

國立臺灣大學管理學院資訊管理學系

碩士論文

Department of Information Management

College of Management

National Taiwan University

Master Thesis

於雲端接取網路中以最佳化為基礎使收益最大化之工
作分配策略

An Optimization-based Task Allocation Strategy for
Maximization of Revenue in Cloud Radio Access
Networks

簡伯銓

Po-Chuan Chien

指導教授：林永松博士

Advisor: Yeong-Sung Lin, Ph.D.

中華民國 106 年 7 月

July, 2017







誌謝

時間過的很快，轉眼間兩年間就過去了，而這篇論文能夠順利完成，在此要感謝許多曾經幫助過我的人。

首先必須謝謝林永松老師，在建構模型遇到困難時給予了許多協助，同時也在情境模擬上給了我許多方向。除了教學上的熱忱以及數理的專業知識之外，老師自然而然散發出來的那份待人處世的那份風度，更是讓我在往後的生活裡能夠持續去效法及實踐的。接著要謝謝蔡益坤老師，謝謝老師當初給了我這份機會讓我接任資料結構和演算法的助教，在這些課程中，除了讓我有機會可以重新溫習大學所學之外，藉著實習課的機會，也對教學技巧上有更進一步的提升，並對自己一些往先可能不甚熟悉的主題能更加地瞭解。同時也感謝溫演福委員、林宜隆委員、莊東穎委員的寶貴意見，使得本論文能更加完備。

此外謝謝研究室的邱漢及演福學長，不時關心我們的論文進度，讓我們可以在規定的期限前順利完成論文的進度，同時也給了我許多可以著手的方向和模型上的意見，使得考量能夠更加全面，以及邱漢學長準備的心靈雞湯時間，讓我們除了學術方面以外，在心靈上也能得到提升。也謝謝研一的士庭、鑫宏和欣宜，有了你們，實驗室的生活變的更加有趣，未來的一年也祝你們都能順利完成碩士學位。這裡要

特別謝謝士庭，在許多個令人難過和崩潰的時候，多虧有你的支持和鼓勵，才讓我更有勇氣面對生活中的各種困難。

接著要謝謝 B02 的專題同學、B03 的資料結構和演算法的同學、B04 的離散數學和資料結構的同學，還有外系一起來修課被我帶到族繁不及備載的同學們，謝謝有你們的陪伴，讓我能在研究所這段期間，拓展了更多生活圈、認識了更多的人，能夠帶到你們這些同學，著實替我的研究所生活增色不少。不論你們以後往後的日子中是否還會用到課堂上學到的這些內容（笑），希望你們之後畢業後也都能夠有個好的出路，在此祝福你們，也希望以後也能在各行各業中再遇到你們大家。

最後謝謝辛苦和家人，因為有你們一路的支持，才能有今天的我，謝謝你們。

簡伯銓謹誌

于國立臺灣大學資訊管理研究所

中華民國一百零六年八月



摘要

為了應付節節攀升的網路流量，網路系統也必須與時俱進，而第五代移動通信系統（5G）正是為了解決此項問題而提出的目標。在5G之中，雲端接取網路因為將基帶處理器與無線寬頻頭端設備分離並集中於中央統一管理，因而可以更有效率的使用計算資源。在本篇論文之中，我們基於拉格朗日鬆弛法，輔以裝箱問題、排程、負載均衡算法，以及在營運時將面臨到的資源限制，提出了一套應用於雲端接取網路的工作分配策略來最大化網路營運商的收益。本篇論文亦模擬了各種情況，並提供實驗模擬數據來說明本方法確能增加網路營運商的利潤。

關鍵詞：第五代移動通信系統、無線接取網路、工作分配策略、最佳化、拉格朗日鬆弛法





Abstract

Due to the rapid increase in the network traffic load, the Internet service system must improve to meet the requirement. Fifth generation (5G) mobile networks aim to deal with this problem, and cloud radio access networks (C-RANs) is a popular approach to this goal. In a C-RAN, baseband processing units are centralized into a pool, which allows us to have a better resource utilization. In this thesis, we use the Lagrangian relaxation method combined with bin packing, scheduling, and traffic shaping to derive a task allocation strategy in a C-RAN that tries to maximize the profit of a network operator who may face multiple kinds of constraints during its operation. After that, we will present the experimental results to show the effectiveness of our proposed method.

Keywords: 5G, cloud radio access network (C-RAN), task allocation strategy, optimization, Lagrangian relaxation





Contents

誌謝	iii
摘要	v
Abstract	vii
1 Introduction	1
1.1 Background	1
1.2 Motivation	3
1.3 Thesis structure	6
2 Related Work	7
2.1 Bin packing	7
2.2 Task scheduling	9
2.3 Load balancing	10
3 Mathematical Model	13
3.1 Problem description	13



3.2	Problem formulation	14
3.3	Task assignment constraints	16
3.4	Capacity constraints	19
3.5	Server switch constraints	20
4	Solution Approach	23
4.1	Lagrangian relaxation method	23
4.2	Lagrangian relaxation objective function and constraints	25
4.3	Task assignment subproblem - SP ₁	27
4.4	Server power on subproblem - SP ₂	28
4.5	Server re-power on related subproblem - SP ₃	30
4.6	Block subproblem - SP ₄	32
4.7	Auxiliary subproblem - SP ₅	33
4.8	Getting primal feasible solution	33
5	Computational Experiments	39
5.1	Experiment on task quantity	41
5.1.1	Uniform arrival	41
5.1.2	Bursty arrival	43
5.2	Experiment on fixed capacity with different number of servers	44
5.2.1	High capacity	45
5.2.2	Medium capacity	47

5.2.3	Low capacity	48
5.3	Experiment on processing time	50
5.4	Experiment on tolerance and waiting time	53
5.5	Experiment on task block penalty	54
5.6	Experiment on task revenue rate	56
5.7	Experiment on server cost rate	58
6	Conclusion and Future Work	61
6.1	Conclusion	61
6.2	Future work	62
	References	65







List of Figures

1.1	Different task allocation strategies	5
5.1	Uniform arrival time	42
5.2	Bursty arrival time	44
5.3	Capacity greater than requirement	46
5.4	Capacity approximately equal to requirement	48
5.5	Capacity less than requirement	50
5.6	Processing time	51
5.7	Different allowed buffer time	54
5.8	Different task penalty	55
5.9	Expected task revenue rate	57
5.10	Cost for turning a server on for an interval	58





List of Tables

1.1	Requirements of 5G	2
3.1	Given parameters	15
3.2	Decision variables	16
5.1	Improvement ratio for uniform arrival	43
5.2	Improvement ratio for bursty arrival	45
5.3	Improvement ratio for high capacity	47
5.4	Improvement ratio for medium	49
5.5	Improvement ratio for low capacity	49
5.6	Improvement ratio for processing time	52
5.7	Improvement ratio for buffer time	55
5.8	Improvement ratio for task block penalty	56
5.9	Improvement ratio for task revenue rate	57
5.10	Improvement ratio for server costs	59





Chapter 1

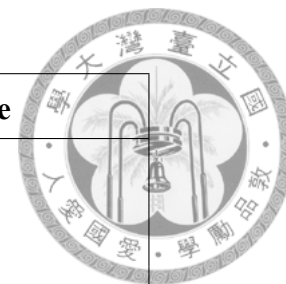
Introduction

1.1 Background

With the ubiquity of access to the Internet and the advance in communication technology, larger amount of traffic services, such as online video streaming, data transfer between servers, and mobile games, are wide usage. The numbers of cell phones, tablets, and other electronic devices are increasing at a high speed. In 2016, there were around 29 millions mobile communication users in Taiwan [1], which is approximately equal to 1.26 devices per person. Although such increase in the number of electronic devices has made life with high convenience, the network traffic load and required amount of processing tasks are growing at a dramatic rate and pose a great burden on the network services and required wireless transmission resource. It is estimated that by 2020, there will be around 11.6 billions of devices worldwide, and the traffic load per month will be 30.6 exabytes [2]. Authors of [3] also stated that the expected data transmission amount of fifth genera-

Table 1.1: Requirements of 5G

Requirement types	Desirable value
Reasonable latency	< 5 ms
High data rate and data volume	1 to 10 Gb per second
Long battery life	10 years
Scalability	Able to serve 300,000 users per AP



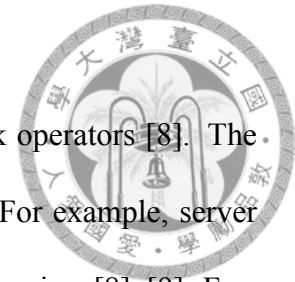
tion (5G) wireless services will be approximately 1,000 times of that than forth generation (4G) networks.

Overcoming such a demand becomes one of the most important challenges of 5G mobile networks. Osseiran et al. proposed some desired goals, which are summarized in Table 1.1, that 5G should meet [4]. The metrics require the system performance and operation strategies so this work will concentrate on the task assignment and resource allocation in the backhaul C-RAN systems.

Peng et al. had discussed heterogeneous networks (HetNets) and provided a possible solution for the heavy traffic load in 5G to achieve the QoS objective [5], [6]. Traditionally, in a HetNet, multiple base stations (BSs) are deployed to satisfy different needs of all kinds user equipments (UEs). A BS consists of a baseband processing unit (BBU), which is used for data transmission and processing, and a remote radio head (RRH), which takes the responsibility of signal process [7]. The BBU component is separated from the front BSs and is centralized in a *BBU pool*, while RRHs are still left non-centralized among all BSs. The resource is centralized and can be shared to serve on demand by cloud servers. The RRHs and C-RAN are connected through the fiber networks. The higher amount of resource can be centralized to support higher traffic load requirement to improve the

network resource utilization.

Such centralization brings high flexibility to the mobile network operators [8]. The centralization management mechanism reduce the operation costs. For example, server site rental fee or cooling energy will be less than un-centralized mechanism [8], [9]. Furthermore, centralization environment provides some directions to improve the utilization and performance. Several existing methods, such as bin packing [10], scheduling [11], and load balancing [12], require to be adopted in C-RAN. These studies had shown the centralization reduces capital expenditure (CAPEX) and operational expenditure (OPEX) [13] than the decentralized network environment and subsequently achieved energy efficiency (EE) [9], [14]. This is one of the reasons to handle resource allocation with non-distributed schemes to enhance the cost-benefit from the the network operators perspective.

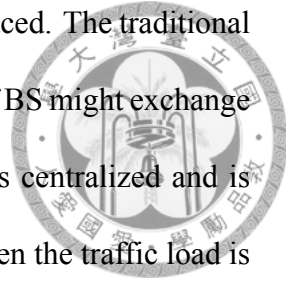


1.2 Motivation

Besides the cooling energy consumption and the site rental save, what matters more is the power consumption of the operation of all servers. Even though centralization may help lower possible cooling costs, the power consumption still poses a great problem to 5G operators, for that around 80% of the energy is consumed by BSs to operate a cellular network [15]. Therefore, EE is a great concern when operators host a network. In the past, when BBUs and RRHs are still tied together in a BS, the BSs must be turned on all the time to be ready to serve UEs; therefore, it creates energy waste if there is no service need.

One of the advantages to separate BBUs and RRHs is to build up the BBUs such

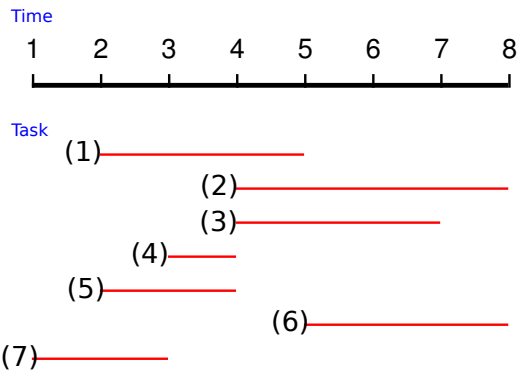
that the resource is efficiently used and process required power is reduced. The traditional resource handling method is individual processed by each BS. A set of BS might exchange message to observe high performance. Oppositely, the BBU pool is centralized and is controlled by a coordinator to assign the tasks in a set of servers. When the traffic load is low, a part of cloud servers can be suspended to save energy. Thus, it would be helpful to achieve higher EE and lower capital and operational expenditures once a cloud host is suspended and even turned off [14], [16]. When a server is turned off during off-peak hours, the energy consumption might save about 60% [5].



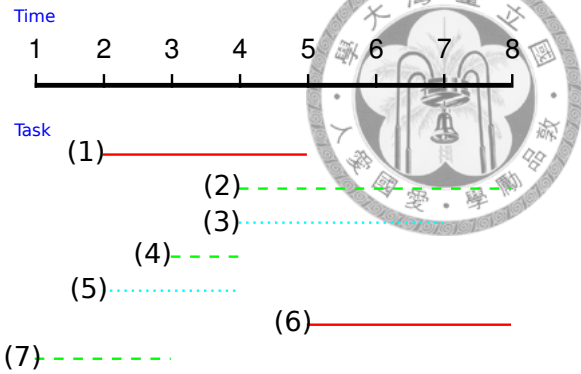
It can be easily inferred that higher number of idle servers results in higher amount of power consumption. Consequently, the strategic of this work is to increase the number of idle servers through a near-optimal task allocation scheme. To show the importance of task allocation strategy quality, we present a simple example below in Figure 1.1.

Figure 1.1A shows a scenario with 8 time intervals and 7 tasks. The black time scale above marks the flow of time, and the red lines below denoted from 1 to 7 stand for the task existence in the system. Task 1 appears from the time period 2 to 5, and tolerable time period of task 6 is from time 5 to 8. Furthermore, this work assumes that a server can only process one task at the same time, and every task is un-weighted. Figure 1.1B is one the optimal solutions for this situation. If we expect every task should be served, then we need to turn on three servers, denoted by red solid line, green dashed line, and cyan dotted line. Such optimum is not unique: if we swap task 2 with task 3, we can still get an optimal solution.

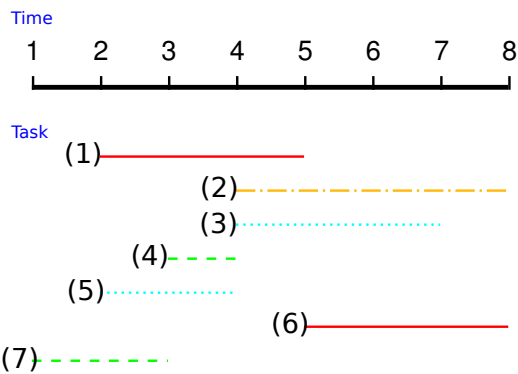
However, a bad task allocation mechanism may fail to assign the tasks properly, as shown in Figure 1.1C and 1.1D. Figure 1.1C shows a situation where the system turns on



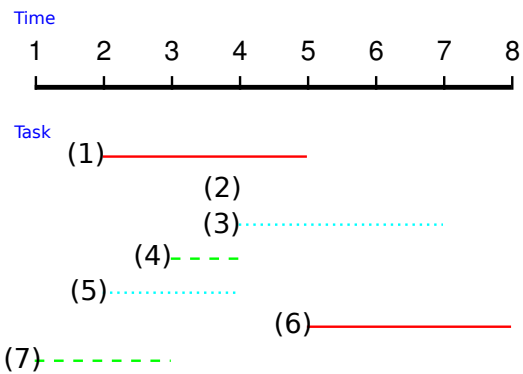
(A) A scenario with 8 intervals and 7 tasks



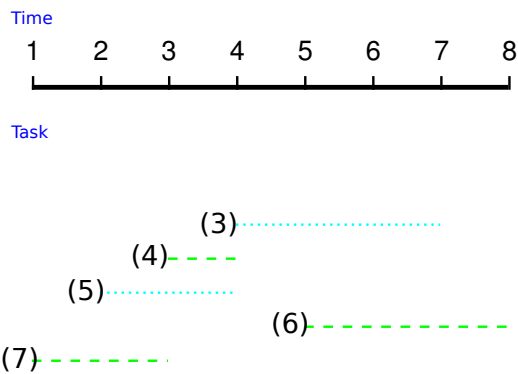
(B) When we have 3 servers, all tasks may enter the system



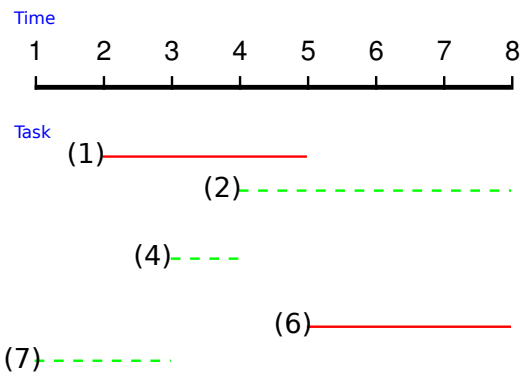
(C) Bad example: use 4 servers to serve the tasks



(D) Bad example: fails to serve the tasks while it is possible



(E) If we only have 2 servers, some tasks will be dropped

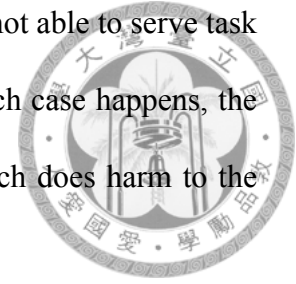


(F) Another choice of tasks if we have 2 servers

Figure 1.1: Different task allocation strategies

four servers to serve all the tasks (the fourth server is marked by an orange dashed dotted line). In this case, the power consumption rise and lowers the final profit. Figure 1.1D

demonstrates other worse task allocation result where the system is not able to serve task 2 while it is possible to do that, as shown in Figure 1.1B. When such case happens, the network operator may have to bear the service failure penalty, which does harm to the revenue.



Nevertheless, the resource is finite, and we cannot always have as many servers as we wish. When the resource is not sufficient, we have to make some trade-off between the tasks. Figure 1.1E and 1.1F show 2 different possible choices for operation. If we can only have 2 servers at most, we can choose to block task (1) and (2) (Figure 1.1E) or task (3) and (5) (Figure 1.1F).

Because properties vary from tasks to tasks, final profit may also differ if we make different decisions. In addition, pursuit of high profit is becoming more touch when there are more tasks, more servers, or when a server can process multiple tasks during one interval. Thus, a good task allocation strategy is indispensable, and this is also the goal of this thesis. In this thesis, we try to incorporate the bin packing, scheduling, and traffic shaping technique to design a task allocation strategy that may help network operators maximize the profit.

1.3 Thesis structure

The remaining part of this thesis is organized as below. Related work is listed in Chapter 2, and in Chapter 3, the mathematical model is described. After that, the solution approach is presented in Chapter 4. Experiments are explained in Chapter 5, and the conclusion is given in Chapter 6.



Chapter 2

Related Work

2.1 Bin packing

The bin packing problem (BPP) describes a situation where a set of various sizes of objects, which to be packed to bins later, and a set of bins exist, and the goal is to use a minimum number of bins to pack all the objects. BPP is proved as an NP-hard problem by [17]. Despite the hardness of this problem, some methods, such as first-fit (FF) and best-fit (BF), are provided by [17]. The first-fit algorithm packs the object to the first bin that can contain the object, while best-fit algorithm finds the bin with minimum residual capacity that can take the item.

These methods are improved by sorting the weights of items in descending order initially, denoted as FFD (first-fit descending) and BFD (best-fit descending), respectively. Both extension methods provide an approximation to the optimal arrangement. [18] gave a proof that FFD and BFD provide worse number of bins was $\frac{11}{9}OPT + 1$ bins, where

OPT is the optimum. The “descending” order is not the necessary condition to achieve optimization. The sorting order leads to observation of high solution such as that a set of tasks is sorted based on the designed weights according to the multipliers observed from Lagrangian relaxation subproblems.



Bin packing technique is adopted by [10] to study the traffic load and server capacity problem. A set of servers denotes the set of bins that packs a set of client requests, which denotes the items. This concept is adopted in this work.

Hu et al. modeled the problem as a 2-D bin packing problem that includes signal processing and bandwidth limits [19]. It shows that the first-fit and best-fit algorithms are still applicable in C-RAN and uses best-fit descending as an approach to solve the problem. This concept inspires us the concept to set multiple limits, such as the CPU and the memory limits in this work when packing tasks into servers, and in the experiment part, we will also use this as an evaluation for our result. The difference mainly consists of 2 parts. The first one is that this work mainly focuses on BBU aggregation and does not assume the tasks weighted. The second one is that the authors assume that the supply of resource unlimited. Therefore, as we will see in the experiment section, some difference exist in the experimental results.

Besides the original definition of bin packing that the goal is to minimize the number of bins used [20]. Different targets, such as power consumption minimization, are set in [21], and our target here is the maximization of an operator’s revenue.

2.2 Task scheduling

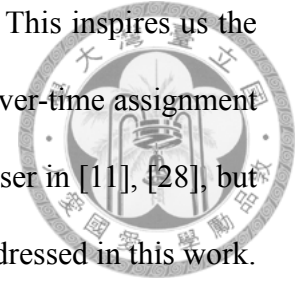


Scheduling refers to the arrangement of work, and is widely applied in many industries, such as manufacturing [22], transportation [23], and online video streaming [24]. Scheduling aims to provide an order that each client receives its required service. Many heuristics have been proposed like first in first out (FIFO), earliest deadline first (EDF), and shortest job first (SJF) to sub-optimally solve the problem due to the problem hardness [25]. Similar to BPP, these heuristics involve sorting. Several sorting targets lead to various schedules. Other than sorting-based heuristics, fair scheduling heuristics (e.g., round-robin scheduling) is also proposed.

Scheduling can be used both in uni-core and multi-core situations, where a single task and multiple tasks are processed simultaneously. Suppose a situation where a server can serve only one task at one time, and multiple servers exist. Such scheduling problem will become an “interval scheduling problem”, as illustrated in Figure 1.1. Task scheduling takes time into consideration in addition to the BPP. In BPP, the packing process is done in one interval. Namely, all items exist in one interval and only stay for only one time slot. However, task requests may not come at the same time and may stay in the system with several time slots. The traditional BPP schemes cannot be applied directly. Thus, this work proposed a new method to solve the problem.

Fair scheduling technique is used in [26] and EDF is suitable for real-time traffic load scheduling, as pointed by [27]. In our experiment, the EDF concept is also utilized during the scheduling process. Authors of [11] decompose the problem into users and BSs, where the BSs are divided into multiple power zones (PZs) and power levels (PLs). A PZ includes

several PLs. Douik et al. further add an upper bound on PLs [28]. This inspires us the idea of scheduling in C-RAN by modeling the problem as a user-server-time assignment problem. The difference is that each PL is exactly allocated to one user in [11], [28], but multiple users can be processed by a server during one interval is addressed in this work.

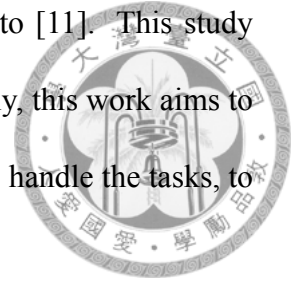


2.3 Load balancing

Several studies had considered load balancing among servers during scheduling. The main idea of the load balancing [29]–[31] is to distribute the traffic load among a given set of cloud servers. The load balancing mechanisms aim to improve the quality of experience [12]. Chabbouh et al. had studied the load balancing through setting the score, which is measured by CPU, memory, and bandwidth utilization rate, in a multi-server environment. They addressed the problem to avoid overload score to prevent items from distributing to other resources [12]. The score concept is also discussed in [32] that the balancing process is triggered when the score is higher than a given threshold. However, the virtual mechanic (VM), which is configured a specific amount of resource before used, is contemplated in this work. Thus, the traffic load is only considered within a VM.

How to pack the VMs, which are the task process units, in a turn-on servers is one of the major issues in this work. VM concept requires low overhead even when the traffic load is high, and we want to minimize the number of active servers in order to lower the cost; hence, the traffic packing among the given power-on servers is the major issue. Furthermore, the arrival tasks might cover several time slots with a given tolerable delay time. The processing time can be distributed in various snippets. Suppose a task only

stays in one server throughout its process period, which is similar to [11]. This study tries to shape the load over a set of processing time slots. Accordingly, this work aims to minimize the number of active cloud servers, which is powered on to handle the tasks, to reduce the required power-on cost.







Chapter 3

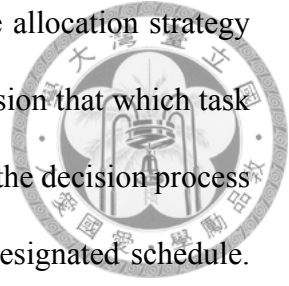
Mathematical Model

3.1 Problem description

In a C-RAN system, there is a resource pool consisting of several servers ready to serve clients in multiple time intervals. The decision here is to let tasks “enter” the system. Because it would be computation-intensive to conduct admission control for every time interval, the process is done at a *batch* scale. In other words, the intervals are divided into several batches, and the decision process will be performed at the beginning of each batch. A task may come at any time, and after its arrival, it will register itself in the system. Because admission control only takes place at the start of a batch, if that task comes during the middle of a batch, it is not until the next batch can it enter the system.

When the next batch begins, the system will decide which tasks to take. The processing time, requirement of CPU and memory, possible revenue, and the status of the resource pool are all possible considerations that may be applied during the admission control pro-

cess. During the process, not only the admission status but also the allocation strategy is made, for there may be multiple BBUs in the system, and the decision that which task will go to which server may bring difference to the final profit. After the decision process is done, service of these admitted tasks is begun according to the designated schedule. Afterwards, when the service is completed (or the task is due, whichever comes first), the system may charge the client, and the allocated resource is released and ready to serve the next client.



Besides the revenue side, we should also consider the expenditure side. In Chapter 1, the idea of the power burden of network operators was introduced. Therefore, when turning on a server for an interval, we should pay the electronic fee. In addition, we consider the start-up cost, which must be paid when a server is turned on. Finally, the block penalty refers to the penalty that is imposed on the network operator if a client is blocked from the system.

The terms used in our model is defined in Table 3.1.

3.2 Problem formulation

The profit of an organization is calculated by gross profit minus cost. Revenue for a firm comes from its clients, and during the service, cost will simultaneously occur. We assume that every client is linked with a charging price and that the cost accrues when the server provides service. The costs taken into consideration are listed in Table 3.1. In this thesis, we model the target problem as an optimization problem of the following objective function:



Table 3.1: Given parameters

Notation	Description
T	The index set of time intervals, which is $\{1, 2, 3, \dots, \tau\}$
I	The index set of tasks, which is $\{1, 2, 3, \dots, i\}$.
D_i	The demand of computing resource of task $i \in I$
R_i	The demand of memory resource of task $i \in I$
V_i	The revenue rate of task $i \in I$
N_i	Penalty of task i if task $i \in I$ is not served
$\delta_{\tau i}$	An indicator function, which is 1 if task $i \in I$ comes at time interval τ , and 0 otherwise
γ_i	Required processing time of task $i \in I$
ϵ_i	Maximum allowed delayed time of task $i \in I$ from its arrival and departure
e_{mn}	An indicator function, which is 0 if task m does not want to be assigned to be the server where task n stays (task m and n are mutually exclusive), and 1 otherwise
θ	Estimated minimum number of blocked tasks
S	The index set of servers in C-RAN, which is $\{1, 2, 3, \dots, s\}$.
C_s	Computing capacity of server s , $s \in S$
M_s	Memory capacity of server s , $s \in S$
A_s	The cost rate of turning on server s for a time interval
E_s	The cost of turning on server s again
O	Minimum required number of turned-on servers

Table 3.2: Decision variables

Notation	Description
$a_{\tau is}$	1 if task i is assigned to server s at time interval τ , and 0 otherwise
b_i	1 if required processing time of task i is not satisfied, i.e., task i is blocked, and 0 otherwise
$x_{\tau s}$	1 if server s is switched on at time interval τ , and 0 otherwise
$y_{\tau s}$	1 if server s is switched on again at time interval τ , and 0 otherwise

$$\max \sum_{\tau \in T} \sum_{i \in I} \sum_{s \in S} V_i a_{\tau is} - \sum_{i \in I} N_i b_i - \sum_{\tau \in T} \sum_{s \in S} A_s x_{\tau s} - \sum_{\tau \in T} \sum_{s \in S} E_s y_{\tau s} \quad (3.1)$$

Decision variables in this equation are shown in Table 3.2. Constraints are explained in details in the following sections.

3.3 Task assignment constraints

In this section, constraints related to task assignment will be discussed.

If a task is permitted to enter the system, then there must be *exactly* one server assigned to this task for processing between the task's arrival and deadline. Also, the assigned server must be turned on in order to serve the client. If a task does not receive required amount of service time, it is considered as *blocked*, and penalty will occur. Therefore, these constraints can be modelled as follows:

$$\sum_{s \in S} a_{\tau is} \leq 1, \forall i \in I, \tau \in T \quad (3.2)$$



$$a_{\tau is} \leq u_{is}, \forall \tau \in T, i \in I, s \in S \quad (3.3)$$

$$\sum_{s \in S} u_{is} \leq 1, \forall i \in I \quad (3.4)$$

$$\sum_{\tau \in T} \sum_{s \in S} a_{\tau is} \leq \gamma_i, \forall i \in I \quad (3.5)$$

$$a_{\tau is} \leq x_{\tau s}, \forall \tau \in T, i \in I, s \in S \quad (3.6)$$

$$\sum_{\tau' \in T} t_{\tau'} \delta_{\tau' i} \leq \sum_{s \in S} a_{\tau is} t_{\tau}, \forall i \in I, \tau \in T \quad (3.7)$$

$$\sum_{s \in S} a_{\tau is} t_{\tau} < \sum_{\tau' \in T} t_{\tau'} \delta_{\tau' i} + \gamma_i + \epsilon_i, \forall i \in I, \tau \in T \quad (3.8)$$

$$a_{\tau ms} + a_{\tau ns} \leq e_{mn} + 1, \forall m, n \in I, \tau \in T, s \in S \quad (3.9)$$

$$\frac{\gamma_i - \sum_{\tau \in T} \sum_{s \in S} a_{\tau is}}{\gamma_i} \leq b_i, \forall i \in I \quad (3.10)$$

$$b_i \leq \gamma_i - \sum_{\tau \in T} \sum_{s \in S} a_{\tau is}, \forall i \in I$$



(3.11)

$$\theta \leq \sum_{i \in I} b_i \tag{3.12}$$

Explanation of the constraints:

- Constraint 3.2: a task may be served by at most one server during a interval.
- Constraint 3.3, 3.4: a task can stay only on one server at most.
- Constraint 3.5: assigned intervals of a task will not be greater than its required processing time.
- Constraint 3.6: a server has to be on to serve tasks.
- Constraint 3.7: the service will begin after a task's arrival.
- Constraint 3.8: the service will terminate before a task's deadline.
- Constraint 3.9: two tasks cannot be assigned to the same server if they do not want to.
- Constraint 3.10: a task is considered as blocked if it does not receive enough service.
- Constraint 3.11: a task is not considered as blocked if it receives enough service.
- Constraint 3.12: at least θ tasks will be blocked from the system (for auxiliary use).

Since the feasible space of dual problems may be large and furthermore increases the

duality gap between the primal problem and the dual problems, we can give an conservative estimation for θ . Here we propose a heuristic, which guarantees that θ will never be overestimated.



To find a conservative estimation of θ , we work from the opposite site: *what is the maximum number of tasks that can be taken?*, and the following procedure is applied in every interval

1. Identify all tasks that exist in this interval and other tasks that overlap with those tasks
2. Store them into a group G , find earliest arrival time A and latest deadline D in G
3. Calculate total resource capacity from A to D
4. Assign tasks from the smallest one, subtract used resource
5. Repeat until no enough resource is available
6. Count the number of allocated tasks
7. Minimum number of blocked tasks is therefore found

3.4 Capacity constraints

The residual computing capacity of a server is calculated as *Total computing capacity* – *Resource used by tasks assigned to this server*. Since the computing and memory resource are considered, two constraints can be set according to this rule.

$$\sum_{i \in I} a_{\tau is} D_i \leq C_s x_{\tau s}, \forall s \in S, \tau \in T$$



(3.13)

$$\sum_{i \in I} a_{\tau is} R_i \leq M_s x_{\tau s}, \forall s \in S, \tau \in T$$

(3.14)

Explanation of the constraints:

- Constraint 3.13: at any time interval, allocated CPU resource will not exceed a server's capacity.
- Constraint 3.14: at any time interval, allocated memory resource will not exceed a server's capacity.

3.5 Server switch constraints

There are always at least O servers on, where O can be set to different values according to the environment requirement. If a server is off in interval $\tau - 1$ and on in interval τ , reopen cost is paid. Reopen cost will never be paid in two adjacent time intervals. By this nature, this cost is paid no more than $\frac{T}{2}$ times, that is, $0 \leq \sum_{\tau \in T} y_{\tau s} \leq \frac{T}{2}$.

$$O \leq \sum_{s \in S} x_{\tau s}, \forall \tau \in T$$

(3.15)

$$x_{\tau s} - x_{(\tau-1)s} \leq y_{\tau s}, \forall s \in S, \tau \in T$$

(3.16)

$$y_{\tau s} + y_{(\tau+1)s} \leq 1, \forall \tau \in T, s \in S$$



Explanation of the constraints:

- Constraint 3.15: at any interval, at least O servers are on.
- Constraint 3.16: reopen happens when a server is from off to on.
- Constraint 3.17: reopen will never happen consecutively.





Chapter 4

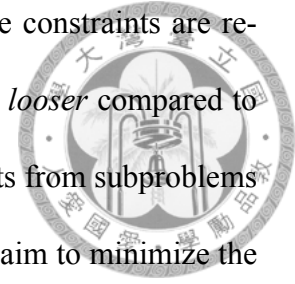
Solution Approach

In this chapter, description of our solution method is given. First, the objective function and constraints are introduced. Next, we are going to solve subproblems related to the decision variables, and finally comes the derivation the primal feasible solution.

4.1 Lagrangian relaxation method

Lagrangian relaxation (LR) is one of the mathematical methods that can be used in constrained optimization problems such as integer programming and combinatorial problems [33]. In a constrained optimization problem, finding the optimal solution may be difficult because one decision variable may affect another one due to the constraints. The advantage of Lagrangian method is that it decomposes the original problem (or the primal problem) into several subproblems (or *dual problems*) that can be solved independently and easily with a more straightforward algorithm by removing difficult constraints and

associating them with corresponding multipliers (μ). Because some constraints are removed (or *relaxed*), the objective value for the subproblems will be *looser* compared to the original problem. In a maximization optimization problem, results from subproblems will be an upper bound (UB) for the primal problem, and we should aim to minimize the results from subproblems iteratively to find feasible solutions.



There are many approaches that can be used to solve the dual problem, and the sub-gradient method [34] is a popular one. In this thesis, we also use this method to solve our dual problem. When the subproblems are solved, we may use the results μ generated there in the primal problem. This allows us to use our proposed method to solve the original problem and to compare the objective value with that from dual problems.

The flow of this method is summarized in algorithm 1.

Algorithm 1: Lagrangian relaxation procedure

- 1 *Initialize the parameters*
 - 2 *While the convergence level is not reached yet:*
 - 3 *Solve each subproblem optimally, record the result*
 - 4 *Use the result from subproblems to solve the primal problems*
 - 5 *Update the optimal result found so far*
 - 6 *Check termination condition*
 - 7 *Adjust multipliers*
-

4.2 Lagrangian relaxation objective function and constraints



When a constraint contains multiple decision variables, this constraint often poses a hindrance on the way toward the desired value. Therefore, those constraints containing more than one decision variables are relaxed, and each of these relaxed constraints becomes an element of LR objective function (LROF, which is also the objective function of the dual problem) in the form of μ . For example, constraint 3.6 is relaxed, so there will be an item

$$\sum_{\tau \in T} \sum_{i \in I} \sum_{s \in S} \mu_{\tau is} (a_{\tau is} - x_{\tau s})$$

in the LR objective function. After a constraint is relaxed, it is excluded from the original constraints, making it easier to find the optimal solution in a less tight condition.

Constraints with multiple decision variables are:

1. Constraint (3.6) - a server is turned on in order to serve
2. Constraint (3.13) - requirement less than CPU capacity
3. Constraint (3.14) - requirement less than memory capacity
4. Constraint (3.16) - reopen cost
5. Constraint (3.10) - blocked if required service is not reached
6. Constraint (3.11) - not blocked if required service is reached
7. Constraint (3.3) - a task will stay only on one server

Relaxing constraints with multiple decision variables, we can derive LROF from the objective function (equation 3.1) as:



$$\begin{aligned}
\min - & \sum_{\tau \in T} \sum_{i \in I} \sum_{s \in S} V_i a_{\tau is} \\
& + \sum_{i \in I} N_i b_i \\
& + \sum_{\tau \in T} \sum_{s \in S} A_s x_{\tau s} \\
& + \sum_{\tau \in T} \sum_{s \in S} E_s y_{\tau s} \\
& + \sum_{\tau \in T} \sum_{i \in I} \sum_{s \in S} \mu_{\tau is}^1 (a_{\tau is} - x_{\tau s}) \\
& + \sum_{\tau \in T} \sum_{s \in S} \mu_{\tau s}^2 \left(\sum_{i \in I} a_{\tau is} D_i - x_{\tau s} C_s \right) \\
& + \sum_{\tau \in T} \sum_{s \in S} \mu_{\tau s}^3 \left(\sum_{i \in I} a_{\tau is} R_i - x_{\tau s} M_s \right) \\
& + \sum_{\tau \in T} \sum_{s \in S} \mu_{\tau s}^4 (x_{\tau s} - x_{(\tau-1)s} - y_{\tau s}) \\
& + \sum_{i \in I} \mu_i^5 \left(\frac{\gamma_i - \sum_{\tau \in T} \sum_{s \in S} a_{\tau is}}{\gamma_i} - b_i \right) \\
& + \sum_{i \in I} \mu_i^6 \left(b_i - \gamma_i + \sum_{\tau \in T} \sum_{s \in S} a_{\tau is} \right) \\
& + \sum_{\tau \in T} \sum_{i \in I} \sum_{s \in S} \mu_{\tau is}^7 (a_{\tau is} - u_{is})
\end{aligned} \tag{4.1}$$

The first four elements of equation 4.1 (LROF, and the objective function of the dual problem) are identical to the objective function. Before relaxation, there are 16 constraints. Since 7 constraints are relaxed, only 9 constraints exist in the LROF:

1. Constraint (3.2) - at a time, a task is served by only one server

2. Constraint (3.3) - service is done by a single server
3. Constraint (3.5) - make sure a task get enough service
4. Constraint (3.7) - service begins after a task's arrival
5. Constraint (3.8) - service is done before a task's deadline
6. Constraint (3.9) - mutually exclusive tasks that cannot be put together
7. Constraint (3.12) - estimated minimum blocked tasks
8. Constraint (3.15) - minimum required amount of turned on servers
9. Constraint (3.17) - reopen cost when a server is turned on again



After the LROF is built, we can start to develop and solve subproblems in respective to the decision variables. To establish a subproblem, we first select one decision variable from the LROF. Next, items containing that decision variable in the LROF are extracted, which furthermore consist the objective function of the subproblem. In this model, there are five decision variables, a , x , y , b , and u , so there will be also five subproblems, which will be discussed later in subsequent sections and denoted as SP_1 to SP_5 .

4.3 Task assignment subproblem - SP_1

Extracting items with a in equation 4.1, we can develop the objective function for this subproblem:

$$\min \sum_{i \in I} \sum_{\tau \in T} \sum_{s \in S} \left(-V_i + \mu_{\tau is}^1 - \frac{\mu_i^5}{\gamma_i} + \mu_i^6 + \mu_{\tau s}^2 D_i + \mu_{\tau s}^3 R_i + \mu_{\tau is}^7 \right) a_{\tau is} \quad (4.2)$$

Constraints: (3.2), (3.5), (3.7), (3.8), and (3.9)

Because setting an element of $a_{\tau is}$ to 1 does not affect whether another element of $a_{\tau is}$ is set to 1 or not, this subproblem can be divided into $T \times I \times S$ subproblems. For each time, we may select the server s with a minimum value that does not host any rivals of this task and set the according element of a . Hence, we can use a *greedy algorithm* to solve this subproblem, which is explained below:

Because a task can stay on only one server, to find the optimal value, we may test all servers and select the best value as the result. The procedure is as follows:

Algorithm 2: Algorithm for subproblem 1

```
1 for each task  $i$ :
2   for each time  $t$  from arrival to deadline:
3     for each server  $s$ :
4       calculate the coefficient  $C[t][i][s]$ 
5       find the server with minimum value and cause no conflict with other tasks
6       select time with minimum value
7   set decision variable  $a$  according to the result
```

4.4 Server power on subproblem - SP₂

Extracting items with x in equation 4.1, we can develop the objective function for this

subproblem:

$$\min \begin{cases} \sum_{\tau \in T} \sum_{s \in S} \left(A_s - \sum_{i \in I} \mu_{\tau i s}^1 - \mu_{\tau s}^2 C_s - \mu_{\tau s}^3 M_s + \mu_{\tau s}^4 - \mu_{(\tau+1)s}^4 \right) x_{\tau s}, & \forall \tau < T \\ \sum_{\tau \in T} \sum_{s \in S} \left(A_s - \sum_{i \in I} \mu_{\tau i s}^1 - \mu_{\tau s}^2 C_s - \mu_{\tau s}^3 M_s + \mu_{\tau s}^4 \right) x_{\tau s}, & \forall \tau = T \end{cases} \quad (4.3)$$



Constraints: (3.15)

Because setting an element of x to 1 does not affect whether another element of x is set to 1 or not, we can use a *greedy algorithm* to solve this subproblem, which is explained below:

Algorithm 3: Algorithm for subproblem 2

```

1  for each time  $t$ :
2      for each server  $s$ :
3          calculate coefficient  $C$  given  $t, s$ 
4          if  $C < 0$ :
5              set  $x[t][s]$  to 1
6          else:
7              store  $C$ 
8          if open too less server:
9              open servers with maximum  $C$ 

```

4.5 Server re-power on related subproblem - SP₃



Extracting items with y in equation 4.1, we can develop the objective function for this subproblem:

$$\min \sum_{\tau \in T} \sum_{s \in S} (E_s - \mu_{\tau s}^A) y_{\tau s} \quad (4.4)$$

Constraints: (3.17)

Because setting an element of y (for instance, y_3) to 1 *prevents* another element of y (in this case, y_2 and y_4) from setting to 1, a simple greedy algorithm cannot be applied.

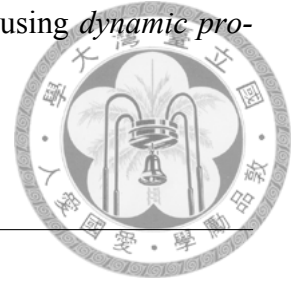
To demonstrate this effect, we will consider a relatively small example. Imagine that if we have 3 elements in the decision set, -4 , -6 , -5 , and this set is ordered and cannot be sorted or modified; that is, the original order must be preserved. A simple greedy algorithm will not work here by choosing -6 ; however, the optimal answer for this set is -9 , by selecting -4 and -5 .

The answer of a server for this subproblem can be derived from the following recurrence relation:

$$Y_t = \begin{cases} 0, & t = 0 \\ \min(C_{\tau s}, 0), & t = 1 \\ \min(Y_{t-1}, Y_{t-2} + C_{\tau s}), & t \geq 2 \end{cases}$$

where Y_t stands for the answer from interval 1 to interval t , and $C_{\tau s}$ means the coefficient of $y_{\tau s}$ in this subproblem. The final answer for this server can be found at Y_T , and aggregating the answer from all servers will produce the final result for this subproblem.

Having this recurrence equation, we can solve this subproblem using *dynamic programming (DP)* approach, which is explained below:



Algorithm 4: Algorithm for subproblem 3

```
1 Let  $Y[ T ]$  = the optimum for the first  $T$  elements
2 Initialize  $Y[ T ]$  to 0
3 for all time  $t$  and server  $s$  pairs  $(t,s)$ :
4   find coefficient  $C[ t ][ s ]$ 
5 for each server  $s$ :
6   find answer for  $Y[ 1 ]$  and  $Y[ 2 ]$ 
7   for all time  $t$  after 2:
8     if  $Y[ t-1 ] > Y[ t-2 ] + C[ t ][ s ]$ :
9       mark  $t$  as used
10     $Y[ t ] = \min( Y[ t-1 ], Y[ t-2 ] + C[ t ][ s ] )$ 
11 from the final time:
12   if  $t$  is marked:
13     set  $y[ t ][ s ]$ 
14     go back 2 time intervals
15   else:
16     go back 1 time interval
```

Proof of this approach can be done by a strong mathematical induction:

Proof.

1. Let Y_n be the optimal answer when first n elements are considered
2. When $n = 1 \vee 2$, Y_1, Y_2 can be found trivially
3. Assume when $n = k$, Y_1, Y_2, \dots, Y_k are correctly found
4. When $n = k + 1$, $Y_{k+1} = \min(Y_k, Y_{k-1} + C_{k+1})$
5. By strong mathematical induction, our method is proven



□

4.6 Block subproblem - SP₄

By eliminating elements other than b , this subproblem can be written as:

$$\min \sum_{i \in I} (N_i - \mu_i^5 + \mu_i^6) b_i \quad (4.5)$$

Constraints: (3.12)

The solution to this subproblem is similar to what is used to solve a and x (*greedy algorithm*).

Algorithm 5: Algorithm for subproblem 4

- 1 **for each** task i :
- 2 *find* coefficient $C[i]$
- 3 *sort* C
- 4 **set** $b[i]$ **to** 1 **if** $C[i] < 0$

5 **if** $\text{sum}(b) < \text{estimated blocked number}$:

6 **set** *unset indices* **to** 1



4.7 Auxiliary subproblem - SP₅

The last element that has not yet been discussed is u . If we extract elements with u , we will get the following objective function for this subproblem:

$$\min \sum_{\tau \in T} \sum_{i \in I} \sum_{s \in S} -\mu_{\tau is}^7 u_{is} \quad (4.6)$$

Constraints: (3.4)

This subproblem is relatively simple by setting u_{is} to 1 if $-\mu_{\tau is}^7$ is negative and 0 otherwise.

4.8 Getting primal feasible solution

After calculation of the five subproblems discussed above is done, we can find some information in the obtained result. Using information obtained from SP_1 to SP_5 , we can build a strategy for task assignment. The brief procedure is as follows:

1. Sort clients according to their μ value
2. Sort servers according to their μ value

3. Assign clients to servers in their respective order
4. Switch on or off servers according to their status
5. Calculate objective value



First, we propose the algorithm called TABLE (**t**ask **a**llocation **b**y **L**R **e**valuation). Through subproblems, importance of tasks and servers are represented in the form of μ ; therefore, the sum of μ values can be used as an index. Furthermore, CPU and memory requirement and deadline of tasks are also important to task allocation strategy. We define the `mu_importance` function as

$$MI = \frac{TV \times ms}{C \times M \times b^2}$$

, where the symbols are explained below in details.

- MI : importance in regards of μ , this is also the base of sorting.
- TV : total value that can be generated from a task.
- ms : sum of μ that is related to the task, which is calculated by

$$\sum_{\tau \in T} \sum_{s \in S} \mu_{\tau is}^1$$

, where i is the index of that task.

- C : CPU requirement of a task.
- M : memory requirement of a task.

- b : residual buffer time of a task.



If task a has a higher $mu_importance$ than task b , then task a will be allocated before task

b . Description of the concept of this algorithm is represented next.

In order to generate a higher revenue, it can be easily seen that we should take tasks with a higher value, so therefore TV appears in the numerator. On the other hand, if a task takes more resource than another one, it is likely that it will prevent other tasks from entering the system; as a result, both C and M appear in the denominator part. Next, we are considering the result generated from subproblem 1. As mentioned earlier, we take $\mu_{\tau is}^1$ into consideration. $\mu_{\tau is}^1$ corresponds to the unmatchness between task i and server s during time interval τ . Because we are considering the importance of a task, we accumulate all $\mu_{\tau is}^1$ with a given task ID i . Hence, a higher sum of $\mu_{\tau is}^1$ means a greater difference, which can be treated as a measure of importance, between the task and the server, and a task with higher μ value should be processed first. Finally, the residual buffer time shows the urgency of a task: when residual buffer time of a task is larger, it usually means that the task are willing to wait and can be processed later. Under this circumstance, the resource can be allocated to tasks that are due soon, and this procedure can prevent potential penalty that may be generated from blocked tasks (tasks that do not receive enough service).

Next, the servers are sorted by their μ values in a similar manner, except that in this case, it is μ^2 and μ^3 that are compared. μ^2 stands for the difference between the CPU capacity and the requirement, while μ^3 represents the memory difference. To find the server with higher priority, we sum all μ^2 and μ^3 as the $mu_importance$ of this server, given the server index s , and sort the servers according to this result.

TABLE algorithm is summarized in the following pseudo code:



Algorithm 6: *TABLE* algorithm

```
1 calculate mu_importance of the tasks
2 sort the tasks in mu_importance descending order
3 calculate mu_importance of the servers
4 sort the tasks in mu_importance descending order
5 for each task:
6     for each server:
7         if server can take the task:
8             break
9         else:
10            take the task
```

Next, when tasks are being assigned to servers, we may find some “peaks” of utilization rate. These peaks may hinder us from allocating service to tasks (due to their deadlines.) When such case happens, we first try to remove some tasks from servers and then assign the new task to the server. After that, we see if all tasks removed in the previous step can be put back. If it succeeds, then this new task, which may not enter the system before this procedure, may enter the system now. This procedure is named as *ROTATE* (reassignment of tasks aim to equilibrium).

ROTATE algorithm is summarized in the following pseudo code:

Algorithm 7: *ROTATE* algorithm

```
1 for each task:
```

```

2   for each server :
3       if server can take the task :
4           break
5       else :
6           remove the tasks already in this server until this task can enter
7           take this task
8           put all removed tasks back
9           if all removed tasks can go back :
10              break # ROTATE method succeeds
11          else :
12              restore the original state (before this task enters)

```



Finally, after the process is done, we check if there is any server that is having a deficit. In other words, the revenue does not cover the costs. If it happens, we further consider that whether turning the server off and marking all tasks belonged to this server as blocked will be better or not (actually a comparison between a greater loss and a smaller loss). If turning off this server is better, we will turn off the server and drop all tasks on this server, and these tasks will be considered as blocked and the block penalty will be imposed. This algorithm is abbreviated as TOWEL (turn off when evaluated loss).

TOWEL algorithm is summarized in the following pseudo code:

Algorithm 8: TOWEL algorithm

```

1   for each server :
2       if loss happens :

```

3 *check the revenue change when turned off*
4 ***if turn off is better :***
5 *turn off this server*
6 ***mark all tasks in this server as blocked***



The overall primal problem algorithm can be written as:

Algorithm 9: Primal problem solution

1 *solve the dual problem*
2 ***for each task:***
3 *TABLE()*
4 *ROTATE()*
5 ***for each server:***
6 *TOWEL()*
7 *calculate the final objective value*



Chapter 5

Computational Experiments

In this chapter, we will list several scenarios and examine the trends in the chart. This chapter will consist of four parts, which focus on different aspects of the system. These parts are listed below:

1. Task quantity - test how task quantity affects the objective function.
2. Server quantity - test how server quantity affects the objective function, given that the total computational capacity is fixed.
3. Task processing time - due to the different nature of clients, some groups of user take a longer time in the system, while other may only take a short period in the system. We will examine what will happen when the average processing time becomes longer.
4. Buffer time - generally, if clients tend to wait longer for them to be served, they will be less likely to be blocked from the system, which subsequently prevents the

operator from being fined. In this part, different values of allowed buffer time will be examined, and the difference between objective values will also be illustrated.

5. Task block penalty - when a task is blocked from the server, its corresponding penalty will be imposed on the network operator. When the penalty changes, so does the objective value. In this part, the relationship between the penalty and the objective value will be examined.
6. Task revenue rate - operators may propose many deals to customers, and different prices may be given to different customers. If the revenue rate is higher, we can expect a higher profit, so we will try different rates and see the change in the result.
7. Server cost rate - as discussed earlier, the electronic fee is a great burden for operators. For instance, in summer, the electronic fee rate will be higher than in winter. This part will try to find the impact that server cost rate brings on the objective value.

The arrival, deadline, demand, and other characteristics are generated randomly using `$RANDOM` in bash. For each experiment, 8 curves are produced. The first curve is the baseline. For baseline, we use the *next-fit* algorithm that is introduced in section 2.1 and denote the solution as *BP-FF*. In addition, we take the approach proposed by [19] and denote it as *BFD* (best-fit descending) in the context and *Xu et al.* in the charts. The next three lines are results for our proposed *TABLE*, *TOWEL*, and *ROTATE* algorithm. Next comes the result that combines the above proposed methods, which is denoted as *Primal*. The last curve is the result from subproblems (LR dual problems), which is an upper bound for the primal problem and is denoted as *Dual* in the following sections.

Also, in each section, the improvement ratio is calculated. The improvement ratio is

calculated by $\frac{Primal - Baseline}{Baseline}$ where *Baseline* is substituted by BP-FF and BFD described above.



5.1 Experiment on task quantity

First, we will examine the trend of the objective function. The control variable here is the number of the tasks. In our experiment, the quantity of tasks ranges from 200 to 800.

5.1.1 Uniform arrival

In the first case, we will test the performance when the clients come steadily. This is a simulation for daily cases, where users come in a regular fashion. In the next case, we will find the result when arrival of the clients is not stable. The result of a steady arrival rate is drawn in Figure 5.1.

The analysis consists of two parts: when task number is less than and greater than 400. In our setting, the demand of 400 tasks is approximately equal to the server capacity, and we will examine these situations separately.

Part 1: $|I|$ from 100 to 400 This is the case where the demand does not exceed (or slightly greater than due to the randomness of data generation) the server capacity. In this part, BFD performs better than our method. In such cases, we can expect that most of the existent tasks will be packed into servers, so there will not be much difference in the revenue part. But on the cost side, BFD starts the process from the task with the highest

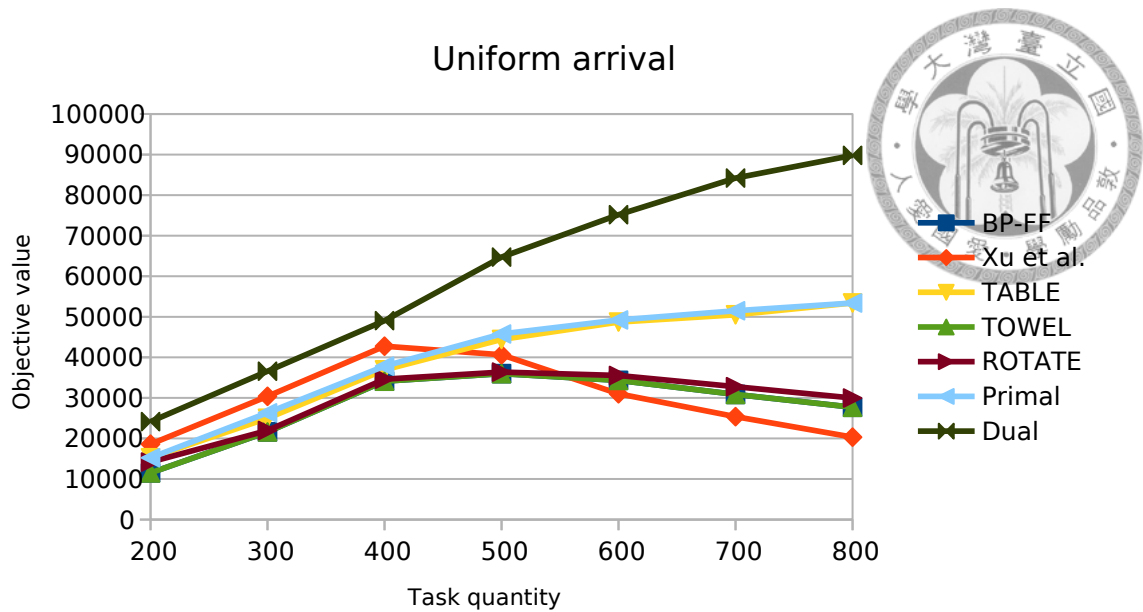


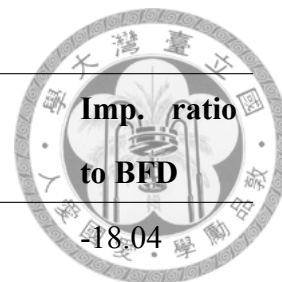
Figure 5.1: Uniform arrival time

demand and packs the tasks to the server with fewest capacity that can accommodate it. This process will then pack task to the servers that are already in use (because their available capacity is smaller), and in the end, fewer servers will be turned on. Hence, the cost from turning on and re-turning on servers will be lower, which helps the revenue.

Part 2: $|I|$ from 500 to 800 When the demand exceeds the capacity, because we can almost be certain that some tasks will be blocked. In BFD, it only considers the demand of the tasks, therefore, it will fail to achieve a higher profit. On the other hand, in TABLE method, because the tasks are sorted by their $\mu_importance$, tasks with higher cost-price ratios will be considered first. In addition, when there are more tasks to choose from, we can select more profitable tasks. Therefore, as shown in the figure, when there are more tasks, the objective value rises and our method has a better performance than BFD.

Table 5.1: Improvement ratio for uniform arrival

Scenario	Primal	BP-FF	Imp. ratio to BP-FF	BFD	Imp. ratio to BFD
200	15236	11511	32.36	18590	-18.04
300	26314	21614	21.74	30378	-13.37
400	37822	34120	10.85	42747	-11.52
500	45770	35973	27.23	40619	12.68
600	49221	34311	43.46	31016	58.70
700	51500	30847	66.95	25360	103.07
800	53395	27683	92.88	20319	162.78



5.1.2 Bursty arrival

This case is very similar to the first scenario, except the pattern of the arrival time of the tasks. In the last experiment, the distribution of arrival time follows uniform distribution. Beyond that, we also tested the overall performance of our method when the tasks come in a bursty pattern. The result is shown in Figure 5.2.

To simulate a bursty pattern, we first select some *burst start points* and the bursty period. After the bursty period is set, the tasks can come only during these periods.

The chart is also similar to Figure 5.1, but however, method TOWEL and ROTATE do not perform as good as when tasks come on a regular basis. This is because BFD, TOWEL and ROTATE do *not* take the importance of the tasks into consideration, and therefore fails to achieve a higher objective value when not all tasks can enter the system.

We can also see how bursty arrival affects the objective value: all results are worse

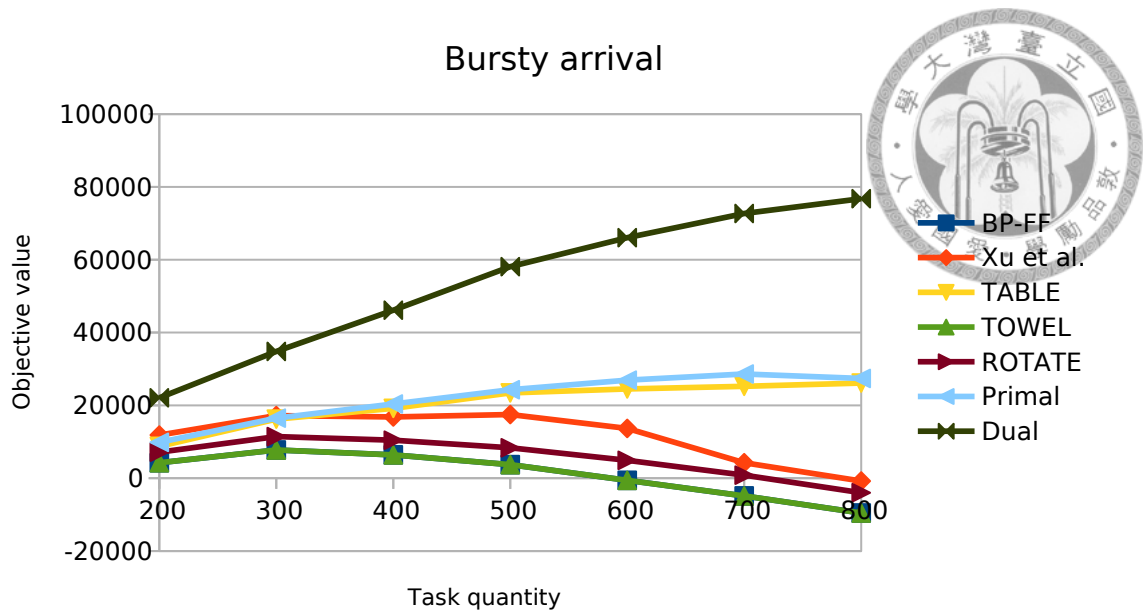


Figure 5.2: Bursty arrival time

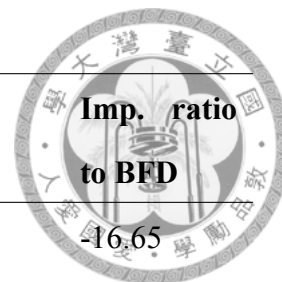
when compared with the last experiment. One explanation is that because the capacity of the servers is finite, when there are many tasks coming in a sudden, there will not be enough buffer time for the servers to serve the tasks. In other words, those tasks that are not served are considered as *blocked*, and the penalty, which lowers the revenue, will be imposed on the objective value. In addition, due to the block penalty, the objective value becomes negative when the penalty exceeds the revenue.

5.2 Experiment on fixed capacity with different number of servers

In the second part of our experiment, we will test how the combination of server quantity and capacity affects the objective value. For example, if the system capacity is fixed at X , then which one will be better, 2 servers with each server with a capacity of $\frac{X}{2}$, or

Table 5.2: Improvement ratio for bursty arrival

Scenario	Primal	BP-FF	Imp. ratio to BP-FF	BFD	Imp. ratio to BFD
200	9856	4236	132.67	11825	-16.65
300	16479	7754	112.52	17181	-4.09
400	20389	6407	218.23	16818	21.23
500	24291	3725	552.11	17474	39.01
600	26919	-583	4717.32	13657	97.11
700	28619	-4861	688.75	4234	575.93
800	27390	-9560	386.51	-773	3643.34



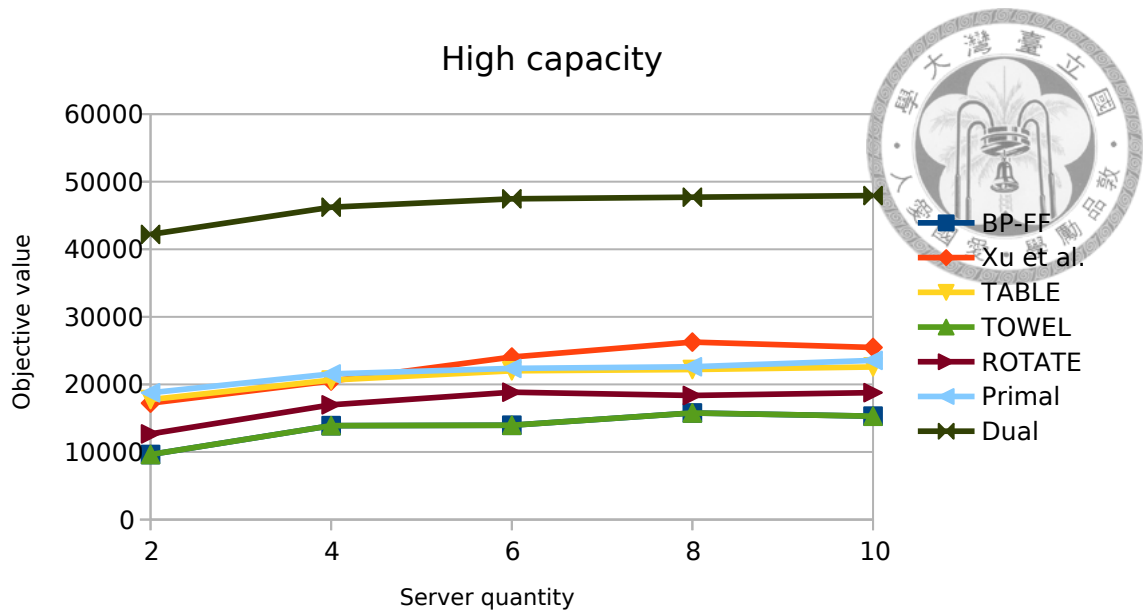
4 servers with each server with a capacity of $\frac{X}{4}$? Or to be more general, when the total capacity is X , and we have n servers with each server with a capacity of $\frac{X}{n}$, which one of a larger or a smaller n will bring a higher revenue?

In addition to that, we tested 3 different conditions: when the server capacity is more than the expected required amount of service of the tasks, when the server capacity is approximately equal to the need of the tasks, and when the server capacity is less than the requirement of the tasks.

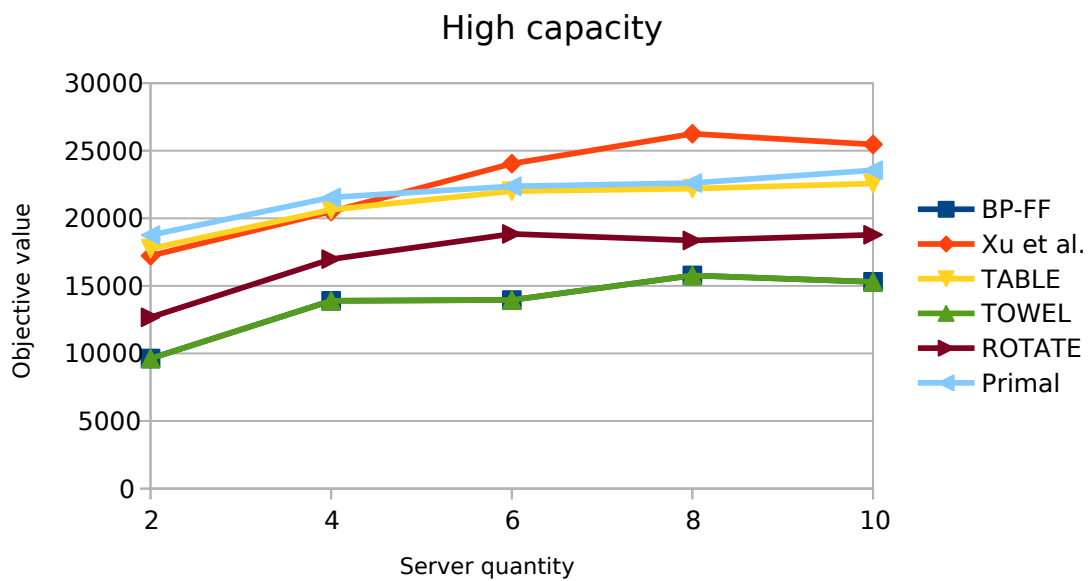
5.2.1 High capacity

We will first examine how the combination of servers affects the objective value. We illustrate the trend using server quantity from 2 to 10. The result is shown in Figure 5.3A.

Exaggerating the trend by excluding the LR curve, we can see the trend better in Figure 5.3B. It can be seen that when there are more (but smaller) servers, the objective value will



(A) With LR curve



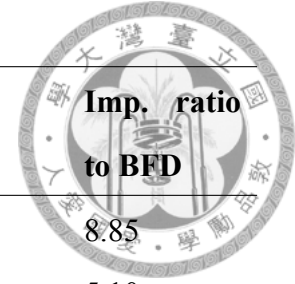
(B) Without LR curve

Figure 5.3: Capacity greater than requirement

also rise. This is because when there are more servers, a more flexible arrangement of tasks between servers can be achieved. To illustrate this effect, we can imagine that requirement is about 60% of available resource. When there are only 2 servers, we have to turn both on in order to serve all tasks. In contrast, when we have 10 servers, we can turn on only

Table 5.3: Improvement ratio for high capacity

Scenario	Primal	BP-FF	Imp. ratio to BP-FF	BFD	Imp. ratio to BFD
2	18752	9618	94.97	17228	8.85
4	21543	13894	55.05	20497	5.10
6	22370	13951	60.35	24040	-6.95
8	22611	15769	43.39	26253	-13.87
10	23554	15301	53.94	25459	-7.48



6 servers out of the 10. In this way, we can save costs, and this will subsequently elevate the operational revenue.

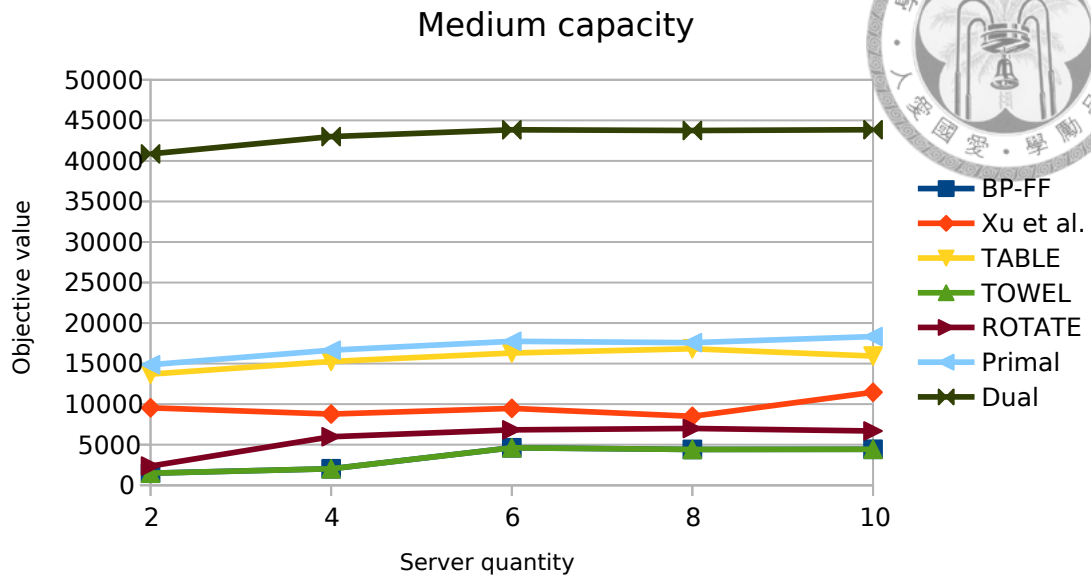
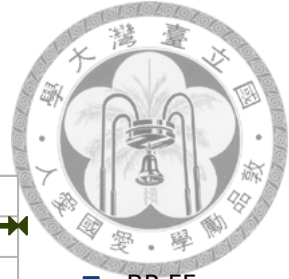
In this situation, BFD performs better than our method. Similar to the discussion in section 5.1, BFD tends to aggregate tasks in as less as possible servers, and therefore lowers the cost for turning on the servers. In the following subsections, we will examine other situations where the capacity is not that abundant.

5.2.2 Medium capacity

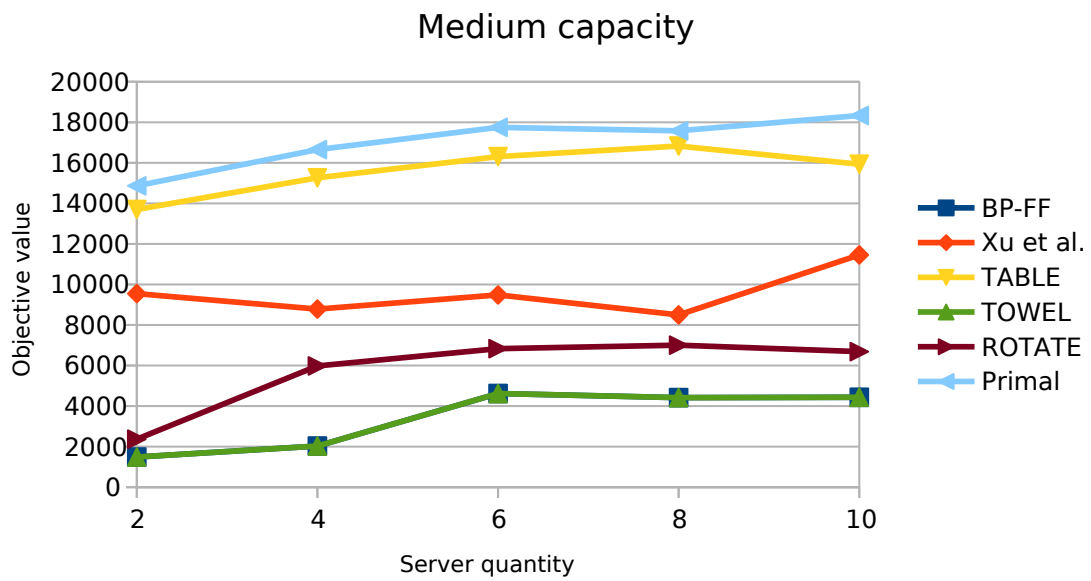
When the system capacity is close to the need of the clients, the chart is very similar to the counterpart of the “higher” version in section 5.2.1. This is reasonable because in these 2 situations, generally no tasks will be blocked due to the sufficient supply of resource. The result is shown in Figure 5.4A and Figure 5.4B.

In section 5.2.1, BFD performs better; however, when the resource is more limited, BFD does not perform as well as the previous result. Like the discussion in section 5.1, when resource are limited, our method has a better performance in finding the “right” tasks

to serve.



(A) With LR curve



(B) Without LR curve

Figure 5.4: Capacity approximately equal to requirement

5.2.3 Low capacity

When the capacity is relatively smaller than the required amount of service, the trend

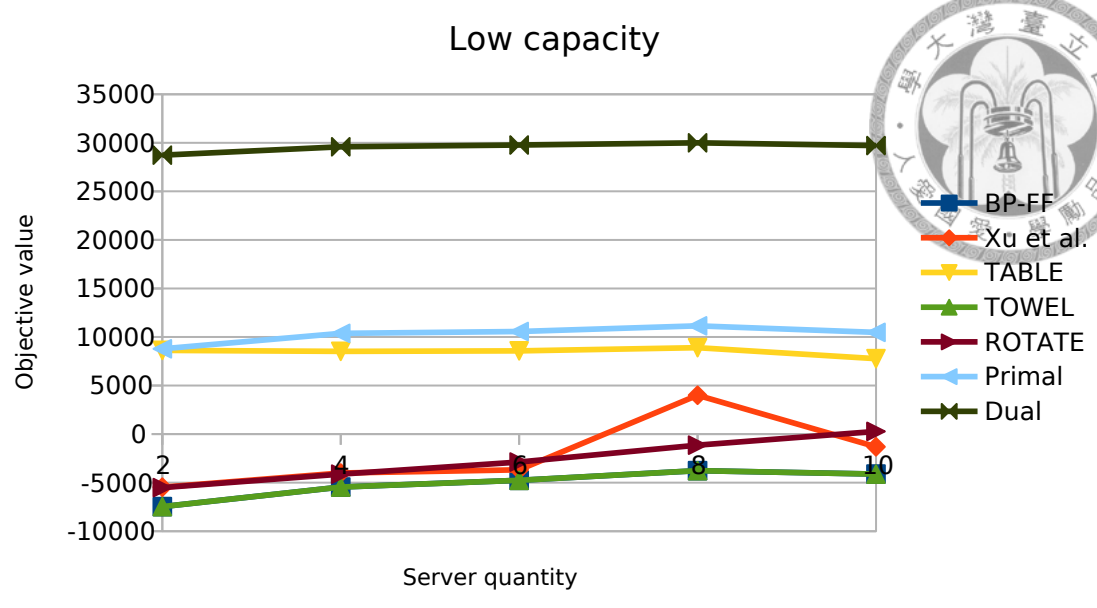
Table 5.4: Improvement ratio for medium

Scenario	Primal	BP-FF	Imp. ratio to BP-FF	BFD	Imp. ratio to BFD
2	14866	1489	898.39	9548	55.70
4	16654	2031	719.99	8782	89.64
6	17750	4620	284.20	9482	87.20
8	17578	4417	297.96	8497	106.87
10	18335	4436	313.32	11459	60.01

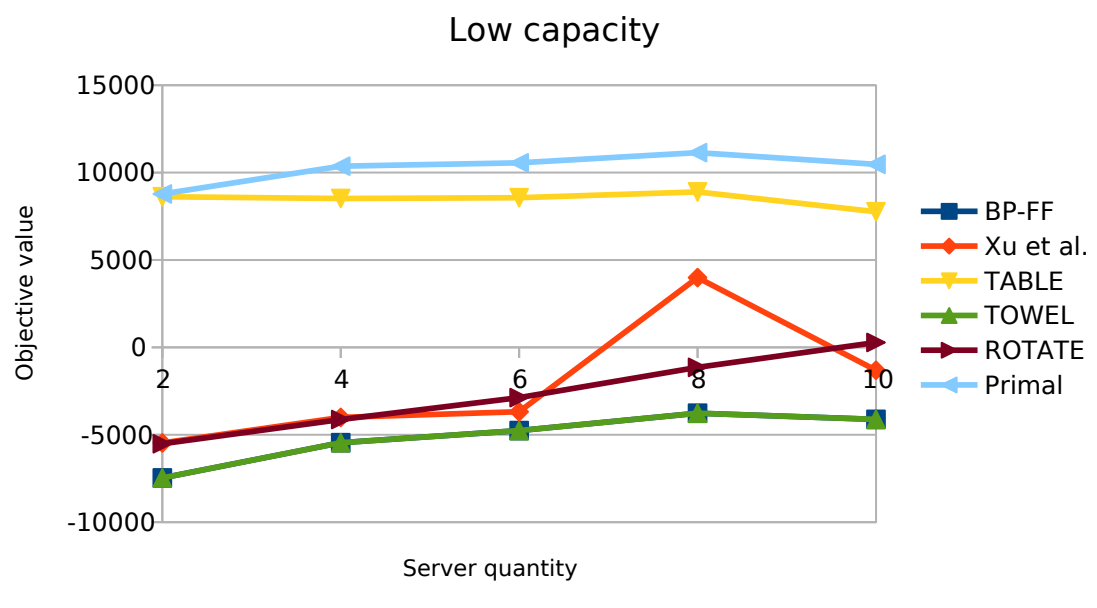
Table 5.5: Improvement ratio for low capacity

Scenario	Primal	BP-FF	Imp. ratio to BP-FF	BFD	Imp. ratio to BFD
2	8776	-7473	217.44	-5469	260.47
4	10368	-5457	289.99	-4010	358.55
6	10558	-4762	321.71	-3684	386.59
8	11140	-3770	395.49	3993	178.99
10	10473	-4122	354.08	-1310	899.47

discussed above does not significantly exist anymore. When the system capacity is less than the need, all servers are very likely to work from beginning to end. This concept is similar to the *bottleneck* that will be discussed in cost accounting. When there is a bottleneck during a product line, that bottleneck must be run at full speed to avoid further decrease to the efficiency. When there are not enough servers, generically all servers must be on all the time in order to server more tasks. However, when all servers are on, the advantage discussed above does not exist, and that is why there is no significant increase in objective value by the increase of the number of servers. Also, in this scenario, our method performs better than BFD. The result is depicted in Figure 5.5A and Figure 5.5B.



(A) With LR curve



(B) Without LR curve

Figure 5.5: Capacity less than requirement

5.3 Experiment on processing time

In the third part, we will discuss the correlation between the max processing time γ and the objective function. In general, when γ becomes larger, it will also be more difficult to

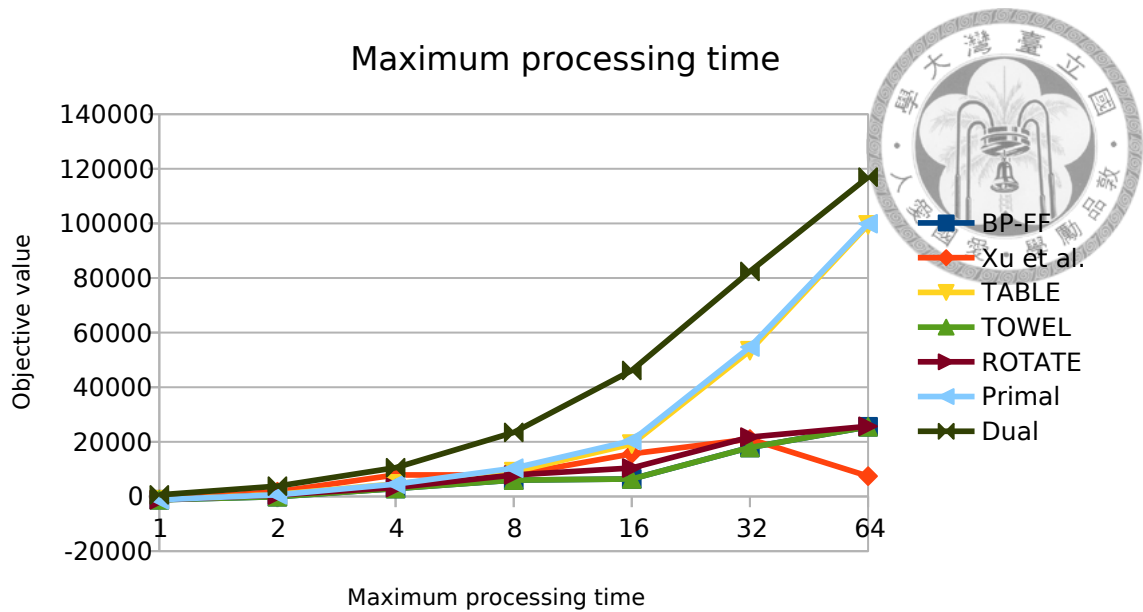


Figure 5.6: Processing time

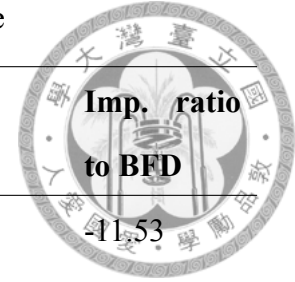
assign the tasks properly because every task now takes more resource in the system and furthermore decreases the available resource that may be reserved for other use when the clients do not stay in the system so long.

In our experiment, processing time of each task follows a uniform distribution. In the following chart, the x axis denotes the max possible processing time for each task. For instance, a value of 10 for x means that the processing time follows a uniform distribution in the interval of $[1, 8]$; that is, the expected processing time is $\frac{1+8}{2} = 4.5$. The result is shown in Figure 5.6.

The first observable phenomenon, which can be easily explained, is that all objective values rise through the increase of tasks' processing time. Since V_i in the objective function denotes the revenue *rate* of task i and $V_i \times \gamma_i$ equals the revenue that can be generated from this task, when γ_i becomes larger, the revenue also rises. From the chart, it can be discovered that when γ_i grows in an exponential fashion, so does the objective function.

Table 5.6: Improvement ratio for processing time

Scenario	Primal	BP-FF	Imp. ratio to BP-FF	BFD	Imp. ratio to BFD
1	-1209	-1209	0.00	-1084	-11.53
2	626	-74	945.95	1751	-64.25
4	4525	2900	56.03	7875	-42.54
8	10351	5998	72.57	7891	31.17
16	20389	6407	218.23	15623	30.51
32	54713	17920	205.32	21014	160.36
64	99893	25519	291.45	7452	1240.49



Next, we can see that when the processing time is relatively small, there is no significant difference between these methods; however, when the overall processing time becomes larger and larger, difference between the methods also gets more noticeable. Here we can also see that because TABLE method takes the importance of tasks into consideration, it succeeds to find the tasks with a higher priority and assign them into the system first. On the contrary, BFD, ROTATE and TOWEL methods fail to do so when the system becomes more “crowded.”

With other things being equal, the increase in processing time (demand per interval is constant) results in the increase in the total demand. Consistent with previous discussion, the performance of BFD is not good when demand exceeds server capacity as the processing time goes above the threshold.

5.4 Experiment on tolerance and waiting time



The final part of the experiment will try to find the correlation between the waiting time ϵ and the objective value. Generally, the larger ϵ is, the more flexibility there will be, and subsequently, the chance of being blocked from the system of a task gets lower and helps improve the result. In this experiment, we will show the results of different ϵ values.

With tolerance, we may achieve traffic shaping because the task can stay in the buffer for a while before begin serviced. In this way, the system can transform the traffic burst into a relatively smooth pattern, and as the tolerance becomes larger, the possibility of traffic shaping also increases.

Figure 5.7 shows the objective value for different ϵ values. The curve is consistent with our guess, which is that ϵ is in a positive correlation with the objective value. Also, BFD performs worse because the resource is limited; however, the objective value obtained from BFD also increases by the increase in the buffer time.

From the chart, we can also find an interesting phenomenon: when $\epsilon \in [1, 20]$, the objective value steadily increases with ϵ , but however, when $\epsilon \in [20, 80]$, the objective value does not increase as noticeably as when ϵ is lower. Such result gives us an implication that when ϵ reaches a threshold, the objective value will be approximately constant. In other words, a value of 20 units for ϵ is large enough to provide the desired amount of flexibility of assignment of the clients to the system in this scenario. This idea can be useful for operators because an operator can set a quality of service (QoS) threshold and tell its users the maximum possible delay for their service. If the delay is not acceptable

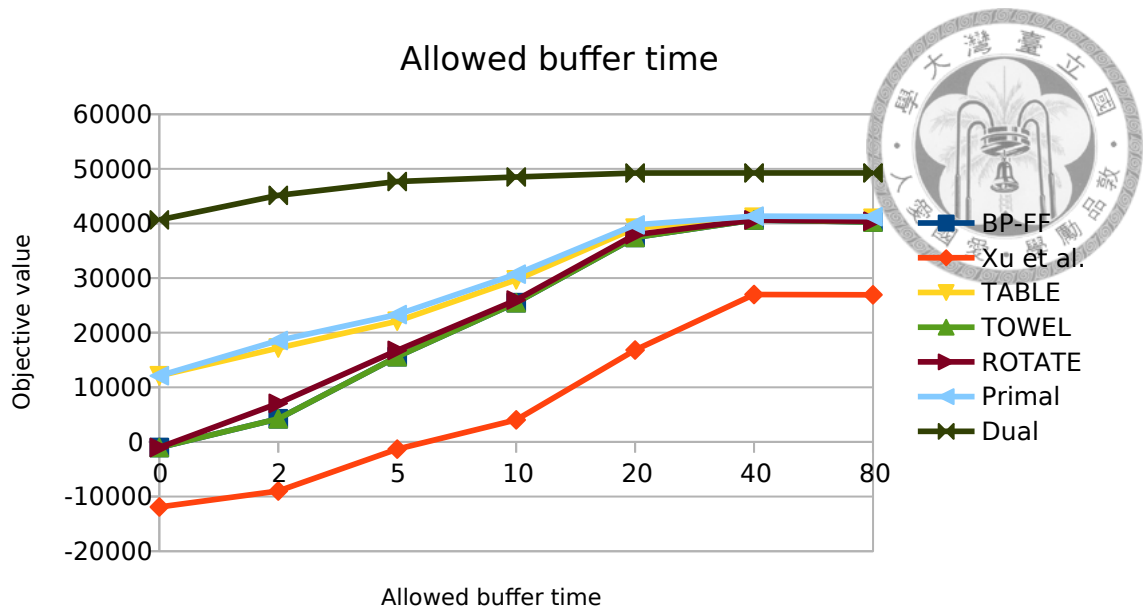


Figure 5.7: Different allowed buffer time

in practice (in this experiment, delay of 20 units of time), then operator should add more servers (or computing resources) to the system.

5.5 Experiment on task block penalty

In this part, we will test how task penalty affects the objective value. It can be easily guessed that when the penalty is higher, the objective value will go down. This is because when the number of tasks are determined, there is no other influence that can be done to the objective value. The result is shown in Figure 5.8.

From the chart, we can find that when the task penalty grows in a linear fashion, so is the objective value, which is consistent with our previous guess.



Table 5.7: Improvement ratio for buffer time

Scenario	Primal	BP-FF	Imp. ratio to BP-FF	BFD	Imp. ratio to BFD
0	12105	-1027	1278.68	-11909	201.65
2	18562	4239	337.89	-8983	306.63
5	23323	15603	49.48	-1324	1861.56
10	30713	25508	20.41	4042	659.85
20	39745	37456	6.11	16818	136.32
40	41341	40641	1.72	27000	53.11
80	41241	40266	2.42	26948	53.04

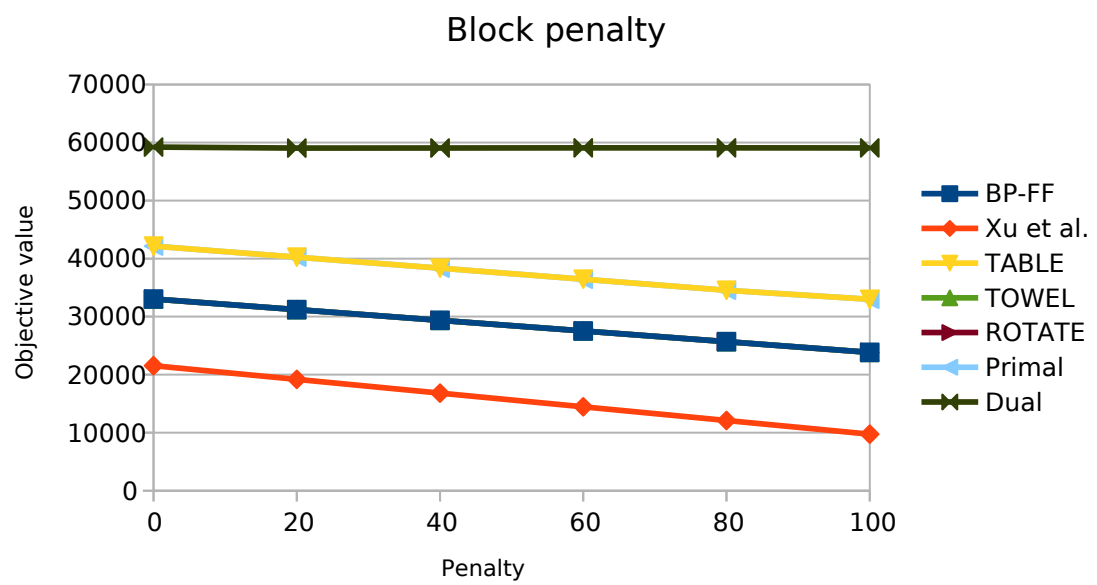
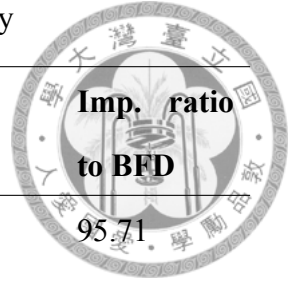


Figure 5.8: Different task penalty

Table 5.8: Improvement ratio for task block penalty

Scenario	Primal	BP-FF	Imp. ratio to BP-FF	BFD	Imp. ratio to BFD
0	42153	33039	27.59	21538	95.71
20	40253	31199	29.02	19178	109.89
40	38353	29359	30.63	16818	128.05
60	36453	27519	32.46	14458	152.13
80	34553	25679	34.56	12098	185.61
100	32986	23839	38.37	9738	238.73



5.6 Experiment on task revenue rate

Next, the relationship between the expected revenue rate and the objective value is examined. If the expected revenue rate is higher (maybe due to the nature of the potential clients), we may be able to serve the tasks with higher revenue rates, which will further increase the profit of a network operator. In this scenario, we will test expected task revenue rate from 20 to 100. The result is shown in Figure 5.9.

When the expected revenue rate is higher, the objective value is higher, too. Compared with BFD, when the revenue rate is higher, the difference between their method and ours slightly becomes larger. This is due to the point that BFD only considers the loading of the tasks and does not consider the revenue. Since a task with higher demand does not guarantee higher profit rate, simply assign from the task with highest demand will fail to find the most profitable tasks.

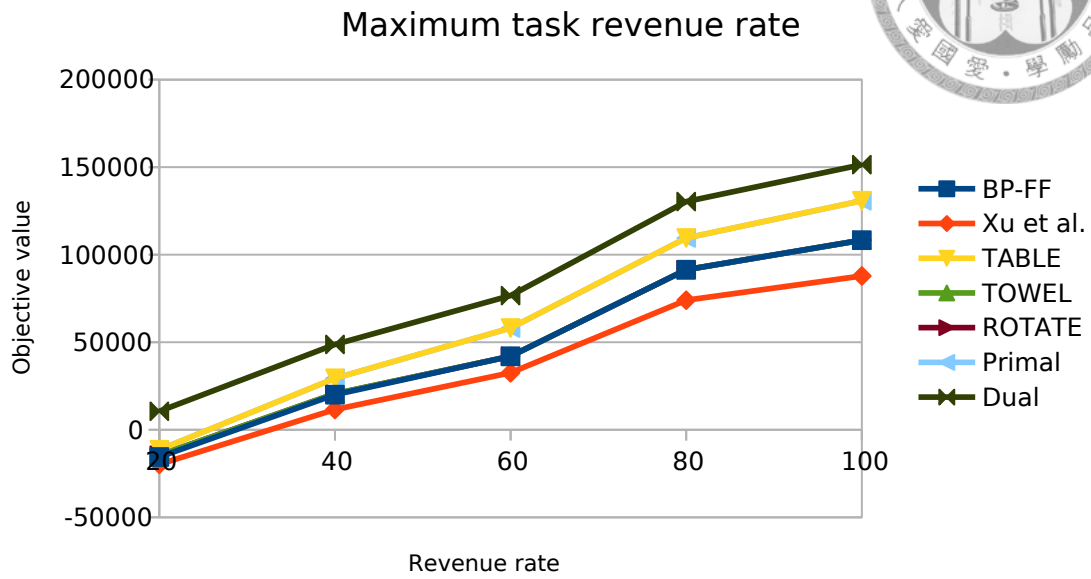


Figure 5.9: Expected task revenue rate

Table 5.9: Improvement ratio for task revenue rate

Scenario	Primal	BP-FF	Imp. ratio to BP-FF	BFD	Imp. ratio to BFD
20	-11452	-15571	26.45	-19752	42.02
40	29358	20009	46.72	11568	153.79
60	58092	41969	38.42	32548	78.48
80	109547	91449	19.79	73968	48.10
100	130840	108309	20.80	87868	48.91



Figure 5.10: Cost for turning a server on for an interval

5.7 Experiment on server cost rate

The last part of our experiments will go through the cost rate of the servers; that is, A and E in our model. The trend of the curve is similar to the task penalty section previously discussed: when cost rate is higher, the objective value will go down. We graph the result in Figure 5.10.

The cost rate and the objective value are in a linear relationship. The reason behind it is also similar to the task penalty: change in the cost rate will bring a determinable increase or decrease to the objective value because we can expect that the number of active servers is fixed, with other things being equal.



Table 5.10: Improvement ratio for server costs

Scenario	Primal	BP-FF	Imp. ratio to BP-FF	BFD	Imp. ratio to BFD
2	38353	29359	30.63	16818	128.05
4	-3257	-14866	78.09	-27307	88.07
6	-44168	-59091	25.25	-71432	38.17
8	-85038	-103316	17.69	-115557	26.41
10	-125727	-147541	14.79	-159682	21.26





Chapter 6

Conclusion and Future Work

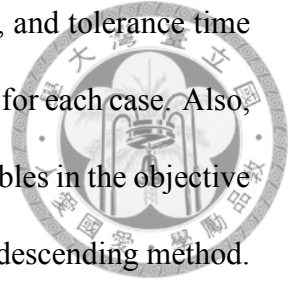
In this chapter, conclusion is given, and we also list some possible improvements that may be helpful to future researchers.

6.1 Conclusion

Task allocation strategy plays an important role for 5G operators, and the development of C-RAN allows us to design various strategies to serve the clients. In this thesis, we modelled the scenario as an optimization problem with constraints that may be encountered in real life and used the Lagrangian relaxation method, from which we derive the task assignment result. Moreover, in addition to the Lagrangian relaxation method, we also combined bin packing, task scheduling, and traffic shaping technique when considering the original task allocation problem in the proposed heuristic.

After that, multiple aspects are considered during the experiment part. We evaluated

the performance when task quantity, server quantity, processing time, and tolerance time are variable respectively, with statistical chart and interpretation given for each case. Also, we tried to find the change in the objective value when the given variables in the objective function changes. The performance was then compared with best-fit descending method. From the result, we can see that when the supply of resource is sufficient, BFD is able to produce a better result. However, when the resource is limited (as often the case in daily life), the Lagrangian multipliers are able to significantly improve the objective value, and when combined with other approaches, the outcome can be further enhanced.



6.2 Future work

In this thesis, we assume that every client can only stay at one server; however, we may further consider the situation where a client may stay on multiple servers throughout its lifetime. In such cases, migration happens. By migration, we can aggregate tasks from multiple servers into one (or fewer) server(s), which may further lower possible cost. On the other hand, migration cost may happen during the migration process. We suggest that future researchers may take the migration technique into consideration.

Also, we do not consider other miscellaneous costs that may occur when our procedure is run on real servers. For example, we do not consider the transmission delay, and in real life, clients may not come in the same pattern as we designed. The difference in the arrival pattern may affect the final result. Also, since the intervals being served need not be consecutive in our model, when the service process is interrupted by the system, the current state must be saved before it can be used later. The swap-in and swap-out cost

is also one of the costs that can be considered. If these criteria can be integrated to the model, we believe that the model will be much more closer to the real life situations.








References


- [1] N. C. Commission. (Apr. 2017). Telecommunication figures in 2016, [Online]. Available: http://www.ncc.gov.tw/english/files/16060/384_1830_170428_1.pdf.
- [2] C. V. N. Index. (Feb. 2016). Global mobile data traffic forecast update 2015–2020 white paper, Feb 2016, [Online]. Available: <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html>.
- [3] H. Dahrouj, A. Douik, O. Dhifallah, T. Y. Al-Naffouri, and M.-S. Alouini, “Resource allocation in heterogeneous cloud radio access networks: advances and challenges,” *IEEE Wireless Communications*, vol. 22, no. 3, pp. 66–73, 2015.
- [4] A. Osseiran, F. Boccardi, V. Braun, K. Kusume, P. Marsch, M. Maternia, O. Queseth, M. Schellmann, H. Schotten, H. Taoka, *et al.*, “Scenarios for 5G mobile and wireless communications: the vision of the metis project,” *IEEE Communications Magazine*, vol. 52, no. 5, pp. 26–35, 2014.


- 
- [5] M. Peng, Y. Li, J. Jiang, J. Li, and C. Wang, “Heterogeneous cloud radio access networks: a new perspective for enhancing spectral and energy efficiencies,” *IEEE Wireless Communications*, vol. 21, no. 6, pp. 126–135, 2014.
- [6] M. Agiwal, A. Roy, and N. Saxena, “Next generation 5G wireless networks: a comprehensive survey,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 1617–1655, 2016.
- [7] F. Han, S. Zhao, L. Zhang, and J. Wu, “Survey of strategies for switching off base stations in heterogeneous networks for greener 5G systems,” *IEEE Access*, vol. 4, pp. 4959–4973, 2016.
- [8] U. Dötsch, M. Doll, H.-P. Mayer, F. Schaich, J. Segel, and P. Sehier, “Quantitative analysis of split base station processing and determination of advantageous architectures for lte,” *Bell Labs Technical Journal*, vol. 18, no. 1, pp. 105–128, 2013.
- [9] J. Zhang, Y. Ji, X. Xu, H. Li, Y. Zhao, and J. Zhang, “Energy efficient baseband unit aggregation in cloud radio and optical access networks,” *Journal of Optical Communications and Networking*, vol. 8, no. 11, pp. 893–901, 2016.
- [10] M. Qian, W. Hardjawana, J. Shi, and B. Vucetic, “Baseband processing units virtualization for cloud radio access networks,” *IEEE Wireless Communications Letters*, vol. 4, no. 2, pp. 189–192, 2015.
- [11] A. Douik, H. Dahrouj, T. Y. Al-Naffouri, and M.-S. Alouini, “Coordinated scheduling for the downlink of cloud radio-access networks,” in *Communications (ICC), 2015 IEEE International Conference on*, IEEE, 2015, pp. 2906–2911.
- [12] O. Chabbouh, S. B. Rejeb, N. Agoulmine, and Z. Choukair, “Service scheduling scheme based load balancing for 5G/hetnets cloud ran,” in *Advanced Information*

Networking and Applications (AINA), 2017 IEEE 31st International Conference on, IEEE, 2017, pp. 843–849.



- [13] J. Li, D. Chen, Y. Wang, and J. Wu, “Performance evaluation of cloud-ran system with carrier frequency offset,” in *Globecom Workshops (GC Wkshps), 2012 IEEE*, IEEE, 2012, pp. 222–226.
- [14] H. Niu, C. Li, A. Papathanassiou, and G. Wu, “Ran architecture options and performance for 5G network evolution,” in *Wireless Communications and Networking Conference Workshops (WCNCW), 2014 IEEE*, IEEE, 2014, pp. 294–298.
- [15] G. Auer, V. Giannini, C. Desset, I. Godor, P. Skillermark, M. Olsson, M. A. Imran, D. Sabella, M. J. Gonzalez, O. Blume, *et al.*, “How much energy is needed to run a wireless network?” *IEEE Wireless Communications*, vol. 18, no. 5, pp. 40–49, 2011.
- [16] P. Demestichas, A. Georgakopoulos, D. Karvounas, K. Tsagkaris, V. Stavroulaki, J. Lu, C. Xiong, and J. Yao, “5G on the horizon: key challenges for the radio-access network,” *IEEE Vehicular Technology Magazine*, vol. 8, no. 3, pp. 47–53, 2013.
- [17] D. S. Johnson, “Fast algorithms for bin packing,” *Journal of Computer and System Sciences*, vol. 8, no. 3, pp. 272–314, 1974.
- [18] M. Yue, “A simple proof of the inequality $FFD(L) \leq 11/9 OPT(L) + 1$, L for the FFD bin-packing algorithm,” *Acta Mathematicae Applicatae Sinica (English Series)*, vol. 7, no. 4, pp. 321–331, 1991.
- [19] X. Xu, J. Zhang, Y. Ji, H. Li, R. Gu, H. Yu, and J. Zhang, “BBU aggregation for maximizing the resource utilization in optical-enabled cloud radio access net-

- works,” in *Optical Communications and Networks (ICOON), 2016 15th International Conference on*, IEEE, 2016, pp. 1–3.
- 
- [20] X. Wang, S. Thota, M. Tornatore, S.-S. Lee, H.-H. Lee, S. Park, and B. Mukherjee, “Green virtual base station in optical-access-enabled cloud-ran,” in *Communications (ICC), 2015 IEEE International Conference on*, IEEE, 2015, pp. 5002–5006.
- [21] X. Wang, S. Thota, M. Tornatore, H. S. Chung, H. H. Lee, S. Park, and B. Mukherjee, “Energy-efficient virtual base station formation in optical-access-enabled cloud-ran,” *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 5, pp. 1130–1139, 2016.
- [22] S. Burnwal and S. Deb, “Scheduling optimization of flexible manufacturing system using cuckoo search-based approach,” *The International Journal of Advanced Manufacturing Technology*, pp. 1–9, 2013.
- [23] S. D. Dao, K. Abhary, and R. Marian, “Optimisation of partner selection and collaborative transportation scheduling in virtual enterprises using ga,” *Expert Systems with Applications*, vol. 41, no. 15, pp. 6701–6717, 2014.
- [24] J. Wu, B. Cheng, Y. Shang, J. Huang, and J. Chen, “A novel scheduling approach to concurrent multipath transmission of high definition video in overlay networks,” *Journal of network and computer applications*, vol. 44, pp. 17–29, 2014.
- [25] C. Zhao, S. Zhang, Q. Liu, J. Xie, and J. Hu, “Independent tasks scheduling based on genetic algorithm in cloud computing,” in *Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on*, IEEE, 2009, pp. 1–4.

- 
- [26] W. Yu, T. Kwon, and C. Shin, “Multicell coordination via joint scheduling, beamforming, and power spectrum adaptation,” *IEEE Transactions on Wireless Communications*, vol. 12, no. 7, pp. 1–14, 2013.
- [27] K. C. Garikipati and K. G. Shin, “Scalable real-time transport of baseband traffic,” *arXiv preprint arXiv:1706.01160*, 2017.
- [28] A. Douik, H. Dahrouj, T. Y. Al-Naffouri, and M.-S. Alouini, “Coordinated scheduling and power control in cloud-radio access networks,” *IEEE Transactions on Wireless Communications*, vol. 15, no. 4, pp. 2523–2536, 2016.
- [29] R. D. Williams, “Performance of dynamic load balancing algorithms for unstructured mesh calculations,” *Concurrency and Computation: Practice and Experience*, vol. 3, no. 5, pp. 457–481, 1991.
- [30] D. R. Karger and M. Ruhl, “Simple efficient load balancing algorithms for peer-to-peer systems,” in *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, ACM, 2004, pp. 36–43.
- [31] L. Doddini Probhuling, “Load balancing algorithms in cloud computing,” *computing*, vol. 2, p. 4, 2013.
- [32] C. Ran, S. Wang, and C. Wang, “Optimal load balancing in cloud radio access networks,” in *Wireless Communications and Networking Conference (WCNC), 2015 IEEE*, IEEE, 2015, pp. 1006–1011.
- [33] M. L. Fisher, “The lagrangian relaxation method for solving integer programming problems,” *Management science*, vol. 27, no. 1, pp. 1–18, 1981.
- [34] ———, “The lagrangian relaxation method for solving integer programming problems,” *Management science*, vol. 50, no. 12_supplement, pp. 1861–1871, 2004.