HAL
open science

Energy-efficient resource provisioning for cloud databases

Chaopeng Guo

To cite this version:

Chaopeng Guo. Energy-efficient resource provisioning for cloud databases. Networking and Internet Architecture [cs.NI]. Université Paul Sabatier - Toulouse III, 2019. English. NNT : 2019TOU30065 . tel-02896451

HAL Id: tel-02896451

https://theses.hal.science/tel-02896451v1

Submitted on 10 Jul 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse 3 - Paul Sabatier

Présentée et soutenue par

CHAOPENG GUO

Le 14 juin 2019

Allocation de ressources efficace en énergie pour les Bases de Données dans le Cloud

---

Ecole doctorale : **EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse**

Spécialité : **Informatique et Télécommunications**

Unité de recherche :

Thèse dirigée par
Jean-Marc PIERSON

Jury
M. Djamal BENSLIMANE, Rapporteur
M. Sébastien MONNET, Rapporteur
M. Shadi IBRAHIM, Examinateur
M. Abdelkader HAMEURLAIN, Examinateur
M. Laurent LEFÈVRE, Examinateur
Mme Joanna MOULIERAC, Examinatrice
M. Jean-Marc PIERSON, Directeur de thèse

Abstract

Today a lot of cloud computing and cloud database techniques are adopted both in industry and academia to face the arrival of the big data era. Meanwhile, energy efficiency and energy saving become a major concern in data centers, which are in charge of large distributed systems and cloud databases.

However, energy efficiency and service-level agreement of cloud databases are suffering from resource provisioning, resource over-provisioning and resource under-provisioning, namely that there is a gap between resource provided and resource required. Since the usage of cloud database is dynamical, resource of the system should be provided according to its workload.

In this thesis, we present our work on energy-efficient resource provisioning for cloud databases that utilizes dynamic voltage and frequency scaling (DVFS) technique to cope with resource provisioning issues. Additionally, a migration approach is introduced to improve the energy efficiency of cloud database systems further. Our contribution can be summarized as follows:

- At first, the behavior of energy efficiency of the cloud database system under DVFS technique is analyzed. Based on the benchmark result, two frequency selection approaches are proposed.

- Then, a frequency selection approach with bounded problem is introduced, in which the power consumption and migration cost are treated separately. A linear programming algorithm and a multi-phases algorithm are proposed. Because of the huge solution space, both algorithms are compared within a small case, while the multi-phases algorithm is evaluated with larger cases.

- Further, a frequency selection approach with optimization problem is introduced, in which the energy consumption for executing the workload and migration cost are handled together. Two algorithms, a genetic based algorithm and a monte carlo tree search based algorithm are proposed. Both algorithms have their pros and cons.

- At last, a migration approach is introduced to migrate data according to the given frequencies and current data layout. A migration plan can be obtained within polynomial time by the proposed Constrained MHTM algorithm.

Résumé

Aujourd'hui, beaucoup de techniques de cloud computing et de bases de données dans le cloud sont adoptées dans l'industrie et le monde universitaire pour faire face à l'arrivée de l'ère du big data. Parallèlement, l'efficacité énergétique et les économies d'énergie deviennent une préoccupation majeure pour les centres de données, qui sont en charge de grands systèmes distribués et de bases de données dans le cloud.

Toutefois, l'efficacité énergétique et l'accord de niveau de service des bases de données dans le cloud souffrent d'un problème d'allocation en ressources, de sur-allocation et de sous-allocation, c'est-à-dire qu'il y a un écart entre les ressources fournies et les ressources requises. Comme l'utilisation des bases de données dans le cloud est dynamique, les ressources du système devraient être fournies en fonction de sa charge de travail.

Dans cette thèse, nous présentons nos recherches sur l'allocation de ressources efficace en énergie pour les bases de données dans le cloud, utilisant des techniques d'ajustement dynamique de la tension et de la fréquence (dynamic voltage and frequency scaling, DVFS for short) pour résoudre les problèmes d'allocation en ressources. De plus, une approche de migration est introduite pour améliorer davantage l'efficacité énergétique des systèmes de bases de données dans le cloud. Notre contribution peut se résumer comme suit:

- Dans un premier temps, le comportement de l'efficacité énergétique du système de base de données dans le cloud utilisant des techniques DVFS est analysé. En fonction des résultats du benchmark, deux approches de sélection des fréquences sont proposées.

- Ensuite, une approche de type problème borné est introduite pour la sélection de la fréquence. Avec cette approche, la consommation d'énergie et le coût de migration sont traités séparément. Un programme linéaire et un algorithme multi-phases sont proposés. Puisque l'espace de solution est très grand, les deux algorithmes sont comparés avec un petit cas, tandis que l'algorithme multi-phases est évalué avec des cas plus grands.

- En outre, une approche de type problème d'optimisation est introduite pour la sélection de la fréquence. Avec cette approche, la consommation d'énergie et le coût de migration sont traités comme un tout. Un algorithme génétique ainsi qu'un algorithme fondé sur la recherche arborescente Monte-Carlo sont proposés. Chacun des deux algorithmes présente des avantages et des inconvénients.

- Enfin, une approche de migration est introduite pour migrer les données en fonction des fréquences données et de leur disposition actuelle. Un plan de migration peut être obtenu en temps polynomial grâce à l'algorithme Constrictif MTHM proposé.

Acknowledgments

In the past three years, I have been working as a PhD candidate in SEPIA group, IRIT Lab at Université Toulouse 3 Paul Sabatier. In the last days of this period, my feeling is quite complicated and beyond description. I'm grateful for having met so many awesome people during this three years' time. Without their help and support, I wouldn't have been able to complete my work so fluently. I really appreciate everyone, and I even start to miss my staying here. Every time when I think my study and life here will end soon, I would fell rather fragile and emotional. Meanwhile, I am also looking forward to the next stage of my life. My feels are really complicated recently. I would like to quote a Chinese poem here to express my feelings for writing this acknowledgment. "岁月蹉跎， 白驹过隙， 三年行来， 感悟良多。 念及离别， 辗转反侧， 子 时提笔，心有所思，信手拈来，忆似水年华，感时光不复。" (Time is just flashing. Three years is like a short moment and has passed quickly. It has indeed inspired me deep in my heart. Considering the coming farewell, I really cannot take easily. With a pen at hand, here I write goodbye to the past three years and all the memories shall stay in my heart eternally.)

First of all, I would like to express my deepest gratitude to my advisor, Prof. Dr. Jean-Marc Pierson for his offer to work with him and his constant support, while still giving me the freedom to research on my own. 6 years ago, we first met each other at Northeastern University in Shenyang, and he deeply impressed me with his talent and passionate. He not only taught me the research principles, but also gave me a lot of support with my publications and this thesis. All the lectures he gave will be the valuable fortune of my life. Besides, I would like to express my appreciation for all the jury members for their suggestions on my thesis.

Secondly, I would like to express my appreciations for China Scholarship Council (CSC) who funded this research, and also my master mentor Prof. Dr. Jie Song who recommended me for this project. Without them, I could not have this opportunity working in the group.

Furthermore, I would like to thank all my colleagues and friends in IRIT.

Malik Irain, Gustavo Rostirolla, Léo Grange, and Tanissia Djemai, it's my pleasure to meet and work with you guys and thank you for making my life so colorful here. Also, I would like to express my appreciations for Tristan Salord, Dr. Ophélie Fraisier, Dr. Minh-Thuyen Thi, Dr. Zongyi Liu and Dr. Jingyi Wang, which I have benefited a lot from your encouragements.

Last but not least, I want to thank my families for their love and support. I was kind of village boy from a small town in China. My parents, Yuzhang Guo and Xianyun Hao, always supported me from my first step with computers until my PhD. Without their support, I would never have been able to become who I am now. When I was in difficulties, they always acted as patient listeners and helped me to realize the goodness of the life and the meaning of love. Also, I would like to express my deepest gratitude to my wonderful partner Dejun Pan, who constantly encourages me to pursue my silly dreams and loves me unconditionally. Three years' long-distance relationship can be extremely challenging for all the couples. We succeeded. I used to think Love was an abstract class until you implemented it and initialized it in my heart. I used to be afraid of the lonely nights, but thanks to you I do not fear it anymore. Thank you for tolerating my unreasonable drama and comforting me from these nightmares.

In the end, I hope this thesis is not the end of academic thinking of mine. I hope that the former sentence is not just a hope.

Chaopeng Guo
Toulouse, April 2019

Contents

1 Introduction **1**
1.1 Motivation 1
1.2 Problem Statement 4
1.3 Research Goals 8
1.4 Overview 9

2 Related Works **11**
2.1 Cloud Database Systems 11
2.2 Energy Efficiency 14
2.2.1 Energy Efficiency Measurement 15
2.2.2 Power Modeling 15
2.2.3 Energy Efficiency Analysis and Improvement 17

3 Modeling **23**
3.1 Energy Efficiency in Cloud Databases 23
3.1.1 Methodology 23
3.1.2 Platform Setup 24
3.1.3 Experimental Cases 26
3.1.4 Result Analysis 27
3.1.5 Lessons Learned 30
3.2 Frequency Selection Model 30
3.2.1 Generic Model 30
3.2.2 Specialized Model for Cloud Databases 32
3.3 Model Simplification Approach 33
3.3.1 Power Consumption Upper Bound 34
3.3.2 Migration Cost Upper Bound 37
3.3.3 Complexity Analysis of Model Simplification 39
3.4 Parameters' Benchmark 40
3.5 Model Extension 40
3.6 Summary 41

4 Frequency Selection with Bounded Problem **43**
4.1 Objective . . . 43
4.1.1 Basic Objectives . . . 43
4.1.2 Bounded problems . . . 44
4.2 Nonlinear Programming Algorithm . . . 44
4.2.1 Nonlinear Programming Model . . . 45
4.2.2 Constraints . . . 47
4.2.3 Nonlinear Programming Objective Function . . . 48
4.3 Multi-Phases Algorithm . . . 49
4.3.1 Frequency Locating Phase . . . 51
4.3.2 Frequency Traversal Phase . . . 57
4.3.3 Frequency Assignment Phase . . . 62
4.3.4 Frequency Filter Phase . . . 65
4.3.5 Frequency Search Phase . . . 66
4.3.6 Frequency Evaluation Phase . . . 66
4.3.7 Summary and Complexity Analysis . . . 67
4.4 Experiment . . . 69
4.4.1 Setup . . . 69
4.4.2 Comparison Experiment . . . 71
4.4.3 Scalability Experiment . . . 73
4.5 Summary . . . 74

5 Frequency Selection with Optimization Problem **75**
5.1 Objective . . . 75
5.2 Genetic Algorithm . . . 76
5.3 Monte Carlo Tree Search Algorithm . . . 77
5.4 Experiment . . . 80
5.4.1 Parameters' Influence . . . 80
5.4.2 Scalability Analysis . . . 83
5.4.3 Optimization Bound Analysis . . . 84
5.4.4 Comparison with Half Hot and Half Cold Approach . . . 85
5.5 Algorithm Robustness Analysis . . . 86
5.6 Summary . . . 92

6 Migration Approach **93**
6.1 Objective . . . 93
6.2 Block Selection . . . 96
6.3 Block Migration Plan . . . 98
6.4 Experiment . . . 107
6.4.1 Migrated Block Comparison . . . 108
6.4.2 Migration Plan Generation Comparison . . . 110

6.4.3 Estimation Approach Evaluation 111
6.5 Summary 113

7 Conclusion and Perspectives **115**
7.1 Conclusion 115
7.2 Perspectives 117

List of Figures

1.1 Without Resource Provision Technique 5
1.2 With Resource Provision Technique 5

2.1 Architecture of Cassandra 13
2.2 Virtual Nodes 14

3.1 Relationship between Request Amount and Throughput 27
3.2 Relationship between Throughput and Energy Efficiency 28
3.3 Energy Consumption for each Workload 29
3.4 Example of Achieving Power Consumption Upper Bound 35
3.5 Example of p^{max} Proof: **Scenario 1** 36
3.6 Example of p^{max} Proof: **Scenario 2** 36
3.7 Example of Achieving Migration Cost Upper Bound 37

4.1 Frequency Solution Space 51
4.2 Example of Frequency Locating and Traverse 53
4.3 Frequency Combination Traverse Examples 61
4.4 Example of the Workload 69
4.5 Execution Time Comparison 71
4.6 Objective Value Comparison 72
4.7 Result of Frequency Selection Comparison 73
4.8 Result of Scalability Experiment 74

5.1 One iteration of the general MCTS approach 77
5.2 Frequency Selection Tree Example 78
5.3 The Influence of Generation Size on Accuracy 81
5.4 The Influence of Population Size on Accuracy 82
5.5 The Influence of Amount of Candidates on Accuracy 82
5.6 Scalability of the Frequency Selection Algorithm 83
5.7 Optimization Bound of the Frequency Selection Algorithm . . . 84
5.8 Comparison with Half Hot and Half Cold Approach 85
5.9 Relationship between ϕ and Root-Mean-Square Deviation 86

5.10 Example of Prediction Error Influence 87
5.11 The Influence of Prediction Errors 89
5.12 The Influence of Capacity Tolerance Factor 91

6.1 Migration Timeline 94
6.2 Result of Migrated Block Comparison 108
6.3 Migration Plan Generation Comparison 109
6.4 Evaluation Result of Migration Cost Estimation Approach . . . 111

List of Tables

3.1 Core Properties of Cassandra 25
3.2 Core Properties of YCBS Workload 25
3.3 Available Frequency Options 26
3.4 Workload Size Per Node 26
3.5 Node's Capacity under Each Frequency Option 28
3.6 Migration Cost for Unit Block 40

4.1 Specifications of the symbols 47
4.2 Example of Frequency Combination Locating 56
4.3 Complexity of Operations 68
4.4 Properties of the Test Cases for Nonlinear Programming Algorithm 71

6.1 1-0 Knapsack Details in Block Selection Phase 96
6.2 Multiple Knapsacks Details in Block Migration Phase 99
6.3 Specifications of the symbols 100
6.4 The Amount of Valid Solutions of 1000 Solutions 108

Chapter 1

Introduction

Nowadays, more and more cloud database systems have been implemented and deployed to meet users' rapidly growing data querying need. With the rapid growth of the data volume and construction of data centers, the problems of energy efficiency and energy waste attract more attention. Energy saving and energy efficiency of cloud database systems get more and more attentions. However, the typical cloud database system suffers from inefficient resource provisioning that causes energy waste. In this thesis, we show our researches on energy-efficient resource provisioning for cloud database system to improve its energy efficiency. In this chapter, we give a comprehensive introduction about our work including motivation, problem statement, research goals and overview of this thesis.

1.1 Motivation

Over the last decades, concerns have been raised about increases in the electricity used by information technologies, and other consumer electronic devices, data centers, and to a much lesser degree, Internet distribution networks [1–3]. According to United States Data Center Energy Usage Report [3], in 2014, data centers in the U.S. consumed an estimated 70 billion kWh, representing about 1.8% of total U.S. electricity consumption. The consumption increased about 4% from 2010-2014 and it is expected to continue slightly increasing in the near future, increasing 4% from 2014-2020. It is estimated that powering digital devices and the supporting infrastructures consumed about 5% of global electricity use in 2012 [4–6]. Janine Morley *et al.* [1] reviewed evidences that network and data centre energy consumption are significant growing. With their consideration about the trends in data demand that underpin the expansion of these infrastructures and energy use, they concluded that the growth in

data traffic continue to outweigh efficiency gains, and increased data flows over mobile and Internet networks represent an increase in energy consumption. In term of the expenditure of energy used by data centers, it is estimated in [7] that the cost of powering and cooling accounts for 53% of the total operational expenditure of data centers, and the energy bills for major cloud service providers are typically the second largest item in their budgets [8]. Under these circumstances, a lot of researches have been made to reduce the energy consumption of data centers and corresponding cloud systems.

Cloud database system is one of the typical cloud systems. To cope with the huge data storage and query needs, massive cloud databases are established, such as, HBase [9], Hive [10], Cassandra [11] and MongoDB [12]. In a common case, a cloud database system contains hundred of nodes within the cluster. For example, the technical group from Instagram claimed that their biggest cluster contains 100 nodes in Cassandra summit 2016 [13]. In this thesis, the optimization goal is to improve energy efficiency of cloud database systems and reduce its energy waste. To improve energy efficiency of cloud database systems, a lot of researches have been done. For example, to achieve green computing, the measurement model and approach of the energy efficiency of cloud database systems were defined by Jie Song *et al.* [14], and the energy efficiency characteristics of cloud database were investigated. Similarly, Willis Lang *et al.* [15] studied the tradeoff between performance and energy efficiency for parallel database management systems to give principles of designing an energy-efficient database system. Typical cloud database systems are wasting energy. Since users' activities are dynamic, the workload of the system varies with time: users' activities are more intense in daytime, whereas they do little at night. In this case, part of the energy is wasted if the system's configuration at night remains the same as the one during daytime. In another case, in the query process, some hotspot areas exist, in which the workload is more intense than on other nodes. If all the nodes keep the same configuration, there might be Service-level agreement (SLA) violation or energy waste, even both. The energy waste and SLA violation come from resource provisioning.

Some self adjusting systems emerged to deal with resource provisioning in cloud systems, i.e. Ursa [16] and WattDB [17]. WattDB is a self adjusting distributed database management system, which switch nodes on and off according to current workload. Ursa, a self adjusting cloud storage system, migrates data from the hotspot nodes to underutilized nodes with minimizing latency and bandwidth manner, and turns off the underutilized nodes to save energy if possible. However, these strategies might not be suitable for cloud database systems since switching nodes on and off would cause an unacceptable migration cost and damage system availability. In WattDB, Daniel Schall

et al. took advantages of solid-state drive to reduce input and output cost during the migration process. In Ursa, Gae-Won You *et al.* tried to turn off the underutilized nodes which do not contain primary replicas. However in cloud database systems, write operations reach all the replicas eventually, while read operations reach one or more replicas according to the strategy of the system [18]. Therefore, Ursa's strategy cannot be applied within cloud database system, since some replicas might be unreachable when the underutilized nodes are shutdown.

Houssem-Eddine Chihoub *et al.* [19] introduced another idea to improve the energy efficiency of cloud database systems, in which they analyzed the tradeoff between energy and consistency of the system using Dynamic Voltage and Frequency Scaling (DVFS). DVFS [20] is an efficient technology to control power consumption of processors. Using DVFS, power control policies can be made. In modern Linux operating systems, five different power schemes (governors) are available to dynamically scale CPU frequency according to CPU utilization. The dynamic tuning strategies use CPU's running information to adjust its frequency, which does not reflect the state of cloud databases' workload lifecycle including memory and disk transfers. The current power schemes do not exactly match their design goal and may even become ineffective in improving energy efficiency [21]. Shadi Ibrahim *et al.* [21] compared energy consumption for three applications, Pi, $Grep$ and $Sort$, in Hadoop System under different frequency governs. They found out that, for all the applications, $PowerSave$ governor did not achieve lowest energy consumption, while $Performance$ governor did not achieve highest performance as well. For example, $Userspace$ governor with 2.53Ghz (the highest CPU frequency in their environment) achieves the best results both in terms of performance and energy efficiency. In Pi application, it achieve 104% better performance by employing the highest CPU frequency rather than the lowest frequency, 1.20 GHz, whereas the average power consumption increases by 48%. Therefore, it makes sense to control frequency in a fine-grained way. In [19], Houssem-Eddine Chihoub *et al.* proposed a property approach, in which they assigned highest frequency to half of the nodes and the lowest frequency to another half, and they achieved 23% of energy saving. However, this approach is a static method for assigning frequencies, and the usage scenario is limited.

The other previous researches [16, 17] show that the energy efficiency of cloud system can be improved by switching underutilized nodes. However, this strategy might unsuitable for cloud database systems because not only the migration process would introduce a huge migration cost, but also the availability of the system might suffer [22]. For example, Sudipto Das *et al.* [22] compared *Stop and Copy* migration strategy with their proposed *Iterative Copy* migra-

tion strategy. *Iterative Copy* has great performance in term of unavailability time, up to 3-5 times less than *Stop and Copy*. Meanwhile, in terms of failed operations,*Iterative Copy* is less than *Stop and Copy* up to 10 times.

In conclusion, it is necessary to propose a novel approach to improve energy efficiency of cloud database systems by providing energy-efficient resource provisioning approach. We identified the following properties:

- Availability: Considering the different constructions of cloud database systems, i.e. file system, consistency requirement and so on, the approach should be based on an abstraction of the cloud database system so that the approach can be applied to different architectures.
- Flexibility: Considering the dynamic workload nature of cloud database systems, the approach should adjust the resources of the system according to its workload.
- Scalability: Considering the scale of the cloud database system, the approach should be scalable to be applied to larger problems.
- Efficiency: By means of the approach, the energy wasted by resource provisioning issue of the system should be reduced, which leads to the improvement of energy efficiency.

1.2 Problem Statement

Workload of a cloud database system vary with time, because users' activities are dynamic. For example users' activities are more intense at day time, whereas they do little at night. In this situation, energy is wasted if the configuration of the system at night remains the same as the one during daytime. Sometimes workload is unbalanced, in which part of system is occupied, whereas the other part remains idle. In this circumstance, energy is wasted because the workload is not scheduled accordingly.

The energy waste mentioned above comes from resource provisioning, i.e. there is a gap between required resource and provided resource. On one hand, over-provisioning increases energy waste and costs. On the other hand, under-provisioning causes Service Level Agreements (SLA) violation and Quality of Service (QoS) dropping [23]. At the cluster level, the situations of over-provisioning and under-provisioning might exist at same time. Figure 1.1 shows an example of resource provisioning for a small cluster, consisting of 10 nodes, where the bars present the required resource of each nodes and the lines present the provided resource. In Figure 1.1, the provided resources by node 1, 2 and

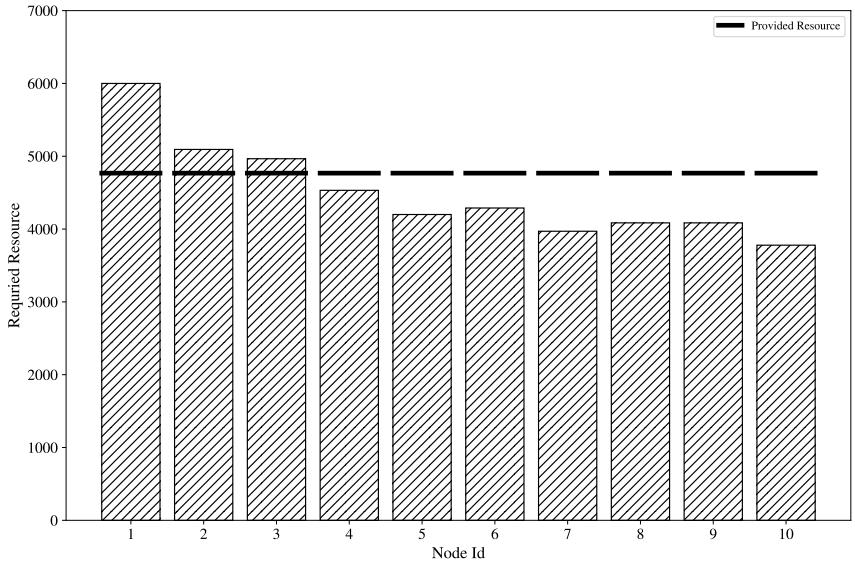


Figure 1.1: Without Resource Provision Technique

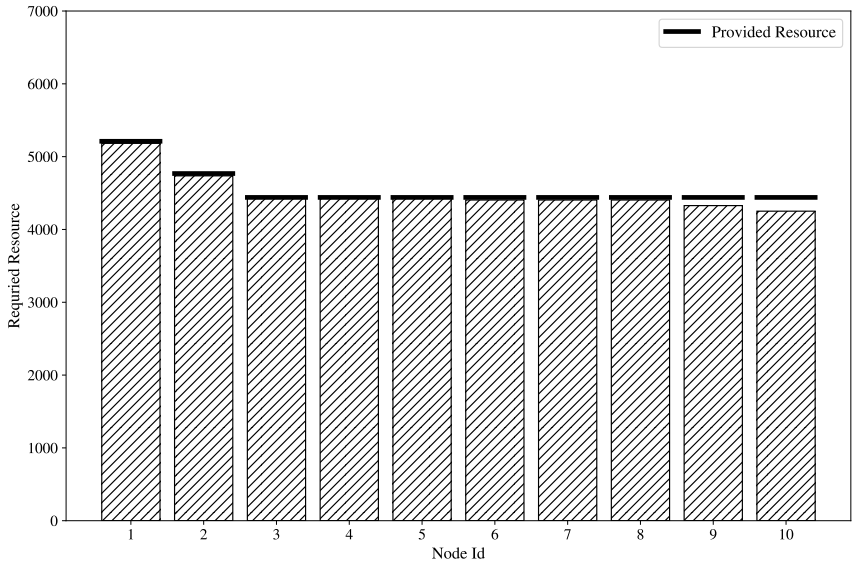


Figure 1.2: With Resource Provision Technique

3 are less than the required resources by the workload. In contrast, in node 4 to node 10, the provided resources are sufficient, but part of resources are wasted. Because of the resource under-provisioning, node 1 to nodes 3 suffer from SLA violation, in which parts of workload cannot be executed. Node 4 to node 10 suffer from energy waste since the resources are over-provisioned. Figure 1.2 shows an ideal case of resource provisioning in which the provided resource and required resource are compatible, and meanwhile SLA violation is eliminated and wasted resource is minimized.

Based on DVFS [20], the system resource can be provided in a fine-grained way. In this work, we propose frequency selection approach, in which the frequency of each node within the system is assigned according to its dynamic workload. By means of frequency selection approach, we make the provided resource and the required resource compatible, which leads to the reduction of SLA violating caused by resource under-provisioning and the reduction of energy waste caused by resource over-provisioning.

The foundation of the frequency selection approach is the relationship between energy efficiency and frequency options. Afterwards, the frequencies can be assigned to the system according to the workload prediction. Therefore the first problem in this thesis is the analysis of the energy efficiency behavior of the system under different frequency options. The next problem is how to assign frequencies to the system according to the workload prediction.

However, compared with the former one, the assignment method is more complex. To improve the energy efficiency of the whole system, we need not only frequency selection but also corresponding migration approach. There are two reasons to support this idea:

1. With the frequency selection approach, two situations which decrease the system's energy efficiency are unavoidable. 1) When a node is assigned with the highest frequency but the workload still needs more resource to be executed, part of the workloads should be migrated to another node to avoid SLA violation. 2) When a node is assigned with the lowest frequency but the workload is still too low, part of the workload from other nodes should be migrated to avoid energy waste. Even if the underutilized nodes are turned off, the migration strategy is still needed, since the data would be accessed by users. If the node is turned off directly, SLA might be violated.

2. With the migration strategy, energy efficiency of the system can be further improved. Intuitively, a node with higher frequency will consume more energy but it can take more workload as well. Meanwhile, applying migration costs energy. Consider the following case: a node has a

resource under-provisioning situation, namely that it requires more resources. With the case, there are 2 choices for this node: 1) assign a higher frequency, which increases energy consumption of the CPU; 2 migrate the workload to another node, which causes certain amount of energy denoted as migration cost. If the migration cost is lower than the energy consumption increment of the node with the assignment of a higher frequency, the energy efficiency of the system is improved.

With migration, there are mainly 3 problems. 1) Migration Selection: Which part of the workload should be migrated. 2) Migration Plan Generation: How to decide the destinations for the migrated workload. 3) Migration Execution: How to complete the migration. This work focuses on the first two parts which are defined as migration process.

To be noticed, the frequency selection approach and the migration approach are not independent. The frequency selection can be treated as searching process. In the solution space, we want to find a frequency configuration that minimize the energy consumption of the system. Within the searching process, the migration approach is applied for each solution candidate to evaluate the given frequency configuration. In terms of practice and implementation of the corresponding algorithms, the main challenge for both approaches is their scalability. At first, the scalability problem of frequency selection approach is caused by its huge solution space. In a small case with 30 nodes and 8 available frequency options, there are 8^{30} frequency combinations in total. In our environment, the optimal solution of above case can be found by the entire searching within 5 hours. However, a common case has hundred nodes within a cluster. For example, the technical group from instagram claimed that their biggest Cassandra system contains 1000+ nodes and biggest cluster contains 100+ nodes in Cassandra summit 2016 [13]. Secondly, the problem of migration strategy is NP-hard since it can be reduced to a knapsack problem [24]. Therefore, combined with the huge solution space of the frequency combinations, the total searching time would be unacceptable which leads to the scalability problem for both approaches.

In frequency selection approach, energy consumption of a cloud database system comes from two parts — the energy used to execute the workload, and the energy used to apply the migration. The former part is related to the workload and execution time, therefore we evaluate it by the power consumption of the system. The latter part is related to the file system and the network module, i.e. router, switch and so on, therefore we define it as migration cost. In practice, user's requirements vary with their cases, environment and budget. In some cases, energy consumption is the only concern of the administrator, then we have to minimize the sum of energy. However, in some cases, users

not only concern the energy, but also have the other constraints. For example, in some green energy system, the power consumption is limited according to the power supply under certain circumstances [25], and in some production environment, huge migration cost is unacceptable since it might lead to high migration overhead or service interruption [22]. Therefore we treat frequency selection approach as two kinds of problem: **Bounded Problem** and **Optimization Problem**. In **Bounded Problem**, the power consumption and the migration cost are treated separately, in which one is treated as the objective and the other one is considered as constraint. With the **Bounded Problem**, we have **Power Bound Problem** and **Migration Cost Bound Problem**. In **Optimization Problem**, we only focus on the energy consumption itself. Both energy consumption are taken into consideration in frequency selection approach.

In conclusion, 3 approaches will be discussed this thesis: 1 **Frequency Selection with Bounded Problem**, 2 **Frequency Selection with Optimization Problem** and 3 **Migration Approach**.

1.3 Research Goals

The initial goal of this research is to improve the energy efficiency of cloud database systems by providing energy-efficient resource provisioning approach, in which the SLA violation and the energy waste are minimized. To reach this goal, we have a few subgoals. At first, the behavior of energy efficiency of the cloud database systems under DVFS technique needs to be analyzed, which is the fundamental conception of the frequency selection approach. The second subgoal is to propose a frequency selection model based on the energy efficiency analysis and abstraction of the cloud database systems to overcome the different characteristics of different cloud database systems. At the end, the third subgoal is to propose simple, efficient and scalable algorithms to cope with **Frequency Selection with Bounded Problem**, **Frequency Selection with Optimization Problem** and **Migration Approach**. However, as discussed above, searching the optimal solution is unrealistic for all the problems because of the high complexity of the problem and the huge solution space. Therefore, we focus on approximation algorithms that give feasible solutions.

1.4 Overview

In this thesis, we present the techniques to provide energy-efficient resource provisioning method within cloud database systems to improve its energy efficiency. After this introduction chapter, the remaining portion of the thesis is organized as follows.

- **Chapter 2** summarizes the related works of nearby fields of the research, covered by this thesis. The related fields include cloud database system, energy efficiency of the system and related improvement approaches.

- **Chapter 3** analyzes energy efficiency of cloud database systems by an empirical methodology first using benchmark on real infrastructure. And then the frequency selection model is proposed. In the modeling, at first a generic frequency selection model is proposed for cloud systems, and then the model is specialized for cloud database systems. Secondly, a model simplification approach is introduced to reduce the complexity for computing the estimations of power consumption and migration cost. Thirdly, a benchmark experiment on real infrastructure is introduced to get the static parameter in the modeling.

- **Chapter 4** solves the frequency selection with bounded problem. At first, the problem type, and corresponding constraints and objective are illustrated, in which the frequency selection problem is regarded as bounded problem. Then, two algorithms, Nonlinear Programming Algorithm and Multi-Phases Algorithm, are proposed, and experiments are introduced to evaluate the proposed algorithms.

- **Chapter 5** solves the frequency selection with optimization problem. At first, the chapter illustrates the constraints and objective used. Then, two algorithms are proposed, Genetic Algorithm and Monte Carlo Tree Search Algorithm, to find approximated solutions to the problem. Both algorithms have their advantages and disadvantages which are discussed in the experiment section.

- **Chapter 6** shows details of the migration approach. In the chapter, the objective of the migration approach is discussed. Then, 2 algorithms are proposed to cope with different phases within migration process, and corresponding experiments are proposed to evaluate the algorithms.

- **Chapter 7** presents the conclusion of this thesis and proposes perspective.

Chapter 2

Related Works

Achieving better energy efficiency has been drawn great attention lately, since the explosive energy consumption and increasing energy bills caused by the growth of the data volume and construction of data centers. Under this circumstance, more and more researches emerge to improve the energy efficiency on cloud systems. As a part of the cloud systems, however, the effort for improving energy efficiency for cloud database has not been done a lot. In the following, we give an overview over related workloads in the fields of cloud database systems, energy efficiency and related improvement approaches within cloud system and cloud database system field.

2.1 Cloud Database Systems

A cloud database is a database that typically runs on a cloud computing platform, and access to the database is provided as-a-service [26]. Generally, the cloud database systems can be categorized into 3 types according to their data model.

- **SQL Database**: SQL Database mainly refers to the **R**elational **D**atabase **M**anagement **S**ystem (**RDBMS**). SQL Database can run in the cloud environment using virtual machine technique or as a service. There are a lot of SQL Database systems in the cloud environment, for example MySQL Cluster [27], Amazon Aurora [28], Oracle Real Application Clusters [29] and so on. As traditional RDBMS, SQL Database in cloud environment still gives fully support of **A**tomicity, **C**onsistency, **I**solation, **D**urability properties (ACID for short). While it can be scaled vertically easily, it is still challenging to scale SQL Database horizontally.

- **NoSQL Database**: The term NoSQL — "Not Only SQL" — refers

to a group of non-relational database management systems [30]. According to http://nosql-database.org [31], there are at least 225 NoSQL databases in current stage, with various features to meet different user cases and requirements. In term of data model, NoSQL databases can be categorized as Key-Value stores, Column-family stores, Document stores and Graph based stores to overcome the diversity of user requirements. In aspect querying method, NoSQL Database provides a ton of APIs depending on the database itself, and some NoSQL databases provide SQL-like querying languages, for example CQL [32] provided by Cassandra, HiveQL [33] provided by Hive. More information about NoSQL database can be referred from the following surveys [11, 30, 34].

- **NewSQL Database**: NewSQLs are a class of modern relational DBMSs that seek to provide the same scalable performance of NoSQL for OLTP read-write workloads while still maintaining ACID guarantees for transactions [34, 35], for example VoltDB [36] Spanner [37] RubatoDB [38]. According to the definition of NewSQL database, NewSQL database can be categorized as following 3 types: 1 novel systems that are built from the ground-up using a new architecture, 2 middleware that re-implement the same sharding infrastructure that was developed in the 2000s by Google and others, and 3 database-as-aservice offerings from cloud computing providers that are also based on new architectures [35].

In this thesis, we use Apache Cassandra [39, 40], a wildly used NoSQL database, as our example within the demonstration. Cassandra has been chosen by following reasons: 1, Cassandra is a popular open-source NoSQL database, 1st within the wide-column store based NoSQL database and 10th within overall cloud databases according to db-engines.com, which can be easy obtained; 2, Cassandra do not rely on the third party distributed file system, like HDFS [41], which leads to easy installation and deployment; 3, Houssem-Eddine Chihoub *et al.* [19] have done some research about performance of Cassandra with DVFS, which gives us a base line of optimization. Since the database system itself is not our main research objective, we only introduce the Cassandra briefly in aspect of its architecture and replication strategy, which are related to our modeling. More details can be found in [39].

Cassandra is a decentralized structured storage system [42]. The fundamental component behind this architecture is gossip protocol [43], a peer-to-peer protocol, which is used to cope with inner communication. The nodes exchange their inner states about themselves and the nodes they know by a heartbeat mechanism using gossip protocol. By switching the state information, Cassandra implements a failure detection mechanism to evaluate whether a node of the system is faulty or dead.

In Cassandra, the primary key (or row key) space is presented as a ring structure, and keys in the ring are defined as tokens. Figure 2.1 shows the logical view of Cassandra architecture, a key ring. A ring is considered as a circle that presents the range of token values. The nodes are placed in the ring, and each node within the cluster is given a token value. By means of the tokens of nodes, a ring is split to several token ranges. When a token belongs to a range between two continuous nodes, the corresponding data is placed on the the latter node. For example, in Figure 2.1, a token whose value is 15 is assigned to the node whose token is 20. In Cassandra, a parameter *Replication Factor* is used to decide the data replications number. In the example, *Replication Factor* is set to 3 indicating that there are 3 copies of the data. The first copy is assigned by the range of tokens, and the other 2 copies are assigned to the following two nodes in the ring. A token is generated according to a partition function which is denoted as partitioner. Partitioner is a key design of Cassandra, whose nature is a map from row key space to token space and it is used to determine how data is distributed across the nodes in the cluster. Currently, there are 3 partitioners: Murmur3Partitioner (default), RandomPartitioner and ByteOrderedPartitioner [44]. Partitioners take advantage of consistent hash method [45] to make sure all the keys are distributed within the ring consistently. However, the tokens for nodes still need to be assigned manually under this strategy.

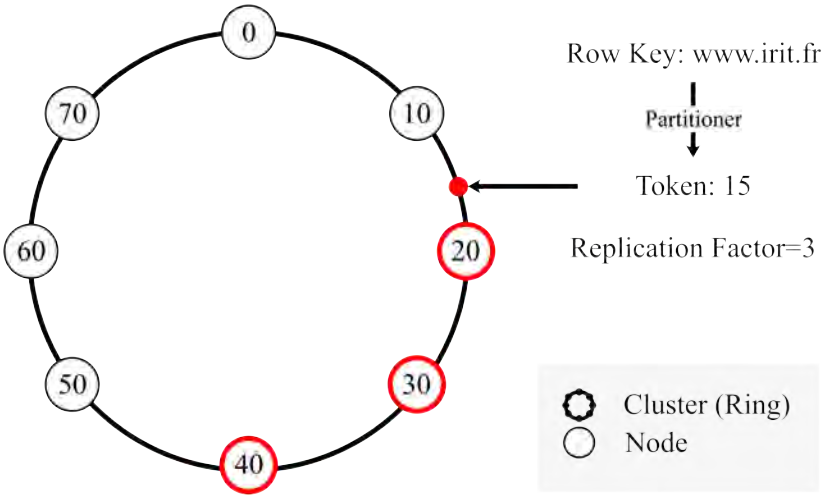


Figure 2.1: Architecture of Cassandra

Load balance is an important issue within cloud computing [46]. However, based on token ring architecture, the tokens of nodes are quite essential for load balance. Virtual Nodes (VNodes for short) [47] are introduced to avoid data skew. Figure 2.2 presents the basic conception of VNodes. The token ring

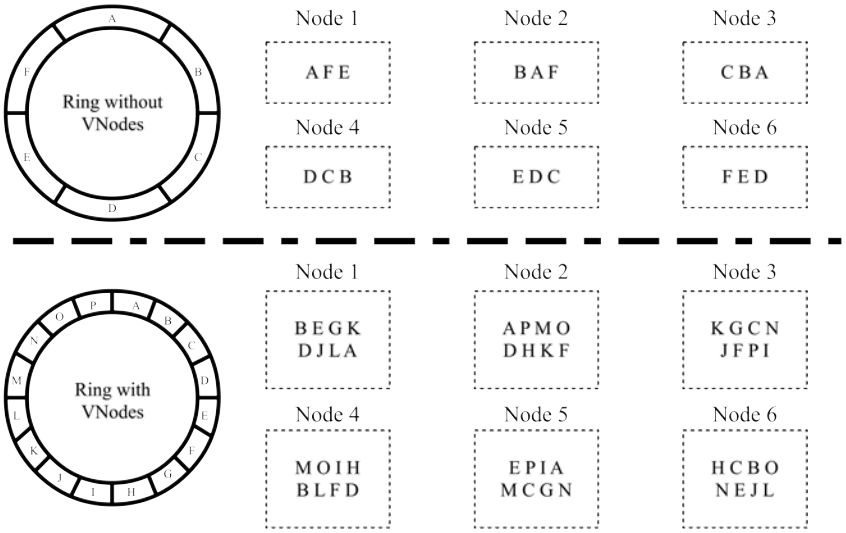


Figure 2.2: Virtual Nodes

with VNodes technique is split into small partitions, and each physical node is allowed to possess more partitions. Therefore the relationship between token ranges and nodes is changed from one-for-one to several-for-one. Speaking of the replications, all replications are assigned randomly across the nodes. Under the VNodes policy, the dataset is placed according to its hashed key (token produced by partitioner) and VNodes placement.

In conclusion, Cassandra is a decentralized NoSQL cloud database system, in which data is organized by tables and identified by a primary key. The key space is presented as a key ring in the system to determine its location. To prevent data skew, Cassandra introduce VNodes for data balance. Each node can contain several VNodes.

2.2 Energy Efficiency

Energy efficiency is a hot topic within cloud-related systems. Xindong You *et al.* made a survey and taxonomy of energy efficiency relevant surveys in cloud-related environments to summarize energy efficiency related works in [48]. In this thesis, we mainly focus on the energy efficiency within cloud database systems in aspect of energy-efficient resource provisioning method. In this section, we summarize the energy efficiency related work within database and cloud database fields.

2.2.1 Energy Efficiency Measurement

In computer science, energy efficiency mainly refers to a ratio between system's useful work and its total energy used for completing the work, shown as Equation 2.1.

$$energy\ efficiency = \frac{work\ done}{energy\ consumption} \tag{2.1}$$

In database and cloud database area, the workload is presented as transactions or operations within the system. Therefore, the energy efficiency of database or cloud database is presented as a ratio between amount of transactions (work done) and energy consumption [49–51], because of the workload of databases, shown as Equation 2.2. Furthermore, a ratio between throughput (or performance) and power consumption (or watts) is used as well [49, 50, 52] in database and cloud database area, shown as Equation 2.3. Equation 2.2 considers energy efficiency of the system within a period of time. In contrast, Equation 2.3 tries to reveal transient energy efficiency of the system.

$$energy\ efficiency = \frac{amount\ of\ transactions}{energy\ consumption} \tag{2.2}$$

$$energy\ efficiency = \frac{throughput}{power\ consumption} \tag{2.3}$$

2.2.2 Power Modeling

In order to evaluate the energy efficiency of a system, energy consumption is required. Generally, energy consumption of computing resources can be obtained by energy sensors, wattmeters and power distribution units [53], or can be estimated by power models [54].

According to different devices and environments, there are different energy consumption estimation methods existing. For example, Miyuru Dayarathna *et al* summarized energy consumption models within data center field in [55], and Raja Wasim Ahmad *et al.* summarized energy consumption models for smartphone applications in [56].

In this thesis, we try to improve the energy efficiency of cloud database systems by taking advantage of DVFS. Therefore, we only focus on CPU related power models. In this section, we introduce widely used power modeling methods.

Digital Circuit Level Energy Consumption Modeling

Complementary **M**etal-**O**xide-**S**emiconductor (CMOS for short) is a technology for constructing integrated circuits. CMOS technology is used in microprocessors, microcontrollers, static RAM, and other digital logic circuits [57]. This power model can be used to accurately model the energy consumption at the micro architecture level of the digital circuits [55].

The power consumption of a CMOS-based microprocessor is defined to be the summation of capacitive, short-circuit and leakage power. The capacitive power (dynamic power dissipation) is the most significant factor of the power consumption [58]. The capacitive power $p_{capacitive}$ can be defined as Equation 2.4, in which where A is the number of switches per clock cycle, C_{ef} is the total capacitance load, V_{dd} is the supply voltage, and f is the frequency [58, 59].

$$p_{capacitive} = AC_{ef}V_{dd}^2 f \tag{2.4}$$

In practice, f can be presented as Equation 2.5, in which V_t denotes threshold voltage and κ denotes hardware-design-specific constant factor [60, 61]. To be noticed, $V_{dd} \geq V_t \geq 0$ and $\kappa > 0$.

$$f = \kappa \frac{(V_{dd} - V_t)^2}{V_{dd}} \tag{2.5}$$

E.N. Elnozahy *et al.* [62] simplified Equation 2.4 as Equation 2.6, in which c_0 presents the power consumption of all components except the CPU and c_1 is a constant. They expressed voltage as a linear function.

$$p = c_0 + c_1 f^3 \tag{2.6}$$

Jun Liu *et al.* assumed the leakage power is a constant, β. They obtained the total power consumption by the summation of the leakage power and the capacitive power, namely by $p_{capacitive} + \beta$. Then, they normalized the power consumption as Equation 2.7, in which f^3 presents the capacitive power and β presents leakage power.

$$p = f^3 + \beta \tag{2.7}$$

System Utilization Based Server Power Models

Fan Xiaobo *et al.* [63] had shown that the linear power model can track the dynamic power usage with a greater accuracy at the PDU level. They presented their power model as Equation 2.8, in which p_{idle} and p_{max} present the power consumptions of when CPU is idle and fully utilized respectively, and ω presents the CPU usage.

$$p = p_{idle} + \omega \times (p_{max} - p_{idle}) \tag{2.8}$$

Combining Equation 2.6 and Equation 2.8, Song Jie *et al.* [51] presented another power model by solving Equation 2.9, in which X_1, X_2, Y_1, Y_2 are constants. They showed their result by Equation 2.10, in which A, B, C, D are system-related coefficients.

$$\begin{cases} \dfrac{\partial p(f^3, \omega)}{\partial \omega} = & X_1 + Y_1 f^3 \\ \dfrac{\partial p(f^3, \omega)}{\partial f^3} = & X_2 + Y_2 \omega \end{cases} \tag{2.9}$$

$$p(f^3, \omega) = Af^3 + B\omega f^3 + C\omega + D \tag{2.10}$$

2.2.3 Energy Efficiency Analysis and Improvement

A variety of the past researches have been done to study and analyze the energy efficiency in cloud systems and cloud database systems. In this section, the related works about analyzing and improving energy efficiency of cloud database systems are summarized.

Related Surveys

Xindong You *et al.* [48] made a survey about energy efficiency relevant surveys in cloud-related environments. This survey summarized and classified energy efficiency related surveys from 2011 to 2017 that gives the comprehensive knowledge about energy efficiency within cloud-related field.

The following surveys are more related to this thesis. Toni Mastelic *et al.* [64] summarized energy efficiency related works within cloud computing infrastructure domain with regard to network, server, cloud management system and applications. Anne-Cecile Orgerie *et al.* [54] summarized techniques and solutions that aim to improve the energy efficiency of computing and network resources. In their survey, the computing resources not only contain the individual nodes (or servers), but also refer to the entire infrastructure, which take advantage of virtualization. For computing resources, they summarized energy efficiency related techniques such as energy consumption estimation method, node-level optimization approach, grid and data centre power management, and virtualization approach. For network resources, they summarized hardware level techniques, shutdown strategy, slowdown strategy, and coordination approach. Davide Careglio *et al.* [65] gave a lot of possible techniques of reducing energy consumption in large scale distributed systems in

hardware level, such as process, memory, disk/flash, and environment. Qaisar Shaheen *et al.* [66] summarized energy saving techniques within computational clouds from hardware level to software level, in which they introduced some researches take advantage of DVFS to achieve better energy efficiency.

Energy Efficiency Analysis

Some related works analyzed the energy efficiency of the relational database management system, parallel databases. In this thesis, we focus on the energy-efficient resource provisioning for cloud database systems using DVFS technique, therefore the energy efficiency of cloud database system with DVFS technique is a fundamental component. The following results inspired our work.

Tsirogiannis Dimitris *et al.* [49] analyzed the relationship between energy efficiency and performance in two RDBMS systems, PostgreSQL [67] and System-X, a commercial DBMS. To be noticed, the performance in their research refers to $\frac{1}{execution\ time}$. In their research, they made a detailed study of the power-performance profiles of core database operations, $HashJoin$, $Scan$ and $Sort$ on modern scale-out hardware. Then, they concluded an investigation on the effects of both hardware and software knobs on the energy efficiency of complex queries in the DBMSs. They found out that with different operations, CPU power used can vary widely, and CPU power is not linear with number of cores used. Then, they found that the best performing configuration was also the most energy-efficient one in most of their experiments. However, in the few cases where this did not hold, energy efficiency did not increase by more than 10%. They believed that this relationship is a result of large up-front power costs in modern server components, i.e. CPU, memory and disk (ssd or hdd). In the end, they gave two directions to improve the energy efficiency of the system: 1) make resource consolidation across underutilized nodes to save power without sacrificing performance, or 2) use alternative energy-efficient hardware to lower fixed-power costs.

Willis Lang *et al.* [15] gave guiding principles for building up the energy-efficient cloud DBMS with query properties and scalability taken into account. At first, they investigated the trade-off in performance versus energy efficiency when performing speedup experiments on TPC-H [68] queries. In their benchmark, they used Vertica [69] and HadoopDB [70] as target parallel databases. They concluded that improving performance of parallel DBMS does not always result in better energy efficiency, and the query bottleneck have to be taken into account to improve the energy efficiency of the system. Secondly, they analyzed the trade-off in performance versus energy efficiency under different type of bottlenecks, i.e. hardware bottleneck (network and disk), algorithm bot-

tleneck (broadcast) and data skew. Under the hardware bottleneck, it causes lower energy efficiency because of underutilized CPU cores. With the algorithmic bottleneck, it reduce energy efficiency since scaling out to more nodes does not speed up broadcast phase of a join. Data skew harms balance of utilization of the cluster nodes. Meanwhile, they implemented a dynamic cluster–the number of nodes can be configured–by P-store [71], a custom-built parallel engine to have more details about the influence of partition-incompatible operations. In the end, they summarized their experimental results to propose cluster design principles: 1) for a highly scalable parallel workload, use all available resources; 2) if the query is not scalable due to the bottlenecks, reduce the performance to meet any required targets (i.e. SLAs), it will achieve higher energy efficiency; 3) within heterogeneous cluster, non-scalable queries may achieve a better energy efficiency.

In this thesis, we analyzed the energy efficiency of cloud database systems under DVFS techniques, Section 3.1. We try to analyze the energy efficiency of the system under different frequency options. Compared with the above research, we mainly focus on NoSQL database rather RDBMS or parallel RDBMS. Meanwhile, our research direction is the same as the mentioned ones. We try to save wasted power caused by under-utilized nodes and reduce SLA violation caused by over-utilized nodes by energy-efficient resource provisioning method.

Switching Nodes and Consolidation Method

To improve the energy efficiency of cloud systems and cloud database systems, a straightforward idea is to switch underutilized nodes off and switch them on when necessary. Besides that, another idea is to use a consolidation method and try to turn off the idle machines. Based on these ideas, there are researches that have been done in cloud systems.

Daniel Schall *et al.* [17, 50, 72] designed and implemented WattDB, which is a distributed DBMS that dynamically adjusts itself switching nodes on and off according to the present workload, and reconfigures itself to satisfy the performance demands. Gae-Won You *et al.* [16] proposed system Ursa which scales to a large number of storage nodes and objects and aims to minimize latency and bandwidth costs during system reconfiguration. At first, Ursa tries to detect the hot spots in the system and re-balance these data with minimized transformation cost. Based on the data migration approach, Ursa implement power management approach in which they use a threshold strategy to maintain the amount of nodes to save energy.

Similarly machine virtualization-based technology that consolidating VMs dynamically and turning off idle servers has been proved effective also. Guangjie

Han *et al.* [73] proposed a remaining utilization-aware (RUA) algorithm for virtual machine placement, and a power-aware algorithm to find proper hosts to shut down for energy saving. Emmanuel Cecchet *et al.* [74] proposed virtualization-driven database provisioning system, Dolly. Dolly took advantage of a new database replica spawning technique that leverages virtual machine cloning in which Dolly makes more replicas when the system is overloaded, and reduces the amount of replicas otherwise. Huangke Chen *et al.* [75] proposed scheduling approach for real time tasks within virtualization environment based on interval number theory, in which they also proposed scale functions to switch nodes on and off according to the workload.

Above works achieved energy efficiency in their domains. However, such approaches naturally require to migrate a large number of data from one node to another in order to switch off underutilized nodes or switch on more nodes. In WattDB [17], Daniel Schall *et al.* took advantage of SDD to reduce the migration cost. In Usra [16], Gae-Won You *et al.* took advantage of replica strategy to shutdown nodes which do not contains any primary replicas. However, in cloud database systems, this technique might be unsuitable since different implementation of the system, they have different consistency strategy. For example, Cassandra provides 5 consistency levels, and HBase provides strong consistency only. In Guangjie Han *et al.*'s work [73], their virtual machine consolidation policy aims to improve resource utilization and reduce the number of active physical servers, therefore they did not consider the migration cost. In Dolly [74], Emmanuel Cecchet *et al.* only considered the down time of the database but not the migration cost. Meanwhile, they only applied the technique for a web application's database layer in which 12 GB amount of data was used in the extreme case of the experiments. Huangke Chen *et al.* [75] scheduled real-time tasks to the virtual machines. When there are overloaded nodes, they scale up the computing resource, namely turn up more virtual machines, and otherwise they reduce the amount of virtual machines. In their work, virtual machines are computing resources to execute the tasks, therefore there is no migration cost at all.

Compared with above related works, we do not chose to switch the nodes on and off, but assign a frequency according to the workload amount. Meanwhile, when SLA violation occurs and no higher frequency can be assigned, a migration is used.

Energy Proportionality Method and DVFS based Method

Balaji Subramaniam *et al.* [76] measured the power consumption and the performance of a Cassandra cluster, and used power and resource provisioning techniques to analyze the energy proportionality of the cloud database sys-

tem.

Houssem-Eddine Chihoub *et al.* [19] explored the tradeoff between consistency and energy efficiency on the energy consumption in Cassandra. Meanwhile a prototype model, Hot-N-Cold, is introduced to reduce energy consumption in Cassandra by means of setting the frequencies manually. In our work, we extend this idea. The frequencies are set by means of frequency selection.

In this work, we use DVFS technique to improve the energy efficiency of cloud systems, especially cloud database systems. There are some researches that have been done using Dynamic Voltage and Frequency Scaling (DVFS) or Dynamic Voltage Scaling (DVS) technique. Yu Lei *et al.* [77] studied the power efficiency scheduling problem of real-time tasks in an identical multi-core system, and presented Node Scaling model to achieve power-aware scheduling. Liu Jun *et al.* [61] studied energy-efficient scheduling of periodic real-time tasks on multi-core processors with voltage islands, in which cores are partitioned into multiple blocks and each block has its own power source to supply voltage. Above works proposed heuristic algorithms to cope with voltage scaling problem in power saving manner. Compared to our work, we assign the frequency to each node of cloud database systems according to the workload. In term of the use case, in our approach, we do not switch nodes on and off to avoid the unacceptable migration cost and unavailability of the system, and for each time window the workloads cannot be totally reassigned but be migrated according to the assigned frequencies and previous workloads.

Chapter 3

Modeling

In this chapter, we mainly focus on the model abstraction. At first, we analyze the energy efficiency of cloud database system under DVFS technique. Then, the frequency selection model is proposed. The model consists of a generic model, and a specialized model. The generic model is proposed to deal with the frequency selection and migration within distributed system, and the specialized model is derived from the generic model to cope with resource provisioning problem within cloud database systems. As mentioned in Section 1.2, the scalability is one of the obstacle of this research. Therefore, in Section 3.3, a model simplification approach is proposed to boost the performance of corresponding approaches. In the end, a parameter benchmark is given to obtained the static parameters within the model, and a discussion about the extension of the model is introduced.

3.1 Energy Efficiency in Cloud Databases

In order to improve energy efficiency of cloud database system by DVFS technique, the first step is to analyze the energy efficiency of the system under different frequency settings. In this section, we take a Cassandra Cluster as an example to analyze its energy efficiency with DVFS technique.

3.1.1 Methodology

In this thesis, we borrowed the energy efficiency definition from Dimitris Tsirogiannis *et al.* [49] which is shown as Equation 3.1, where Δt indicate a time window, $\overline{p(\Delta t)}$ is the average power consumption (watts), and $l(\Delta t)$ is the throughput (operations per second) during the time window.

$$ee(\Delta t) = l(\Delta t)/\overline{p(\Delta t)} \tag{3.1}$$

Intuitively, energy efficiency of the system refers to a ratio between the workload done by the system and its energy consumption [51]. However, it is not suitable to describe the energy efficiency in cloud database systems for the following reasons.

1. Cloud database systems such as Cassandra and HBase behave like a long-term servicing system, which is different from the typical cloud computing system such as Hadoop and Spark [78] where batch tasks are submitted and have a finite duration. Therefore, Equation 3.1 considers the energy efficiency of the system for a certain time window Δt.

2. Operation types in cloud database mainly are querying, inserting, updating and scanning, whereas in cloud computing system the tasks mainly refer to typical batch tasks, i.e file scanning, aggregation and so on.

In order to evaluate the relationship between energy efficiency and different frequency settings, a benchmark experiment is designed. For a given frequency settings, the variation of energy efficiency can be obtained by increasing the amount of workload and collecting the corresponding power consumption.

3.1.2 Platform Setup

In this section, the details of the benchmark implementation are introduced.

Environment. This benchmark is executed on Grid5000 [79] testbed. Grid5000 is designed to provide a scientific tool for computer scientists similar to the large-scale instruments used by physicists, astronomers and biologists. In the benchmark, a database system Cassandra [11] with 10 nodes belonging to the Nancy site graphene cluster is deployed and the key parameters of the database settings are shown as Table 3.1. To be noticed that, the cache related parameters are set to 0 to avoid the influence of cache mechanism to the experiment results. *num_tokens* gives the number of virtual nodes for each node. 256 is the maximal *num_tokens* in current Cassandra Version (3.0) and higher *num_tokens* is good for data balance. The nodes from graphene cluster are equipped with a 4 cores Intel Xeon X3440 and 16 GB of RAM. The energy consumption values are collected by Power Distribution Units (PDU) and Kwapi API [80]. In this benchmark, the maximum throughputs under each available frequency option are obtained. The reasons for choosing the Nancy site and graphene cluster are: 1) The processors in graphene support manual tuning. In the other site, like Lyon, the processors may need to decrease their

Table 3.1: Core Properties of Cassandra

Property	Value
num_tokens	256
max_hints_file_size_in_mb	128
key_cache_size_in_mb	0
row_cache_size_in_mb	0
concurrent_reads	32
concurrent_writes	32

Table 3.2: Core Properties of YCBS Workload

Property	Value
recordcount	30000000
fieldlength	1000
readproportion	0.95
updateproportion	0.05
requestdistribution	uniform
threadcount	500

driver to provide this function. 2) In graphene cluster, it was more easier to create a resource limitation scenario. In contrast, for Lyon site orion cluster, it was impossible to simulate a resource limitation scenario since the cluster is quite powerful equipped with 2 cpu (16 cores for each) and 32 RAM.

Dataset and benchmark framework. To simulate the real workload and test cases, the **Y**ahoo! **C**loud **S**erving **B**enchmark (YCSB) framework [81] is selected as benchmark executor. Apache Cassandra [11] is chosen as target cloud database. YCSB is an open-source specification and program suite for evaluating retrieval and maintenance capacities of computer programs. It is aimed to develop a framework and common set of workloads for evaluating the performance of different "key-value" and "cloud" serving stores. A common YCSB experiment consists of two parts — a YCSB client, an extensible workload generator, and a few core workloads, a set of workload scenarios to be executed by the generator. A workload profile is used to describe the properties of the use case including the amount of records, reading proportion, update proportion, request distribution and so on. By the client, YCSB can generate the simulation data and execute the queries automatically. The core properties used in YCSB workload profile in our experiment are shown in Table 3.2. To be noticed, Table 3.2 is used to simulate one of data query usages within the system. In practice, the benchmark needs to be executed multiple times for different workload. Table 3.2 shows a heavy read workload, in which

readproportion and *updateproportion* indicate that the workload includes 0.95 reading operations and 0.05 updating operations. *recordcount* shows the total number of records within the system and *fieldlenth* shows the size (bytes) of each record. Combining *recordcount* and *fieldlenth*, we have 30GB data within the system. *requestdistribution* shows the query method, in which *uniform* is used, namely that in the queries the row key is picked uniformly. *threadcount* shows that the client will use 5000 thread to conduct the queries.

3.1.3 Experimental Cases

Table 3.3: Available Frequency Options

	f_1	f_2	f_3	f_4
Freq(GHz)	2.53	2.40	2.13	2.00
	f_5	f_6	f_7	f_8
Freq(GHz)	1.73	1.60	1.33	1.20

Table 3.4: Workload Size Per Node

	Q_1	Q_2	Q_3	Q_4
Items	4k	8k	16k	32k
	Q_5	Q_6	Q_7	Q_8
Items	64k	128k	256k	512k
	Q_9	Q_{10}	Q_{11}	Q_{12}
Items	1024k	2048k	4096k	8192k

Within graphene cluster, there are 8 available frequency options, which are shown in Table 3.3. In this benchmark, for a given frequency option, we load more and more queries into the system. We use 12 query tasks, denoted as Q_1 to Q_{12} shown in Table 3.4. For example the items queried per node in $Q1$ is 4k, which indicates 4000×10 random rows queried in Q_1. When a query task is finished, the corresponding execution time, energy consumption are recorded. And, the energy efficiency is obtained by Equation 3.1.

To be noticed, from Q_1 to Q_{12} more and more queries are loaded into the system. However, we do not try to control the throughput directly. In terms of throughput, there are two concepts — the system throughput and the required throughput. The required throughput is users' activity speed, namely the amount of queries that users submit per second. In contrast, the system throughput is the system's capacity of handling users' queries, which is impacted by the nodes' hardware configuration and software configuration. In

Equation 3.1, the workload refers to the throughput of the operations in the time window, namely that it refers to the system throughput. In practice, the system throughput has an upper limit due to different nodes' configurations. If the required throughput is below the nodes' maximum throughput, the nodes can handle the requests steadily. In contrast, if the required throughput tries to exceed the nodes' maximum throughput, the latency will be much longer due to resource usage competition. In this condition, the expected required throughput is unreachable. In YCSB profile, a property key *throughput* exists. This key only controls the maximum user query throughput, but it cannot ensure the system throughput we set.

Test Cases. Combining available frequency options f_i(Table 3.3) and query tasks Q_j (Table 3.4), there are 96 cases executed in this benchmark experiment.

3.1.4 Result Analysis

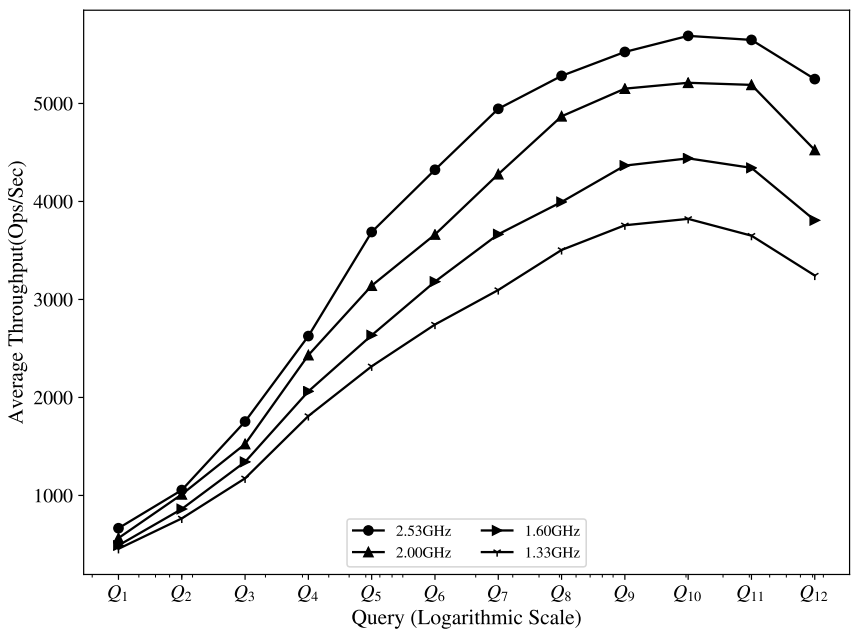


Figure 3.1: Relationship between Request Amount and Throughput

For workload Q_i $i \in [1, 12]$, the total amount of operations is $4000 \times 2^{i-1}$. Figure 3.1 shows the trends of system throughput along with the increasing operations for the system under frequency 2.53GHz, 2.00GHz, 1.60GHz and 1.33GHz. The trends have a same pattern. Along with the increasing operations, the system throughputs are increasing at first and then decline. During

Table 3.5: Node's Capacity under Each Frequency Option

Frequency	Capacity	Frequency	Capacity
2.53GHz	5690	1.73GHz	4768
2.40GHz	5518	1.60GHz	4440
2.13GHz	5357	1.33GHz	3822
2.00GHz	5211	1.20GHz	3520

the fluctuation, the system throughputs under different frequencies reach the highest point. We define this highest point as the capacity of the node under its frequency settings. The capacity can be treated as the system throughput upper limit. At beginning, the request throughput is lower than the node's capacity. Therefore, the system throughput increases as well. However, after the highest point, the request throughput tries to exceed the node's capacity, but some requests cannot finish because of the resource competition. The result is that the system throughput declines. For all the frequencies, the capacities are different. When the frequency is higher, the capacity is larger. The capacities for all frequency options are listed in Table 3.5. Note that, the capacity is related to the hardware and software configuration of the system. When the configuration changes, the capacity needs to be reevaluated.

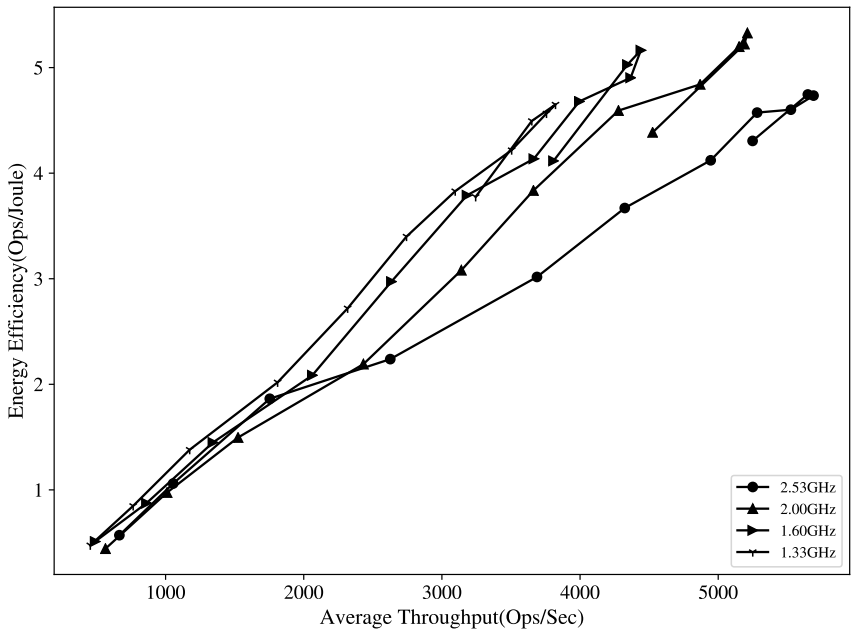


Figure 3.2: Relationship between Throughput and Energy Efficiency

Figure 3.2 shows the relationship between energy efficiency and the system

throughput under different frequencies. Along with the increase of the system throughput, the energy efficiency increases as well under each frequency. To be noticed that each line has a few coincident parts because when the request throughput tries to exceeds the capacity, the system throughput declines. Each frequency has its maximum energy efficiency value and the energy efficiency value reaches its maximum value at its maximum system throughput.

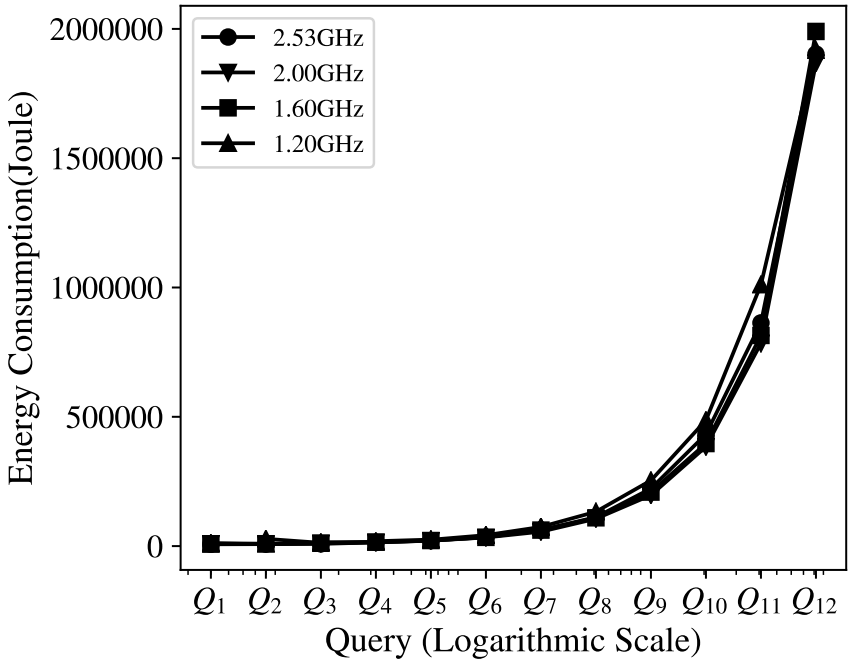


Figure 3.3: Energy Consumption for each Workload

Figure 3.3 shows the energy consumption for each workload. Along with the increase of the operations, the energy consumption increases as well for each frequency configuration. However with the same request amount, the energy consumptions under different frequency option do not have big differences. Since the maximum throughput of the system is tested, a lot of operations are loaded into the system. With different frequency configuration, the throughput of the system is different. For example with 2.53GHz, the average throughput is 5590 Ops/Sec and with 1.33 GHz, the average throughput is 3822 Ops/Sec. Therefore with the same request amount, the execution time under two frequency option is quite different. For example, with Q_{12}, with frequency 2.53 GHz execution is 1778 seconds, while it is 2837 with frequency 1.33GHz. Same query amount, with higher frequency, the system has higher power consumption while it has lower execution time. As a consequence, the energy consumptions do not have big difference in the end.

3.1.5 Lessons Learned

In this benchmark, the energy efficiency of a Cassandra Database was analyzed. By analyzing the result, we draw the following conclusions:

1. For a certain frequency setting, the system throughput of each node within the system has a maximum value. The maximum throughput value is defined as the node capacity under its frequency setting. With lower frequency, the capacity of the node is lower as well.

2. With a given frequency setting, the request throughput cannot exceed its capacity.

3. The energy efficiency of a node has correlation with its throughput. When the node reaches higher throughput, it achieves also higher energy efficiency.

According to the conclusions, the goal in frequency selection and migration approach is to select frequency settings and a migration plan which lead each node within the cluster to reach its capacity.

3.2 Frequency Selection Model

In this section, the frequency selection model is proposed based on the energy efficiency conclusions we obtained in the benchmark experiment. At first, a generic model is proposed to cope with resource provisioning within cloud systems. Then the model is specialized for cloud database systems by redefining the key functions.

3.2.1 Generic Model

A cluster $\mathbf{C}$ consists of n nodes. To simplify the description, the nodes are considered homogeneous. The extension of the model to heterogeneous nodes is discussed in Section 3.5.

The total running time of a system is made up of time windows. The length of a time window Δt is denoted as $|\Delta t|$. In practice, the size of the time window should fit the users' request activities. The principles of dividing running time are:

- Within a time window, the users' activities should be stable and the frequency of accessing the cloud database should be uniform.

- Between time windows, the frequency of the users' activities could be different.

The frequency selection approach and migration approach are applied for each time window. Therefore, in the following discussion we focus on a time window Δt. In Δt, the state of the system, $s_{\Delta t}(\mathbf{F}_{\Delta t}, \mathbf{W}_{\Delta t})$, is the state where the nodes are assigned to a frequency vector $\mathbf{F}_{\Delta t}$ and a workload vector $\mathbf{W}_{\Delta t}$. To simplify the description, the notation Δt is omitted. A frequency f_i, ($f_i \in \mathbf{F}$), is assigned to a node c_i. Similarly, a workload w_i, ($w_i \in \mathbf{W}$), is assigned to c_i. The size of workload w_i is denoted as $|w_i|$. It should be noticed that $\mathbf{W}$ is a predicted value. The maximum amount of workload that can be handled by a node under a frequency is defined as its capacity. Let the capacity measurement function be $z(c_i, f_i)$. When the current workload exceeds the node's capacity, the workload cannot be completed, which causes SLA violation. Meanwhile, we consider that SLA violation is not allowed, namely that all requests of a workload must be completed during the time window.

In order to avoid SLA violation, a migration process is introduced. Let the workload migration function be m. The migration process is considered as a state transformation process, namely $s(\mathbf{F}, \mathbf{W}) \xrightarrow{m} s^*(\mathbf{F}, \mathbf{W}^*)$. The state transformation process is denoted as $\widetilde{s}$. The workload for c_i after the migration is denoted as w_i^*. Energy used by the migration process is defined as migration cost. The migration cost estimation function is denoted $mc(\widetilde{s})$ and the system power consumption estimation function is $p(s)$. The energy consumption e of the system in Δt is:

$$e = p(s^*) \times |\Delta t| + mc(\widetilde{s}) \tag{3.2}$$

Energy efficiency of a system is defined by Equation 3.3 in which the energy efficiency in Δt is a ratio between the amount of workload processed and the energy consumption in Δt. Since the amount of workload is constant during a time window, the objective is to minimize e to improve the energy efficiency of the system.

$$ee = \frac{\sum_i^n |w_i|}{e} \tag{3.3}$$

For a given frequency vector, the power consumption and the migration cost can be estimated by $p(s)$ and $mc(\widetilde{s})$ respectively. Finding the most energy efficient configuration is then to find the best frequency for each node: It is a search problem within the frequency combinations space.

The conditions for applying the model are:

1. The system's running time can be divided into time windows. In a time window, the workload should be stable hence the power consumption can be estimated.

2. The node's capacity can be measured and part of the workload can be migrated when the current workload exceeds its capacity.

3. The workload of the next time window can be predicted according to previous running information.

4. The power consumption and the migration cost can be estimated according to the frequencies and the workloads.

3.2.2 Specialized Model for Cloud Databases

In this section, the generic model is specialized for cloud database systems. The workload w_i, the capacity measurement function $z(c_i, f_i)$, the migration function m, the power consumption estimation function $p(s)$ and the migration cost estimation function $mc(\tilde{s})$ must therefore be identified.

In a cloud database system, the dataset $\mathbf{D}$ consists of h data blocks, in which the size of block b_g is denoted as $|b_g|$, and the blocks are distributed within the cluster. In order to meet data integrity and fault-tolerance requirements, cloud databases use a replication factor to control the number of replicas of the data blocks. The dataset with a replication factor r is denoted as $\mathbf{D}^r = \{b_{1,1}, b_{1,2}..., b_{1,r}...b_{h,1}, b_{h,2}...b_{h,r}\}$, in which $b_{g,k} \in \mathbf{D}^r, k \leq r$ is the kth replica of b_g. The workload w_i of a cloud database system is defined as data query throughput. The probability of $b_{g,k}$ being accessed is denoted as $\varphi_{g,k}$. The total throughput of the system is denoted as l and $\sum_g^h \sum_k^r \varphi_{g,k} = 1$. $\varphi_{g,k}$ and l are predicted values, which can be given by data mining techniques and machine learning techniques, such as time series data mining and linear regression. For the corresponding techniques, we refer readers to the literature [23]. Let the block set assigned to c_i be $\mathbf{D}_i^r = \{b_{g,k} \mid b_{g,k} \in \mathbf{D}^r \text{and } b_{g,k} \text{ is assigned to } c_i\}$. The workload w_i is defined by Equation 3.4. According to the workload definition, we define the throughput of a block b_{gk} as block throughput, which equals to $l \times \varphi_{gk}$.

$$w_i = \sum_{b_{g,k} \in \mathbf{D}_i^r} l \times \varphi_{g,k} \tag{3.4}$$

The capacity measurement function $z(c_i, f_i)$ is a discrete function. Using benchmarks, the maximum throughput of a cloud database under each frequency can be obtained, Section 3.1.4.

The frequency option set is denoted as η. A frequency f is one of the available frequency options. Let the idle power consumption and the maximum power consumption of a node under a frequency f be c_f^{idle} and c_f^{max} respectively. $\forall f_p,\ f_q \in \eta$ and $f_p > f_q$, $c_{f_p}^{idle} > c_{f_q}^{idle}$ and $c_{f_p}^{max} > c_{f_q}^{max}$. With a higher frequency, the system provides more resources to support workloads, but consumes more energy. If a fraction ψ of CPU is used under the frequency f, the power consumption estimation function is defined by Equation 3.5 [82]. In cloud database systems, ψ_i is defined by Equation 3.6, in which w_i^* is the workload after the migration, hence $\psi_i \leqslant 1$.

$$p(s^*) = \sum_{i}^{n} \left(c_{f_i}^{idle} + \psi_i \times \left(c_{f_i}^{max} - c_{f_i}^{idle} \right) \right) \tag{3.5}$$

$$\psi_i = \frac{w_i^*}{z(c_i, f_i)} \tag{3.6}$$

In terms of migration process, there are mainly 3 types of migrations in the cloud environment according to network topology, namely migration within a rack, migration between racks and migration between data centers. We only focuses on the first two types. Let $\mathbf{M}_{\text{In}}$ and $\mathbf{M}_{\text{Out}}$ denote the sets of blocks migrated within a rack and the set of blocks migrated between racks, respectively. Let e_{In} and e_{Out} denote the energy costs per mega byte of migration within a rack and between racks respectively. Then the migration cost is defined by Equation 3.7. In Equation 3.7, $\sum_{b_{g,k} \in \mathbf{M}_{\text{In}}} |b_{g,k}|$ and $\sum_{b_{g,k} \in \mathbf{M}_{\text{Out}}} |b_{g,k}|$ shows the total size of data migrated within a rack and between racks, respectively.

$$mc(\widetilde{s}) = e_{\text{In}} \times \sum_{b_{g,k} \in \mathbf{M}_{\text{In}}} |b_{g,k}| + e_{\text{Out}} \times \sum_{b_{g,k} \in \mathbf{M}_{\text{Out}}} |b_{g,k}| \tag{3.7}$$

By redefining the key concepts and key functions of the generic model, the model is specialized for cloud database systems to cope with the resource provisioning problem.

3.3 Model Simplification Approach

The power consumption and the migration cost can be estimated by Equation 3.5 and Equation 3.7. However, both values are obtained after the migration process. In Equation 3.5, the workload after the migration w_i^* is required, while in Equation 3.7, the migrated block set $\mathbf{M}_{\text{In}}$ and $\mathbf{M}_{\text{Out}}$ are required. The problem is that migration strategy is NP-hard, because it can be considered as multiple knapsacks problem [24] (more details are discussed in Chapter 6). By means of approximation algorithms, (Section 6.2 and Section 6.3), a migration plan can be obtained within polynomial time. However, when the total

amount of possible frequency vectors increases, the evaluation time becomes unacceptable, namely that the algorithm cannot give a valid solution within a time window Δt. To overcome this performance issue, a model simplification approach is proposed in this section to obtain the upper bounds of the power consumption and the migration cost, which will be used later to evaluate the frequency vectors. The initial idea of the model simplification is to reduce costs for computing the power consumption and the migration cost values using a relaxation approach.

3.3.1 Power Consumption Upper Bound

According to Equation 3.5, power consumption is related to node's frequency and CPU usage. Considering a block $b_{g,k}$ is assigned to c_i, the power consumption of c_i increases because of it, i.e. it increases with the throughput of the block $l \times \varphi_{g,k}$. The increment is $\frac{l \times \varphi_{g,k}}{z(c_i,f_i)} \times \left(c_{f_i}^{max} - c_{f_i}^{idle}\right)$ in which the constant factor $\frac{\left(c_{f_i}^{max} - c_{f_i}^{idle}\right)}{z(c_i,f_i)}$ is defined as the **p**ower **c**onsumption **c**ontribution **f**actor of c_i (PCCF for short). In order to achieve the maximum power consumption, the blocks are assigned to the nodes with the highest PCCF value as much as possible.

To simplify the assignment, a relaxation is introduced, in which the blocks are continuous : one block can be split and put to multiple nodes. Let the total ordered node set be $\widetilde{\mathbf{C}} = (\mathbf{C}, \leq)$, in which if $\forall i, j$ $\frac{\left(c_{f_i}^{max} - c_{f_i}^{idle}\right)}{z(c_i,f_i)} \geqslant \frac{\left(c_{f_j}^{max} - c_{f_j}^{idle}\right)}{z(c_j,f_j)}$, then $c_i \leqslant c_j$. p^{max} is obtained by Equation 3.8.

$$
\begin{aligned}
p^{max} = & \left(\sum_{i=1}^{n} w_i - \sum_{i=1}^{\widetilde{i}} z\left(c_i, f_i\right)\right) \times \frac{c_{f_{\widetilde{i}+1}}^{max} - c_{f_{\widetilde{i}+1}}^{idle}}{z\left(c_{\widetilde{i}+1}, f_{\widetilde{i}+1}\right)} \\
& + c_{f_{\widetilde{i}+1}}^{idle} + \sum_{i=1}^{\widetilde{i}} c_{f_i}^{max} + \sum_{i=\widetilde{i}+2}^{n} c_{f_i}^{idle}
\end{aligned}
\tag{3.8}
$$

In Equation 3.8, index $\widetilde{i}$ of $\widetilde{\mathbf{C}}$ satisfies $\sum_{i=1}^{\widetilde{i}} z(c_i, f_i) \leqslant l \leqslant \sum_{j=1}^{\widetilde{i}+1} z(c_j, f_j)$, namely that index $\widetilde{i}$ is a pivot node: the blocks are assigned to the nodes c_i with $i \leqslant \widetilde{i}+1$. The nodes c_i with $i \leqslant \widetilde{i}$ reach their maximal power consumption, and the nodes c_i with $i \geqslant \widetilde{i}+2$ reach their idle power consumption because there is no block assigned to them. The power consumption of $c_{\widetilde{i}+1}$ is computed by Equation 3.5. Figure 3.4 gives an example of achieving power consumption upper bound. In the figure, c_1 to $c_{\widetilde{i}}$ reach their maximal power consumption because their capacities are fully utilized by the assigned blocks. To be noticed,

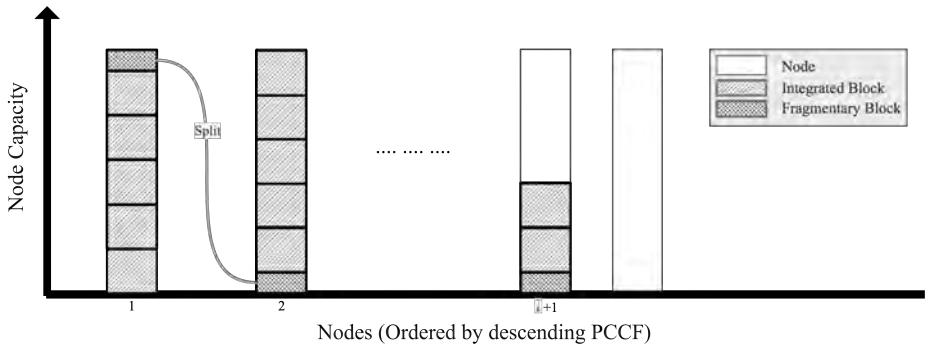


Figure 3.4: Example of Achieving Power Consumption Upper Bound

between c_1 and c_2, there is a fragmentary block that is split and put into both nodes. $c_{\tilde{i}+2}$ reach its idle power consumption because there is no block assigned. $c_{\tilde{i}+1}$ is the pivot node in the example, in which assigned blocks occupied part of its capacity. Therefore, the power consumption of $c_{\tilde{i}+1}$ can be computed by Equation 3.5.

Proposition 1. *For a given frequency vector* $\mathbf{F}$*, the power consumption achieved by Equation (3.8) is the maximal power consumption.*

Proof. For a given frequency vector $\mathbf{F}$, p^{max} is obtained by Equation 3.8. Assume that p^* exists, and $p^* > p^{max}$. Since $p^* \neq p^{max}$, we say that they have different block assignment plan. Without loss of generality, we assume that nodes are sorted by descending their PCCF values. In order to achieve higher power consumption than p^{max}, namely to achieve p^*, the key is to move the workload within $c_{\tilde{i}+1}$ to the previous nodes. To do so, consider the following scenarios.

Scenario 1 $b_{g,k}$ is assigned to $c_{\tilde{i}+1}$ in p^{max}. In order to achieve higher power consumption in p^*, $b_{g,k}$ is assigned to c_i $i \in [1, \tilde{i}]$, while other blocks keep the same assignment. Figure 3.5 shows an example of this scenario.

Scenario 2 $b_{g,k}$ is assigned to $c_{\tilde{i}+1}$ in p^{max}. In order to achieve higher power consumption in p^*, $b_{g,k}$ is switched with a block that is assigned to c_i, $i \in [1, \tilde{i}]$, while other blocks keep the same assignment. Figure 3.6 shows an example of this scenario, in which $b_{g,k}$ within $c_{\tilde{i}+1}$ is switched with a block within c_2.

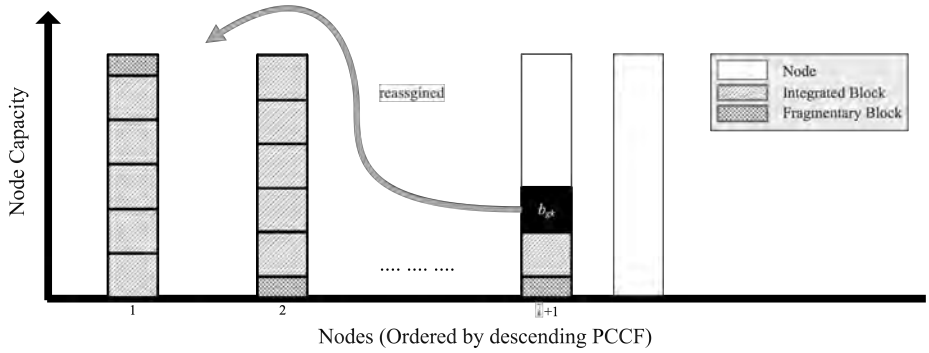


Figure 3.5: Example of p^{max} Proof: **Scenario 1**

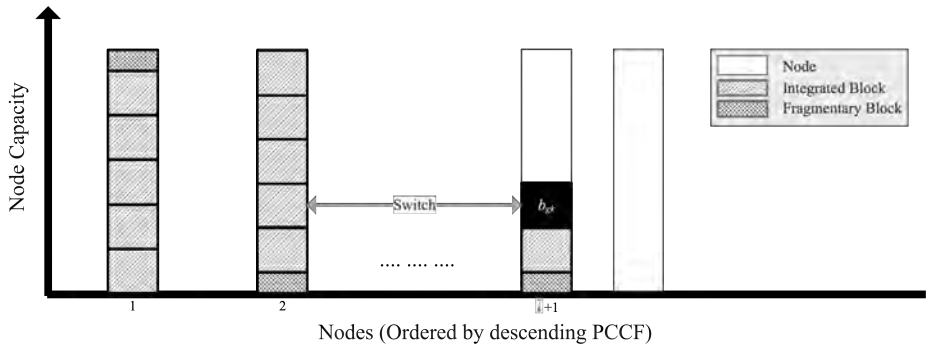


Figure 3.6: Example of p^{max} Proof: **Scenario 2**

For **Scenario** 1, in Figure 3.5, there is no more place for $b_{g,k}$ within nodes c_1 to $c_{\bar{i}}$, because all the capacities of these nodes are occupied by the other blocks. Therefore p^* does not exist.

For **Scenario** 2, no matter the switch method, the total workload l is constant. Therefore, the workload remains on $c_{\bar{i}+1}$ is $l - \sum_{i=1}^{\bar{i}} z(c_i, f_i)$. Therefore, $p^{max} = p^*$.

By means of above discussion, we can conclude that p^{max} is the maximal power consumption under the given frequency vector.

□

3.3.2 Migration Cost Upper Bound

In order to achieve the upper bound of migration cost, we try to migrate the migrated blocks in a worse case that produces highest migration cost within all the migration plan. It means maximizing the total size of migrated block between racks first, then in a rack. To simplify the migration process two relaxation conditions are introduced: 1. the nodes with free capacity in the same rack are combined to form a big knapsack with larger free capacity; 2. the migrated blocks are continuous: one block can be split and put to multiple nodes.

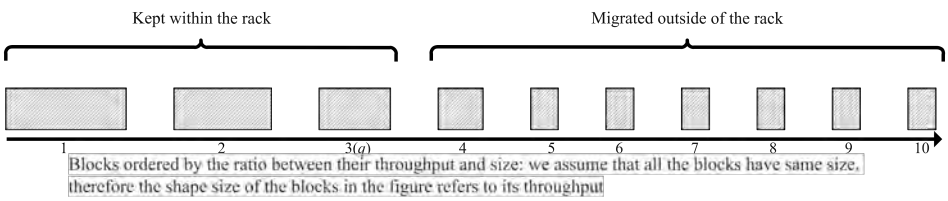


Figure 3.7: Example of Achieving Migration Cost Upper Bound

An example is shown in Figure 3.7 to give our intuitive idea. To be noticed, the remained capacity for a rack is constant value. For a rack γ_u, the remained capacity equals to $\sum_{c_i \in \mathbf{C}_u^{in}} (z(c_i, f_i) - w_i)$ for $\mathbf{C}_u^{in} = \{c_i | c_i \in \gamma_u \text{ and } w_i < z(c_i, f_i)\}$. To have the worse migration plan, we try to keep the migrated blocks with higher throughput but smaller size as much as possible in the rack to maximize the total migrated block size between racks. In the figure, we assume there are 10 migrated blocks, and furthermore, we consider that all the blocks have the same block size and sorted by a descending ratio between their throughputs and sizes. Therefore, the shape size of blocks in the figure refer to their throughputs. By picking up blocks from the beginning of the sorted migrated block

list until no capacity remains, then the maximal migration cost is achieved, because we can have the smallest number of kept migrated block in the example.

To compute the maximal migration cost, consider the following conditions:

1. The number of racks is denoted as U, and a rack is denoted as γ_u. For γ_u, the migrated block set is denoted as $\mathbf{M}_u$. The total ordered set of $\mathbf{M}_u$ is denoted as $\widetilde{\mathbf{M}}_u = (\mathbf{M}_u, \leqslant) = \{b_1, b_2, ...\}$ in which the blocks are sorted by descending the ratio between their throughputs and sizes, namely $\forall b_i, b_j \in \widetilde{\mathbf{M}}_u$, if $\frac{l \times \varphi_i}{|b_i|} \geqslant \frac{l \times \varphi_j}{|b_j|}$, then $b_i \leqslant b_j$.

2. The blocks with higher ratio values are kept within racks and the other blocks are migrated to other racks. $\widetilde{\mathbf{M}}_u^{\text{In}}$ and $\widetilde{\mathbf{M}}_u^{\text{Out}}$ are partitions of $\widetilde{\mathbf{M}}_u$. $\widetilde{\mathbf{M}}_u^{\text{In}}$ contains blocks that migrated within rack γ_u. While $\widetilde{\mathbf{M}}_u^{\text{Out}}$ contains blocks migrated out of the rack. $\widetilde{\mathbf{M}}_u^{\text{In}}$ and $\widetilde{\mathbf{M}}_u^{\text{Out}}$ satisfy the following conditions:

   - b_q is a pivot block, which is split into 2 parts b_q^1 and b_q^2 and put into $\widetilde{\mathbf{M}}_u^{\text{In}}$ and $\widetilde{\mathbf{M}}_u^{\text{Out}}$ respectively.
   - $\widetilde{\mathbf{M}}_u^{\text{In}} = \{b_1, ..., b_q^1\}$, $\widetilde{\mathbf{M}}_u^{\text{Out}} = \{b_q^2, b_{q+1}, b_{q+2}, ...\}$.
   - Let the set of partial nodes in γ_u that have extra capacity be $\mathbf{C}_u^{in} = \{c_i | c_i \in \gamma_u \text{ and } w_i < z(c_i, f_i)\}$. Then, the index q of $\widetilde{\mathbf{M}}_u$ satisfies $\sum_{j=1}^{q-1} l \times \varphi_j \leqslant \sum_{c_i \in \mathbf{C}_u^{in}} (z(c_i, f_i) - w_i) < \sum_{j=1}^{q} l \times \varphi_j$, and $\sum_{j=1}^{q-1} l \times \varphi_j + \frac{|b_q^1|}{|b_q^2|} \times \frac{l \times \varphi_q}{|b_q|} = \sum_{c_i \in \mathbf{C}_u^{in}} (z(c_i, f_i) - w_i)$

With above conditions, the maximum migration cost can be obtained by Equation 3.9

$$mc^{max} = e_{\text{In}} \times \sum_{b_{\text{In}} \in \bigcup_{p=1}^{U} \widetilde{\mathbf{M}}_u^{\text{In}}} |b_{\text{In}}| + e_{\text{Out}} \times \sum_{b_{\text{Out}} \in \bigcup_{p=1}^{U} \widetilde{\mathbf{M}}_u^{\text{Out}}} |b_{\text{Out}}| \tag{3.9}$$

where U is the amount of racks. Since the blocks with lower throughputs but larger sizes are migrated to other racks, the migration cost is the highest among all migration plans.

Proposition 2. *For a given migrated block set* $\mathbf{M}$*, the migration cost achieved by Equation 3.9 is the maximal migration cost.*

Proof. In this section, two relaxations are introduced. For a rack γ_u, the migration approach becomes a fractional knapsack problem under the relaxations. The size of knapsack is $\sum_{c_i \in \mathbf{C}_u^{in}} (z(c_i, f_i) - w_i)$. The items are migrated

blocks. The weight of the item is its throughput, and the value of the item is its block size. The objective is to minimize the total value of picked items. In Equation 3.9, the kept blocks are picked according to descending ratio between their throughput and their size, name by the ratio $\frac{weight}{value}$. Therefore, by means of a greedy algorithm, the optimal solution, minimal total size of kept migrated block size can be achieved. Therefore, the migration cost for the rack is maximal, and mc^{max} can be obtained as well.

□

In the model simplification, the migration plan is the worst case under the given migrated blocks, i.e., data blocks with higher block size per unit of throughput are migrated out of their own rack first. Therefore the upper bound of migration cost is achieved for the given migrated blocks.

3.3.3 Complexity Analysis of Model Simplification

In this section, a model simplification approach was introduced to reduce the complexity for computing the power consumption and the migration cost. Without the model simplification, the power consumption and the migration cost are obtained by means of Equation 3.5 and Equation 3.7. However, both processes require the migration. To compute the upper bound of power consumption using Equation 3.8, the required operation is to sort the nodes by their power consumption contribution factor. Therefore, the complexity for obtaining the upper bound of power consumption is $O(n\log n)$ (n is the amount of nodes). To compute the upper bound of migration cost using Equation 3.9, the required operation is to obtain the migrated blocks and sort the migrated blocks by their ratio between throughputs and block sizes. In migration process, the complexity for generating migration block set is $O(\sum_{i=1}^{n} |\mathbf{D}_i^r| \log |\mathbf{D}_i^r|)$, which can be further improved to $O(\sum_{i=1}^{n} |\mathbf{D}_i^r|)$=$O(m)$. For computing the upper bound of migration cost, we need to sort the migrated blocks for each rack. We assume each rack has same number of nodes. Let γ be the number of nodes within each rack, and there are Γ racks. Then complexity for computing upper bound of migration cost would be $O(\sum_{p=1}^{\Gamma} \sum_{i=1}^{\gamma} \mathbf{M}_i \log \gamma) = O(\sum_{i=1}^{n} \mathbf{M}_i \log \gamma)$, in which $\mathbf{M}$ presents the migration block set of node c_i . Therefore, in the end the obtain the upper bound of migration cost, the complexity is $O(\max\{m, \sum_{i=1}^{n} \mathbf{M}_i \log \gamma)\})$. An experiment is made in Section 6.4.3 to evaluate the model simplification.

3.4 Parameters' Benchmark

In Equation 3.7, the migration cost is obtained using two static parameters, the energy costs per mega byte of migration within a rack and between racks, namely e_{In} and e_{Out}. To obtain these values, a benchmark is executed.

This benchmark is executed in Grid5000 [79] at Nancy site graphene cluster. A Cassandra [11] system is deployed on 2 nodes. After the loading process, the system is waiting for a few minutes (5 minutes in the experiment) to obtain the power consumption in the idle status which is denoted as p_{Idle}. Then, a decommission process is executed on one of the nodes, which causes all the blocks in the node to be migrated to another one. The energy consumption within the decommission process is denoted as $e_{\text{Migration}}$. By means of choosing different node combinations, e_{In} and e_{Out} can be obtained. e_{In} and e_{Out} are calculated by Equation 3.10, in which $t_{\text{Migration}}$ indicates the execution time of the decommission process. The values of the parameters are shown in Table 3.6. In the Migration Execution phase, "Iterative Copy" [22] is adopted: the system still provides services while the migration is executed. Therefore, the static energy consumption is calculated within the power consumption, namely in Equation 3.5. Therefore, in Equation 3.10, the static energy consumption, $p_{\text{Idle}} \times t_{\text{Migration}}$, is omitted.

$$e_{\text{In}|\text{Out}} = \frac{e_{\text{Migration}} - p_{\text{Idle}} \times t_{\text{Migration}}}{\sum_{b_{g,k} \in \mathbf{M}_{\text{In}|\text{Out}}} |b_{g,k}|} \tag{3.10}$$

Table 3.6: Migration Cost for Unit Block

Parameter	Value
e_{In}	0.8 Joule/MB
e_{Out}	1.0 Joule/MB

3.5 Model Extension

In Section 3.2.1, the nodes are considered homogeneous. With the following extensions, the model can be applied to a heterogeneous cluster.

1. The nodes in a heterogeneous cluster can be categorized according to their architectures. Otherwise, the efforts for obtaining static parameters are unacceptable. Considering the types of the workload, the amount of architectures, and the number of available frequency options, the efforts to conduct the benchmark, Section 3.1 would be to high.

2. The capacity measurement function $z(c_i, f)$ should be specialized for different categories of nodes since the frequency options may not be the same.

3. The power consumption estimation function should be specialized for different categories of nodes.

4. When computing the migration cost, e_{In} and e_{Out} should consider the difference of node's architecture.

In general, the model's static parameters which are related to the node's architecture should be obtained according to the different architectures.

3.6 Summary

In this chapter, we first investigate the energy efficiency behavior of the system under different frequency settings. We found out that, for a certain frequency setting, the node within the cluster has a maximum throughput limitation. We define this value as the capacity of the node under the frequency setting. Based on this conclusion, we propose a generic model for frequency selection approach, in which we define the system state transformation process. Then, a specialized model is derived from the generic model to cope with resource provision within cloud database, in which the core concepts of the generic model, including the estimation of power consumption, the estimation of migration cost, and the migration process, are redefined. In the model simplification approach, the upper bound values of power consumption and migration cost are obtained without the entire migration process.

Chapter 4

Frequency Selection with Bounded Problem

In this chapter, we treat frequency selection problem as a bounded problem. According to different objectives, we have 2 kinds of bounded problem in this chapter. At first, the constraints and objectives are introduced. Then we present two algorithms, namely Nonlinear Programming Algorithm and Multi-Phases Algorithm to solve the bounded problems. At last, a series of experiments are given to evaluate the algorithms in terms of accuracy and performance.

4.1 Objective

In this section, we first introduce two basic objectives about power consumption and migration cost respectively. Then, the basic objectives are combined into two kinds of bounded problems, power consumption bounded problem and migration cost bounded problem.

4.1.1 Basic Objectives

Energy efficiency of cloud database systems is a ratio between the workload and the energy consumption according to Equation 3.3. When the workload is constant, the energy consumption should be minimized in order to improve the energy efficiency of the system. In the modeling, energy consumption of the system within a time window Δt is estimated by Equation 3.2. The energy consumption comes from two parts, energy for executing the workloads and energy for the migration process. Therefore, there are two objectives naturally, namely a power consumption related objective and a migration cost related

objective.

The power consumption of the cloud database system is estimated by Equation 3.5. The power consumption related objective is shown by Equation 4.1, in which $\mathbb{P}$ is the maximum power consumption needed to satisfy the system's need. The objective is to minimize $\mathbb{P}$.

$$p(s) \leq \mathbb{P} \tag{4.1}$$

The migration cost is defined by Equation 3.7. Intuitively, Equation 4.2 is treated as the migration cost objective, in which $\mathbb{MC}$ represents the maximum energy needed to complete the migration process. The objective is to minimize $\mathbb{MC}$.

$$mc(\widetilde{s}) \leq \mathbb{MC} \tag{4.2}$$

4.1.2 Bounded problems

Bounded problems appear when the user sets constraints to the objectives. In general, there are 2 types of bounded problems within frequency selection problem. One is the bounded power consumption problem, the other one is the bounded migration cost problem.

The Bounded Power Consumption Problem. The power consumption constraint $\mathbb{P}^b$ refers to the maximum power consumption can be provided to the system. With this constraint, the objective is simplified into minimizing the migration cost which is denoted as $\mathbb{P}^b\mathbb{MC}^{min}$. In other words, $\mathbb{P}^b\mathbb{MC}^{min}$ represents the scenario that minimizes migration cost within a maximum power consumption.

The Bounded Migration Cost Problem. The migration cost constraint $\mathbb{MC}^b$ refers to the maximum migration cost. With this constraint, the objective is simplified into minimizing the power consumption which is denoted as $\mathbb{MC}^b\mathbb{P}^{min}$. In other words, $\mathbb{MC}^b\mathbb{P}^{min}$ represents the lowest achievable power consumption within the migration cost requirement.

In this chapter, our objective is to present corresponding algorithms to solve above bounded problems.

4.2 Nonlinear Programming Algorithm

Intuitively, the bounded problems can be treated as a nonlinear programming problem [83]. By means of the linear programming solver, Gurobi [84], the optimal solution can be obtained. In this section, a nonlinear programming

model, based on specialized model (Section 3.2.2) is introduced. Then the corresponding constraints and objectives are presented.

4.2.1 Nonlinear Programming Model

To apply the nonlinear programming solver, the following parameters are introduced. There are U racks within the cluster, and there are γ nodes within each rack. The node set $\mathbf{C}$ is a consecutive set, denoted as $\mathbf{C} = c_1, c_2..., c_\gamma, c_{\gamma+1}, c_{\gamma+2}, ..., c_{2\gamma}.....$ In here, we assume that every cluster has the same amount of nodes. Some virtual nodes can be introduced otherwise. The available frequency options set is denoted as η, and $|\eta| = m$. Equation 4.3 defines a binary parameter x_p^i which equals to 1 when and only when frequency $f_p \in \eta$ is assigned to the node c_i. Therefore, the capacity of c_i is denoted as z_i, which can be presented as Equation 4.4.

$$x_p^i = \begin{cases} 1 \text{ If } c_i \text{ is assigned with } f_p \\ 0 \text{ If } c_i \text{ is not assigned with } f_p \end{cases} \tag{4.3}$$

$$z(c_i, f_i) = \sum_{p=1}^{m} z(c_i, f_p) \times x_p^i \tag{4.4}$$

Equation 4.5 defines a binary parameter $a_{g,k}^i$, which indicates the block assignment. $a_{g,k}^i$ equals to one when and only when block $b_{g,k}$ is located on node c_i. Correspondingly, the workload w_i of c_i can be presented as Equation 4.6. Combining Equation 4.4 and Equation 4.6, the power consumption estimation function Equation 4.7 can be presented as Equation 4.7.

$$a_{g,k}^i = \begin{cases} 1 \text{ If } b_{g,k} \text{ is assigned to } c_i \\ 0 \text{ If } b_{g,k} \text{ is not assigned to } c_i \end{cases} \tag{4.5}$$

$$w_i = \sum_{g=1}^{h} \sum_{k=1}^{r} \left(a_{g,k}^i \times l \times \varphi_{g,k} \right) \tag{4.6}$$

$$p(s^*) = \sum_{i=1}^{n} \left(\sum_{p=1}^{m} \left(x_p^i \times c_{f_p}^{idle} \right) + \frac{\sum_{g=1}^{h} \sum_{k=1}^{r} \left(a_{g,k}^i \times l \times \varphi_{g,k} \right)}{\sum_{p=1}^{m} z(c_i, f_p) \times x_p^i} \times \sum_{p=1}^{m} \left(x_p^i \times \left(c_{f_p}^{max} - c_{f_p}^{idle} \right) \right) \right) \tag{4.7}$$

According to Equation 3.7, the migration sets $\mathbf{M}_{\text{In}}$ and $\mathbf{M}_{\text{Out}}$ are required to estimate the migration cost. However, the concept of migration sets is

between two time windows. To show the block assignment of the last time window (time window $\Delta t - 1$), a superscript -1 is applied. For example ${a^i_{g,k}}^{-1}$ indicates whether block $b_{g,k}$ was placed in node c_i in the time window $\Delta t - 1$. To describe the migration sets in nonlinear programming solver, the following parameters are introduced.

$$A_{g,k} = \sum_{i=1}^{n} \left(i \times a^i_{g,k} \right) \tag{4.8}$$

Equation 4.8 defines parameter $A_{g,k}$ which gives the index of the node where the block $b_{g,k}$ is placed. Correspondingly, the location index of $b_{g,k}$ in the last time window is denoted as ${A_{g,k}}^{-1}$.

$$\begin{aligned} m_{g,k} &= \begin{cases} 0 \text{ if } {A_{g,k}}^{-1} = A_{g,k} \\ 1 \text{ if } {A_{g,k}}^{-1} \neq A_{g,k} \end{cases} \\ m^{\text{Out}}_{g,k} &= \begin{cases} 0 \text{ if } \left\lfloor \frac{{A_{g,k}}^{-1}}{\gamma} \right\rfloor = \left\lfloor \frac{A_{g,k}}{\gamma} \right\rfloor \\ 1 \text{ if } \left\lfloor \frac{{A_{g,k}}^{-1}}{\gamma} \right\rfloor \neq \left\lfloor \frac{A_{g,k}}{\gamma} \right\rfloor \end{cases} \end{aligned} \tag{4.9}$$

In Equation 4.9, binary parameter $m_{g,k}$ and $m^{\text{Out}}_{g,k}$ are introduced. $m_{g,k}$ equals to one when and only when ${A_{g,k}}^{-1}$ does not equal to $A_{g,k}$. Since ${A_{g,k}}^{-1}$ and $A_{g,k}$ reveal the placements of $b_{g,k}$ in time window $\Delta t - 1$ and Δt respectively, $m_{g,k}$ indicate whether block $b_{g,k}$ has been migrated in Δt. γ shows the number of nodes in each cluster. $m^{\text{Out}}_{g,k}$, $\left\lfloor \frac{A_{g,k}}{\gamma} \right\rfloor$ gives the rack index which contains $b_{g,k}$ in Δt. By means of comparing two rack indexes of $b_{g,k}$ in $\Delta t - 1$ and Δt, $m^{\text{Out}}_{g,k}$, a binary parameter, reveals whether $b_{g,k}$ is migrated between two racks. Therefore, the migrated block sets can be obtained based on following facts:

- If a block is migrated ($m_{g,k} = 1$), but it is not migrated between racks ($m^{\text{Out}}_{g,k} \neq 1$), then the block belongs to $\mathbf{M}_{\text{In}}$;
- If a block is migrated ($m_{g,k} = 1$), and it is migrated between racks ($m^{\text{Out}}_{g,k} = 1$), then the block belongs to $\mathbf{M}_{\text{Out}}$.

By means of $m_{g,k}$ and $m^{\text{Out}}_{g,k}$, the migration cost estimation function 3.7 can be rewritten as Equation 4.10. In Equation 4.10, the migration cost weight for every block is computed by $m_{g,k} \times \left(m^{\text{Out}}_{g,k} \times e_{\text{In}} + \left(1 - m^{\text{Out}}_{g,k} \right) \times e_{\text{Out}} \right)$. Clearly, when $b_{g,k}$ is not migrated, the migration cost weight is 0, since $m_{g,k} = 0$. Correspondingly, the migration cost weight is e_{In} if the block is migrated within the rack while the weight is e_{Out} if the block is migrated between racks.

$$mc(\widetilde{s}) = \sum_{g=1}^{h} \sum_{k=1}^{r} m_{g,k} \times \left(m_{g,k}^{\text{Out}} \times e_{\text{In}} + \left(1 - m_{g,k}^{\text{Out}}\right) \times e_{\text{Out}} \right) \times |b_{g,k}| \tag{4.10}$$

The specialized model is redefined in nonlinear programming model manner by introducing extra parameters. The symbols used in nonlinear programming model are concluded by Table 4.1.

Table 4.1: Specifications of the symbols

Symbol	Type	Comment
l	Float	The predicted system throughput in Δt
$\varphi_{g,k}$	Float	The predicted accessed probability of block $b_{g,k}$
$z(c_i, f)$	Float	The capacity (maximum throughput) of c_i under frequency f
x_p^i	Binary	A binary indicator, which equals 1 only when c_i is assigned with frequency f_p
$a_{g,k}^i$	Binary	A binary indicator which equals 1 only when $b_{g,k}$ is assigned to node c_i
$A_{g,k}^i$	Integer	The index of a node which contains $b_{g,k}$
$m_{g,k}$	Binary	A binary indicator which equals 1 only when $b_{g,k}$ is migrated in Δt
$m_{g,k}^{\text{Out}}$	Binary	A binary indicator which equals 1 only when $b_{g,k}$ is migrated to another rack in Δt

To be noticed, in Table 4.1, only x_p^i and $a_{g,k}^i$ are the parameters that need to be computed by the nonlinear programming solver. l, $\varphi_{g,k}$, $z(c_i, f)$ are known parameters, and $A_{g,k}^i$, $m_{g,k}$ $m_{g,k}^{\text{Out}}$ are computed according to $a_{g,k}^i$. Therefore, The bounded problems can be treated as mixed integer nonlinear programming problems since only binary parameters are involved. The amount of parameters is shown as Equation 4.11, where $|\eta|$ is number of frequency options, n is number of nodes, and h, r are number of blocks and number of replicas for each block respectively.

$$|\eta| \times n + h \times r \times n \tag{4.11}$$

4.2.2 Constraints

According to the descriptions in previous section, some constrains are shown as below:

$$\forall i \in [1,n], \forall p \in [1,m] \quad x_p^i \in \{0,1\} \tag{4.12}$$

$$\forall i \in [1,n] \quad \sum_{p=1}^{m} x_p^i = 1 \tag{4.13}$$

$$\forall g \in [1,h], \forall k \in [1,r], \forall i \in [1,n] \quad a_{g,k}^i \in \{0,1\} \tag{4.14}$$

$$\forall g \in [1,h], \forall k \in [1,r] \quad \sum_{i=1}^{n} a_{g,k}^i = 1 \tag{4.15}$$

$$\forall i \in [1,n], \forall g \in [1,h] \quad \sum_{k=1}^{r} a_{g,k}^i \leq 1 \tag{4.16}$$

$$\forall g \in [1,h], \forall k \in [1,r] \quad \varphi_{g,k} \in [0,1], \sum_{g=1}^{h} \sum_{k=1}^{r} \varphi_{g,k} = 1 \tag{4.17}$$

$$\forall i \in [1,n], \quad \sum_{p=1}^{m} z(c_i, f_p) \times x_p^i \geq \sum_{g=1}^{h} \sum_{k=1}^{r} \left(a_{g,k}^i \times l \times \varphi_{g,k} \right) \tag{4.18}$$

Constraints 4.12 and 4.13 show the rules of frequency selection for each node. Constraint 4.12 indicates that every node should be configured or not configured to a given frequency. Constraint 4.13 shows that every node should be under only one frequency.

Constraints 4.14, 4.15 and 4.16 show the rules of block assignment. Constraints 4.14 and 4.15 indicate that each data block should be assigned to only one node. Constraint 4.16 means that if two data blocks are replicas of the same block, they cannot be assigned to the same node.

Constraint 4.17 shows that the sum of the probabilities of all blocks being accessed should equal to 1 (at least one block is accessed during Δt).

Constraint 4.18 shows that the amount of workload allocated to one node cannot exceed the capacity of the node.

4.2.3 Nonlinear Programming Objective Function

As described in Section 4.1, there are 2 bounded problems, $\mathbb{P}^b\mathbb{MC}^{min}$ and $\mathbb{MC}^b\mathbb{P}^{min}$. For the Nonlinear Programming Algorithm, there are 2 types of objective function ($\mathbb{P}^{min}$, $\mathbb{MC}^{min}$) and 2 different types of constraint functions, $\mathbb{P}^b$, $\mathbb{MC}^b$. Due to Gurobi's properties, the power consumption estimation function need to be changed mathematically to meet the solver's requirements.

Equation 4.7 can be used to estimate the power consumption in nonlinear programming model. However, it cannot be applied in practice. In Equation 4.7, the main parameters are $a_{g,k}^i$ and x_p^i. However, Gurobi cannot manipulate parameters which are denominators. Therefore, we rewrite Equation 4.7 as

Equation 4.19. In Equation 4.19, the binary parameter x_p^i is moved at the outermost of the equation. From a mathematic point of view, x_p^i is to determine a frequency f_p for c_i. Therefore $\sum_{p=1}^{m} x_p^i \times z(c_i, f_p)$ is determined and constant when a certain frequency f_p is configured to c_i. When x_p^i equals to 0, then the total energy consumption of the node is 0 under frequency f_p since the corresponding frequency is not assigned to the node.

$$p(s^*) = \sum_{i=1}^{n} \sum_{p=1}^{m} x_p^i \times \left(c_{f_p}^{idle} + \frac{\sum_{g=1}^{h} \sum_{k=1}^{r} \left(a_{g,k}^i \times l \times \varphi_{g,k} \right)}{z(c_i, f_p)} \times \left(c_{f_p}^{max} - c_{f_p}^{idle} \right) \right) \quad (4.19)$$

Combining Equation 4.19 and Equation 4.10, we have 2 nonlinear programming objective functions shown as Equation 4.20 and Equation 4.21 that represent $\mathbb{P}^b \mathbb{MC}^{min}$ and $\mathbb{MC}^b \mathbb{P}^{min}$ respectively.

$$\begin{aligned} \textbf{minimize} \quad & mc(\widetilde{s}) \\ \textbf{subject to} \quad & p(s*) \leq \mathbb{P}^b \end{aligned} \quad (4.20)$$

$$\begin{aligned} \textbf{minimize} \quad & p(s*) \\ \textbf{subject to} \quad & mc(\widetilde{s}) \leq \mathbb{MC}^b \end{aligned} \quad (4.21)$$

Using a nonlinear programming solver Gurobi, the optimal solution can be found. However, because of the complexity of the problem and the amount of parameters, the usage of nonlinear programming model is limited. The corresponding experiments are given in Section 4.4

4.3 Multi-Phases Algorithm

The Nonlinear Programming Algorithm has an outstanding advantage to obtain the optimal solution for the bounded problems. Due to the complexity of the algorithm, the bottleneck is its performance for large scale problems. In order to overcome this bottleneck, we introduce a Multi-Phases Algorithm which gives suboptimal solutions.

The frequency selection problem can be treated as a search problem within the frequency combination space. Assume there are 3 available frequency options and 2 nodes, then the amount of frequency combinations is $3^2 = 9$. We can evaluate the 9 solutions and compute the estimated power consumption

and estimated migration cost according to Equation 4.7 and Equation 3.7 after the migration process. However, in a larger case, there are 8 frequency options and 30 nodes, and the amount of frequency combinations is 8^{30}. The time of entire search is unacceptable. In our environment, the optimal solution of above case can be found by the entire searching within 5 hours.

In order to solve the bounded problem faster, a search algorithm, Multi-Phases Algorithm, is proposed in this section. The initial idea of Multi-Phases Algorithm is to reduce the solution space of the frequency selection problem. A certain frequency combination is defined as one of the solutions. For a given solution, the maximum supported workload can be obtained as $\sum_{i=1}^{n} z(c_i, f_i)$. According to the total capacity of the cluster and the predicted throughput l, we have the following constraints for the solutions:

1. For a given solution, when $\sum_{i=1}^{n} z(c_i, f_i) < l$, then the solution is abandoned since it cannot provide enough resources to execute the workload in next time window.

2. For a given solution, when $\sum_{i=1}^{n} z(c_i, f_i) > l + \epsilon$, in which ϵ is a threshold parameter, then the solution is abandoned also, since it provides too much resources for the next time window.

The above constraints are given based on the intuition about resource provisioning problem. If under provisioning occurs, i.e. $\sum_{i=1}^{n} z(c_i, f_i) < l$, then the system cannot get enough resources, which leads to SLA violation. If over provisioning occurs, i.e. $\sum_{i=1}^{n} z(c_i, f_i) > l + \epsilon$, then it increases energy waste and costs [23]. Therefore in Multi-Phases Algorithm, we barely consider the solutions which provide capacities between l and $l + \epsilon$. Then all the solutions are evaluated according to problem type and corresponding constraints.

Multi-Phases Algorithm includes 6 phases:

1. **Frequency Locating Phase**. In this phase, the boundary frequency combinations are located. The boundary frequency combinations include the frequency combinations providing larger but closest capacity with l and the frequency combinations providing lower but closest capacity with $l + \epsilon$.

2. **Frequency Traversal Phase**. The frequency combinations between boundary frequency combinations are traversed. The founded combinations are treated as candidate solutions, denoted as *s_comb*s.

3. **Frequency Assignment Phase**. The frequencies in each *s_comb* are assigned to the cluster.

4. **Frequency Filter Phase**. In this phase *s_comb*s are filtered according to the upper bound value on the considered constraints.

5. **Frequency Search Phase**. According to the objective of the problem and the upper bound value, a few candidate solutions are searched.

6. **Frequency Evaluation Phase**. The candidate solutions are evaluated by the migration process (Chapter 6) and the best is chosen as the final solution according to problem type ($\mathbb{P}^b\mathbb{MC}^{min}$ or $\mathbb{MC}^b\mathbb{P}^{min}$).

This section includes 7 subsections. In first 6 subsections, each phase of Multi-Phases Algorithm is introduced. In Section 4.3.7, the behavior and the complexity of the algorithm is analyzed.

4.3.1 Frequency Locating Phase

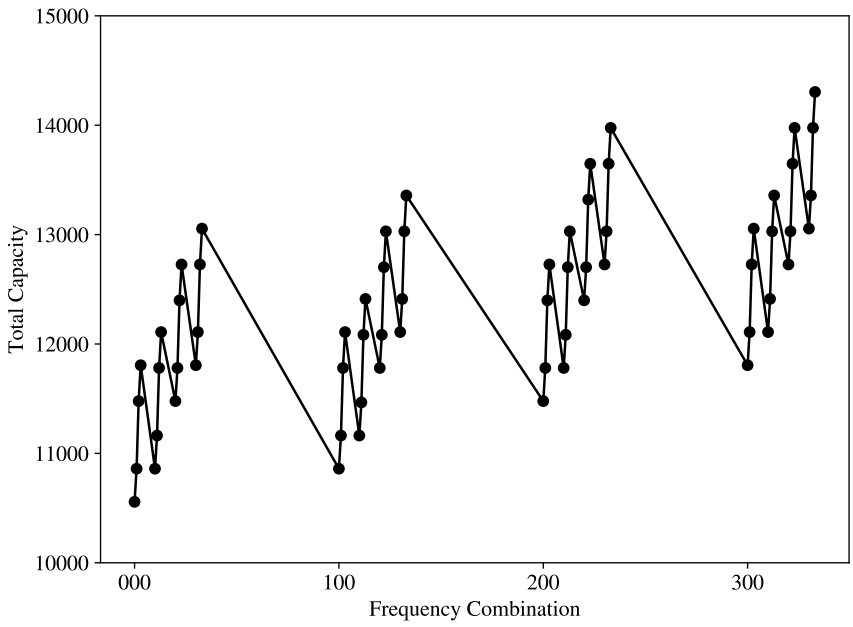


Figure 4.1: Frequency Solution Space

In frequency combination space, we have m^n combinations, since there are n nodes and each node can be assigned with m frequency options. An example with a small case, 3 nodes and 4 available frequency options, is shown as Figure 4.1. In Figure 4.1, frequency configuration for the cluster is denoted as frequency combinations. For example 000 is a frequency combination, in which f_0 is assigned to all the nodes. In the figure, we can observer that, some

combinations provide the same total capacity, for example both 311 and 131 assign f_3 to one node and f_1 to the other two nodes. However, in this case, the frequency combinations provide assignment information. For example, 311 assigns f_3 to c_1, f_1 to c_2 and c_1 to c_3. However, in the other hand, this assignment information creates lots of redundancy. In order to reduce the solution space, we introduce the frequency combination encoding method. In the encoding process, we have following constraints:

1. Frequency combinations are coded as m base digital numbers. For example if there are 11 available frequencies and 3 nodes, the minimal combination is 000, representing that f_0 is assigned to all the nodes. While the maximal combination is aaa, in which f_{10} is assigned to all the nodes.

2. For each encoded frequency combination, a digit should not be smaller than its following digits.

Constraint (1) defines an encoding method for frequency combinations. Constraint (2) is used to eliminate redundancies within the frequency combinations. For example, with constraint (2), solution 001 and 010 are invalid. We would prefer 100 in this case.

Using the constraints above, the solution space is reduced from m^n to C^n_{m+n-1}. With the encoding method, the frequency combination turn into a k-multi-combination problem in which k in our case represents the number of nodes, namely n. The proof of the combination number can be given by stars and bars theory [85]. Figure 4.2 shows an example of frequency locating and traverse in which the frequency combination are encoded. It can be observed that the solution space is reduced greatly.

The introduction of the encoding method has its pros and cons. The most important advantage is that it can reduce the solution space tremendously. For example, with 30 nodes and 8 available frequency options, the solution reduced from $8^{30} \approx 1.238e^{27}$ to $C^{30}_{8+30-1} \approx 1.03e^7$. The disadvantage is that the encoding method losses frequency assignment information. For example, with 3 nodes and 2 available frequency options, frequency combination 110 and 101 are treated as the same since 101 is invalid according to the constraints. Both of them will assign 2 nodes with f_1 and 1 node with f_0. However, if we consider the assignments, we have C^3_3 different assignment methods, for example 110, 101, 011. How Multi-Phases Algorithm deals with the assignment problem is discussed in Section 4.3.6.

In Multi-Phases Algorithm, we only consider the frequency combinations which provides total capacity between l and $l + \epsilon$. ϵ is a user predefined

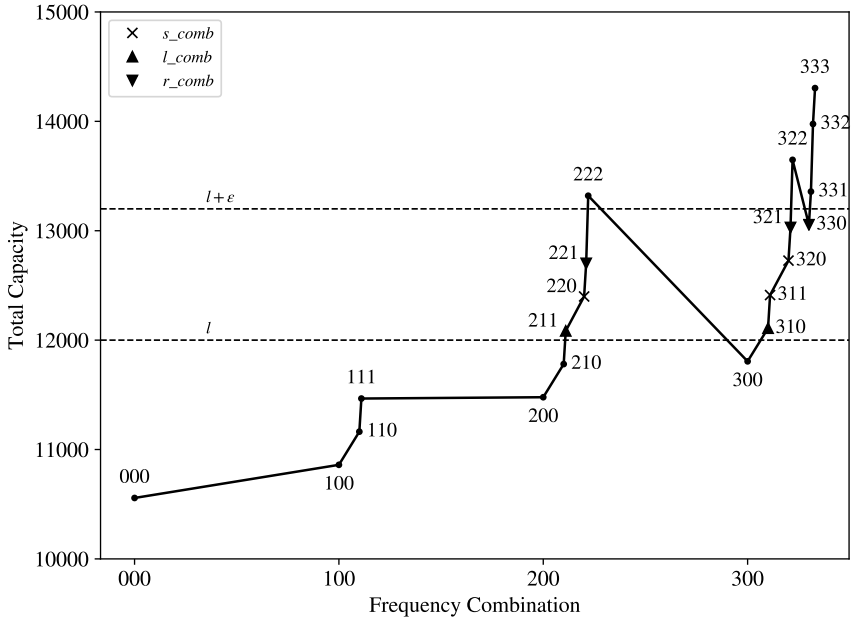


Figure 4.2: Example of Frequency Locating and Traverse

parameter. In this phase, we mainly focus on the boundary combinations. There are two kinds of boundary combinations: 1. the frequency combinations providing larger but closest capacity with l which is denoted as *l_comb* (left combination), and 2. the frequency combinations providing lower but closest capacity with $l + \epsilon$ which is denoted as *r_comb* (right combination).

Figure 4.2 shows an example of frequency combination locating and traverse. In the figure, we shows two boundary lines: l and $l + \epsilon$, and three type of frequency combination sets: *l_comb*, *r_comb* and *s_comb*. s_{comb} indicates the combinations whose total capacities are between l and $l + \epsilon$, which are traversed in next phase, Section 4.3.2.

In this phase, the combinations *l_comb* and *r_comb* are located. For a given frequency combination, the total capacity is $\sum_{i=1}^{n} z(c_i, f_i)$. However, it is not easy to have the frequency combinations when the total capacity is given. There are 2 reasons for this bottleneck. Firstly, given a particular capacity value, the corresponding combination might not exist. Secondly, even if the boundary combinations (upper bounded or lower bounded) are required, there might exist several combinations. In the general point of view, the relationship between combinations and their total capacity is not linear, for example in Figure 4.2, the capacity of 300 is lower than 222. However, in a certain range, the maximal total capacity of the range and the minimal total capacity can be located. For example, within range 300 to 333, the combination that achieves

maximal total capacity of the range is 333, and the combination that achieve minimal total capacity of the range is 300. Recursively, the same fact occurs within its sub range also, for example range 300 to 310, and 320 to 322. Based on the above fact, the frequency combination locating algorithm, Algorithm 1, is propose.

Algorithm 1 shows the core function $LocateFrequency$ in this phase, which locates the boundary combinations which meet the requirement. The basic idea of frequency locating algorithm is to find the junction points between traverse line and boundary lines. When the minimal ranges of the points are found, then the bound points are located. For example, the first minimal range of junction point is 210 to 211. Since the junction point is between traverse line and the lower boundary line, then 211 is returned as one of l_comb.

Parameters of the function are l, ϵ, $digitNum$, $digitMax$, $lower$. l indicates the total predicted workload and the value should be l or $l+\epsilon$. $digitNum$ indicates the number of digits in the combination and $digitMax$ gives the maximum value for each digit. Namely, $digitNum$ and $digitMax$ refer to the number of nodes and the number of available frequency options respectively. $lower$ is a binary parameter, which is used to indicate which bound is required. If $lower$ is $True$ then the upper bound is searched, otherwise the lower bound is searched. At the end, $LocateFrequency$ returns the boundary combinations according to the parameters. The main idea for $LocateFrequency$ is to narrow the search space to find the boundaries for l. $LocateFrequency$ takes advantage of the fact that for the combinations that share the same prefix, the combinations providing maximum/minimum total capacity can be located. Therefore, a check between ranges and their sub-ranges is applied. l is tested for each range of the frequency traversing order, and by narrowing down the range, the boundary combinations can be obtained.

In Algorithm 1, line 4 to line 12 check the basic ranges (only the first digital number is checked). To be noticed, the checking starts from 1 because the combination $000 - 000$ is not a range. In range checking, if l is between the provided throughputs by the minimal combination of the range and the maximal combination of the range (line 9), then the range is stored (line 10) and awaits for future narrow down. Meanwhile, in each loop, the combination from the last previous range is computed (line 5) and compared with the first combination in the current range (line 8) by a function ChooseComb. If the l is between $prev$ and $left$ then the eligible one is put to result set.

From line 13 to line 26, the ranges are narrowed down by each digit, and the possible ranges are selected. However, in each process of the range narrowing down, the combination from previous sub-range is computed(line 18) and compared within first combination in the current sub-range (line 21). To

Algorithm 1 Frequency Combination Locating Algorithm

```
function LocateCombination(l, ϵ, digitNum, digitMax, lower)
    result ← []
    prefixs ← []
    for j ∈ [1, digitMax] do
        prev ← [j − 1] × digitNum
        left ← [j] + [0] × (digitNum − 1)
        right ← [j] × digitNum
        result ← result ∪ ChooseComb(prev, left, l, ϵ, lower)
        if left <= l <= right then
            prefixs ← prefixs ∪ [[j]]
        end if
    end for
    for i ∈ [(digitNum − 1), ..., 2] do
        tempPrefix ← []
        for prefix ∈ prefixs do
            lastMax = prefix[−1]
            for j ∈ [1, lastMax] do
                prev ← prefix + [j − 1] × i
                left ← prefix + [j] + [0] × (i − 1)
                right ← prefix + [j] × i
                result ← result ∪ ChooseComb(previous, left, l, ϵ, lower)
                if left <= l <= right then
                    tempPrefix ← tempPrefix ∪ (prefix + [j])
                end if
            end for
        end for
        prefixs ← tempPrefix
    end for
    for prefix ∈ prefixs do
        lastMax = item[−1]
        for j ∈ [1, lastMax] do
            left ← prefix + [j − 1]
            right ← prefix + [j]
            result ← result ∪ ChooseComb(previous, left, l, ϵ, lower)
        end for
    end for
    return result
end function
```

$l = 12000, \epsilon = 1500, digitNum = 3, digitMax = 3, lower = False$				
for	$loop$	$result$	$prefixs/tempPrefixs$	$combs(prev/left - right)$
1	1	[]	[]	000/100-111
1	2	[]	[[2]]	111/200-222
1	3	[]	[[2],[3]]	222/300-333
2	1	[]	[]	200/210-211
2	2	[]	[[21]]	300/310-311
2	3	[310]	[[21]]	311/320-322
2	4	[310]	[[21]]	322/330-333
3	1	[310]	[[21]]	None/210-211
3	2	[310,211]	-	-
RETURN		[310,211]		

Table 4.2: Example of Frequency Combination Locating

be noticed, the last digit in a combination is not checked (line 13), because with the last digit, the provided throughputs are monotonic. For example, the throughputs provided by 330, 331, 332, 333 are monotonic increasing. Line 29 to line 36, the boundaries combinations are selected. The boundary combinations are returned in the end(line 37).

Algorithm 2 Choose Combination Algorithm

```
1: function ChooseComb(l_comb, r_comb, l, ϵ, lower)
2:     if l_comb ≤ l ≤ r_comb and not lower then
3:         return r_comb
4:     end if
5:     if l_comb ≤ l + ϵ ≤ r_comb and lower then
6:         return l_comb
7:     end if
8:     return NULL
9: end function
```

Algorithm 2 shows the function *ChooseComb*. The function takes two combinations, *l_comb* and *r_comb*, two boundary line, l and $l + \epsilon$, and bound type *lower* as input parameters. When the resources provided by two combinations have a junction with the boundary lines, the proper bound point is returned.

Table 4.2 shows the executing process of Algorithm 1 for locating the *l_comb*, [211, 310], in Figure 4.2. The input parameters are $l = 12000, \epsilon = 1500, digitNum = 3, digitMax = 3, lower = False$. Table 4.2 shows the changes of variables within the algorithm. 5 columns exist in the table: for

represents the "For Blocks" within the algorithm, line 4 to line 12, line 13 to line 28, and line 29 to line 36; *loop* shows the current executing loop within each "For Block"; $result$ represents the variable of $result$ that is returned when the algorithm is terminated; $prefixs/tempPrefixs$ represents variable of $prefixs$ within the first and last "For Block", and variable of $tempPrefixs$ within the second 'For Block'; combs represents the temporary variable within the algorithm, $prev$, $left$ and $right$. Since $digitMax$ is set to 3, the first "For Block", $for = 1$, has 3 loops, and 3 ranges are examined. In Figure 4.2, the lower boundary line has two intersection points with the total capacity line, and the points go through range 200-222 and 300-333. Therefore, in the end of first "For Block", $prefixs$ equals to [[2],[3]]. In the second "For Block", the algorithm explorers the reminded digitals, and the last digital is checked in the third "For Block". To be noticed, the second "For Block" consist of 2 nested "For Block", however in the example, it is shown as 1 "For Block" because there is only 1 digital is checked. In the second "For Block" and second *loop*, the range 300/310-311 is checked. However, when the range 300-311 is checked by $ChooseComb$ in line 21, 311 is matched result condition and is put into the result set. In the third "For Block", 210 is found out as one of the boundary point and is put into the result set. In the end, the result [310, 211] is returned.

4.3.2 Frequency Traversal Phase

In this phase, the candidate frequency combinations that provided the capacities between l and $l+\epsilon$ are generated. In the previous phase, the boundary combinations are located, which are denoted *l_comb* and *r_comb* for lower boundary combinations and upper boundary combinations respectively. The candidate frequency combinations are denoted as *s_comb* (**S**elected **Comb**inations). An example of boundary combinations and selected combinations are shown in Figure 4.2, in which the selected combinations between all the boundary combinations are found.

Frequency traversal approach is based on a basic fact that:

1. The frequency combinations after a *l_comb* might be a selected combination, but the frequency combinations before the *l_comb* are not.

2. The frequency combinations before a *r_comb* might be a selected combination, but the frequency combinations after the *r_comb* are not.

Based on the above facts, the Frequency Generation Algorithm, Algorithm 3, is proposed. The input of the algorithm is *boundaries* which is a sorted list of all the boundary combinations generated in the frequency locating phrase, and the output of the algorithm is *s_comb* list including *l_comb* and *r_comb*.

Algorithm 3 Frequency Generation Algorithm

```
1: function GenerateCombinations(boundaries, l, ϵ)
2:     result ← []
3:     amount ← len(boundaries)
4:     last ← NULL
5:     for i ∈ [0, amount) do
6:         start ← boundaries[i]; end ← NULL
7:         if start = l_comb and i < amount − 1 then
8:             end ← boundaries[i + 1]
9:         end if
10:        if start = r_comb and i > 0 then
11:            end ← boundaries[i − 1]
12:        end if
13:        if curt = l_comb then
14:            result ← result ∪ ForwardTraversal(start, end)
15:            last ← result[−1]
16:        else
17:            result ← result ∪ BackwardTraversal(start, end, last)
18:        end if
19:    end for
20:    return result
21: end function
```

In the algorithm, line 5 to line 19 traverse the frequency combinations from each boundary combination. For *l_comb*, it uses Frequency Forward Traversal Algorithm, Algorithm 4, to get all the selected frequency combinations. Comparably, it applies Frequency Backward Traversal Algorithm, Algorithm 5, for *r_comb*.

Algorithm 4 shows the key function of Frequency Generation Algorithm, $ForwardTraversal$, in which the selected combinations after the given boundary combination are traversed. The inputs of the algorithm are *begin* combination (*l_comb*), *end* combination (*l_comb*, *r_comb* or $NULL$), workload l and ϵ. Line 2 to line 4 initialize some parameters, in which *result* is the *s_comb* list, *next* is an intermediate variable that indicates the next frequency combination within the traverse process, and i indicates the position that is used to increase the combination. Line 4 to line 30 is the traversing process. The stop criteria condition of the traverse process is to meet the *end* combination or to meet one combination that falls outside of the boundary lines. Within the loop, line 6 to line 23, the next combination is constructed and the value of i is updated for next loop. Line 24 to line 29, the provided capacity of the *next*

Algorithm 4 Frequency Forward Traversal Algorithm

```
function ForwardTraversal(begin, end, l, ϵ)
    result ← [begin]
    next ← begin
    i ← argmin(begin)
    while next ≠ end do
        next[i] ← next[i] + 1
        if i ≠ len(next) − 1 then
            for j ← [i + 1, len(next)] do
                next[i] ← 0
            end for
        end if
        if i = len(begin) then
            j ← len(begin) − 1
            k ← j − 1
            while next[i] = next[j] and k > 0 do
                j ← j − 1 and k ← k − 1
            end while
            if k = 0 and begin[j] = begin[k] then
                i ← k
            else
                i ← j
            end if
        end if
        capacity ← Capacity(next)
        if l ≤ capacity ≤ l + ϵ then
            result ← result ∪ next
        else
            break
        end if
    end while
    return result
end function
```

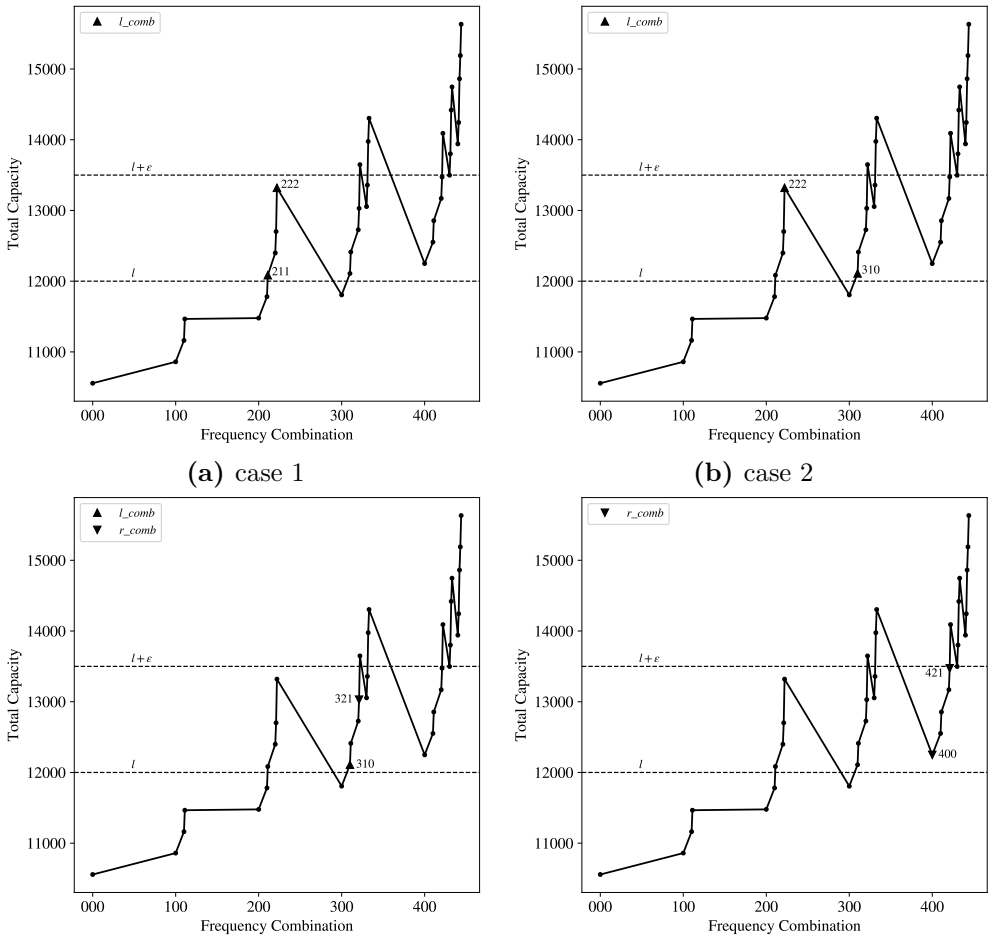
Algorithm 5 Frequency Backward Traversal Algorithm

```
1: function BackwardTraversal(begin, end, last, l, ϵ)
2:     result ← [begin]
3:     next ← begin
4:     check_list ← [len(begin) − 1, ..., 0]
5:     while next ≠ end and next ≠ last do
6:         pos ← −1
7:         for i ∈ check_list do
8:             if next[i] ≠ 0 then
9:                 pos ← i
10:                break
11:            end if
12:        end for
13:        next[pos] ← next[pos] − 1
14:        if pos ≠ len(next) − 1 then
15:            for j ∈ [pos + 1, len(next)) do
16:                next[j] ← next[pos]
17:            end for
18:        end if
19:        capacity ← Capacity(next)
20:        if l ≤ capacity ≤ l + ϵ then
21:            result ← result ∪ next
22:        else
23:            break
24:        end if
25:    end while
26:    return result
27: end function
```

combination is examined. If the total capacity value is between the boundary lines, then the combination is put into the result list. Otherwise, the traverse process is terminated in line 28.

Algorithm 5 shows the function, TraverseReversely, in which the selected combinations before the given boundary combination are traversed. The inputs of the algorithm are *begin* combination (*l_comb*), *end* combination (*r_comb* or $NULL$), *last* combination (*s_comb* or $NULL$), workload l and ϵ. *last* shows the last combination within the current traverse process. To be noticed, in Algorithm 3, *last* is only updated after $ForwardTransversal$ to avoid the repetitive traverse by the following $BackwardTraversal$. Line 2 to line 4 initialize some parameters, in which *check_list* is used to produce the next

combination within the traverse process. Line 5 to line 25 show the traverse process within the algorithm. The stop criteria condition is to meet the *end* or *last* combination, or to meet one combination that falls outside of the boundary lines. Within the loop, line 6 to line 18, the next combination is constructed. Line 19 to line 24, the provided capacity of the *next* combination is examined. If the total capacity value is between the boundary lines, then the combination is put into the result list. Otherwise, the traverse process is terminated in line 23.



(a) case 1 **(b)** case 2

(c) case 3 **(d)** case 4

Figure 4.3: Frequency Combination Traverse Examples

In Figure 4.3 4 examples are made to show different cases for the traverse process.

Figure 4.3a shows two adjacent *l_comb*s, 211 and 222, in which 222 is treated as *l_comb* for the reason that the capacity range $222 - 300$ is crossed

with the lower boundary line, and 222 is within the range $[l, l+\epsilon]$. To get the *s_comb* starts from 211, function $ForwardTraversal$, Algorithm 4, is applied. The algorithm is terminated since the *end* combination 222 is met.

Figure 4.3b shows two adjacent *l_comb*s, 222 and 310. Same with Figure 4.3a, function, $ForwardTraversal$, Algorithm 4 is applied, and the parameters of *begin* and *end* are 222 and 310 respectively. However, the result of the algorithm is [222] since an undesirable combination, 300, whose capacity is outside of $[l, l+\epsilon]$ is met.

Figure 4.3c shows two adjacent *l_comb* and *r_comb*, 310 and 321. At beginning, the traverse process starts with 310, and, function $ForwardTraversal$, Algorithm 4 is applied. When the algorithm meets 321, the function returns $[310, 311, 320]$. Meanwhile, the *last* combination is set to 320 in Algorithm 3. Afterwards, the traverse process starts with 321 and function $BackwardTraversal$, Algorithm 5, is used since 321 is a *r_comb*. The parameter of Algorithm 5 is $start = 321$, $end = 310$ and $last = 320$. The algorithm is returned with list [321] since the last combination is met. The *last* variable within the traverse process is to avoid the repetitive traverse.

Figure 4.3d shows two adjacent *r_comb*s, 400 and 421. With same reason that 222 is treated as *l_comb*, 400 is treated as *r_comb*. When the traverse algorithm meets 400, it applies function $BackwardTraversal$, Algorithm 5, to get *s_combe* by passing the parameter $start = 400$, $end = 331$ and $last = 331$. The function only returns the result list [400] because the undesirable combination 333 is met. Afterward, when the traverse algorithm meets 421, it applies function $BackwardTraversal$, Algorithm 5, to get *s_combe* by passing the parameter $start = 421$, $end = 400$ and $last = 331$. The function returns $[421, 420, 411, 410]$ when the *end* is met.

4.3.3 Frequency Assignment Phase

In Section 4.3.1, frequency combinations are encoding to reduce the solution space. However, the encode method losses frequency assignment information. In **Frequency Assignment Phase**, the frequencies in each frequency combination are assigned to the nodes, namely the relationship between frequencies and nodes are decided. The objective of **Frequency Assignment Phase** is to generate a frequency assignment which minimizes the total migrated workload and maximizes the workload migrated within racks. The reasons for this objective are: 1) the migration cost cannot be obtained before the whole migration process is done; 2) when a frequency combination is assigned to the nodes, the migrated workload is determined which is related to the migration process and affects the migration cost.

The steps of **Frequency Assignment Phase** is shown as following:

1. Sort the nodes by descending order of their predicted workload, namely by $\forall i \in [1, n]\ w_i$.

2. Assign the frequencies in the combination to the sorted nodes. In the frequency combination, frequencies are sorted by descending frequency value.

3. For two nodes which are in different racks, swap the frequency assignments if they satisfy the conditions below.

   - For both nodes, their capacities are either lager or lower than their workloads.
   - The total migrated workload within racks increases because of the swap.

4. At the end of assignment process, the upper bound of power consumption and the upper bound of migration are obtained according to the model simplification method (Section 3.3).

Using step 1 and step 2, a frequency combination is assigned to the nodes sorted by descending order of the predicted workload under current block layout, by which the total migrated workload is minimized (Proposition 3). Using step 3, the migrated workload within a rack can be maximized (Corollary 1).

Proposition 3. *When a frequency combination is assigned to the nodes that are sorted by descending order of their predicted workload under current block layout, the total size of migrated workload is minimum.*

Proof. Without loss of generality, we say the node set $\mathbf{C}$ is sorted by descending order of the predicted workload under current block layout, namely by w_i. The frequency assigned to c_i is denoted as f_i. According to the assignment method, if $i < j$, then $f_i \geq f_j$ and $w_i \geq w_j$. Furthermore, $z(c_i, f_i) \geq z(c_j, f_j)$. The total migrated workload is denoted as w_{Migrated}. w_{Migrated} can by obtained by Equation 4.22 in which $\mathbf{I}$ is an index set indicating the nodes whose predicted workload exceed their capacities.

$$\begin{aligned} \mathbf{I} &= \{i \mid w_i > z(c_i, f_i)\ i \in [1, n]\} \\ w_{\text{Migrated}} &= \sum_{i \in \mathbf{I}} w_i - z(c_i, f_i) \end{aligned} \tag{4.22}$$

In order to achieve an assignment which gives less total migrated workload, the frequencies are swapped between c_i and c_j. Without loss of generality, we assume $i < j$. Basically there are 4 scenarios.

Scenario 1 $z(c_i, f_i) > w_i$ and $z(c_j, f_j) > w_j$

Scenario 2 $z(c_i, f_i) < w_i$ and $z(c_j, f_j) < w_j$

Scenario 3 $z(c_i, f_i) > w_i$ and $z(c_j, f_j) < w_j$

Scenario 4 $z(c_i, f_i) < w_i$ and $z(c_j, f_j) > w_j$

Scenario 1 When f_i and f_j are swapped, $z(c_j, f_i) > w_j$ can be obtained because $f_i > f_j$, $z(c, f)$ is a monotonic increasing function with f, and $z(c_j, f_j) > w_j$. Since the relationship between $z(c_i, f_j)$ and w_i is unknown, there are 2 sub-scenarios.

Scenario 1.1 $z(c_i, f_j) > w_i$
According to Equation 4.22, the increment of w_{Migrated} under this sub-scenario is 0.

Scenario 1.2 $z(c_i, f_j) < w_i$
According to Equation 4.22, the increment of w_{Migrated} is $w_i - z(c_i, f_j) > 0$.

According to the above analysis, when the swap happens under **Scenario 1**, there is no chance for reducing the total migrated workload.

Scenario 2 When f_i and f_j are swapped, $z(c_i, f_j) < w_i$ is obtained because $f_i > f_j$ and $z(c_i, f_i) < w_i$. Since the relationship between $z(c_j, f_i)$ and w_j is unknown, there are 2 sub scenarios.

Scenario 2.1 $z(c_j, f_i) > w_j$
According to Equation 4.22, the increment of w_{Migrated} under this sub scenarios is $(w_i - z(c_i, f_j)) - (w_i - z(c_i, f_i) + w_j - z(c_j, f_j)) = z(c_j, f_i) - w_j > 0$. The scenario is valid because c_i and c_j have the same capacity under the same frequency, namely c_i and c_j are homogenous.

Scenario 2.2 $z(c_j, f_i) < w_j$
According to Equation 4.22, the increment of w_{Migrated} under this sub scenarios is 0.

According to the above analysis, when the swap happens under **Scenario 2**, there is no chance for reducing the total migrated workload.

Scenario 3 When f_i and f_j are swapped, $z(c_j, f_i) > w_i \geq w_j > z(c_i, f_j)$ is obtained. The increment of the total migrated workload is $(w_i - z(c_i, f_j)) -$

$(w_j - z(c_j, f_j)) = w_i - w_j \geq 0$. when the swap happens under **Scenario 3**, there is no chance for reducing the total migrated workload.

Scenario 4 When f_i and f_j are swapped, $w_i > z(c_j, f_i) > z(c_i, f_j) > w_j$ can be obtained. The increment of the total migrated workload is $(w_i - z(c_i, f_j)) - (w_i - z(c_i, f_i)) = z(c_i, f_i) - z(c_i, f_j) > 0$. when the swap happens under **Scenario 4**, there is no chance for reducing the total migrated workload.

According to the discussion above, when a frequency combination is assigned to the nodes sorted by descending order of the predicted workload, the total migrated workload cannot be minimized by means of swapping 2 assignments. Therefore, the total amount of migrated workload is minimum under this assignment. □

Corollary 1. *The swap cannot reduce the total migrated workload, but it can increase the migrated workload within a rack in some cases.*

Proof. In the proof of **Proposition 3**, two swapping scenarios, **Scenario 1.1** and **Scenario 2.2**, do not increase the migrated workload. However, in some situations, they increase the migrated workload within racks.

We assume c_i and c_j are from different racks. Considering **Scenario 2.2**, the migrated workloads from the nodes before the swapping are $w_i - z(c_i, f_i)$ and $w_j - z(c_j, f_j)$ respectively. After the swapping the migrated workloads from the nodes are $w_i - z(c_i, f_j)$ and $w_j - z(c_j, f_i)$ respectively. According to the conditions of the scenario, the relationships, $w_i \geq w_j > z(c_i, f_i) = z(c_j, f_i) > z(c_j, f_j) = z(c_i, f_j)$ can be obtained. Using the swapping, more workload is migrated from c_i and correspondingly, less workloads are migrated from c_j. If the rack of c_i needs some workload to migrate inside and the rack of c_j needs some workload migrated outside, then the amount of migrated workload within the rack of c_i increases, and correspondingly, the amount of migrated workload between racks of rack c_j decreases. For **Scenario 1.1**, the same case can be constructed as well. Therefore, **Corollary 1** is justified. □

4.3.4 Frequency Filter Phase

In this phase, *s_comb*s are filtered according to the problem's bound constraint. By means of **Frequency Filter Phase**, all the frequency candidates are valid in terms of the bound constraint.

$\mathbb{P}^b\mathbb{MC}^{min}$ defines the problem that minimizes the migration cost fulfilling the basic requirement of the power consumption bound. In **Frequency Filter Phase**, *s_comb*s are filtered according to $\mathbb{P}^b$. For each frequency combination, if $p^{max} \leq \mathbb{P}^b$, then the combination is kept otherwise the combination

is discarded. Due to the filter process, all the kept combinations satisfy $\mathbb{P}^b$ constraint.

$\mathbb{MC}^b\mathbb{P}^{min}$ defines the problem that minimizes the achievable power consumption within the migration cost bound. In order to solve the problem, *s_comb*s are filtered by means of $\mathbb{MC}^b$ in **Frequency Filter** sub-process. For each solution, if $mc^{max} \leq \mathbb{MC}^b$, then the solution is kept otherwise the solution is discarded. Due to the filter process, all the kept solutions can satisfy $\mathbb{MC}^b$ constraint.

4.3.5 Frequency Search Phase

After the **Frequency Filter Phase**, a **Frequency Search** mechanism is introduced. In **Frequency Filter**, all *s_comb*s are ensured to satisfy the bounded requirement. However, the amount of solutions might be still too large. Therefore, the solution space is narrowed down further by **Frequency Search Phase**. The purpose of **Frequency Search Phase** is to find a few promising solution candidates. In **Frequency Search Phase**, a top-k algorithm [86] is used to obtained κ solutions according to problem's objective. In $\mathbb{P}^b\mathbb{MC}^{min}$ problem, the top κ combinations with minimized migration cost upper bound value are searched. Correspondingly, the top κ combinations with minimized power consumption upper bound value are obtained for $\mathbb{MC}^b\mathbb{P}^{min}$.

4.3.6 Frequency Evaluation Phase

In this phase, the κ frequency candidates generated in **Frequency Search Phase** are evaluated, and the solution that meets all the constraints and minimizes the objective is obtained. In **Frequency Search Phase**, the solution space is reduced to κ solution candidates. Therefore, in **Frequency Evaluation Phase**, the migration process (Chapter 6) is applied to these κ solution candidates. In the process, a migration plan is generated according to each given frequency assignment and workload prediction. Meanwhile, the estimated power consumption and the estimated migration cost are obtained according to the migration result by Equation 4.7 and Equation 3.7. The final solution is obtained according to the objective target. For $\mathbb{P}^b\mathbb{MC}^{min}$, the solution with the minimal migration cost is chosen, and for $\mathbb{MC}^b\mathbb{P}^{min}$, the solution with the minimal power consumption is chosen. By means of evaluation phase, the best feasible solution is obtained according to problem's constraints and objective.

4.3.7 Summary and Complexity Analysis

In previous sections, the details of the Multi-Phases Algorithm are introduced. In this section, the Multi-Phases Algorithm is introduced in the view of bounded problems, and the complexity of the algorithm is analyzed.

The feasible solution can be found by Multi-Phases Algorithm, which consists of 6 phases: **Frequency Locating Phase**, **Frequency Traversal Phase**, **Frequency Assignment Phase**, **Frequency Filter Phase**, **Frequency Search Phase**, and **Frequency Evaluation Phase**. The key of Multi-Phases Algorithm is to reduce the solution space and to optimize the solution according to the problem type with different aspects in each phase.

In **Frequency Locating**, the frequency combinations are encoded by the encoding constraints which reduce the solution space significantly. Then, the solution space is reduced again in **Frequency Traversal** by boundary lines, which indicate the maximal total provided capacity and the minimal total provided capacity. The frequency combinations whose total capacities are between the boundary lines are taken into consideration in the following process. In **Frequency Filter**, the solution space is narrowed down further by omitting the solution candidates that are not valid in terms of bound constraint ($\mathbb{P}^b$ or $\mathbb{MC}^b$). At the end, only κ solution candidates are kept by **Frequency Search Phase** according to the objective of the problem.

In terms of the optimization, Multi-Phases Algorithm takes the combinations whose total capacities are between l and $l+\epsilon$ into consideration to avoid under provisioning and over provisioning in **Frequency Locating Phase** and **Frequency Traversal Phase**. In **Frequency Assignment Phase**,the assignment mechanism is in favor of reducing migrated workloads, which leads to lower migration cost. In **Frequency Filter Phase**, the combinations are filtered according to the bound constraint of the bounded problem, which makes sure all the solution candidates are valid in point view of the bound constraint. Afterwards, only κ candidate solutions are obtained within **Frequency Search Phase**. **Frequency Search Phase** utilizes top-k algorithm to get the solution candidates with lower upper bound value of its objective, which is in favor of obtaining a lower objective value of the final solution. At the end, the solution is obtained based on the migration process.

The complexities of each phase are listed in Table 4.3. In the table, U is the number of racks, n represents the number of nodes, and m represents the number of frequency options. δ represents the number of frequency combinations which are generated in frequency selection phase. The number of blocks for each node is denoted as $|\mathbf{D}_i^r|$ in which $\mathbf{D}_i^r = \{b_{g,k} \mid b_{g,k} \text{ is assigned to } c_i\}$.

- Phase 1: In **Frequency Locating Phase** the operation of locating combination checks the frequency option for each node. Afterward, the

Table 4.3: Complexity of Operations

Phase	Operation	Complexity
1	Frequency Locating	$O(nm^2)$
2	Frequency Traversal	$O(n\delta)$
3	Frequency Assignment	$O(n^2\delta)$
4	Frequency Filter	$O(\delta \max_{i=1}^{n} (\lvert\mathbf{D}_i^r\rvert \log \lvert\mathbf{D}_i^r\rvert))$
5	Frequency Search	$O(\kappa \ln \widetilde{\delta})$
6	Frequency Evaluation	$O(\lvert\sum_{i=1}^{n} \mathbf{D}_i^r\rvert^3)$

ranges are narrowed down from each digital (encoded positions within frequency combination), therefore the complexity is $O(nm^2)$.

- Phase 2: In **Frequency Traversal Phase**, each selected frequency combination is checked to make sure it fits to the boundary. However, when generating next frequency combination, each digit within the combination is rechecked to make sure that the combination is valid according to the encoded method. Therefore the complexity for traversing combinations is $O(n\delta)$, in which δ is the number of generated frequency combinations.

- Phase 3: In **Frequency Assignment Phase**, frequencies in each combination are exchanged between nodes to generate the assignment which is in favor of minimizing migrated workload. The complexity of assigning frequency is $O(n^2)$.

- Phase 4: In **Frequency Filter Phase**, frequency combinations are filtered according to the upper bound values and the problem constraint ($\mathbb{P}^b$ or $\mathbb{MC}^b$). However, in order to obtain the upper bound migration cost, the migrated block sets are required (Section 3.3). The detail of the block selection is discussed in Section 6.2. The complexity of block selection operation for a given frequency combination is $O\left(\max_{i=1}^{n} (\lvert\mathbf{D}_i^r\rvert log \lvert\mathbf{D}_i^r\rvert)\right)$, therefore The complexity of the **Frequency Filter Phase** is $O\left(\delta \max_{i=1}^{n} (\lvert\mathbf{D}_i^r\rvert log \lvert\mathbf{D}_i^r\rvert)\right)$.

- Phase 5: In **Frequency Search Phase**, κ candidate solutions are searched from filtered combinations by top-k algorithm. The number of filtered combinations is denoted as $\widetilde{\delta}$, therefore the complexity of the phases is $O(\kappa \log \widetilde{\delta})$

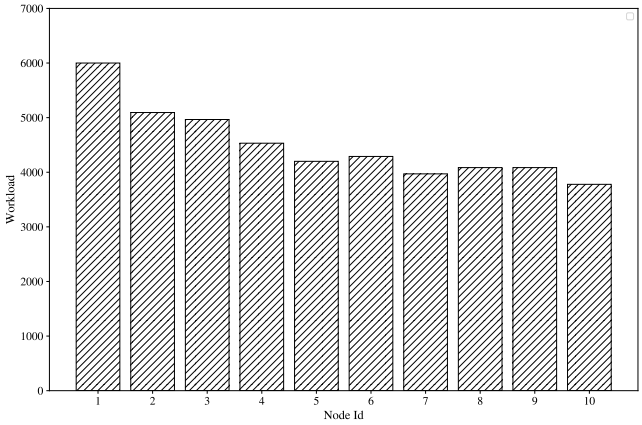


Figure 4.4: Example of the Workload

- Phase 6: In **Frequency Evaluation Phase**, the migration plan is obtained to compute the estimated power consumption and estimated migration cost. At the end, the solution candidate with the minimal objective value is taken as the final solution. In block migration operation, the algorithm CMTHM is adopted (Section 6.3). The complexity of CMTHM depends on the amount of items because of exchange process. Therefore, the complexity of CMTHM algorithm is $O(|\sum_{i=1}^{n} \mathbf{D}_i^r|^3)$

4.4 Experiment

In this section, proposed algorithms— Nonlinear Programming Algorithm and Multi-Phases Algorithm — are evaluated using simulation experiment. Nonlinear Programming Algorithm can find the optimal solution for both frequency selection and migration process, however it cannot be applied to large scale problems. In contrast, Multi-Phases Algorithm can find the sub-optimal solutions and has good scalability. In this section, a comparison experiment is made between Nonlinear Programming Algorithm, Multi-Phases Algorithm and HHHC approach from Houssem-Eddine Chihoub *et al.* [19]. Then, a scalability experiment is implemented for Multi-Phases Algorithm.

4.4.1 Setup

To conduct the simulation experiments, some simulation test cases are generated. The method for generating test cases are shown as below.

- **System**. A system is denoted as $d\{NodeAmount\}$. Meanwhile, it is assumed that there are 2 racks in total. For example $d20$ represents a

database system with 20 nodes, and there are 2 racks in which each rack contains 10 nodes.

- **Block and Block layout**. The number of blocks is set to 64 per node in the experiments and the replica factor is set to 3. Therefore in $d10$, there are 1920 blocks in total. The size of blocks is uniformly generated within 10Mb to 30Mb. For the replicas, they have same block size. To be noticed this method is used to give the number of blocks. The blocks are distributed by following rules: 1) the first replica is placed on one of the nodes by means of round-robin strategy; 2) the second replica is placed on another rack; 3) the third replica is placed on the same rack of the second, but on a different node chosen at random.

- **Block accessed probability**. The block accessed probability in Frequency Selection model refers to the parameter $\varphi_{g,k}$. In practice, these are predicted values. In the experiment, probability is generated by Zipf distribution [87] and the distribution factor is set to 2.5 in our experiment.

- **Workload**. The workload in this paper refers to the parameter l which is the predicted throughput. The value of l is given by the throughput per node. For example, if we say the workload of one test case is set to 3500 Ops/Sec to $d10$, then for the whole system, the throughput is set to 35000 Ops/Sec. However, this method is a way to set the throughput value for the whole system, the throughput for each node depends on the assigned blocks and their accessed probabilities. An example of a case with 10 nodes can be find in Figure 4.4, in which the bars show the workload in Ops/Sec of each node.

- **Bound**. There are two problem types in this chapter, $\mathbb{P}^b\mathbb{MC}^{min}$ and $\mathbb{MC}^b\mathbb{P}^{min}$. For both problems, the boundary value of $\mathbb{P}^b$ and $\mathbb{MC}^b$ should be provided. In the experiments, if not specified, then the boundary value is obtained as follow. To obtain the boundary value $\mathbb{P}^b$, the corresponding test case is solved by $\mathbb{MC}^b\mathbb{P}^{min}$ in which $\mathbb{MC}^b$ is set to infinite denoted as $\mathbb{MC}^{Inf}$. When the value of $\mathbb{P}^{min}$ is obtained, the boundary value is set to $1.1 \times \mathbb{P}^{min}$. The boundary value $\mathbb{MC}^b$ is obtained by the same way.

A test case is a combination with a system(d), a workload (l) and a problem type ($\mathbb{P}^b\mathbb{MC}^{min}$ or $\mathbb{MC}^b\mathbb{P}^{min}$). For example $(d10, 3500, \mathbb{P}^b\mathbb{MC}^{min})$ presents a test case in which there are 10 nodes (5 for each rack), the workload throughput is set to $3500 \times 10 = 35000$ Ops/Sec. In $(d10, 3500, \mathbb{P}^b\mathbb{MC}^{min})$, a power consumption boundary value is provided and the objective is to minimize migration cost.

Table 4.4: Properties of the Test Cases for Nonlinear Programming Algorithm

Property	Description
System	The system includes 6 nodes denoted as $d6$.
Replica	2
Block Amount	3 per node(36 blocks in total)

4.4.2 Comparison Experiment

Nonlinear Programming Algorithms vs Multi-Phases Algorithm

Since the Nonlinear Programming Algorithm cannot be applied to a large scale problem, two special test cases, denoted as $(d6, 4000, \mathbb{P}^b\mathbb{MC}^{min})$ and $(d6, 4000, \mathbb{MC}^b\mathbb{P}^{min})$, are introduced. The detail about the properties of these test cases are listed in Table 4.4. Compared with the test case $(d20, 4000, \mathbb{P}^b\mathbb{MC}^{min})$, $(d6, 4000, \mathbb{P}^b\mathbb{MC}^{min})$ and $(d6, 4000, \mathbb{MC}^b\mathbb{P}^{min})$ are smaller in terms of the number of nodes and the number of blocks. However, in our experiment, these are the biggest cases for the Nonlinear Programming Algorithm, otherwise the algorithm cannot produce a solution within a few minutes. The comparison result between the Nonlinear Programming Algorithm and the Multi-Phases Algorithm is shown in Figure 4.5 and Figure 4.6.

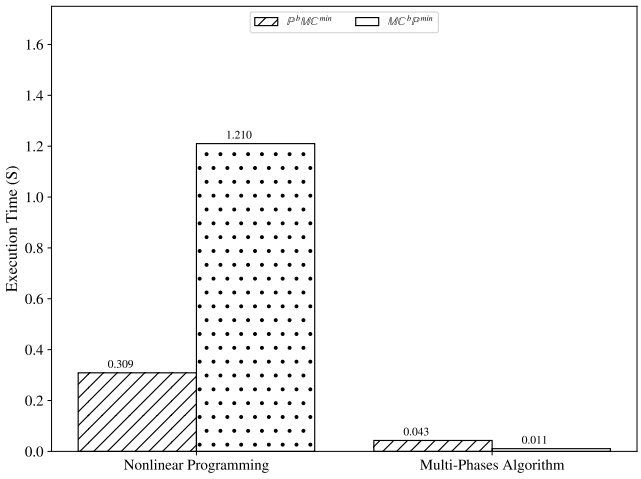


Figure 4.5: Execution Time Comparison

In Figure 4.5, the execution time of Multi-Phases Algorithm is lower than the corresponding execution time of Nonlinear Programming Algorithm for both test cases. In term of execution time, Multi-Phases Algorithm is faster than Nonlinear Programming Algorithm up to 7 to 10 times under our test cases. Figure 4.6 shows the objective value given by both algorithm for the

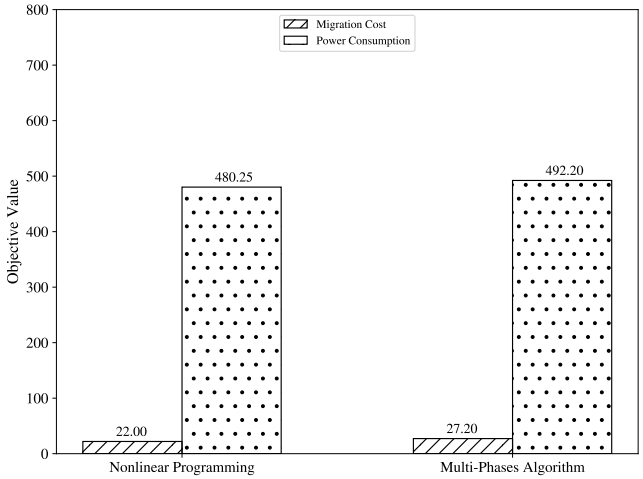


Figure 4.6: Objective Value Comparison

test cases. Generally, the objective values given by Nonlinear Programming Algorithm is lower than the corresponding objective values given by Multi-Phases Algorithm. For $\mathbb{P}^b\mathbb{MC}^{min}$ migration cost produced by Multi-Phases Algorithm is 1.23 times larger than the value obtained by Nonlinear Programming Algorithm. While for $\mathbb{MC}^b\mathbb{P}^{min}$, the power consumption value achieved by Multi-Phases Algorithm is 1.03 times larger than the value obtained by Nonlinear Programming Algorithm. The reason of this situation is that Multi-Phases Algorithm takes advantages of approximation algorithms in each phase to obtain a feasible solution of the problem. In contrast, Nonlinear Programming Algorithm uses nonlinear optimization method to solve the problem and obtain the global optimal solution.

Multi-Phases Algorithm vs Half Hot and Half Cold Approach

In next comparison experiment, the Multi-Phases Algorithm is compared with a method from Houssem-Eddine Chihoub *et al.* [19]. The method is defined as HHHC(Half Hot and Half Cold) in our experiment. In HHHC, half of the nodes are set to the highest frequency(2.53GHz) and another half of the nodes are set to the lowest frequency (1.20GHz). In Multi-Phases Algorithm, the frequencies are selected according to the predicted workload.

In this experiment, 4 test cases are involved, which are denoted as $(d20, l, \mathbb{T}^{Inf}\mathbb{P}^{min})$ $l \in [5000, 4500, 4000, 3500]$. Since there is no boundary for migration cost, minimize power consumption is the objective for both algorithms. All the cases are solved by Multi-Phases Algorithm and HHHC and the result is shown by Figure 4.7.

In Figure 4.7, The power consumption given by multi-phases algorithm are

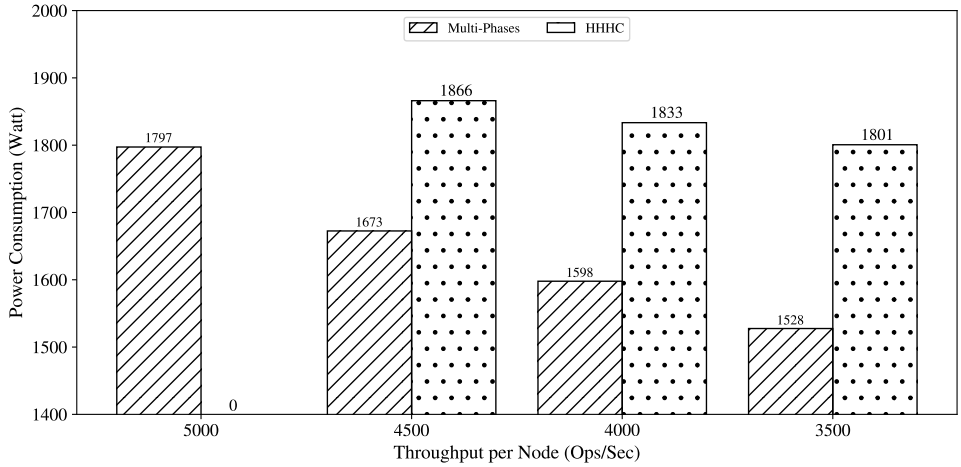


Figure 4.7: Result of Frequency Selection Comparison

lower than the corresponding result given by HHHC. The average improvement of Multi-Phases Algorithm compared with HHHC is 12.89%. When l is set to 5000 Ops/Sec, HHHC cannot produce a valid solution. Theoretically, when HHHC is applied, the system with 20 nodes can support any workloads with throughput under $10 \times 5690 + 10 \times 3520 = 92080$ Ops/Sec, however with the setting 5000 Ops/Sec for each node, the system does not have enough resources to support it. Therefore the corresponding power consumption is recorded as 0. The main drawback of HHHC is its flexibility. HHHC sets the frequencies statically, while the Multi-Phases Algorithm chooses frequencies according to the workload predictions.

4.4.3 Scalability Experiment

A series of tests is made to evaluate the scalability of the Multi-Phases Algorithm. In this experiment, test cases $(d\{NodeAmount\}, 4000, \mathbb{P}^b\mathbb{MC}^{min})$ and $(d\{NodeAmount\}, 4000, \mathbb{P}^b\mathbb{MC}^{min})$ are used, in which $NodeAmount \in [10, 20, 30, 40, 50, 60, 70]$. The execution time for each case is shown by Figure (4.8) in logarithmic scale.

In Figure (4.8), the execution time of the Multi-Phases Algorithm increases with the increase of the number of nodes for both problem type. When the number of nodes increases, the total frequency combinations increases as well. Therefore, the number of frequency combinations which are evaluated in the Multi-Phases Algorithm increases. The amount of migrated blocks increases as well since the workload increases along with the node amount. Then, the execution time for both frequency selection and workload migration increases. As a consequence, the execution time of the Multi-Phases Algorithm increases.

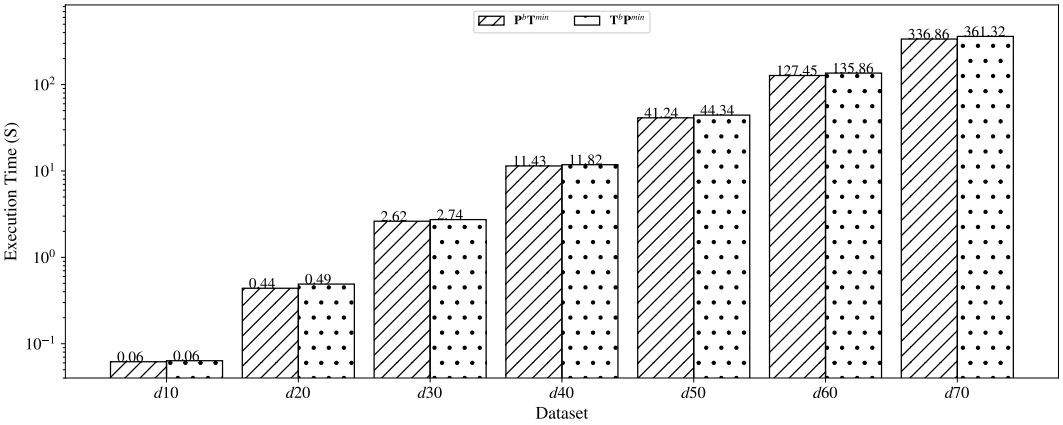


Figure 4.8: Result of Scalability Experiment

With the same node amount, the execution time for $\mathbb{P}^b\mathbb{MC}^{min}$ is shorter than the execution time for $\mathbb{MC}^b\mathbb{P}^{min}$ especially for larger problems (i.e $d60$ and $d70$). For problem $\mathbb{P}^b\mathbb{MC}^{min}$, the Multi-Phases Algorithm filters frequency combinations after the frequency assignment phase by p^{max}, and sort frequency combinations after the block selection phase by mc^{max}. However, for problem $\mathbb{MC}^b\mathbb{P}^{min}$, the Multi-Phases Algorithm sorts frequency combinations after the frequency assignment phase by p^{max}, and filters frequency combinations after the block selection phase by mc^{max}. When the same number of frequency combinations generated in the frequency selection phase, the execution time for $\mathbb{P}^b\mathbb{MC}^{min}$ is shorter because less frequency combinations are evaluated in block selection phase. When a frequency combination is filtered, then the combination is discarded. In $\mathbb{P}^b\mathbb{MC}^{min}$, the filter operation happens before the block selection phase which causes less execution time.

4.5 Summary

In this chapter, the frequency selection problem is considered as two bounded problems: $\mathbb{P}^b\mathbb{MC}^{min}$ and $\mathbb{MC}^b\mathbb{P}^{min}$. Nonlinear Programming Algorithm and Multi-Phases Algorithm are proposed to solve them. Nonlinear Programming Algorithm can obtain a global optimal solution but has a poor scalability to large scale problems. In contrast, Multi-Phases Algorithm has a good scalability for large scale problems by means of approximation method. According to the experiment results up to 21.5% power can be saved.

Chapter 5

Frequency Selection with Optimization Problem

In previous chapter, we consider power consumption and migration cost are independent variables and solve the frequency selection problem under two bounded problems. In this chapter, both of them are considered as a whole in the optimization problem. We propose two approximation algorithms, genetic based algorithm and monte carlo tree based algorithm, to obtain the feasible solution for the optimization problem. We introduce the objective of the problem first, and then two algorithms are proposed. In experiment section, we evaluate the algorithms in aspect of performance, accuracy and scalability. In the end, the pros and cons for both algorithms are discussed.

5.1 Objective

Energy consumption of the system is obtained by Equation 3.2, in which energy consumption consist of two parts energy consumption used for executing workload, $p(s^*) \times |\Delta t|$, and energy consumption used for conducting migration, $mc(\widetilde{s})$. In optimization problem, we consider both parts as a whole. Equation 5.1 shows the maximal energy consumption required by the system to execute the workload and conduct migration for Δt. The objective of the optimization problem is to minimize $\mathbb{E}$.

$$p(s^*) \times |\Delta t| + mc(\widetilde{s}) \leq \mathbb{E} \tag{5.1}$$

5.2 Genetic Algorithm

Genetic **A**lgorithm (GA for short) is a type of algorithms for randomly searching suboptimal solutions, which is guided by evaluation and natural genetics [88]. Generally, GA includes several phases in each iteration : 1, encoding; 2, generation of initial population; 3, evaluation; 4 selection; 5 crossover; 6 mutation and 7 stopping criteria. In this section, the essential parts of GA—encoding and evaluation—are introduced. The influence of parameters of GA is discussed in Section 5.4.1

The objective of frequency selection is to generate the frequency vector $\mathbf{F}$ for the cloud system, which minimizes energy consumption within each time window. An encoded frequency vector $\mathbf{F}$ is regarded as a chromosome. In the encoding process, frequency $f_i \in \mathbf{F}$ is replaced by its index within the frequency option set. Let the frequency option set be η and function $I_\eta(f_i)$ gives the corresponding index of the frequency option f_i. The chromosome is represented by $< I_\eta(f_1), I_\eta(f_2), ..., I_\eta(f_n) >$.

In the evaluation phase, a fitness function is required to qualify chromosomes. The power consumption and the migration cost are estimated according to chromosomes. However, as discussed above, the fitness function may become a bottleneck and the model simplification approach is applied. The pseudocode of the fitness function is shown in Algorithm 6.

Algorithm 6 Fitness Function of Genetic Algorithm

Require: *chromosome* encoded frequencies
Ensure: *score* fitness value

1: **function** EVALUATE(*chromosome*)
2: $\quad\mathbf{F} \leftarrow$ DECODE(*chromosome*)
3: $\quad p^{max} \leftarrow$ MAXPOWERCONSUMPTION($\mathbf{F}$)
4: $\quad mc^{max} \leftarrow 0$
5: $\quad$**for** $u \leftarrow [1, ..., U]$ **do**
6: $\quad\quad \mathbf{M}_p^{\text{In}}, \mathbf{M}_p^{\text{Out}} \leftarrow$ SELECTMIGRATIONBLOCKS(γ_u)
7: $\quad\quad mc^{max} \leftarrow mc^{max} +$ MIGRATIONCOST($\mathbf{M}_u^{\text{In}}, \mathbf{M}_u^{\text{Out}}$)
8: $\quad$**end for**
9: $\quad$**return** $p^{max} \times |\Delta t| + mc^{max}$
10: **end function**

The function gives a score for each *chromosome*. Firstly, the *chromosome* is decoded (line 2). Secondly, the maximum power consumption is obtained by Equation 3.8 (line 3). Thirdly, the maximum migration cost is calculated within the loop (line 4 to line 8). There are U racks, and the migration blocks are selected for each rack γ_u (line 6), and the migration cost for each rack

is obtained by Equation 3.9. Finally, the maximum energy consumption is returned as the score.

Since the power consumption and the migration cost are replaced by their upper bound values, the solution of GA may not be optimal. In order to improve the accuracy of the algorithm, GA is applied multiple times to generate several candidates. Afterwards, the migration process is applied to all candidates and the solution with the minimum energy consumption is chosen.

5.3 Monte Carlo Tree Search Algorithm

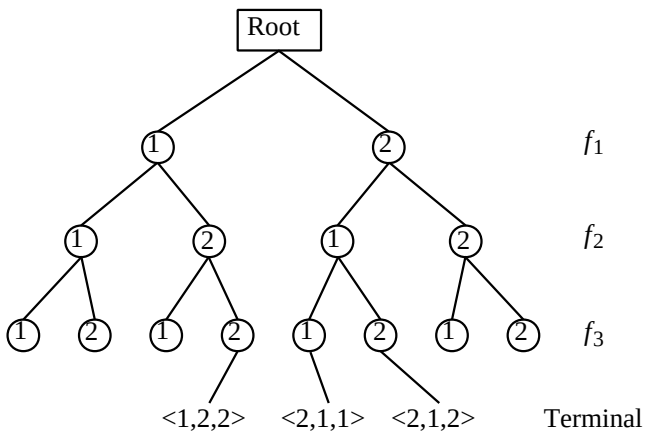


Figure 5.1: One iteration of the general MCTS approach

Monte Carlo Tree Search Algorithm (MCTS) is a method for finding the suboptimal decision in a given domain by taking random samples in the decision space and building a search tree according to the results. Over the last few years, MCTS has achieved great success with many games, complex real-world planning, optimization and control problems [89].

MCTS is based on Monte-Carlo process model. The model consists of a set of states, a set of actions, a transition model, and a reward function. The decision is presented as a pair of a state and an action, and the next state is chosen by a probability distribution built up by the current state and available actions. The link between state and actions is defined as *policy* and the aim is to find the special $policy^*$ generating highest reward.

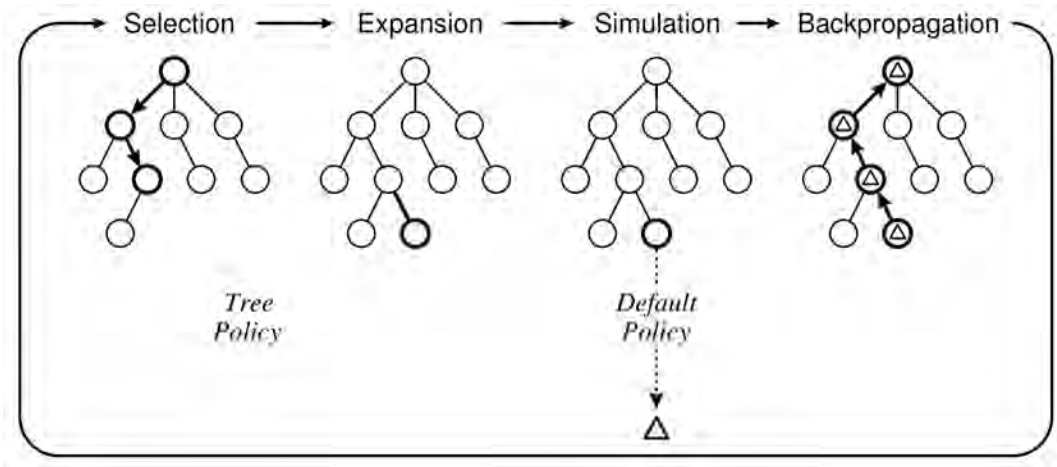


Figure 5.2: Frequency Selection Tree Example

Under the frequency selection approach, the set of states are the frequency options. The action refers to choosing a frequency option for the next node. Figure 5.1 shows a structure of the frequency selection tree under a small case with 3 nodes and 2 available frequency options. In each layer, the frequency for the node is chosen. At beginning, f_1 has two options. When f_1 is set to frequency 1, there are 2 actions: set frequency 1 to f_2 and set frequency 2 to f_2. Therefore there are 4 states in second layer. Since the frequency option for the next node is not related to the current state, the frequency selection tree is a complete $|\eta|$-ary tree. When the searching process arrives at leaf nodes, the terminal condition is reached. A path of the tree is denoted as a frequency vector. For example, in Figure 5.1, $< 1, 2, 2 >$ is one of the frequency vectors. The task of MCTS is to find the frequency vector which produces the minimum energy consumption.

Figure 5.2 from the survey [89] explains the general process in MCTS including 4 phases in each iteration: selection, expansion, simulation, and back propagation.

1. Selection: Starting at the root node, a child selection policy is recursively applied to descend through the tree until the most urgent expandable node is reached. When a node is not fully expanded, then the most urgent expandable child is built according to the untiled actions. Otherwise, a best child is selected (tree policy). A node is expandable if it represents a non-terminal state and has unvisited (i.e. unexpanded) children.

2. Expansion: One (or more) child nodes are added to expand the tree, according to the available actions.

3. Simulation: A simulation (default policy) is run from the new node(s) according to the default policy to produce an outcome.

4. Back-propagation: The simulation result is backed up through the selected nodes to update their statistics.

Basically, the process is controlled by two functions, a tree policy and a default policy. A node on the search tree is denoted as v and a child node of v is denoted as v'. Let function N show how many times the node has been visited. Let function Q give the score of v. The tree policy chooses the best child with the maximum Upper Confidence Bounds value (UCT for short). The UCT function is shown by Equation 5.2, in which C is a constant factor. In UCT function, the exploitation (visiting the expanded nodes) and the exploration (visiting the unexpended nodes) are balanced. If a node is not visited before, the tree policy chooses a node randomly. In the default policy, one of the paths is evaluated by a reward function R. Based on all nodes on the path, an evaluation score is required and the score is back propagated to all nodes on the path to update its scores, Q.

$$UCT = \frac{Q(v')}{N(v')} + C\sqrt{\frac{2\ln N(v)}{N(v')}} \tag{5.2}$$

The base idea of the default policy in this work is the same with the fitness function of GA that evaluates a current solution and gives a score. Therefore, Algorithm 6 is used as the reward function of the default policy. However, the input parameter *chromosome* is replaced by the paths of the tree. In the tree policy, a dedicated UCT function, shown in Equation 5.3, is adopted. The difference between Equation 5.2 and 5.3 is the method for calculating scores. In MCTS, the value of the node's score is between 0 and 1, and the node with highest UCT value is selected. In the default policy of this work, the energy consumption is returned as score of a node. By means of $1 - (Q(v')/e^{max})$, the value is converted within 0 and 1. The highest value of $1 - (Q(v')/e^{max})$ indicates the path with the minimum energy consumption.

$$UCT = \frac{1 - (Q(v')/e^{max})}{N(v')} + C\sqrt{\frac{2\ln N(v)}{N(v')}} \tag{5.3}$$

In Equation 5.3, the value of e^{max} is required. Equation 3.2 shows that the energy consumption consists of two parts, namely the energy consumption of the running system and the energy consumption of workload migration. In most of the cases, the energy consumption of the running system is the dominant part. In this case, the maximum energy consumption scenario is

that each node is assigned with its highest frequency. Using Equation 3.5 and Equation 3.7, the maximum energy consumption e^{max} can be obtained. In contrast, when migration cost is the dominant part, the frequency vector for achieving e^{max} cannot be constructed directly. In this case, GA is applied to find e^{max}, which makes it meaningless to apply MCTS since the frequency vector for achieving e^{min} can be found by GA also. Therefore, MCTS may be inapplicable for the cases in which the energy consumption of the workload migration is the dominant part. Like the GA approach, it should be noticed that the final solution of MCTS may not be optimal. MCTS is applied multiple times to obtain several solutions, and the solution with the minimum energy consumption is chosen.

5.4 Experiment

In this section the experiment consists of 4 parts: parameter influence, scalability analysis, optimization bound analysis, and comparison experiment to evaluate GA and MCTS in aspects of performance and accuracy. In the experiment, we follow the same test case definitions from Chapter 4 (Section 4.4.1). A test case is a combination of a system(d), a workload(l) and an algorithm(a). For example $(d10, 5000, GA)$ indicates a test case, in which GA is applied to system $d10$ and the workload is set to 5000 Ops/Sec for each node. The value of throughput per node is a standard to simulate the total workload for the cases, and the throughput for each node is decided by the block accessed probability φ_{gk}. The value of throughput per node is set to 5000 Ops/Sec by default if not otherwise specified, for example case $(d10, 5000, GA)$ is denoted as $(d10, GA)$

5.4.1 Parameters' Influence

In this section, the influence of parameters on the algorithms is examined. As the number of blocks on each node does not impact the performance of the frequency selection algorithm, the number of blocks is set to 64 for each node, and the access probabilities are generated by Zipf's law (distribution factor is set to 2.5).

In order to evaluate the accuracy of the algorithms, the solutions for the cases $(d10, Optimal)$, $(d20, Optimal)$ and $(d30, Optimal)$ are obtained, in which *Optimal* indicates the complete search where all possible frequency vectors are evaluated. The energy consumption for a case is denoted as $E(case)$ and the corresponding execution time is denoted as $T(case)$. The accuracy of a case $A(case)$ is defined by Equation (5.4) in which $d \in [d10, d20, d30]$ and $a \in [GA, MCTS]$.

$$A(d,a) = 1 - \frac{(E(d,a) - E(d,Optimal))}{E(d,Optimal)} \tag{5.4}$$

The influence of generation size on GA

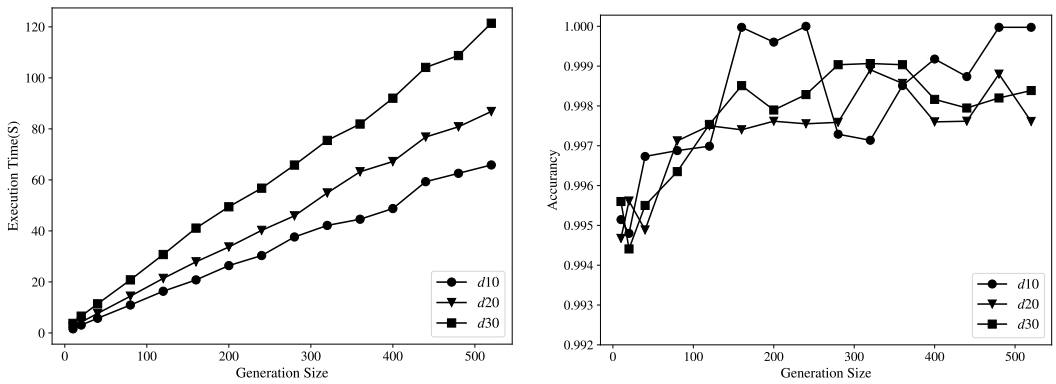


Figure 5.3: The Influence of Generation Size on Accuracy

The result is shown in Figure 5.3. In each case, the population size is set to 100 and the amount of candidates is set to 10. In Figure 5.3, $\forall i \in [10, 20, 30]$, $T(di, GA)$ increases with the increment of generation size for the reason that more generations lead to more iterations. With the same population size, $\forall i > j$ $T(di, GA) > T(dj, GA)$. More nodes lead to longer chromosome in GA, because the length of a chromosome is the number of nodes. In terms of accuracy, the range of $A(d, GA)$ is $[0.994, 0.999]$. As shown in Figure 5.3, $A(di, GA)$ increases at beginning with the increment of generation size. However, when generation size exceeds some points (100 for $d10$, $d20$ and 150 for $d30$), $A(di, GA)$ does not increase significantly and sometimes $A(di, GA)$ even decreases a little. More generations lead to more iterations of GA. At beginning, it leads to more evolutions, which improves $A(di, GA)$. However afterwards the search process is close enough to an optimal point and the iterations keep the solutions around the optimal point.

The influence of population size on GA

The result is shown in Figure 5.4. In each case, generation size is set to 150 and the amount of candidates is set to 10. In Figure 5.4, with the same population size, $\forall i > j$, $T(di, GA) > T(dj, GA)$, because more chromosomes are evaluated in one iteration. Increasing population size improves $A(d, GA)$ at the beginning. However, at some points (80 for $d10$, 160 for $d20$ and 320

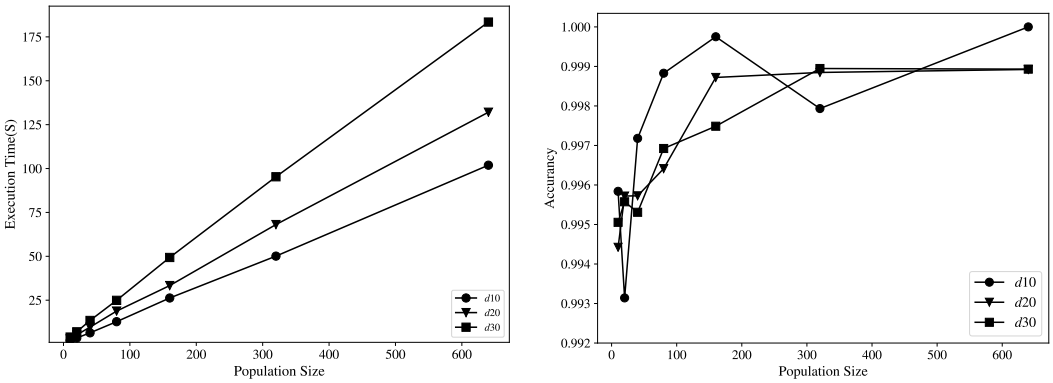


Figure 5.4: The Influence of Population Size on Accuracy

for $d30$), the increment of population size does not improve the accuracy any more.

The influence of number of candidates

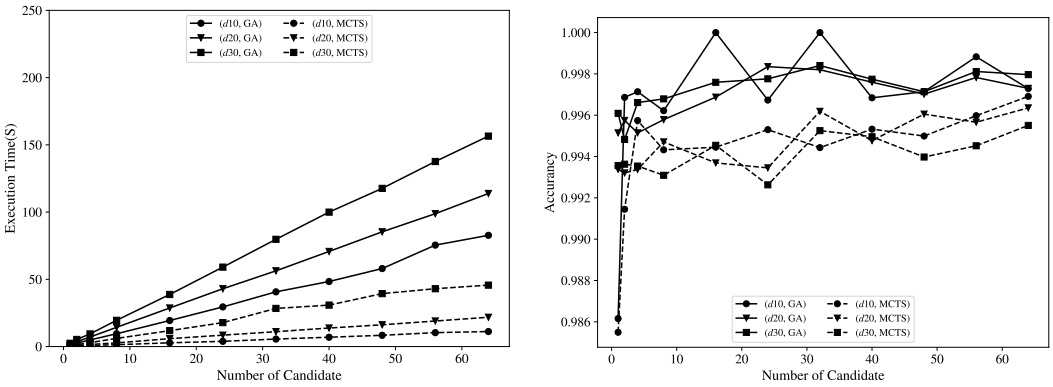


Figure 5.5: The Influence of Amount of Candidates on Accuracy

GA and MCTS cannot find the optimal solution because of the model simplification approach. Therefore, both algorithms are executed multiple times to find several candidates, and the solution with the minimum energy consumption is chosen. For GA, generation size is set to 150 and population size is set to 100. The result is shown in Figure 5.5. In Figure 5.5, $T(d, GA)$ and $T(d, MCTS)$ increase with the amount of candidates. With the same amount of candidates, $T(d, GA) > T(d, MCTS)$. The reason is that GA is based on the evolutions while MCTS is based on the tree searching technique. The computation cost is higher for GA (see Section 5.4.2). In term of accuracy, the increment of the amount of candidates improves $A(d10, GA)$

and $A(d10, MCTS)$ significantly at beginning. $A(d10, GA)$ goes up and down when more candidates are involved because in some cases, a close to optimal solution is found occasionally. In other cases, the accuracy of both algorithms increases slightly in general when more candidates are involved. Generally, $A(d, GA) > A(d, MCTS)$. In MCTS, the search space is organized by a tree structure. The leaf nodes are not ordered and there is no tendency among all the solutions. Considering Figure 5.1 and the maximum power consumption of terminal nodes, we have $p^{max}(< 1, 2, 2 >) > p^{max}(< 2, 1, 1 >)$ and $p^{max}(< 1, 2, 2 >) = p^{max}(< 2, 1, 2 >)$. Therefore, the random sampling method doesn't perform well in this situation, which impacts negatively the overall accuracy. The maximum accuracy of MCTS is 99.6%.

5.4.2 Scalability Analysis

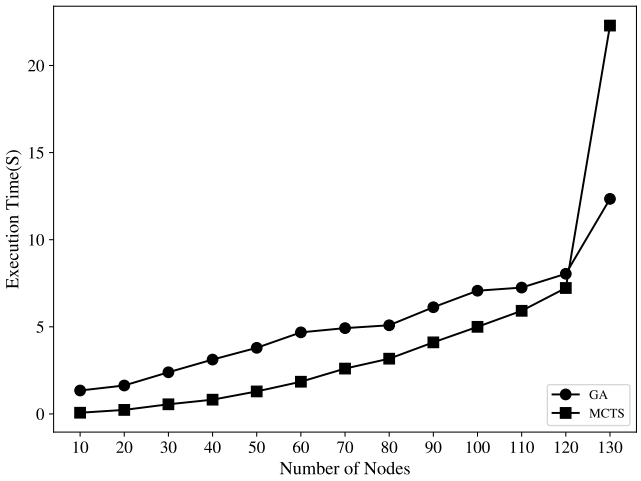


Figure 5.6: Scalability of the Frequency Selection Algorithm

In this section, there are 12 systems ($d10$ to $d130$) involved. Only one candidate is required in each case. For GA, generation size is set to 150 and population size is set to 100. The result is shown in Figure 5.6. Generally, $T(d, GA) > T(d, MCTS)$. When the system is smaller, the difference is more dramatic. For example, $T(d10, GA)$ is nearly 19 times $T(d10, MCTS)$. However, the growth rate of the execution time of GA is lower than MCTS. For example, $T(d130, GA)$ is 9 times $T(d10, GA)$ while $T(d130, MCTS)$ is 327 times $T(d10, MCTS)$. In GA, the increasing amount of nodes leads to the longer length of chromosome. When generation size and population size are constant, the length of chromosome only influences the performance in each evaluation. It leads to linear increment. However, in MCTS, when the amount

of node is increased by 1, the height of the tree is increased by 1 which leads to exponential growth of the leaf nodes. The search of solutions is an exponential function, which makes the execution time grows exponentially.

5.4.3 Optimization Bound Analysis

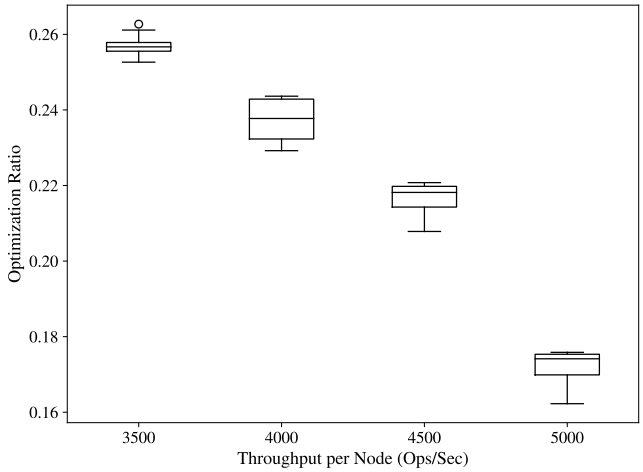


Figure 5.7: Optimization Bound of the Frequency Selection Algorithm

In this section, optimization ratio is introduced to evaluate how much energy can be saved using the frequency selection approach. The optimization ratio is defined by Equation 5.5 in which f^{max} refers to the highest frequency, 2.53Ghz in our environment. The optimization ratio indicates the ratio between the saved energy by frequency selection approach and the energy consumption under highest frequency. The higher the ratio is, the more energy is saved. In this section, each case is solved by GA. In GA, generation size is set to 150, population size is set to 100, and the amount of candidates is set to 10. There are 12 systems involves ($d10$ to $d120$), and the cases are divided into 4 categories based on their throughputs per each node. The throughputs for each node are 3500 Ops/Sec, 4000 Ops/Sec, 4500 Ops/Sec and 5000 Ops/Sec.

$$O(d, l, a) = \frac{E(d, l, f^{max}) - E(d, l, a)}{E(d, l, f^{max})} \tag{5.5}$$

The result is shown in Figure (5.7). The optimization ratio depends on the value of throughput per node. For $\forall l_1, l_2 \in \{3500, 4000, 45000, 5000\}$ $l_1 > l_2$, $O(d, l_1) < O(d, l_2)$. With the same value of throughput per node, the optimization ratios are concentrated. With the increment of value of throughput per node, the optimization ratio decrease. The maximum of optimization ratio

is 26.2% for the case $(d10, 3500, GA)$, and the minimum of optimization ratio is 16% for the case $(d110, 5000, GA)$.

If the power consumption is the only concern of the system's administration, the maximum optimization ratio can be constructed as follows. The node's throughput is set to 3520 Ops/Sec and the node is set to the performance mode. According to equation (5.5), the maximum optimization ratio is calculated as equation (5.6). 26.43% is the optimization bound of the model theoretically. The higher optimization ratio bound could be obtained by means of decreasing workload. However when the workload is too low, the value is meaningless, because the cluster is fully under-utilized. In the real cases, the optimization ratio might be lower than 26.43% because of the migration cost.

$$\begin{aligned} O(d1, 3520) &= \frac{P(< 2.53Ghz, 3520 >) - P(< 1.20Ghz, 3520 >)}{P(< 2.53Ghz, 3520 >)} \\ &= \frac{c^{idle}_{2.53Ghz} + \frac{3520}{5690} \times (c^{max}_{2.53Ghz} - c^{idle}_{2.53Ghz}) - c^{max}_{1.20Ghz}}{c^{idle}_{2.53Ghz} + \frac{3520}{5690} \times (c^{max}_{2.53Ghz} - c^{idle}_{2.53Ghz})} \\ &= 26.43\%. \end{aligned} \tag{5.6}$$

5.4.4 Comparison with Half Hot and Half Cold Approach

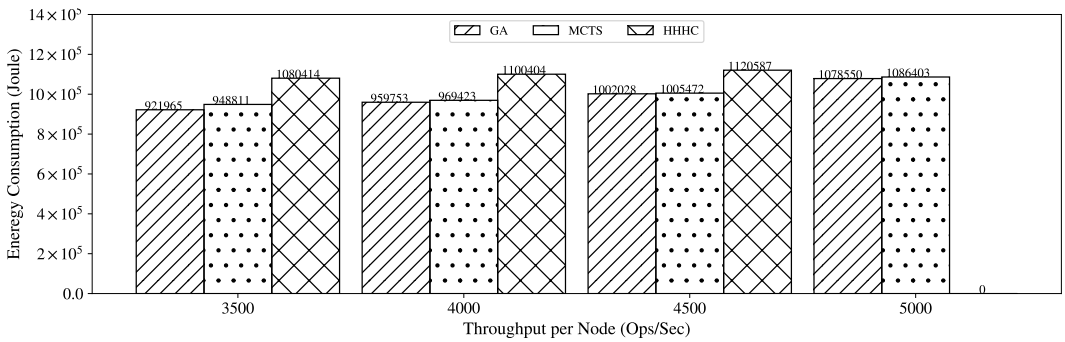


Figure 5.8: Comparison with Half Hot and Half Cold Approach

In this section, a comparison between Half Hot and Half Cold Approach (HHHC) [19], GA and MCTS is made. The system used in this section is $d20$. For the cases, the throughput per node is set to 3500 Ops/Sec, 4000 Ops/Sec, 4500 Ops/Sec and 5000 Ops/Sec respectively.

The results are shown as Figure (5.8). The results given by GA and MCTS are better than the corresponding results given by HHHC. The average improvement of GA compared with HHHC is 12.65%, and the average improve-

ment of MCTS is 11.44%. When the value of the throughput on each node is set to 5000 Ops/Sec, HHHC approach cannot produce a valid result. Theoretically, when HHHC approach is applied, the system with 20 nodes can support any workloads with throughput under 92100 Ops/Sec, however with the setting 5000 Ops/Sec for each node, the system does not have enough resources to support it. Therefore the corresponding energy consumption is recorded as 0. The main drawback of HHHC is its flexibility. GA and MCTS choose the frequency vector according to the workload predictions, while HHHC sets the frequencies statically.

5.5 Algorithm Robustness Analysis

In previous sections, the frequency selection approach and corresponding algorithms are based on the predictions of the workload. Because the prediction errors are inevitable, the robustness of the algorithms are analyzed in this section.

To analyze the robustness of algorithms, systems $d10$, $d20$ and $d30$ are used in this section. In the specialized model, the block access possibility φ_{gk} is the key to define the workload for cloud database systems.

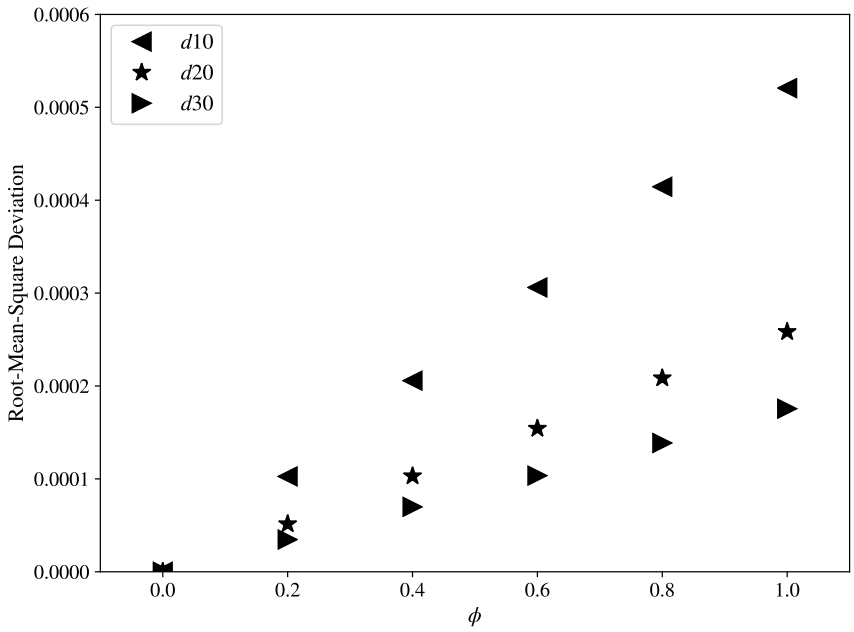
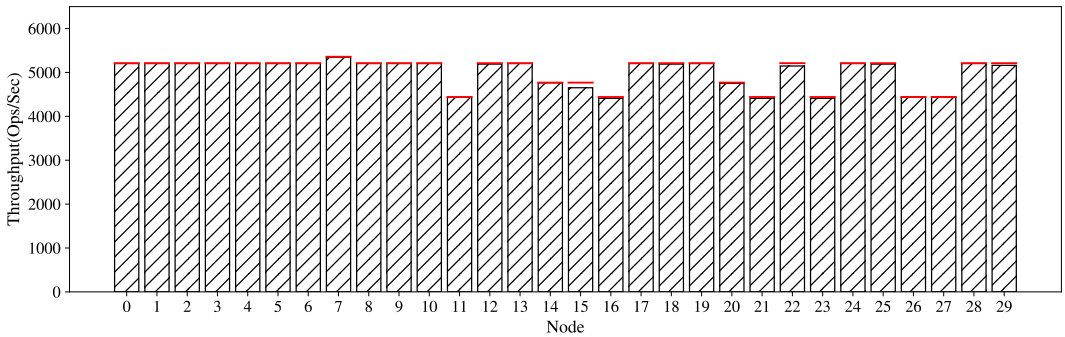


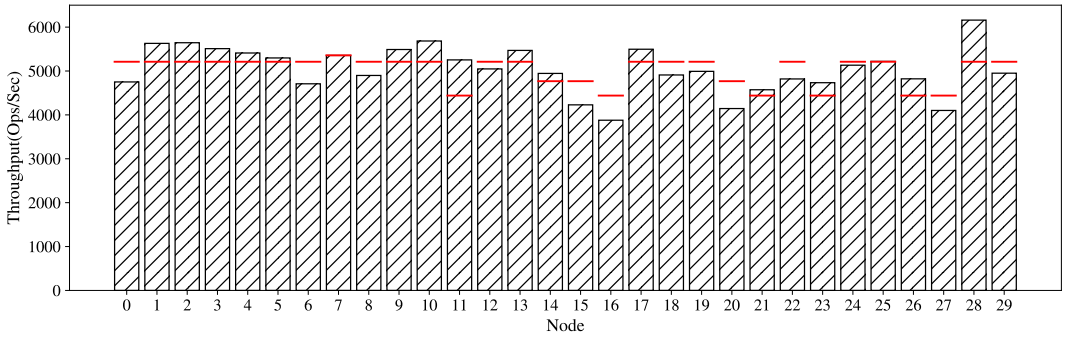
Figure 5.9: Relationship between ϕ and Root-Mean-Square Deviation

At first the prediction errors extracted from the normalized distribution $N(\mu, \sigma^2)$. In each case, μ is set to 0 to make sure half of errors are neg-

ative and other half of errors are positive. The range of values extracted from the normalize distribution is within $[\mu - 3\sigma, \mu + 3\sigma]$. To make sure the ranges of errors are the same with corresponding case, and the σ is set to $\overline{\varphi}\phi$ in which ϕ is the value from $[0, 0.2, 0.4, 0.6, 0.8, 1]$ and $\overline{\varphi}$ is average value of φ_{gk}. ϕ controls the total errors. Specially, $\phi = 0$ indicates that there is no error introduced. Secondly, the errors are added to the corresponding φ_{gk} to simulate the cases with prediction errors, and the cases are denoted as $(d\{NodeAmount\}, \phi)$. The root-mean-square deviation ($RMSD$) values of errors are calculated for each case, and the result is shown by Figure (5.9). $RMSD$ represents the amount of errors introduced. A higher $RMSD$ value refers to more errors are introduced. With the same node amount, the increment of ϕ leads to the increment of $RMSD$, because larger ϕ value increases the possibility of generating larger error values. With the same ϕ value, when $i > j$ $\mathbf{RMSD}(di, \phi) < \mathbf{RMSD}(dj, \phi)$. This scenario is caused by corresponding $\overline{\varphi}$, since $\sigma = \overline{\varphi} \times \phi$. For each case, every node is assigned with 64 blocks and $\overline{\varphi} = \frac{1}{64 \times NodeAmount}$. When $\overline{\varphi}$ increases, the RMSB increases. Therefore, when the case with more nodes, **RMSD** value is lower.



(a) Original Result of Frequency Selection Approach



(b) Corrected Result of Frequency Selection Approach

Figure 5.10: Example of Prediction Error Influence

When GA and MCTS are applied to cases with prediction errors, corre-

sponding selected frequencies and migration plans are collected. Because of the prediction errors, two scenarios arise:

1. The workload of a node exceeds its capacity;
2. Some of the resources of a node are wasted since the assigned workloads are too low.

To describe these scenarios, the execution result of GA for case $(d30, 1)$ is shown as Figure 5.10a and Figure 5.10b. In the figures, the bars present the workloads for each node, and the lines represent the node's capacities which are selected by GA. Figure 5.10a shows the original result of frequency selection result, in which $w_i^* < z(c_i, f_i)$. However because of existence of prediction error, the workload of each node is not correct. Figure 5.10b shows the corrected result, in which the workload of each node is recomputed according to the real $\varphi_{g,k}$. In Figure 5.10b, some nodes exceed their capcities, i.e. node 1, node 2 and so on. Some nodes waste their resources, i.e. node 0, node 6 and so on.

According to the experiment 3.1.4, when request throughput (workload) tries to exceed the node's capacity, the system throughput declines due to the resource limitation and the operation failure. To make sure the node can reach its capacity, if the workload exceeds the node capacity, part of the requests are refused. The ratio between refused requests and succeeded requests is defined as error ratio for the node. To describe the error ratio for the whole system, maximum error ratio (MER) is introduced which is defined by Equation 5.7. MER is used to describe scenario 1.

$$MER = \max \left\{ \frac{\sum\limits_{b_{gk} \in \mathbf{D}_i^r} l \times \varphi_{gk} - z(c_i, f_i)}{z(c_i, f_i)}, i \in [1, n] \right\}. \tag{5.7}$$

The ratio between wasted resource with the node's capacity is defined as waste ratio. Same with the MER, MWR, defined by equation 5.8, indicates the maximum percentage of resources wasted amongst all nodes, which is used to describe scenario 2.

$$MWR = \max \left\{ \frac{z(c_i, f_i) - \sum\limits_{b_{gk} \in \mathbf{D}_i^r} l \times \varphi_{gk}}{z(c_i, f_i)}, i \in [1, n] \right\}. \tag{5.8}$$

The results of the influence of prediction errors are shown in Figure 5.11. By means of GA and MCTS, the frequencies are selected, and the indicators,

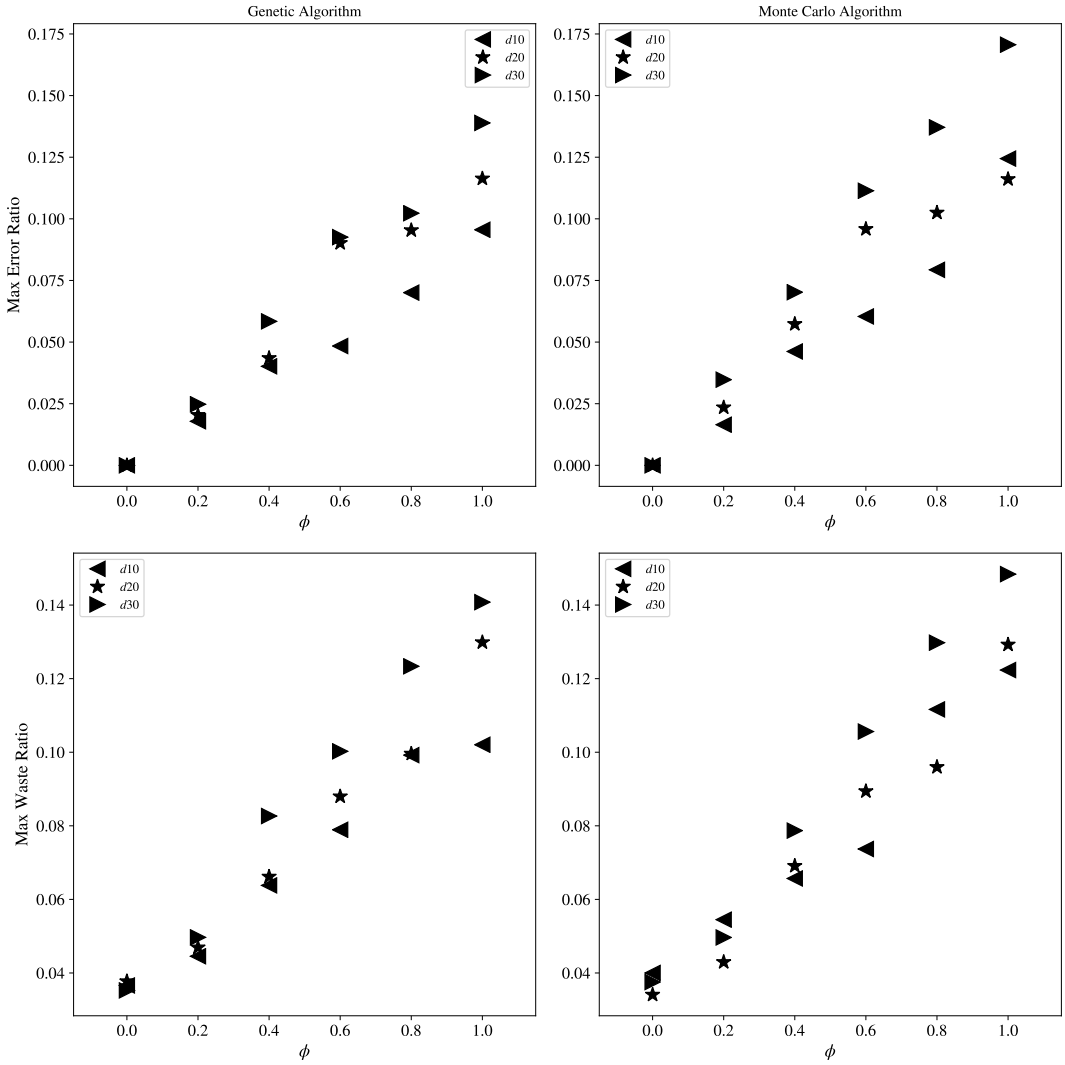


Figure 5.11: The Influence of Prediction Errors

MER and MWR, are calculated by Equation 5.7 and Equation 5.8. With the same node amount d, if $\phi_1 > \phi_2$, then $MER(d, \phi_1) > MER(d, \phi_2)$ and $MWR(d, \phi_1) > MWR(d, \phi_2)$. For cases with same node amount, the increment of ϕ leads to increment of $RMDS$ which indicates more errors are introduced, and more errors lead to higher MER and MWR. Specifically, when there is no error ($\phi = 0$) for $\forall d \in [d10, d20, d30]$, $MER(d, 0) = 0$ and $MWR(d, 0) > 0$. In perspective of frequency selection model, SLA cannot be violated, therefore no request is refused when no prediction errors are introduced. However, since the blocks are not continuous, some energy is wasted when the migration process generates migration plan. But the algorithms try to minimize the energy consumption. When $\phi = 0$, MWR is around 0.04.

According to Figure 5.11, when prediction error exists, it results in higher MER and MWR. In order to decrease them, the capacity tolerance factor is introduced, which is denoted as G. With the capacity tolerance factor, the new capacity $z^*(c_i, f_i)$ of node c_i with frequency f_i is shown by Equation (5.9). When $G < 0$, the capacity of the nodes is regarded lower than its original capacity. When the amount of workloads is constant, the higher frequencies will be selected to make sure enough resources to execute the workload. In contrast when $G > 0$, the capacity of the nodes is regarded higher than its original capacity and the lower frequencies are likely to be selected to avoid energy waste.

$$z^*(c_i, f_i) = z(c_i, f_i) \times (1 + G). \tag{5.9}$$

The values of $G \in [-0.1, -0.05, 0, 0.05, 0.1]$ are adopted to evaluate the influence of the capacity tolerance factor. Since the conclusions are quite similar, part of the results, cases with ϕ 0.2 and 1, are shown in Figure 5.12. Generally, for each case, MER increases with the increment of G, while MWR is decreasing. The reason for this scenario is that when $G < 0$, the higher frequencies are selected, which leads to lower MER. Correspondingly, more resources are wasted which leads to higher MWR. When $G > 0$, the lower frequencies are selected, which avoid resource waste. However, it causes higher MER. According to the experiment result, there is a trade off between MER and MWR by means of tuning the capacity tolerance factor. With the increment of capacity tolerance factor (from negative to positive), MER increases which indicates the increment of SLA violation, and meanwhile MWR decreases, which indicates the decrease of energy waste. By means of comparison of results from GA and MCTS, with the same ϕ and G, there are no big difference between them in terms of MER and MWR. For example, in Figure 5.12 $\phi(0.2) - Genetic$, the range of MER is from 0% to 14% which is the same with $\phi(0.2) - MCTS$. In term of MWR, there is the same result. The influence of

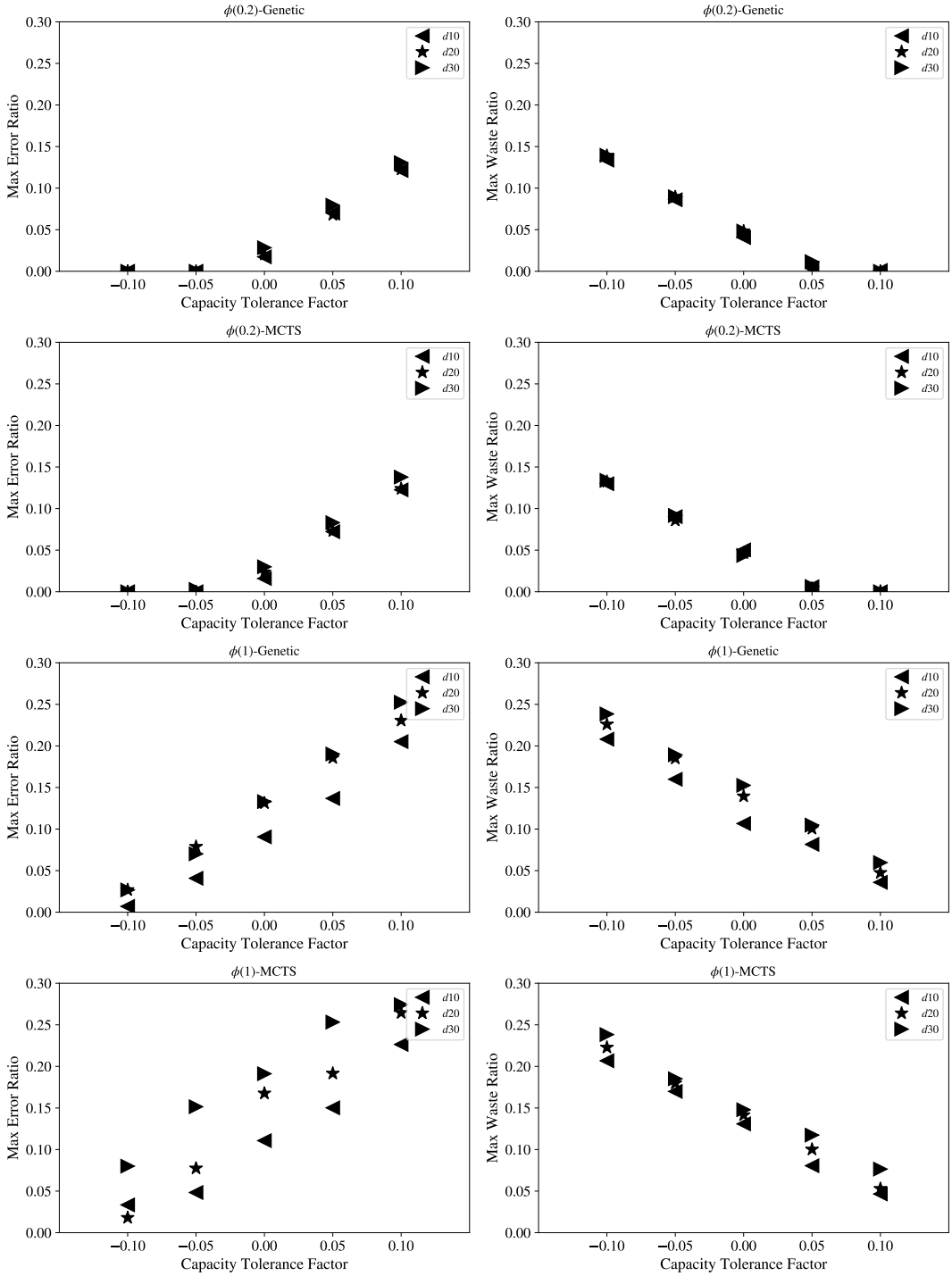


Figure 5.12: The Influence of Capacity Tolerance Factor

capacity factor decreases with more prediction errors, because more prediction errors lead to higher MER and MWR. For example, with $C = -0.01$ and GA, $MER(d30, 0.2) < MER(d30, 1)$.

In this section, the robustness of corresponding algorithms was analyzed. Prediction errors cause SLA violations and energy waste. In order to eliminate the influence of the prediction errors, the capacity tolerance factor is introduced. By tuning the capacity tolerance factor, the trade off between MER and MWR can be found. However, when the case has higher MER, it leads to more requests failure. Therefore, in practice, the MER should be kept low.

5.6 Summary

In this Chapter, we mainly introduced frequency selection approach with optimization problem, in which algorithms are proposed to minimize energy consumption in Δt. The proposed algorithms include a genetic based algorithm and a monte carlo tree search based algorithm. Both algorithms have its advantages and disadvantages. The results of the experiment show that both algorithms have great scalability with reasonable accuracy (up to 99.9% and 99.6% for two algorithms respectively). Since the prediction errors are inevitable, the robustness of the algorithms is analyzed. The prediction errors might cause SLA violations and energy waste. By means of tuning the capacity tolerance factor, the trade off between SLA violation and energy wasting can be found. In practice, the error ratio should be kept in a lower range.

Chapter 6

Migration Approach

To cope with resource provisioning within cloud database system, migration process is an indispensable part. In pervious chapters, we mainly introduced frequency selection approach with different objectives. In this chapter, the details of migration approach are given with respect to block selection and block migration. In the end, a series of experiments are implemented to evaluate block selection algorithm and block migration algorithm. Besides, an experiment is carried out to compare the estimation approach (Section 3.3) with the migration approach.

6.1 Objective

Migration Approach can be considered as a data layout transformation method, in which the workload changes accordingly. In cloud database system, the basic migration unit is data blocks $b_{g,k}$. Generally, the migration approach consists of following 3 phases:

1. **Block Selection**: Under a given frequency vector $\mathbf{F}$, a migrated block set is generated for each node, which contains the blocks to be migrated out of the node. The migrated block set for node c_i is denoted as $\mathbf{M}_i$

2. **Migration Plan Generation**: A valid migration destination for each migrated block is determined in this phase. The collection of migrated blocks and their corresponding destinations is defined as migration plan. In cloud database system, a replication factor is applied to control the number of replicas, a valid destination refers to that the destination node is not the original node that contains the migrated block, and the node does not contain the same replica of the given migrated block as well.

3. **Migration Plan Execution**: Migration is executed according to the migration plan.

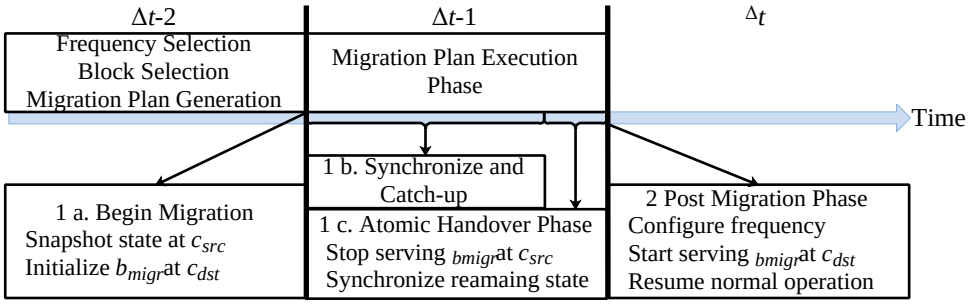


Figure 6.1: Migration Timeline

In this thesis, we focus on the former two phases for the following reasons. 1) The primary objective of migration approach is to conduct the migration process with minimal migration cost, and the former two phases play a decisive role. According to migration cost estimation function, Equation 3.7, migration cost is determined by the size of migrated blocks and the migration directions (within a rack or between racks). Therefore, the migration cost is obtained after phase 2. 2) For **Migration Plan Execution**, a live database migration technique from Sudipto Das *et al.* [22] is adopted.

A migration execution technique considers service downtime and migration overhead [22]. In service downtime, the database or part of database is unavailable. The migration overhead indicates the additional work done and the corresponding impact on operations to facilitate migration, for example response delay, cache rebuild. Intuitively, a migration process can be done within parallel database systems or could database systems by Stop and Copy technique, in which the operations involving migrated data are aborted until the migration process is done. This technique introduces considerable service downtime, in which the services are unavailable. One of the improvements for Stop and Copy technique is On Demand Migration [90]. On Demand Migration takes advantage of cache technique. When the migration happens, the destination node fetches the needed data from the original node and caches them in the local storage. This technique reduces service interruption but introduces high post delay time because of the expensive cache misses. To overcome the service interruption and migration overhead, Sudipto Das *et al.* [22] proposed live migration technique.

Figure 6.1 shows the timeline of a live migration process for time window Δt. In time window $\Delta t-2$, the frequency selection and migration approach are

applied to obtain the frequency vector and migration plan for Δt according to the predicted workload value. The migration plan execution phase is conduct in time window $\Delta t - 1$. The execution phase includes 3 sub-phases which are summarized briefly as following:

1. Phase 0 **Pre-Migration**: Use frequency selection approach and migration approach to obtain the frequency vector and migration plan for Δt.

2. Phase 1 **Migration**: The migration process is executed according to migration plan. An example is made for block b_{migr} which is migrated from node c_{src} to node c_{dst}. This phase can be divided into 3 steps further.

   (a) **Begin Migration**: A snapshot is taken in c_{src}, which is transformed to c_{dst}, and b_{migr} is initialized in c_{dst}. During this step, all the operations involve with b_{migr} are process within c_{src}.

   (b) **Iterative Copy**: After the snapshot copy, c_{src} still works on the operations involving b_{migr}, therefore b_{migr} on c_{src} and c_{dst} are not synchronized. In this step, the updates of b_{migr} since the snapshot are copied from c_{src} to c_{dst} iteratively.

   (c) **Atomic Handover**: In this step, the ownership of b_{migr} is transferred from c_{src} to c_{dst}. This step is an atomic operation to deal with the failures. The operations during this step are aborted in c_{src} and restarted in c_{dst}.

3. Phase 2 **Post-Migration**: In this phases, the frequencies of the cluster are reset according to the frequency vector obtained in $\Delta t - 2$. Operations involving b_{migr} are sent to c_{dst} directly.

By means of live migration, the migration plan execution phase can be concluded with minimized service disruption manner. According to Sudipto Das *et al.* [22] results, their approach reduces 3 to 10 times database unavailable time compared with Stop and Copy technique.

In this thesis, we focus on **Pre-Migration** phase that includes **Block Selection** and **Migration Plan Generation**. The primary goal of migration process is divided into 2 targets which related to each phase. The migration cost is determined by the total size of migrated blocks and their migration directions. The objective of **Block Selection** is to generate migrated block sets with minimizing the total size of migrated blocks. The objective of **Migration Plan Generation** is to maximize the total size of migrated blocks within racks.

6.2 Block Selection

In this phase, a set of blocks for the migration are selected in a manner of minimizing the total size of migrated blocks.

When a frequency vector $\mathbf{F}$ is assigned to the system, the nodes, in which the blocks are migrated, are determined. The migration node set is denoted as $\mathbf{C}_{\text{Out}} = \{c_i | z(c_i, f_i) < w_i \ i \in [1, n]\}$ i.e. it consists of nodes for which the workload exceeds their capacity. For $\forall c_i \in \mathbf{C}_{\text{Out}}$, the migrated workload size is $w_i - z(c_i, f_i)$. To minimize the migration cost, the migration selection problem can be interpreted in another way: which blocks should stay in place to achieve the node's capacity while the total size of the kept blocks is maximized? This problem can be treated as a 0/1 knapsack problem. The item is block, and the weight and the profit of the item are the throughput of the block and the block size, respectively. The volume of the knapsacks is the capacity of the node. The detail of the knapsack problem is shown by Table 6.1.

Table 6.1: 1-0 Knapsack Details in Block Selection Phase

Term	Description
Item	Block whose accessed probability is greater than 0.
Knapsack	$c_i \in \mathbf{C}_{\text{Out}}$; Knapsack size $= z(c_i, f_i)$.
Weight	Throughput of the blocks, namely $l \times \varphi_{g,k}$.
Profit	Block size $\lvert b_{g,k} \rvert$.
Objective	Maximize the sum of the kept block sizes.

Using dynamic programming algorithm [91], the optimal kept blocks can be found. The complexity of the algorithm is considered as $O(nC)$ in which n is the number of items and C is the size of knapsack. However within 0-1 knapsack problem, the complexity is pseudo-polynomial [92]. Considering the number of item, 64 in our experiment and 256 in practice, and the size of knapsack, from minimal capacity to maximal capacity, dynamic programming algorithm cannot solve the block selection problem with an acceptable time. In our experiment, it takes minutes to solve one single case. What is worse, the block selection process is required to obtain the upper bound of migration cost in Section 3.3 , namely the algorithm is used as many as frequency vectors. Eventually, the block selection part would be a bottleneck within frequency selection approach.

In order to overcome the performance issue of dynamic programming algorithm, a greedy algorithm, *GreedyBlockSelection*, is used to solve the block selection, in which the items are sorted and selected by their $profit/weight$ ratio, namely by $\lvert b_{g,k} \rvert / l \times \varphi_{g,k}$. *GreedyBlockSelection* is shown as Algorithm 7.

Algorithm 7 Greedy Block Selection

Require: $\mathbf{D}^r$ Block sets, $\mathbf{Cap}$ Capacities of the nodes
Ensure: $\mathbf{D}_{\text{Migr}}$ Migrated Block set

```
function GreedyBlockSelection(D^r, Cap)
    B ← ∅, n ← |Cap|
    Diff ← {SumThroughput(D_i^r) − Cap_i | i ∈ [1, ..., n]}
    C_max ← − min(Diff)
    for i ← [1, ..., n] do
        B̃ ← ∅
        if Diff_i > 0 then
            SD_i^r ← SortBlocksByDescending(D_i^r)
            for b ← D_i^r do
                if Diff_i > 0 and b.tp ≤ C_max then
                    B̃ ← B̃ ∪ b
                    Diff_i ← Diff_i − b.tp
                else
                    break
                end if
            end for
        end if
        B ← B ∪ B̃
    end for
    return D_Migr
end function
```

The input parameters of Algorithm 7 are $\mathbf{D}^r$, the set of assigned block set including block set assigned to each node, **Cap**, the capacity set of the nodes obtained according to the given frequency vector. The assigned block set for node c_i is denoted $\mathbf{D}_i^r \in \mathbf{D}^r$. The blocks belonging to $\mathbf{D}_i^r$ are denoted as $b \in \mathbf{D}_i^r$, in which each block has two properties. $b.tp$ represents the throughput of the block, namely $l \times \varphi_{g,k}$, and $b.w$ represents the size of the block, namely $|b_{g,k}|$. The output of the algorithm is a set of migrated block set, **B**. In Algorithm 7, line 2 and line 3 initializes the result set and the number of nodes. Line 3 computes a set **Diff** in which the differences between current workload of the nodes and corresponding capacity. Function $SumThroughput$ gives the workload of the node, namely $\sum_{b_{g,k} \in \mathbf{D}_i^r} l \times \varphi_{g,k}$. Line 4 obtained the maximum remaining capacity of the nodes. Line 5 to line 19 obtain the migrated block set for each node. For node c_i, line 8 sorts the blocks by $|b_{g,k}|/l \times \varphi_{g,k}$ in descending order. Line 9 to line 16 evaluate each block. When the workload exceeds the capacity, namely $\mathbf{Diff}_i > 0$, and the throughput of block $b.tp$ smaller than the maximum remaining capacity of the nodes, the block is marked as a migrated block. In the end, line 20 returns the set of migrated block set.

The complexity of Algorithm 7 is $O(\sum_{i=1}^{n} |\mathbf{D}_i^r| \log |\mathbf{D}_i^r|)$ because of the sort operation in line 8. In practice, the performance of the algorithm can be further improved by removing the sort operation outside of the algorithm since the block assignment is unchanged during the frequency selection process. Therefore, the complexity of Algorithm 7 can be reduced to $O(\sum_{i=1}^{n} |\mathbf{D}_i^r|)$ because of block selection process from line 9 to line 15. Compared with dynamic programming algorithm, the greedy algorithm has polynomial complexity.

6.3 Block Migration Plan

The goal of this phase is to generate a migration plan which gives the destination for the migrated blocks. The objective is to reduce the total migration cost. According to Equation 3.7, the migration cost is related to the migrated block sizes and their destinations (i.e. within a rack or between racks). The migrated blocks are determined by Migration Selection phase, which means the block sizes are determined as well. Therefore, the objective of this phase is to generate a migration plan which keeps the migrated blocks in their racks as much as possible. The main process of this phase is shown as follow.

1. Migrated blocks are placed within their own racks first.

2. If some blocks cannot be placed within their own rack, then they are redistributed within the whole cluster.

The migration plan generation can be interpreted in another way: Which blocks should be kept within their own rack in order to minimize the migration cost? Actually, the migration problem can be treated as a constrained multiple knapsack problem. The multiple knapsacks are the interspaces of the nodes whose predicted workloads do not exceed their capacity. The item's weight and profit are block workload and block size respectively. The problem objective is to maximize the block sizes which are kept within the rack. The detail of the multiple knapsack problem is shown by Table 6.2.

Term	Description
Item	Migrated Blocks which are selected in Block Selection Phase.
Knapsacks	The nodes with extra capacity. $\{c_i \mid (z(c_i, f_i) - w_i) > 0 \ i \in [1, n]\}$. Knapsack size= $z(c_i, f_i) - w_i$.
Weight	Throughput of the block $l \times \varphi_{g,k}$.
Profit	Block size $\lvert b_{g,k} \rvert$.
Objective	Maximize the sum of block sizes which are migrated within their own racks.

Table 6.2: Multiple Knapsacks Details in Block Migration Phase

In order to solve the multiple knapsack problem, a constrained exchange algorithm, **C**onstrained **MTHM** (CMTHM for short), is introduced. This algorithm is inspired by MTHM algorithm [93]. In MTHM algorithm, there are 4 steps basically:

1. Choose the item using a greedy algorithm for each knapsacks (items are sorted by descending $profit/weight$ and knapsacks are sorted by descending volumes).

2. Rearrange the selected items to make sure not all items of similar profile per unit weight are stored in the same knapsack.

3. Interchange assigned items and try to insert an unassigned item.

4. Exchange assigned items with unassigned items if the exchange can improve the total profile.

Table 6.3: Specifications of the symbols

Symbol	Type	Comment
n	Int	The number of nodes (knapsacks)
m	Int	The number of migrated blocks (items)
i	Int	Index of nodes
j	Int	Index of migrated blocks
p, q	Int	Temporary Indexes of a node
g, k, t	Int	Temporary Indexes of a migrated block
$\mathbf{B}$	Collection	Migrated block collection consists of all the migrated blocks of a certain rack. A migrated block is denoted as $\mathbf{B}_j$. A block contains two properties. $\mathbf{B}_j.tp$ represents the throughput of the block, namely $l \times \varphi_{g,k}$, and $\mathbf{B}_j.w$ represents the size of the block, namely $\lvert b_{g,k}\rvert$. Without loss of generality we assume that $\mathbf{B}$ is sorted by ratio of ${}^{\mathbf{B}_j.w}/_{\mathbf{B}_j.tp}$ in descending order.
d	Float	Throughput difference between two blocks, namely $\mathbf{B}_p.tp - \mathbf{B}_q.tp$
$\mathbf{RCap}$	Collection	A collection of remaining capacities. For a rank γ, $\mathbf{RCap} = \{z(c_i, f_i) - w_i \mid c_i \in \gamma, \text{ and } z(c_i, f_i) - w_i > 0\}$. Without loss of generality we assume that $\mathbf{RCap}$ is sorted by value of $z(c_i, f_i) - w_i$ in descending order. To be noticed, we do not distinguish the difference between the index of node and the index of $\mathbf{RCap}$ and we assume $n = \lvert\mathbf{RCap}\rvert$
$\widetilde{\mathbf{RCap}}$	Collection	A copy of $\mathbf{RCap}$
$\mathbf{Y}$	Array	A migrated block assignment. When a migrated block $\mathbf{B}_j$ is not assigned, $\mathbf{Y}_j = 0$. If $\mathbf{B}_j$ is assigned to node c_i, $\mathbf{Y}_j = i$ and $\mathbf{Cap}_i = \mathbf{Cap}_i - \mathbf{B}_j.tp$
s	Float	The total size of assigned migrated blocks
$NoReplica(i, j)$	Function	A function shows whether block $\mathbf{B}_j$ in node c_i has a replica. When there is a replica of $\mathbf{B}_j$, the function return $True$.

In CMTHM, the replicas of a block are regarded as conflict with each other when items are moving, i.e. two replicas of the same block cannot be put into the same node. If a conflict occurs, the exchange continues until the next exchange opportunity is found. In order to minimize the migration cost, the migration process is applied to each rack first. If there are some remaining blocks, the process is applied to the whole cluster. The latter one is denoted as global migration. Compared with the migration within racks, the knapsacks in the global migration are the nodes of the cluster which do not reach their capacities. By applying CMTHM $U + 1$ times (U is the amount of racks), a migration plan for the given frequency vector can be obtained.

To illustrate CMTHM, we introduce an example first, which will be used in the following discussion. Example 1 shows the parameters. The example consists of 3 nodes, $n = 3$, and 15 blocks, $m = 15$. Without loss of generality, we say the start of each set is 1. The set of pairs of block throughput and block size is shown as $\mathbf{B}$. In this example, $\mathbf{B}$ is sorted descending by the ratio of block size over block throughput. To be noticed, we assume all the blocks have the same size 1. The capacities of nodes are shown as $\mathbf{RCAP}$, in which we assume all the nodes have the same capacity, 100 in our example, to simplify the description. $\mathbf{CF}$ shows the conflict within the migration process that represents replica conflict. For example, $\mathbf{CF}_1 = [7, 10]$ means block $\mathbf{B}_7$ and $\mathbf{B}_{10}$ cannot be placed in node 1. $\mathbf{Y}$ shows the block assignment. If block $\mathbf{B}_1$ is assigned to node 2, then $\mathbf{Y}_1$ is set to 2. However, if a block is not assigned, then the corresponding value in $\mathbf{Y}$ is assigned to 0. According to $\mathbf{CF}$, $\mathbf{Y}_7$ and $\mathbf{Y}_10$ cannot be set to 1. s is the total profile. Since block sizes are set to 1, s can be considered as the number of assigned blocks.

$$
\begin{aligned}
n =& 3, m = 15 \\
\mathbf{B} =& [(10,1),(12,1),(13,1),(14,1),(14,1), \\
& (18,1),(19,1),(20,1),(22,1),(22,1), \\
& (22,1),(25,1),(27,1),(28,1),(30,1)] \\
\mathbf{RCAP} =& [100,100,100] \\
\mathbf{CF} =& [[7,10],[4,5],[12]] \\
\mathbf{Y} =& [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] \\
s =& 0
\end{aligned}
$$

Example 1: Parameters

CMTHM is shown as Algorithm 8 to Algorithm 12, in which Algorithm 8 shows a greedy migration function, and Algorithm 9 to Algorithm 12 show each

Algorithm 8 Greedy Migration

Require: i a given node index, $\mathbf{B}$ Sorted Migrated Blocks, s Total Assigned Block Size, $\mathbf{Y}$ Block Assignment, $\mathbf{RCap}$ Remaining Node Capacities

Ensure: s Total Assigned Block Size, $\mathbf{Y}$ Block Assignment, $\mathbf{RCap}$ Remainding Node Capacities

1: **function** GreedyMigration(i, $\mathbf{B}$, z, $\mathbf{Y}$, $\mathbf{RCap}$)
2: $\quad m \leftarrow |\mathbf{B}|$
3: $\quad$ **for** $j \leftarrow [1, ..., m]$ **do**
4: $\quad\quad$ **if** $\mathbf{Y}_j = 0$ **and** $\mathbf{RCap}_i \geq \mathbf{B}_j.tp$ *and* NoReplica(i, j) **then**
5: $\quad\quad\quad \mathbf{Y}_j \leftarrow i$
6: $\quad\quad\quad \mathbf{RCap}_i \leftarrow \mathbf{RCap}_i - \mathbf{B}_j.tp$
7: $\quad\quad\quad s \leftarrow s + \mathbf{B}_j.w$
8: $\quad\quad$ **end if**
9: $\quad$ **end for**
10: $\quad$ **return** $s, \mathbf{Y}, \mathbf{RCap}$
11: **end function**

phase of CMTHM. To simplify the descriptions, we list some specifications of the symbols in the algorithms in Table 6.3.

Algorithm 8 gives a greedy migration algorithm for node c_i. In Algorithm 8, $\mathbf{B}$ is sorted by the ratio between block size and block throughput in descending order. Line 3 to line 9, the unassigned migrated blocks are put into node c_i as much as possible. Line 4 gives the assignment conditions: 1) $Y_j = 0$ refers that the block $\mathbf{B}_j$ is not assigned to any node; 2) $\mathbf{RCap}_i \geq \mathbf{B}_j.tp$ indicates that the throughput of $\mathbf{B}_j$ is less than the remaining capacity of c_i; 3) $NoReplica(i, j)$ is used to ensure there is no replica of $\mathbf{B}_j$ in c_i.

Algorithm 9 Constrained MTHM Phase 1

Require: $\mathbf{B}$ Sorted Migrated Blocks, $\mathbf{RCap}$ Remaining Node Capacities

Ensure: $\mathbf{Y}$ Block Assignment

1: **function** InitialSolution($\mathbf{B}$, $\mathbf{RCap}$)
2: $\quad s \leftarrow 0,\ \mathbf{Y} \leftarrow \{0\},\ \widetilde{\mathbf{RCap}} \leftarrow \mathbf{RCap},\ n \leftarrow |\mathbf{RCap}|$
3: $\quad$ **for** $i \leftarrow [1, ..., n]$ **do**
4: $\quad\quad s,\ \mathbf{Y},\ \widetilde{\mathbf{RCap}} \leftarrow$ GreedyMigration(i, $\mathbf{B}$, s, $\mathbf{Y}$, $\widetilde{\mathbf{RCap}}$)
5: $\quad$ **end for**
6: $\quad$ **return** $\mathbf{Y}$
7: **end function**

Algorithm 9 shows the first phase of CMTHM in which a migration solution is obtained by Greedy Migration method. Line 2 initializes same parameters,

and line 2 to line 4 obtain the migrated block assignment for all the nodes. To be noticed, the solution $\mathbf{Y}$ is a temporary solution, since only the assignment $\mathbf{Y}$ is returned in line 6, which will be passed into next phase as a base solution for rearrangement. The complexity of this phase is $O(nm)$ because of the greedy assignment process.

$$\begin{aligned}\mathbf{RCAP} =& [19, 17, 23]\\ \mathbf{Y} =& [1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 0, 3, 3, 0]\\ s =& 13\end{aligned}$$

Example 2: Result of Phase 1

Example 2 shows the result of phase 1. Clearly, in phase 1, we only apply Greedy Migration. However, because of the conflict, $\mathbf{B}_{12}$ is not placed on node 3. The remaining capacities are shown as **RCAP**.

Algorithm 10 shows the second phase of CMTHM, *Rearrangement*. In CMTHM, the main concept is to take advantage of exchanging and inserting method to improve the objective value, namely the total size of the assigned migrated blocks. However, the exchange method is more effective when the items in the knapsack are dissimilar [93]. Therefore in this phase, a rearrangment solution is obtained based on the greedy solution. The input parameters of algorithm are $\mathbf{B}$, sorted migrated block set, $\mathbf{Y}$, the migrated block assignment generated by Algorithm 9, and **RCap**, Remaining node capacities. In Algorithm 10, line 1 initializes the parameters. To be noticed, the total assigned block size s is reset to 0, since a migration solution will be obtained. Line 3 to line 18 go through all the assigned migrated blocks within the greedy migration solution. line 6 to line 10 try to find a node index in which $\mathbf{B}_j$ can be assigned with. The sequence of node searching is $[i, ..., n] \cup [1, ..., i-1]$, which indicates the blocks are assigned in a round robin style. If the index is found, then the block is assigned to the node and corresponding parameter is updated (line 13 to line 16), otherwise the block is marked unassigned (line 12). In the end of algorithm (line 19 to line 21), the greedy migration is called again to make sure the blocks are assigned as much as possible. To be noticed, compared with phase 1, the results of greedy migration s, $\mathbf{Y}$ and **Cap** are recorded, which are used to improve the solution in phase 3 and phase 4. The complexity of this phase is $O(nm)$ because of the node searching for each assigned blocks and the greedy migration process. To be noticed, $NoReplicaCheck$ function can be implement by a *hashmap* technique, which gives a searching time $O(1)$.

Algorithm 10 Constrained MTHM Phase 2

Require: **B** Sorted Migrated Blocks, **Y** Block Assignment, **RCap** Remaining Node Capacities

Ensure: s Total Assigned Block Size, **Y** Assigned Indexes, **RCap** Remaining Node Capacities

```
function REARRANGEMENT(B, Y, RCap)
    i ← 1, s ← 0, m ← |B|, n ← |Cap|
    for j ← [m, ..., 1] do
        if Y_j > 0 then
            p ← 0
            for q ← [i, ..., n] ∪ [1, ..., i − 1] do
                if B_j.tp ≤ RCap_q and NOREPLICACHECK(q, j) then
                    p ← q
                end if
            end for
            if p = 0 then
                Y_j ← 0
            else
                Y_j ← p, RCap_p ← RCap_p − B_j.tp, s ← s + B_j.w
                if p < n then i ← p + 1 else i ← 1 endif
            end if
        end if
    end for
    for i ← [1, ..., n] do
        s, Y, RCap ← GREEDYMIGRATION(i, B, s, Y, RCap)
    end for
    return s, Y, RCap
end function
```

$$\mathbf{RCAP} = [0, 20, 14]$$
$$\mathbf{Y} = [3, 2, 1, 3, 1, 3, 2, 1, 3, 2, 3, 1, 2, 1, 0]$$
$$s = 14$$

Example 3: Result of Phase 2

Example 3 shows the result of phase 2. In phase 2, assigned blocks are rearranged first in robin style, and the greedy algorithm is called again to make sure we assign blocks as much as possible. To be noticed, from our result **Y**, the assigned blocks are not rearranged in a totally robin style because of the conflict **CF**. By phase 2, we have 14 assigned blocks in total.

Algorithm 11 Constrained MTHM Phase 3

Require: $\mathbf{B}$ Sorted Migrated Blocks, s Total Assigned Block Size, $\mathbf{Y}$ Block Assignment, $\mathbf{RCap}$ Remaining Node Capacities

Ensure: s Total Assigned Block Size, $\mathbf{Y}$ Block Assignment, $\mathbf{RCap}$ Remaining Node Capacities

```
function FirstImprovement(B, s, Y, RCap)
    n ← |Cap| ,  m ← |B|
    for j ← [1, ..., m]  do
        if Y_j > 0 then
            for o ← [j + 1, ..., m] do
                if Y_o > 0 and Y_o ≠ Y_j and NoReplicaCheck(Y_o, j) and
NoReplicaCheck(Y_j, o) then
                    p ← argmax(B_j.tp, B_o.tp)
                    q ← argmin(B_j.tp, B_o.tp)
                    d ← B_p.tp − B_q.tp
                    if d ≤ Cap_{Y_q} then
                        V ← {v|v ∈ [1, ..., m],  Y_v = 0,  B_v.tp ≤ Cap_{Y_p} + d,
                                NoReplicaCheck(Y_p, v)}
                        if V ≠ ∅ then
                            t ← argmax({B_v.w|v ∈ V})
                            Cap_{Y_p} ← Cap_{Y_p} + d − B_t.tp
                            Cap_{Y_q} ← Cap_{Y_q} − d
                            Y_t ← Y_p, Y_p ← Y_q, Y_q ← Y_t
                            s ← s + B_t.w
                        end if
                    end if
                end if
            end for
        end if
    end for
    return s,  Y, Cap
end function
```

Algorithm 11 shows the third phase of CMTHM, *FirstImprovement*. In this phase, an attempt is done to exchange two assigned items and insert an unassigned item. Line 3 to line 23 of the algorithm shows the exchange process. For an assigned block $\mathbf{B}_j$ $j \in [1, m]$, the exchange is tried with each block $\mathbf{B}_o$ $o \in [j+1, m]$. Line 6 evaluates the exchange process first: 1) two blocks are not assigned to same node $\mathbf{Y}_o \neq \mathbf{Y}_j$; 2) $\mathbf{B}_j$ does not have same replicas in node $\mathbf{Y}_o$; 3) $\mathbf{B}_o$ does not have same replicas in node $\mathbf{Y}_j$; In line 7 to line 9, a difference of throughput between two blocks is obtained, and index p is used to indicate the block with larger throughput while q refers to the other one. Line 9 shows another condition that $\mathbf{B}_p$ can be placed into $\mathbf{Y}_q$ by $d \leq \mathbf{Cap}_{\mathbf{Y}_q}$. Line 11 obtains a insertable block list, $\mathbf{V}$, in which all the unassigned blocks are evaluated. When $\mathbf{V}$ is not empty, the most valuable item, $\mathbf{B}_t$, is picked and inserted into $\mathbf{Y}_p$, line 12 to line 17. The complexity of this phase is $O(m^3)$ because of the evaluation process between two assigned blocks and the insertable blocks searching process.

$$\begin{aligned}\mathbf{RCAP} =& [0, 2, 2]\\ \mathbf{Y} =& [2, 2, 1, 3, 1, 3, 2, 1, 3, 3, 3, 1, 2, 1, 2]\\ s =& 15\end{aligned}$$

Example 4: Result of Phase 3

Example 4 shows the result of phase 3. Phase 3 exchanges the assigned blocks if an unassigned block can be inserted. In our example, $\mathbf{B}_1$ assigned to node 3 in Phase 2, and $\mathbf{B}_{10}$ assigned to node 2 in Phase 2, are exchanged. Afterward, $\mathbf{B}_{15}$ is inserted to node 2. All blocks are assigned in this phase, $s = 15$.

Algorithm 12 shows the last phase of CMTHM, *SecondImprovement*. In this phase, an exchange process is done within assigned blocks and unassigned blocks. Line 3 to line 18 implement the exchange process. Each assigned block $\mathbf{B}_j$ is evaluated. If $\mathbf{B}_j$ was removed, a new capacity value, c, of $\mathbf{Y}_j$ is obtained in line 5. Line 7 to line 12 evaluate all unassigned blocks, if the block $\mathbf{B}_o$ can be assigned to $\mathbf{Y}_j$, then it is put into collection $\mathbf{V}$. If the total value (total size) of blocks in $\mathbf{V}$ is larger than $\mathbf{B}_j$, then $\mathbf{B}_j$ is marked unassigned and blocks in $\mathbf{V}$ is assigned to $\mathbf{Y}_j$, line 14 to line 17. In the end, the final solution $\mathbf{Y}$ and the total size of assigned block s is returned. The complexity of this phase is $O(m^2)$ since the assigned block evaluation process and unassigned block searching process. In this phase, the assignment of our example is remained same since all blocks are assigned in phase 3.

The phases of CMTHM are executed sequently. Therefore the complexity of CMTHM algorithm is $O(m^3)$, namely $O(|\sum_{i=1}^{n} \mathbf{D}_i^r|^3)$.

Algorithm 12 Constrained MTHM Phase 4

Require: $\mathbf{B}$ Sorted Migrated Blocks, s Total Assigned Block Size, $\mathbf{Y}$ Block Assignment, $\mathbf{Cap}$ Remaining Node Capacities

Ensure: s Total Assigned Block Size, $\mathbf{Y}$ Block Assignment

```
function SecondImprovement(B, s, Y, Cap)
    n ← |Cap| , m ← |B|
    for j ← [1, ..., m] do
        if Y_j ≠ 0 then
            c ← Cap_{Y_j} + B_j.tp
            V ← ∅
            for o ← [1, ..., m] do
                if Y_o = 0 and B_o.tp ≤ c and NoReplicaCheck(Y_j, o) then
                    V ← V ∪ {o}
                    c ← c − B_o.tp
                end if
            end for
            if Σ_{v∈V} B_v.w > B_j.w then
                for v ← V do
                    Y_v ← Y_j, s ← s + B_v.w, Cap_{Y_j} ← Cap_{Y_j} − B_v.tp
                end for
                Y_j ← 0, s ← s − B_j.w, Cap_{Y_j} ← Cap_{Y_j} + B_j.tp
            end if
        end if
    end for
    return s, Y
end function
```

6.4 Experiment

In this section, 2 experiments are conducted to evaluate block selection algorithm, Greedy Selection, and migration algorithm, CMTHM. In the end, an experiment is implemented to compare the estimation approach, Section 3.3, with the migration approach. The test cases used in this experiment are same with Section 4.4. To be noticed, in order to evaluate the migration approach, a few frequency vectors are generated within each experiment accordingly using frequency selection approach with optimization problem for following reasons: 1) migration cost only relies on the migration approach; 2) compared with frequency selection approach with bounded problem, frequency selection approach with optimization problem is practical in the experiment since there is

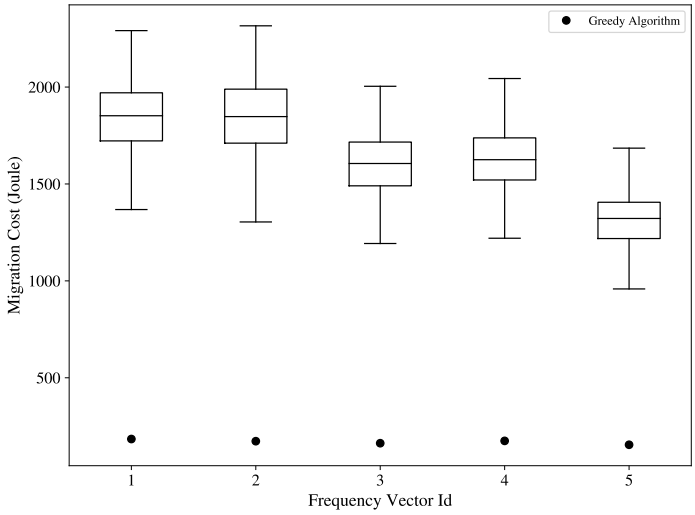


Figure 6.2: Result of Migrated Block Comparison

Algorithm	Valid Solution Amount
First Fit	877
CMTHM	913

Table 6.4: The Amount of Valid Solutions of 1000 Solutions

no requirement for the bound value.

6.4.1 Migrated Block Comparison

In migration selection, migrated blocks are selected by the Greedy algorithm, Algorithm 7, to reduce migration cost. In this section, the greedy selection is compared with random selection.

The test case $(d20, GA)$ is used in this experiment. 5 frequency vectors are given by Genetic Algorithm, Section 5.2. For each frequency vector, the random selection approach is applied 1000 times and the corresponding migration cost values are obtained using Equation(3.7). To be noticed, the frequency vectors are given to minimize the energy consumption in the time window. Therefore, each frequency vector is the suboptimal solution. All of them try to provide a frequency setting which fits to the workload predictions. Since the frequency vector determines the total migrated workloads for each node, the migration cost is only determined by the migrated blocks and the migration plan. In this experiment, the migrated blocks are obtained by two algorithms, Random Selection and Greedy Selection. The migration plan is generated by CMTHM. The comparison results is shown by Figure 6.2.

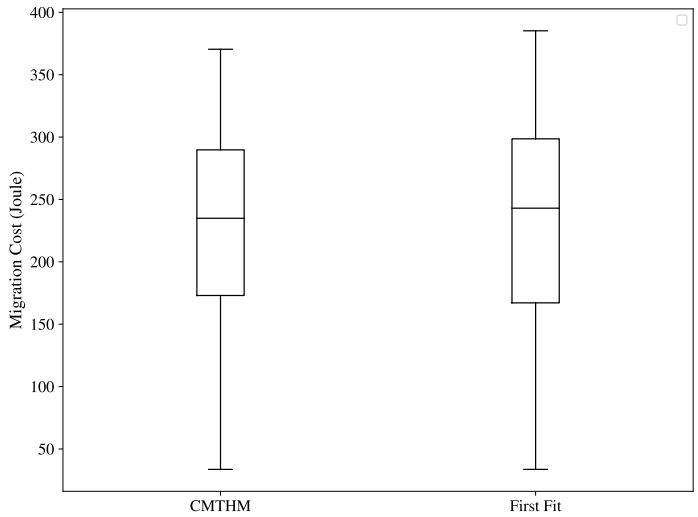


Figure 6.3: Migration Plan Generation Comparison

In Figure 6.2, the migration cost values obtained by the random selected migrated blocks are shown in the form of box charts. According to the experiment result, the migration costs obtained by the greedy algorithm are much lower than the random selected migrated blocks for each frequency vector. Before the block selection phase, the total amount of migrated workloads are determined. Therefore, the objective in block migration phase is to generate a migrated block set which satisfies the migrated workload constraint, and produces minimum migrated block sizes. In Greedy Algorithm, the blocks are sorted and selected by the ratio between their throughput and block size. Therefore, the blocks with higher throughputs but smaller block sizes are selected as migrated blocks, which maximizes the sum of kept block sizes and minimizes the migration cost in the end.

Another phenomenon of this experiment is that the migration cost does not depend on the frequency vectors, since the migration costs are close for different frequency vector. The reason for this situation is that the frequency vectors are given by Frequency Selection Approach in which the frequency vector is produced to minimize the energy consumption of the time window. For each frequency vector, the provided capacities of the nodes are close to their workloads, thus the migrated workloads are similar which leads to the migration costs being similar as well.

6.4.2 Migration Plan Generation Comparison

In migration plan generation phase, the migrated blocks are redistributed within the cluster. CMTHM is used to choose the destination for each migrated block. In this experiment, CMTHM is compared with First Fit algorithm. In First Fit, the nodes with extra capacities are sorted by the descending available capacity and the migrated blocks are sorted by descending the ratio between their block size and their throughput. Meanwhile, the blocks are migrated within their own rack first. If there is no space for the blocks, the remaining blocks are redistributed to other racks.

The test case used in this experiment is $(d20, GA)$. 1000 frequency vectors are used in this experiment which are obtained by Frequency Selection Approach. For each frequency vector, the migrated blocks are given by the Greedy Algorithm. Then, the blocks are migrated by both CMTHM and First Fit. In the end of migration, the migration costs are collected. The results are shown in Figure 6.3 and Table 6.4.

According to Figure 6.3, the range of the migration cost given by CMTHM is slightly lower than the results given by First Fit. The migration cost for migrating a block between racks is slightly larger than the migration cost for migrating a block within its rack according to Table 3.6 . Both algorithms try to place the migrated blocks in their own racks first. However, because of the exchange mechanism, CMTHM can place more migrated block within their own racks, which leads to lower migration cost. Among all the results, 83.8% of results given by CMTHM are lower than the results given by First Fit. In CMTHM, in order to improve the profit, the items are rearranged and swapped after the First Fit (first step of CMTHM). In most of the cases, these operations improve the profit of kept items, however occasionally they do not. In general, CMTHM is better than First Fit. However, the differences between the migration costs given by both algorithm are not high. The reason for that is their frequency vectors are given by Frequency Selection Approach. The frequency vectors are suboptimal to the current time window for minimizing the energy consumption. Therefore, the capacities of the nodes are limited, which gives less optimization space for minimizing the migration cost.

A given frequency vector can provide enough resources to execute the workload, however, it may not produce a valid migration plan because blocks are not continuous, i.e. a block cannot be split and put to multiple nodes and some blocks may not be placed. In our experiment, 1000 frequency vector are used, the amount of valid solutions produced by both algorithms are shown in Table 6.4. Compared with First Fit, CMTHM can produce more valid solutions because of the exchange mechanism which uses the size of each knapsack more efficiently.

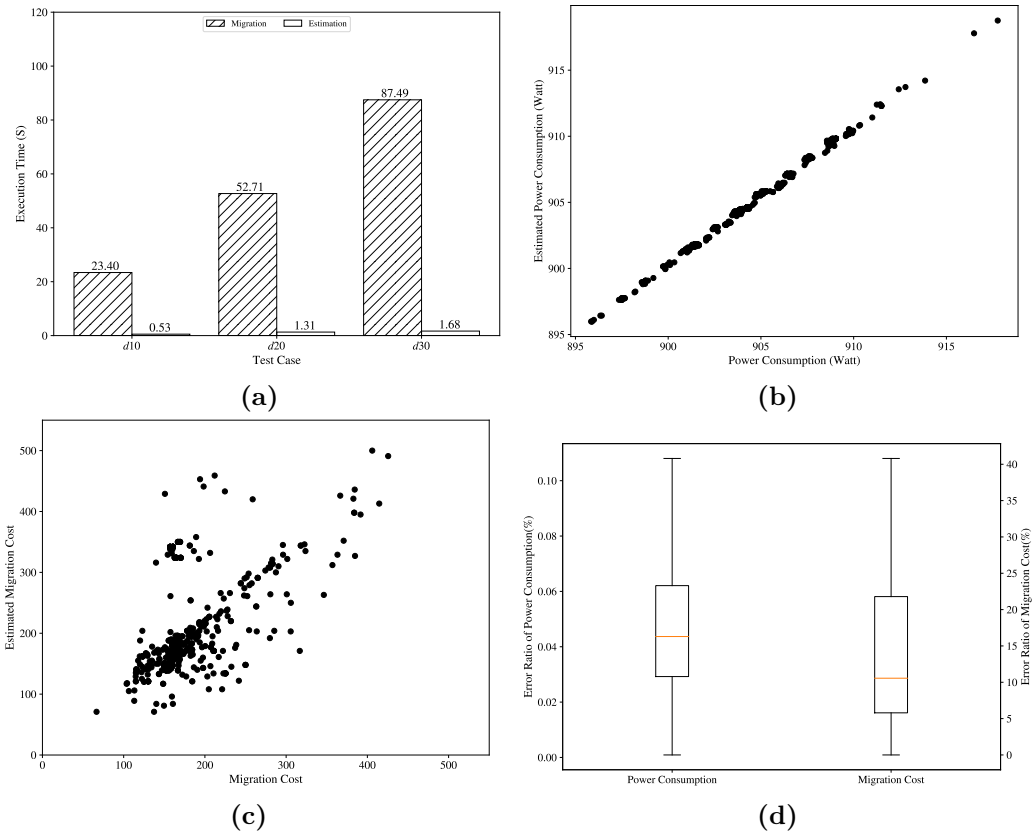


Figure 6.4: Evaluation Result of Migration Cost Estimation Approach

6.4.3 Estimation Approach Evaluation

In order to reduce the complexity for computing the migration cost and the power consumption in the frequency selection process, an estimation approach is proposed in this work. In this section, the approach is evaluated in 4 aspects.

At first, the performance of the estimation approach is evaluated. In this experiment, test cases $(d10, GA)$, $(d20, GA)$, $(d30, GA)$ are used. For all the test cases, the power consumption and the migration cost are collected by both the estimation approach and the migration process. The processes are executed 1000 times for both approaches and the total execution time is collected and shown by Figure 6.4a. The execution time for the cases solved by the migration process is denoted as $T(\text{TestCase}, \mathcal{M})$, and correspondingly the execution time for the cases solved by the estimation approach is denoted as $T(\text{TestCase}, \mathcal{E})$. With the increase of block amount, the execution time increases as well for both approaches. The optimization ratio of estimation approach for each case is denoted by Equation 6.1. $O(d10, GA) = 97.7\%$,

$O(d20, GA) = 97.5\%$, $O(d30, GA) = 98.0\%$. Therefore by means of the estimation, the average improvement is 97.7%. The reason for the improvement is that in the estimation approach, only the Migration Selection is required. In contrast, there are Migration Selection and Migration Plan Generation in the migration process.

$$O(\text{TestCase}) = 1 - \frac{T(\text{TestCase}, \mathcal{E})}{T(\text{TestCase}, \mathcal{M})} \tag{6.1}$$

With the introduction of estimation approach, the performance of the frequency vector evaluation is improved. In this experiment, the relationship between the power consumption (the migration cost) and the estimated power consumption (the estimated migration cost) is evaluated. Test case $(d20, GA)$ is used. 1000 cases are generated with the configuration of $(d20, GA)$. For each case, a frequency vector is selected by the frequency selection approach and the corresponding values are collected. The results are shown by 6.4b and 6.4c. Figure 6.4b shows the relationship between the power consumption and the estimated power consumption. Figure 6.4c shows the relationship between the migration cost and the estimated migration cost. Generally, both values are positive relative. With the increment of power consumption (migration cost), the estimated power consumption (the estimated migration cost) increases.

$$\begin{aligned} \text{Error Ratio of Power Consumption} &= \frac{p^{max} - p(s^*)}{p(s^*)} \\ \text{Error Ratio of Migration Cost} &= \frac{mc^{max} - mc(\widetilde{s})}{mc(\widetilde{s})} \end{aligned} \tag{6.2}$$

Figure 6.4d shows the error ratios for the estimated power consumption and the estimated migration cost, which are calculated by Equation 6.2. The median error ratios are 0.04% for estimated power consumption and 20.10% for estimated migration cost respectively. According to Equation 3.5, the power consumption is obtained by frequency vector and the workload. In Frequency Selection Approach, the frequency vector is given by the workload. In the estimation approach, the estimated power consumption is given by the worst workload assignment under the frequency vector. Thus, the estimated power consumption is more stable. However, in contrast, the migration cost depends on the migrated block selection and the migration plan generation. In this work, the estimation migration cost is obtained by migrated blocks and the relaxation approach. Therefore the error occurs for the reason that there is difference migrating a block between racks and migrating a block within its own rack. The upper bound of migration cost is obtained using a worst case of migration plan.

In order to avoid the influence of the errors, within the frequency selection process, the frequency selection approach is applied multiple times to generate several candidates. Afterwards, migration process is applied to candidates and the solution with the minimum energy consumption can be chosen.

6.5 Summary

In this chapter, the details of migration approach is introduced. The objective of migration approach in our proposal is to generate a migration plan in minimal migration cost manner. In migration execution phase, a live migration technique from Sudipto Das *et al.* [22] is adopted. The objective of block selection phase is to generate the migration block set for each node that minimizes the total size of migrated blocks. Consider the performance obstacle of the algorithm, a greedy selection algorithm is applied. In migration plan generation phase, the objective is to obtain a migration plan which reduces the total migration cost under given migrated block set. A CMTHM algorithm is used in this phase. By means of the migration process, a migration plan can be obtained in polynomial time. Meanwhile a migration cost estimation approach is provided to evaluate the given frequency configuration quickly. The experimental results show that by means of the migration approach a migration plan which minimizes migration cost under a given frequency configuration can be obtained. Using estimation approach, the performance of evaluating a frequency configuration is improved up to 97.7%.

Chapter 7

Conclusion and Perspectives

This thesis was dedicated to propose energy-efficient resource provisioning method for cloud database system. In our approach, we mainly took advantage of DVFS technique to allocate resources by workload prediction. Besides, a migration approach is proposed to further improve the energy efficiency within cloud database systems. Here, we summarize our findings and point out our contributions. In the end, the perspectives of this research is presented.

7.1 Conclusion

Because of the increment of energy usage within all IT area and the large energy bills, energy efficiency has been an emerging concern within all data centers and cloud service vendors. Cloud database as a big component within data centers, the energy efficiency of the system has drawn a lot of attention. However, the energy waste is still a big issue within cloud database systems because of resource provisioning. In this thesis, we presented our research on energy-efficient resource provisioning for cloud database system. The general idea is to use DVFS technique to allocate resources for the system according to its workload prediction, namely a frequency selection approach is introduced to improve the energy efficiency of the system. Also, a migration approach is proposed to further improve the energy efficiency. We conclude this thesis as follows.

First of all, we took Cassandra as our example, and the energy efficiency of cloud database systems under different frequency settings is analyzed by a benchmark experiment. The system under a given frequency setting has a maximum throughput value. Meanwhile, the energy efficiency of the system has correlation with its throughput. Therefore, to achieve higher frequency, we need to keep the system reach its maximum throughput. Based on the

above conclusions, a generic frequency selection model is proposed for cloud systems to cope with energy-efficient resource provisioning problem, in which we give the definition of time window based energy efficiency of a cloud system, and the method to estimate the energy usage under a given frequency setting. Then, the generic model is specialized for cloud database systems by redefining the key concepts.

Secondly, a frequency selection approach for bounded problem is proposed, in which the frequency selection problem is treated as power consumption bounded problem, $\mathbb{P}^b\mathbb{MC}^{min}$, and migration cost bounded problem, $\mathbb{MC}^b\mathbb{P}^{min}$. To solve them, a nonlinear programming algorithm and a multi-phases algorithm are proposed. Nonlinear programming algorithm can obtain the global optimal solution but has a poor scalability to large scale problems. In contrast, multi-phases algorithm has a good scalability for large scale problems by means of approximation method. The experimental results show that the multi-phases algorithm has great scalability which can be applied to a cloud database system deployed on 70 nodes. Meanwhile by means of the approach, it can save up to 21.5% of the energy consumption of the running time.

Thirdly, a frequency selection approach for optimization problem is proposed, in which we try to minimize the total energy consumption from the running time and from the migration process. The proposed algorithms include a genetic based algorithm and a monte carlo tree search based algorithm. Both algorithms have their advantages and disadvantages. The results of the experiment show that both algorithms have great scalability with reasonable accuracy (up to 99.9% and 99.6% for two algorithms respectively). Since the prediction errors are inevitable, the robustness of the algorithms is analyzed. The prediction errors might cause SLA violation and energy wasting. By means of tuning the capability tolerance factor, the trade off between SLA violation and energy wasting can be found. In practice, the error ratio should be kept in a lower range.

In the end, a migration approach based on the frequency selection circumstances is presented in this thesis to further improve the energy efficiency for cloud database systems. We separated the migration process into 3 phases: block selection, block migration and migration execution. We treated block selection problem as a single 0-1 knapsack problem, and a greedy algorithm is used to solve it. While, we treated block migration problem as a multi-knapsacks problem, and $CMTHM$ algorithm is presented. For migration execution, a live migration approach from [22] is adopted.

7.2 Perspectives

The work presented in this thesis is a primary step to improve the energy efficiency of cloud database system by providing energy-efficient resource provisioning method. As a perspectives, there are certainly remaining directions to improve our approach in the further. In this section, we give a few promising directions as follows.

- **Multiple Resources.** In this thesis, we only consider the influence of CPU resource. In term of power consumption estimation method, only CPU power is taken into consideration. In practice, more resource types can be considered in the approach, for example I/O resource, memory resource, network resource and so on. When considering the multiple resources, the resource provisioning will be more complex. In this direction, the influence of each resource need to be investigated first.

- **Multi-System Environment.** We assume there is only one cloud system is in execution. However, in the production environment, the environment is more complex. There are different cloud systems running in the cluster at same time. For example, cloud database system, remote filesystem, monitoring system and so on. Therefore, in multi-system environment, the resource provisioning has more challenges. At first, the resource isolating method should be investigated. Then, according to the character of each system, the particular resource assignment method can be made. In the end, a consolidation approach for all the systems should be proposed to reduce the resource over-provisioning and resource under-provisioning.

- **Migration Attention Mechanism.** In migration approach, we searched for a migration plan to minimize migration cost for certain Δt. However, considering the time sequences, there are more factors. For example, if a block is hotpot within two continues time windows, then the best strategy is that we do not change the location of the block in case any operation failures. If we consider multiple time windows, a better migration plan can be made.

Publications

1. **Chaopeng Guo**, Jean-Marc Pierson, Hui Liu, Jie Song. Frequency Selection Approach for Energy Aware Cloud Database. *IEEE ACCESS*, IEEE, 2018, DOI: 10.1109/ACCESS.2018.2885765.

2. **Chaopeng Guo**, Jean-Marc Pierson, Jie Song, Christina Herzog. Hot-N-Cold model for energy aware cloud databases. *Journal of Parallel and Distributed Computing*, Elsevier, Vol. 09, 10, 2018.

3. **Chaopeng Guo**, Jean-Marc Pierson. Frequency Selection Approach for Energy Aware Cloud Database (regular paper). *International Symposium on Computer Architecture and High-Performance Computing (SBAC-PAD 2018)*, Lyon, 24/09/18-27/09/18, IEEE Computer Society, p. 1-8, 09 2018.

4. Jie Song, **Chaopeng Guo**, Zhi Wang, Yichan Zhang, Ge Yu, Jean-Marc Pierson. HaoLap: A Hadoop based OLAP System for Big Data. *Journal of Systems and Software*, Elsevier, Vol. 4, 09, 2014.

Bibliography

[1] J. Morley, K. Widdicks, and M. Hazas, "Digitalisation, energy and data demand: The impact of Internet traffic on overall and peak electricity consumption," *Energy Research & Social Science*, vol. 38, pp. 128–137, Apr. 2018.

[2] M. Avgerinou, P. Bertoldi, and L. Castellazzi, "Trends in Data Centre Energy Consumption under the European Code of Conduct for Data Centre Energy Efficiency," *Energies*, vol. 10, p. 1470, Sept. 2017.

[3] A. Shehabi, S. Smith, D. Sartor, R. Brown, M. Herrlin, J. Koomey, E. Masanet, N. Horner, I. Azevedo, and W. Lintner, "United States Data Center Energy Usage Report," Tech. Rep. LBNL–1005775, 1372902, June 2016.

[4] W. Van Heddeghem, S. Lambert, B. Lannoo, D. Colle, M. Pickavet, and P. Demeester, "Trends in worldwide ICT electricity consumption from 2007 to 2012," *Computer Communications*, vol. 50, pp. 64–76, Sept. 2014.

[5] A. Andrae and T. Edler, "On Global Electricity Usage of Communication Technology: Trends to 2030," *Challenges*, vol. 6, pp. 117–157, Apr. 2015.

[6] J. Malmodin, Å. Moberg, D. Lundén, G. Finnveden, and N. Lövehagen, "Greenhouse Gas Emissions and Operational Electricity Use in the ICT and Entertainment & Media Sectors," *Journal of Industrial Ecology*, vol. 14, pp. 770–790, Oct. 2010.

[7] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: State-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, pp. 7–18, May 2010.

[8] M. Zakarya and L. Gillam, "Energy efficient computing, clusters, grids and clouds: A taxonomy and survey," *Sustainable Computing: Informatics and Systems*, vol. 14, pp. 13–33, June 2017.

[9] Mehul Nalin Vora, "Hadoop-HBase for large-scale data," in *Proceedings of 2011 International Conference on Computer Science and Network Technology*, (Harbin, China), pp. 601–605, IEEE, Dec. 2011.

[10] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, "Hive - a petabyte scale data warehouse using Hadoop," in *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, (Long Beach, CA, USA), pp. 996–1005, IEEE, 2010.

[11] J. Han, H. E, G. Le, and J. Du, "Survey on NoSQL database," in *2011 6th International Conference on Pervasive Computing and Applications*, (Port Elizabeth, South Africa), pp. 363–366, IEEE, Oct. 2011.

[12] V. Anand and C. M. Rao, "MongoDB and Oracle NoSQL: A technical critique for design decisions," in *2016 International Conference on Emerging Trends in Engineering, Technology and Science (ICETETS)*, (Pudukkottai, India), pp. 1–4, IEEE, Feb. 2016.

[13] D. Gu, "Cassandra at Instagram 2016," 2016.

[14] J. Song, T. Li, X. Liu, and Z. Zhu, "Comparing and Analyzing the Energy Efficiency of Cloud Database and Parallel Database," in *Advances in Computer Science, Engineering & Applications: Proceedings of the Second International Conference on Computer Science, Engineering & Applications (ICCSEA 2012), May 25-27, 2012, New Delhi, India. Volume 2* (D. C. Wyld, J. Zizka, and D. Nagamalai, eds.), pp. 989–997, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[15] W. Lang, S. Harizopoulos, J. M. Patel, M. A. Shah, and D. Tsirogiannis, "Towards energy-efficient database cluster design," *Proceedings of the VLDB Endowment*, vol. 5, pp. 1684–1695, July 2012.

[16] G.-W. You, S.-W. Hwang, and N. Jain, "Ursa: Scalable Load and Power Management in Cloud Storage Systems," *ACM Transactions on Storage*, vol. 9, pp. 1–29, Mar. 2013.

[17] D. Schall and T. Härder, "WattDB - A Journey towards Energy Efficiency," *Datenbank-Spektrum*, vol. 14, pp. 183–198, Nov. 2014.

[18] W. Vogels, "Eventually consistent," *Communications of the ACM*, vol. 52, p. 40, Jan. 2009.

[19] H.-E. Chihoub, S. Ibrahim, Y. Li, G. Antoniu, M. S. Perez, and L. Bouge, "Exploring Energy-Consistency Trade-Offs in Cassandra Cloud Storage

System," in *2015 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, (Florianopolis, Brazil), pp. 146–153, IEEE, Oct. 2015.

[20] A. P. Florence, V. Shanthi, and C. B. S. Simon, "Energy Conservation Using Dynamic Voltage Frequency Scaling for Computational Cloud," *The Scientific World Journal*, vol. 2016, pp. 1–13, 2016.

[21] S. Ibrahim, T.-D. Phan, A. Carpen-Amarie, H.-E. Chihoub, D. Moise, and G. Antoniu, "Governing energy consumption in Hadoop through CPU frequency scaling: An analysis," *Future Generation Computer Systems*, vol. 54, pp. 219–232, Jan. 2016.

[22] S. Das, S. Nishimura, D. Agrawal, and A. E. Abbadi, "Live Database Migration for Elasticity in a Multitenant Database for Cloud Platforms," Tech. Rep. 2010-09, University of California, Santa Barbara, USA, June 2010.

[23] M. Amiri and L. Mohammad-Khanli, "Survey on prediction models of applications for resources provisioning in cloud," *Journal of Network and Computer Applications*, vol. 82, pp. 93–113, Mar. 2017.

[24] K. Li, H. Zheng, and J. Wu, "Migration-Based Virtual Machine Placement in Cloud Systems," in *2013 IEEE 2nd International Conference on Cloud Networking (CloudNet)*, pp. 83–90, Nov. 2013.

[25] A. Krioukov, C. Goebel, S. Alspaugh, Y. Chen, D. Culler, and R. Katz, "Integrating Renewable Energy Using Data Analytics Systems: Challenges and Opportunities," *Bulletin of the IEEE Computer Society Technical Committee*, vol. 34, pp. 3–11, Mar. 2011.

[26] Wikipedia, "Cloud Database." https://bit.ly/2NhMaqZ, 2019.

[27] M. Ronstrõm and L. Thalmann, "MySQL Cluster Architecture Overview," tech. rep., Apr. 2004.

[28] A. Verbitski, X. Bao, A. Gupta, D. Saha, M. Brahmadesam, K. Gupta, R. Mittal, S. Krishnamurthy, S. Maurice, and T. Kharatishvili, "Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases," in *Proceedings of the 2017 ACM International Conference on Management of Data - SIGMOD '17*, (Chicago, Illinois, USA), pp. 1041–1052, ACM Press, 2017.

[29] M. Vallath, *Oracle Real Application Clusters*. Elsevier Digital Press, 2014.

[30] A. B. M. Moniruzzaman and S. A. Hossain, "NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison," *International Journal of Database Theory and Application*, vol. 6, no. 4, p. 14, 2013.

[31] Edlich, "NoSQL Database." http://nosql-database.org, 2019.

[32] DATASTAX, "CQL reference — CQL for Cassandra 3.0." https://bit.ly/2O2SzXC, 2019.

[33] R. Kumar, N. Gupta, Shilpi Charu, Somya Bansal, and K. Yadav, "Comparison of SQL with HiveQL," *International Journal for Research in Technological Studies*, vol. 1, pp. 28–30, Aug. 2014.

[34] K. Grolinger, W. A. Higashino, A. Tiwari, and M. A. Capretz, "Data management in cloud environments: NoSQL and NewSQL data stores," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 2, no. 1, p. 22, 2013.

[35] A. Pavlo and M. Aslett, "What's Really New with NewSQL?," *ACM SIGMOD Record*, vol. 45, pp. 45–55, Sept. 2016.

[36] M. Stonebraker and A. Weisberg, "The VoltDB Main Memory DBMS," *IEEE Technical Committee on Data Engineering*, vol. 36, pp. 21–27, 2013.

[37] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford, "Spanner: Google's Globally Distributed Database," *ACM Transactions on Computer Systems*, vol. 31, p. 22, Aug. 2013.

[38] L.-Y. Yuan, L. Wu, J.-H. You, and Y. Chi, "A Demonstration of Rubato DB: A Highly Scalable NewSQL Database System for OLTP and Big Data Applications," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD '15*, (Melbourne, Victoria, Australia), pp. 907–912, ACM Press, 2015.

[39] Apache Software Foundation, "Apache Cassandra." http://cassandra.apache.org/, 2019.

[40] S. Anand, P. Singh, and B. M. Sagar, "Working with Cassandra Database," in *Information and Decision Sciences:*, Advances in Intelligent Systems and Computing, pp. 531–538, 2018.

[41] W. Jing, D. Tong, G. Chen, C. Zhao, and L. Zhu, "An optimized method of HDFS for massive small files storage," *Computer Science and Information Systems*, vol. 15, no. 3, pp. 533–548, 2018.

[42] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, p. 35, Apr. 2010.

[43] C. Esposito, A. Castiglione, F. Palmieri, and M. Ficco, "Improving the gossiping effectiveness with distributed strategic learning (Invited paper)," *Future Generation Computer Systems*, vol. 71, pp. 221–233, June 2017.

[44] R. Aniceto, R. Xavier, V. Guimarães, F. Hondo, M. Holanda, M. E. Walter, and S. Lifschitz, "Evaluating the Cassandra NoSQL Database Approach for Genomic Data Persistency," *International Journal of Genomics*, vol. 2015, pp. 1–7, 2015.

[45] Z. Chen, S. Yang, S. Tan, G. Zhang, and H. Yang, "Hybrid Range Consistent Hash Partitioning Strategy – A New Data Partition Strategy for NoSQL Database," in *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, (Melbourne, Australia), pp. 1161–1169, IEEE, July 2013.

[46] V. N. Volkova, L. V. Chemenkaya, E. N. Desyatirikova, M. Hajali, A. Khodar, and A. Osama, "Load balancing in cloud computing," in *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, (Moscow), pp. 387–390, IEEE, Jan. 2018.

[47] E. Casalicchio, L. Lundberg, and S. Shirinbad, "An Energy-Aware Adaptation Model for Big Data Platforms," in *2016 IEEE International Conference on Autonomic Computing (ICAC)*, (Wuerzburg, Germany), pp. 349–350, IEEE, July 2016.

[48] X. You, Y. Li, M. Zheng, C. Zhu, and L. Yu, "A Survey and Taxonomy of Energy Efficiency Relevant Surveys in Cloud-Related Environments," *IEEE Access*, vol. 5, pp. 14066–14078, 2017.

[49] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah, "Analyzing the energy efficiency of a database server," in *Proceedings of the 2010 International Conference on Management of Data - SIGMOD '10*, (Indianapolis, Indiana, USA), p. 231, ACM Press, 2010.

[50] D. Schall, *Energy Efficiency in Database Systems*. PhD thesis, Feb. 2015.

[51] J. Song, T. Li, Z. Wang, and Z. Zhu, "Study on energy-consumption regularities of cloud computing systems by a novel evaluation model," *Computing*, vol. 95, pp. 269–287, Apr. 2013.

[52] TPC, "TPC-Energy." http://www.tpc.org/tpc_energy/, 2019.

[53] Wikipedia, "Power distribution unit." https://bit.ly/2GdutWt, Sept. 2018.

[54] A.-C. Orgerie, M. D. de Assuncao, and L. Lefevre, "A survey on techniques for improving the energy efficiency of large-scale distributed systems," *ACM Computing Surveys*, vol. 46, pp. 1–31, Mar. 2014.

[55] M. Dayarathna, Y. Wen, and R. Fan, "Data Center Energy Consumption Modeling: A Survey," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 732–794, Firstquarter 2016.

[56] R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shojafar, A. I. A. Ahmed, S. A. Madani, K. Saleem, and J. J. Rodrigues, "A survey on energy estimation and power modeling schemes for smartphone applications: Energy Estimation and Power Modeling Schemes for Smartphone Apps," *International Journal of Communication Systems*, vol. 30, p. e3234, July 2017.

[57] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. Pearson Education India, 2015.

[58] Y. C. Lee and A. Y. Zomaya, "Energy Conscious Scheduling for Distributed Computing Systems under Different Operating Conditions," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, pp. 1374–1381, Aug. 2011.

[59] T. Mudge, "Power: A first-class architectural design constraint," *Computer*, vol. 34, pp. 52–58, Apr. 2001.

[60] X. Qi and D.-K. Zhu, "Energy Efficient Block-Partitioned Multicore Processors for Parallel Applications," *Journal of Computer Science and Technology*, vol. 26, pp. 418–433, May 2011.

[61] J. Liu and J. Guo, "Energy efficient scheduling of real-time tasks on multicore processors with voltage islands," *Future Generation Computer Systems*, vol. 56, pp. 202–210, Mar. 2016.

[62] M. Elnozahy, M. Kistler, and R. Rajamony, "Energy-Efficient Server Clusters," in *Power-Aware Computer Systems* (B. Falsafi and T. N. Vijaykumar, eds.), pp. 179–197, Springer Berlin Heidelberg, 2003.

[63] X. Fan, W.-D. Weber, and L. A. Barroso, "Power Provisioning for a Warehouse-sized Computer," in *In Proceedings of the ACM International Symposium on Computer Architecture*, (San Diego, CA, USA), pp. 13–23, June 2007.

[64] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, and A. V. Vasilakos, "Cloud Computing: Survey on Energy Efficiency," *ACM Computing Surveys*, vol. 47, pp. 1–36, Dec. 2014.

[65] D. Careglio, G. Da Costa, and S. Ricciardi, "Hardware Leverages for Energy Reduction in Large-Scale Distributed Systems," in *Large-Scale Distributed Systems and Energy Efficiency* (J.-M. Pierson, ed.), pp. 17–40, Hoboken, NJ, USA: John Wiley & Sons, Inc, Apr. 2015.

[66] Q. Shaheen, M. Shiraz, S. Khan, R. Majeed, M. Guizani, N. Khan, and A. M. Aseere, "Towards Energy Saving in Computational Clouds: Taxonomy, Review, and Open Challenges," *IEEE Access*, vol. 6, pp. 29407–29418, 2018.

[67] M.-G. Jung, S.-A. Youn, J. Bae, and Y.-L. Choi, "A Study on Data Input and Output Performance Comparison of MongoDB and PostgreSQL in the Big Data Environment," in *2015 8th International Conference on Database Theory and Application (DTA)*, (Jeju Island, South Korea), pp. 14–17, IEEE, Nov. 2015.

[68] P. Boncz, T. Neumann, and O. Erling, "TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark," in *Performance Characterization and Benchmarking* (D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, R. Nambiar, and M. Poess, eds.), vol. 8391, pp. 61–76, Cham: Springer International Publishing, 2014.

[69] C. Bear, A. Lamb, and N. Tran, "The vertica database: SQL RDBMS for managing big data," in *Proceedings of the 2012 Workshop on Management of Big Data Systems - MBDS '12*, (San Jose, California, USA), p. 37, ACM Press, 2012.

[70] K. S. Sangeetha and P. Prakash, "Big Data and Cloud: A Survey," in *Artificial Intelligence and Evolutionary Algorithms in Engineering Systems* (L. P. Suresh, S. S. Dash, and B. K. Panigrahi, eds.), pp. 773–778, Springer India, 2015.

[71] S. Harizopoulos, V. Liang, D. J. Abadi, and S. Madden, "Performance Tradeoffs in Read-optimized Databases," in *Proceedings of the 32Nd International Conference on Very Large Data Bases*, VLDB '06, pp. 487–498, VLDB Endowment, 2006.

[72] D. Schall and T. Härder, "Approximating an Energy-Proportional DBMS by a Dynamic Cluster of Nodes," in *Database Systems for Advanced Applications* (S. S. Bhowmick, C. E. Dyreson, C. S. Jensen, M. L. Lee, A. Muliantara, and B. Thalheim, eds.), pp. 297–311, Springer International Publishing, 2014.

[73] G. Han, W. Que, G. Jia, and L. Shu, "An Efficient Virtual Machine Consolidation Scheme for Multimedia Cloud Computing," *Sensors*, vol. 16, p. 246, Feb. 2016.

[74] S. Savinov and K. Daudjee, "Dynamic database replica provisioning through virtualization," in *Proceedings of the Second International Workshop on Cloud Data Management - CloudDB '10*, (Toronto, ON, Canada), p. 41, ACM Press, 2010.

[75] H. Chen, X. Zhu, H. Guo, J. Zhu, X. Qin, and J. Wu, "Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment," *Journal of Systems and Software*, vol. 99, pp. 20–35, Jan. 2015.

[76] B. Subramaniam and W.-c. Feng, "On the Energy Proportionality of Distributed NoSQL Data Stores," in *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation* (S. A. Jarvis, S. A. Wright, and S. D. Hammond, eds.), pp. 264–274, Springer International Publishing, 2015.

[77] L. Yu, F. Teng, and F. Magoules, "Node Scaling Analysis for Power-Aware Real-Time Tasks Scheduling," *IEEE Transactions on Computers*, vol. 65, pp. 2510–2521, Aug. 2016.

[78] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *Proceeding HotCloud'10 Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, (Boston, MA, USA), p. 7, USENIX, June 2010.

[79] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, and O. Richard, "Grid'5000: A large scale and highly reconfigurable grid experimental testbed," in *The 6th IEEE/ACM International*

Workshop on Grid Computing, 2005., (Seattle, WA, USA), p. 8 pp., IEEE, 2005.

[80] Grid5000, "Power Monitoring Devices." https://bit.ly/2E2WHSR, 2019.

[81] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM Symposium on Cloud Computing - SoCC '10*, (Indianapolis, Indiana, USA), pp. 143–154, ACM Press, 2010.

[82] D. Borgetto, H. Casanova, G. Da Costa, and J.-M. Pierson, "Energy-aware service allocation," *Future Generation Computer Systems*, vol. 28, pp. 769–779, May 2012.

[83] Wikipedia, "Nonlinear Programming." https://bit.ly/2SP6A0y, Dec. 2018.

[84] B. Meindl and M. Templ, "Analysis of commercial and free and open source solvers for linear optimization problems," Tech. Rep. CS-2012-1, Vienna, Austria, Mar. 2012.

[85] Wikipedia, "Stars and bars (combinatorics)." https://bit.ly/1h49E13, Dec. 2018.

[86] G. Das, "Top-k Algorithms and Applications," in *Database Systems for Advanced Applications* (X. Zhou, H. Yokota, K. Deng, and Q. Liu, eds.), pp. 789–792, Springer Berlin Heidelberg, 2009.

[87] Wikipedia, "Zipf's law." https://bit.ly/2l6RJgY, 2017.

[88] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1st ed., 1989.

[89] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, pp. 1–43, Mar. 2012.

[90] C. Curino, E. Jones, Y. Zhang, S. Madden, and E. Wu, "Relational Cloud: The Case for a Database Service," Tech. Rep. MIT-CSAIL-TR-2014-014, MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, USA, Mar. 2010.

[91] S. Martello, D. Pisinger, and P. Toth, "Dynamic Programming and Strong Bounds for the 0-1 Knapsack Problem," *Management Science*, vol. 45, pp. 414–424, Mar. 1999.

[92] Wikipedia, "Pseudo-polynomial time." https://bit.ly/2HFia7G, Mar. 2019. Page Version ID: 832590030.

[93] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990.