups. In the reverse order, it cuts the VM-cluster set into the pod-level VM-cluster groups first, and then cuts each pod-level VM-cluster group into the rack-level VM-cluster groups. As shown in Figure (5.17), the normal order performs better than the reverse order.

5.5 Summary

This chapter explores virtual network resource allocation problem in multi-tenant data centers. This chapter proposes a solution to map a virtual network into a data center network while minimizing the total numbers of servers and switches required under the constraints of the link bandwidth capacity and server computation capacity. The solution consists of three phases: traffic-aware virtual machine (VM) packing, network-aware virtual network (VN) mapping, and power-aware flow routing. VM Packing minimizes the number of physical machines (servers) required to host all the VMs. VN Mapping assigns VM-clusters to physical servers in a way that VM pairs with more traffic are placed closer to minimize network traffic. Flow routing consolidates flows to a smaller number of links/switches, and thus leaves more idle links and switches to be powered off. Algorithms in all three phase work under the bandwidth guarantees for flows. The experimental results show our solutions significantly outperform the existing VM placement solutions.

CHAPTER 6

Conclusions

Cloud data centers are becoming increasingly popular for providing computing and storage resources. However, the energy consumption of these data centers has skyrocketed and is becoming a heavy burden to the cloud providers. The energy consumption of cloud computing data centers comes mainly from three sources: servers, network switches, and cooling system. There are some energy-efficient techniques used to reduce energy consumption in data centers such as virtualization, Dynamic Voltage/Frequency Scaling (DVFS), and Dynamic Power Management (DPM). Virtualization can consolidate the workload of a data center into the minimum number of servers and switches, and then turn off unneeded servers and switches to save power. DVFS can reduce the power consumption of a server by adjusting the working speed of the server. DPM can switch server power modes between the sleep mode and the active mode to reduce the power consumption. However, reducing power consumption of data centers may affect data center performance. For example, the response time of a job may be extended so long that it misses its deadline, or the bandwidth of a flow is not satisfied that affects the application performance. Therefore, the performance of applications must be taken into account while minimizing the energy consumption of data centers.

This thesis analyzed server and network energy consumption and proposed solutions to improve energy efficiency of cloud data centers under the performance constraint.

First, this thesis introduces a smart power saving scheme, PowerSleep, which aims at power saving for a single server. PowerSleep can minimize power consumption of a single server under the mean job response time constraint. PowerSleep adjusts the server working frequency during running time by DVFS, and puts the server into the sleep mode once the queue is empty, and activates the server to work once a new job arrives. To overcome the transition overhead, PowerSleep adds procrastination sleep period when a new job arrives while the severs still in the sleep mode. The server keeps sleeping during the procrastination sleep period to collect more jobs into the queue. After the procrastination sleeping time, the sever wakes up to process jobs. This approach reduces the mode transition overhead, but it increases the job response time.

Second, the thesis investigates energy-efficient real-time task scheduling in multi-core processors with voltage island model. This thesis introduces an energy-efficient algorithm Voltage Island Largest Capacity First (VILCF) to schedule real-time tasks into cores. VILCF fully utilizes the remaining capacity of cores without increasing power consumption of a chip. This thesis presents detailed theoretical analysis of the approximation ratio of the proposed VILCF algorithm in terms of energy efficiency.

Finally, the thesis explores the energy-efficient resource allocation problem for tenants in cloud data centers while under bandwidth guarantees. The solution consists of three phases:VM packing, VN mapping, and Flow routing. It is a complex problem, even each phase is an NP-hard problem. The first two phases VM packing and VN mapping are a graph partitioned problem. The goal is to put VMs with heavy traffics as close as possible. The third phase Flow routing aggregates VM-pair flows to the minimum number of switches. The proposed solution enhances resource utilization in data centers and thus reduce the energy consumption.

In conclusion, substantial contributions of this thesis in optimizing the energy consumption of servers and networks in data centers will enable cloud providers to offer scalable services with lower energy consumption, costs, and CO2 emissions.

BIBLIOGRAPHY

[1] Amazon, E., “Amazon elastic compute cloud (Amazon EC2),” *Amazon Elastic Compute Cloud (Amazon EC2)*, 2010.

[2] Thibodeau, P., “Data centers are the new polluters,” http://www.computerworld.com/article/2598562/data-center/data-centers-are-the-new-polluters.html, Accessed: 07-15-2015.

[3] Liu, J., Zhao, F., Liu, X., and He, W., “Challenges towards elastic power management in internet data centers,” *Distributed Computing Systems Workshops, 2009. ICDCS Workshops' 09. 29th IEEE International Conference on*, IEEE, 2009, pp. 65–72.

[4] Srikantaiah, S., Kansal, A., and Zhao, F., “Energy aware consolidation for cloud computing,” *Proceedings of the 2008 conference on Power aware computing and systems*, Vol. 10, San Diego, California, 2008.

[5] Lackey, D. E., Zuchowski, P. S., Bednar, T. R., Stout, D. W., Gould, S. W., and Cohn, J. M., “Managing power and performance for system-on-chip designs using voltage islands,” *Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on*, IEEE, 2002, pp. 195–202.

[6] Buurma, J. and Cooke, L., “Low-power design using multiple VTH ASIC librarie,” *SoC central*, 2004.

[7] Choi, K., *Dynamic Voltage and Frequency Scaling for energy-efficient system design*, University of Southern California, 2005.

[8] Teodorescu, R. and Torrellas, J., “Variation-aware application scheduling and power management for chip multiprocessors,” *ACM SIGARCH Computer Architecture News*, Vol. 36, IEEE Computer Society, 2008, pp. 363–374.

[9] Suh, J., Kang, D.-I., and Crago, S. P., “Dynamic power management of multiprocessor systems,” *Parallel and Distributed Processing Symposium, International*, Vol. 2, IEEE Computer Society, 2002, pp. 0097b–0097b.

[10] Ballani, H., Costa, P., Karagiannis, T., and Rowstron, A., “Towards predictable datacenter networks,” *ACM SIGCOMM Computer Communication Review*, Vol. 41, ACM, 2011, pp. 242–253.

[11] Lee, J., Turner, Y., Lee, M., Popa, L., Banerjee, S., Kang, J.-M., and Sharma, P., "Application-driven bandwidth guarantees in datacenters," *Proceedings of the 2014 ACM conference on SIGCOMM*, ACM, 2014, pp. 467–478.

[12] Popa, L., Yalagandula, P., Banerjee, S., Mogul, J. C., Turner, Y., and Santos, J. R., "Elastic-switch: practical work-conserving bandwidth guarantees for cloud computing," *ACM SIGCOMM Computer Communication Review*, Vol. 43, ACM, 2013, pp. 351–362.

[13] Guo, C., Lu, G., Wang, H. J., Yang, S., Kong, C., Sun, P., Wu, W., and Zhang, Y., "Secondnet: a data center network virtualization architecture with bandwidth guarantees," *Proceedings of the 6th International COnference*, ACM, 2010, p. 15.

[14] Benson, T., Akella, A., Shaikh, A., and Sahu, S., "CloudNaaS: a cloud networking platform for enterprise applications," *Proceedings of the 2nd ACM Symposium on Cloud Computing*, ACM, 2011, p. 8.

[15] Wang, S., Liu, J., Chen, J.-J., and Liu, X., "Powersleep: a smart power-saving scheme with sleep for servers under response time constraint," *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, Vol. 1, No. 3, 2011, pp. 289–298.

[16] Liu, J. and Guo, J., "Energy efficient scheduling of real-time tasks on multi-core processors with voltage islands," *Future Generation Computer Systems*, 2015.

[17] Wang, S., Munawar, W., Liu, J., Chen, J.-J., and Liu, X., "Power-saving design for server farms with response time percentile guarantees," *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012 IEEE 18th*, IEEE, 2012, pp. 273–284.

[18] Wang, S., Chen, J.-J., Liu, J., and Liu, X., "Power saving design for servers under response time constraint," *Real-Time Systems (ECRTS), 2010 22nd Euromicro Conference on*, IEEE, 2010, pp. 123–132.

[19] Liu, J. and Guo, J., "Voltage Island Aware Energy Efficient Scheduling of Real-Time Tasks on Multi-core Processors," *High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS), 2014 IEEE Intl Conf on*, IEEE, 2014, pp. 645–652.

[20] Liu, J. and Guo, J., "Traffic and Energy Aware Virtual Network Resource Allocation in Data Centers," *The 7th IEEE International Conference on Cloud Computing Technology and Science*, IEEE, 2015.

[21] Von Kaenel, V., Macken, P., and Degrauwe, M. G., "A voltage reduction technique for battery-operated systems," *Solid-State Circuits, IEEE Journal of*, Vol. 25, No. 5, 1990, pp. 1136–1140.

[22] Kim, N. S., Austin, T., Baauw, D., Mudge, T., Flautner, K., Hu, J. S., Irwin, M. J., Kandemir, M., and Narayanan, V., "Leakage current: Moore's law meets static power," *computer*, Vol. 36, No. 12, 2003, pp. 68–75.

[23] Kolpe, T., Zhai, A., and Sapatnekar, S. S., "Enabling improved power management in multi-core processors through clustered DVFS," *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, IEEE, 2011, pp. 1–6.

[24] Raje, S. and Sarrafzadeh, M., "Variable voltage scheduling," *Proceedings of the 1995 international symposium on Low power design*, ACM, 1995, pp. 9–14.

[25] Chang, J.-M. and Pedram, M., "Energy minimization using multiple supply voltages," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, Vol. 5, No. 4, 1997, pp. 436–443.

[26] Wu, H., Liu, I.-M., Wong, M. D., and Wang, Y., "Post-placement voltage island generation under performance requirement," *Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on*, IEEE, 2005, pp. 309–316.

[27] Benini, L., Bogliolo, A., and De Micheli, G., "A survey of design techniques for system-level dynamic power management," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, Vol. 8, No. 3, 2000, pp. 299–316.

[28] Brock, B. and Rajamani, K., "Dynamic power management for embedded systems," *Proceedings of the IEEE SOC Conference*, 2003, pp. 1–25.

[29] Soteriou, V. and Peh, L.-S., "Dynamic power management for power optimization of interconnection networks using on/off links," *High Performance Interconnects, 2003. Proceedings. 11th Symposium on*, IEEE, 2003, pp. 15–20.

[30] Geer, D., "Chip makers turn to multicore processors," *Computer*, Vol. 38, No. 5, 2005, pp. 11–13.

[31] Krishnan, V. and Torrellas, J., "A chip-multiprocessor architecture with speculative multithreading," *Computers, IEEE Transactions on*, Vol. 48, No. 9, 1999, pp. 866–880.

[32] Talpes, E. and Marculescu, D., "Toward a multiple clock/voltage island design style for power-aware processors," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, Vol. 13, No. 5, 2005, pp. 591–603.

[33] Hu, J., Shin, Y., Dhanwada, N., and Marculescu, R., "Architecting voltage islands in core-based system-on-a-chip designs," *Proceedings of the 2004 international symposium on Low power electronics and design*, ACM, 2004, pp. 180–185.

[34] Lee, W.-P., Liu, H.-Y., and Chang, Y.-W., "Voltage island aware floorplanning for power and timing optimization," *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, ACM, 2006, pp. 389–394.

[35] Yang, S., Wolf, W., Vijaykrishnan, N., and Xie, Y., "Reliability-aware soc voltage islands partition and floorplan," *Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium on*, IEEE, 2006, pp. 6–pp.

[36] Pillai, P. and Shin, K. G., "Real-time dynamic voltage scaling for low-power embedded operating systems," *ACM SIGOPS Operating Systems Review*, Vol. 35, ACM, 2001, pp. 89–102.

[37] Aydin, H., Melhem, R., Mossé, D., and Mejia-Alvarez, P., "Dynamic and aggressive scheduling techniques for power-aware real-time systems," *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS)*, IEEE, 2001, pp. 95–105.

[38] Chen, J.-J., Hsu, H.-R., and Kuo, T.-W., "Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems," *Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE*, IEEE, 2006, pp. 408–417.

[39] Aydin, H. and Yang, Q., "Energy-aware partitioning for multiprocessor real-time systems," *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, 2003, pp. 113–121.

[40] Mishra, R., Rastogi, N., Zhu, D., Mossé, D., and Melhem, R., "Energy aware scheduling for distributed real-time systems," *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, 2003, pp. 21–21.

[41] AlEnawy, T. A. and Aydin, H., "Energy-aware task allocation for rate monotonic scheduling," *Proceedings of the 11th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS)*, IEEE, 2005, pp. 213–223.

[42] Seo, E., Jeong, J., Park, S., and Lee, J., "Energy efficient scheduling of real-time tasks on multicore processors," *Parallel and Distributed Systems, IEEE Transactions on*, Vol. 19, No. 11, 2008, pp. 1540–1552.

[43] Qi, X. and Zhu, D.-K., "Energy Efficient Block-Partitioned Multicore Processors for Parallel Applications," *Journal of Computer Science and Technology*, Vol. 26, No. 3, 2011, pp. 418–433.

[44] Ozturk, O., Kandemir, M., and Chen, G., "Compiler-directed energy reduction using dynamic voltage scaling and voltage islands for embedded systems," *Computers, IEEE Transactions on*, Vol. 62, No. 2, 2013, pp. 268–278.

[45] Pagani, S. and Chen, J.-J., "Energy efficiency analysis for the single frequency approximation (SFA) scheme," *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 13, No. 5s, 2014, pp. 158.

[46] Devadas, V. and Aydin, H., "Coordinated power management of periodic real-time tasks on chip multiprocessors," *Green Computing Conference, 2010 International*, IEEE, 2010, pp. 61–72.

[47] Fanxin, K., Wang, Y., and Qingxu, D., "Energy-Efficient Scheduling of Real-Time Tasks on Cluster-Based Multicores ," *EDAA*, 2011.

[48] Sheshadri, V., Agrawal, V. D., and Agrawal, P., "Power-aware SoC test optimization through dynamic voltage and frequency scaling," *Very Large Scale Integration (VLSI-SoC), 2013 IFIP/IEEE 21st International Conference on*, IEEE, 2013, pp. 102–107.

[49] Bari, M. F., Boutaba, R., Esteves, R., Granville, L. Z., Podlesny, M., Rabbani, M. G., Zhang, Q., and Zhani, M. F., "Data center network virtualization: A survey," *Communications Surveys & Tutorials, IEEE*, Vol. 15, No. 2, 2013, pp. 909–928.

[50] Stage, A. and Setzer, T., "Network-aware migration control and scheduling of differentiated virtual machine workloads," *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, IEEE Computer Society, 2009, pp. 9–14.

[51] Beloglazov, A. and Buyya, R., "Energy efficient resource management in virtualized cloud data centers," *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, IEEE Computer Society, 2010, pp. 826–831.

[52] Wu, C.-M., Chang, R.-S., and Chan, H.-Y., "A green energy-efficient scheduling algorithm using the DVFS technique for cloud datacenters," *Future Generation Computer Systems*, Vol. 37, 2014, pp. 141–147.

[53] Wang, L., Zhang, F., Arjona Aroca, J., Vasilakos, A. V., Zheng, K., Hou, C., Li, D., and Liu, Z., "GreenDCN: A general framework for achieving energy efficiency in data center networks," *Selected Areas in Communications, IEEE Journal on*, Vol. 32, No. 1, 2014, pp. 4–15.

[54] Wang, M., Meng, X., and Zhang, L., "Consolidating virtual machines with dynamic bandwidth demand in data centers," *INFOCOM, 2011 Proceedings IEEE*, IEEE, 2011, pp. 71–75.

[55] Popa, L., Kumar, G., Chowdhury, M., Krishnamurthy, A., Ratnasamy, S., and Stoica, I., "FairCloud: sharing the network in cloud computing," *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, ACM, 2012, pp. 187–198.

[56] Rodrigues, H., Santos, J. R., Turner, Y., Soares, P., and Guedes, D., "Gatekeeper: Supporting Bandwidth Guarantees for Multi-tenant Datacenter Networks." *WIOV*, 2011.

[57] Jiang, J. W., Lan, T., Ha, S., Chen, M., and Chiang, M., "Joint VM placement and routing for data center traffic engineering," *INFOCOM, 2012 Proceedings IEEE*, IEEE, 2012, pp. 2876–2880.

[58] Wang, S.-H., Huang, P. P.-W., Wen, C. H.-P., and Wang, L.-C., "EQVMP: Energy-efficient and QoS-aware virtual machine placement for software defined datacenter networks," *Information Networking (ICOIN), 2014 International Conference on*, IEEE, 2014, pp. 220–225.

[59] Meng, X., Pappas, V., and Zhang, L., "Improving the scalability of data center networks with traffic-aware virtual machine placement," *INFOCOM, 2010 Proceedings IEEE*, IEEE, 2010, pp. 1–9.

[60] Dias, D. S. and Costa, L. H. M., "Online traffic-aware virtual machine placement in data center networks," *Global Information Infrastructure and Networking Symposium (GIIS), 2012*, IEEE, 2012, pp. 1–8.

[61] Fang, W., Liang, X., Li, S., Chiaraviglio, L., and Xiong, N., "VMPlanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers," *Computer Networks*, Vol. 57, No. 1, 2013, pp. 179–196.

[62] Li, X., Wu, J., Tang, S., and Lu, S., "Let's stay together: Towards traffic aware virtual machine placement in data centers," *INFOCOM, 2014 Proceedings IEEE*, IEEE, 2014, pp. 1842–1850.

[63] Duffield, N. G., Goyal, P., Greenberg, A., Mishra, P., Ramakrishnan, K. K., and van der Merive, J. E., "A flexible model for resource management in virtual private networks," *ACM SIGCOMM Computer Communication Review*, Vol. 29, ACM, 1999, pp. 95–108.

[64] Zhu, J., Li, D., Wu, J., Liu, H., Zhang, Y., and Zhang, J., "Towards bandwidth guarantee in multi-tenancy cloud computing networks," *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, IEEE, 2012, pp. 1–10.

[65] Ballani, H., Jang, K., Karagiannis, T., Kim, C., Gunawardena, D., and O'Shea, G., "Chatty Tenants and the Cloud Network Sharing Problem." *NSDI*, 2013, pp. 171–184.

[66] Xie, D., Ding, N., Hu, Y. C., and Kompella, R., "The only constant is change: incorporating time-varying network reservations in data centers," *ACM SIGCOMM Computer Communication Review*, Vol. 42, No. 4, 2012, pp. 199–210.

[67] Shen, M., Gao, L., Xu, K., and Zhu, L., "Achieving bandwidth guarantees in multi-tenant cloud networks using a dual-hose model," *Performance Computing and Communications Conference (IPCCC), 2014 IEEE International*, IEEE, 2014, pp. 1–8.

[68] Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S., "VL2: a scalable and flexible data center network," *ACM SIGCOMM Computer Communication Review*, Vol. 39, ACM, 2009, pp. 51–62.

[69] Niranjan Mysore, R., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., Subramanya, V., and Vahdat, A., "Portland: a scalable fault-tolerant layer 2 data center network fabric," *ACM SIGCOMM Computer Communication Review*, Vol. 39, ACM, 2009, pp. 39–50.

[70] Al-Fares, M., Loukissas, A., and Vahdat, A., "A scalable, commodity data center network architecture," *ACM SIGCOMM Computer Communication Review*, Vol. 38, No. 4, 2008, pp. 63–74.

[71] Guo, C., Wu, H., Tan, K., Shi, L., Zhang, Y., and Lu, S., "Dcell: a scalable and fault-tolerant network structure for data centers," *ACM SIGCOMM Computer Communication Review*, Vol. 38, ACM, 2008, pp. 75–86.

[72] Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., Tian, C., Zhang, Y., and Lu, S., "BCube: a high performance, server-centric network architecture for modular data centers," *ACM SIGCOMM Computer Communication Review*, Vol. 39, No. 4, 2009, pp. 63–74.

[73] Heller, B., Seetharaman, S., Mahadevan, P., Yiakoumis, Y., Sharma, P., Banerjee, S., and McKeown, N., "ElasticTree: Saving Energy in Data Center Networks." *NSDI*, Vol. 10, 2010, pp. 249–264.

[74] Shang, Y., Li, D., and Xu, M., "Energy-aware routing in data center network," *Proceedings of the first ACM SIGCOMM workshop on Green networking*, ACM, 2010, pp. 1–8.

[75] Alonso, M., Coll, S., Martínez, J.-M., Santonja, V., López, P., and Duato, J., "Dynamic power saving in fat-tree interconnection networks using on/off links," *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, IEEE, 2006, pp. 8–pp.

[76] Kliazovich, D., Bouvry, P., and Khan, S. U., "DENS: data center energy-efficient network-aware scheduling," *Cluster computing*, Vol. 16, No. 1, 2013, pp. 65–75.

[77] Barroso, L. A. and Hölzle, U., "The case for energy-proportional computing," *Computer*, , No. 12, 2007, pp. 33–37.

[78] Levy, H. and Kleinrock, L., "A queue with starter and a queue with vacations: delay analysis by decomposition," *Operations Research*, Vol. 34, No. 3, 1986, pp. 426–436.

[79] Michael, R. G. and David, S. J., "Computers and intractability: a guide to the theory of NP-completeness," *WH Freeman & Co., San Francisco*, 1979.

[80] Chen, J.-J., Hsu, H.-R., Chuang, K.-H., Yang, C.-L., Pang, A.-C., and Kuo, T.-W., "Multiprocessor energy-efficient scheduling with task migration considerations," *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*, IEEE, 2004, pp. 101–108.

[81] Andreev, K. and Racke, H., "Balanced graph partitioning," *Theory of Computing Systems*, Vol. 39, No. 6, 2006, pp. 929–939.

[82] Anderson, R. J., Mayr, E. W., and Warmuth, M. K., "Parallel approximation algorithms for bin packing," *Information and Computation*, Vol. 82, No. 3, 1989, pp. 262–277.

[83] Leinberger, W., Karypis, G., and Kumar, V., "Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints," *1999 International Conference on Parallel Processing*, IEEE, 1999, pp. 404–412.