Resource Scheduling through Data Allocation and Processing for Mobile Cloud Computing

by

HUI ZHAO

DISSERTATION

Presented to the Faculty of

Pace University

in Partial Fulfillment of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN

COMPUTER SCIENCE

PACE UNIVERSITY

May 2018

Abstract

With the rapid development of the computer software and hardware technologies, various mobile devices have been broadly applied in people's daily life, such as smart phones and smart watches. However, mobile devices usually have the disadvantages of weaker performance than traditional desktop computers. Cloud computing can efficiently overcome the weakness of mobile devices. *Mobile Cloud Computing* (MCC) is a paradigm that integrates cloud computing, mobile computing and wireless networks to bring rich computational resources to mobile users, network operators, and cloud computing providers. Making MCC efficient has become an important issue that can be addressed in two aspects. The first aspect is to improve data allocations via the proposed resource scheduling methods on mobile cloud servers. The other aspect is to increase the capacities and capabilities of data processing in wireless networks by optimizing resource scheduling. This proposed work aims to obtain an optimized mobile cloud computing systems that can enhance efficiency and reduce network traffics by using the proposed resource scheduling algorithms. For reaching this goal, we focus on the following three research problems: (1) how can we improve data processing performance through data allocation on mobile cloud servers by using dynamic programming? (2) how can we improve wireless network performance of MCC by reducing network traffics? (3) how can we save wireless power consumption of MCC by finding the optimal transfer plan? (4) how can we improve system performance of MCC by data allocation and processing? To solve the problems above, we design a series of approaches and algorithms, and the proposed methods have been partially examined in the experimental evaluations. The proposed approaches can reach the expected performance, according to the current experimental findings.

Contents

Chapter 1

Introduction

With the rapid development of the computer software and hardware technologies, various mobile devices have been broadly applied in people's daily life, such as smart phones and smart watches. Compare with traditional desktop computers mobile devices have the advantage in portability. However, mobile devices usually have the disadvantages of weaker performance than traditional desktop computers, including storage space, data processing ability, and stand-by time.

Cloud computing is a kind of Internet-based computing that provides shared resources to computers and other devices on demand. In cloud computing model, the service provider provide various services, such as storage, applications, and databases for user. These services often are provided by a group of servers. In cloud computing, users just choose what service they want. Cloud providers implement and maintain the services. Users do not need to care about the implementation details of services. Therefore, cloud computing can overcome the weakness of mobile devices efficiently. The improvement of mobile devices' data processing ability, storage space, and stand-by time will encounter technology bottleneck because of their limited size and requirement of portability. Moving data processing and data storage to cloud side can help mobile devices solve this problem.

Mobile Cloud Computing (MCC) is a paradigm that integrates cloud computing, mobile computing and wireless networks to bring rich computational resources to mobile users, network op-

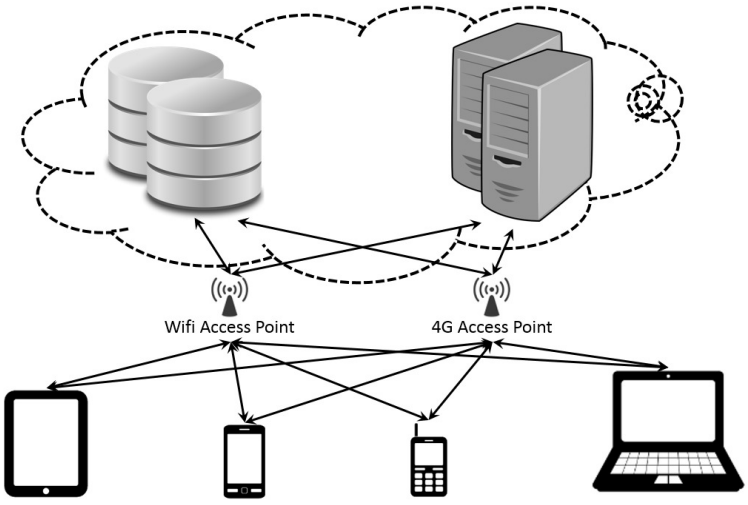


Figure 1.1: Architecture of Mobile Cloud Computing

erators, and cloud computing providers. The main goal of MCC is to improve the performance of mobile devices by integrating the three technologies, including cloud computing, mobile computing, and wireless networks. Improving the performance of MCC is an useful research on this background. Fig. 1.1 shows the brief architecture of MCC.

The mobile devices connect mobile cloud servers, which include processing cloud server and data cloud server, via wifi access point or 4G access point. In this scenario, the main function of mobile devices is providing a UI and showing the results that are transformed from cloud side in MCC. The complex data processing is done on the cloud side. Thus, the mobile devices can response the user's request and provide the high performance service to user without powerful computing ability. We can improve the performance of MCC in some aspects.

One is to improve the data processing performance of mobile cloud servers. Mobile cloud servers play an important role in an MCC environment. They are engineers of the whole MCC. The ability of data processing is an important factor for the performance of MCC. The response time will be reduced when the speed of the data processing on mobile cloud servers are efficient; then, the user experience will be improved.

The other is to improve the performance of wireless network. Wireless network is another important part in MCC. The execution time of mobile apps will be long when the wireless network

is not good even the cloud server can get the result in time. Users can receive benefit from reducing the wireless network response time in MCC environment. Thus, reducing the wireless network response time is another important issue for MCC.

Moreover, the power consumption of wireless network is heavy in modern smart mobile devices. Nowadays, multi-channel wireless communications are used on more and more intelligent mobile devices, such as smart mobile phones, smart watches, and tele-health devices. These multiple wireless communications have different energy costs and successful probability when they transfer data packets. Therefore, finding out an optimal transfer plan for saving wireless energy consumptions is an important issue for MCC, too.

Focus on these aspects, our research proposed some schemes to improve the performance of MCC via efficient resource scheduling techniques. There are various types of resources, such as computing resources, data resources, memory resource, energy resource, and network resources, etc. In different environments, resources have different meanings. For example, data processing time is a main resource when the task of data processing is busy. In addition, the network workload, as a manner of the networking resource, is a main deterministic factor for influencing the system performance. When the data processing job on the remote cloud server is small, the objective will become to making the volume of the data transmission less. Therefore, it is important to schedule different types of resources in a heterogeneous environment to improve the performance of MCC.

As one of the popular approaches supporting software-controlled on-chip memories, *Scratch-Pad Memory* (SPM) has been broadly implemented in a variety of industries while the workload of cloud servers larger and larger. The data accesses are controlled by a program stored in SPMs, which can process data by dynamic manipulations. On cloud side, we can schedule resources through data allocation to improve the performance of cloud servers that employ SPM. We proposed an approach to find the optimal data allocation plan to minimize the data processing cost to improve the cloud server performance by using dynamic programming.

Wireless network resource is another important resource in MCC environment. There are many repeated data in the wireless network flow where the mobile devices communicate with the cloud servers. Lots of wireless network traffic will be saved if we can reduce these repeated data in

network flow. Reducing network traffic can efficiently shorten wireless network response time. Scheduling resource through data processing to improve the performance of MCC is our research goal in network aspect.

The basic idea is finding these repeated data and avoid transfer them repeatedly. This include two different situation. One is that the receivers start the transmission. The other is that the senders start the transmission actively.

Moreover, energy is an important resource for mobile devices, such as smart mobile phones, smart watches, and tele-health devices. Multi-channel wireless communications are used on more and more intelligent mobile devices currently. These multi-channel wireless communications have different energy costs and successful probabilities when transferring data packets. It this necessary to find an optimal solution to saving the power cosumption for mobile devices in the context of MCC and design an intelligent energy-aware communication strategy.

According to the analysis above, we have proposed four sub-research problems that are described as follows:

1. How can we improve data processing performance through data allocation on mobile cloud servers by using dynamic programming (i.e., data allocation in mobile cloud)?

2. How can we improve wireless network performance of MCC by reducing network traffics?

3. How can we save wireless power consumption of MCC by finding the optimal transfer plan(i.e., data processing in mobile cloud)?

4. How can we improve system performance of MCC by data allocation and processing (i.e., data allocation and processing in mobile cloud)?

The structure of this research project is shown in Figure 1.2. This dissertation is organized as the following order: In Chapter 1, we introduce the research background , our contribution and the architecture of this dissertation.

In Chapter 2, we describe the problem "how to schedule data resources through data allocation on mobile cloud servers that implement SPMs" and give the solution of this problem. Software-controlled on-chip memories, *Scratch-Pad Memory* (SPM) is a novel memory technology that uses

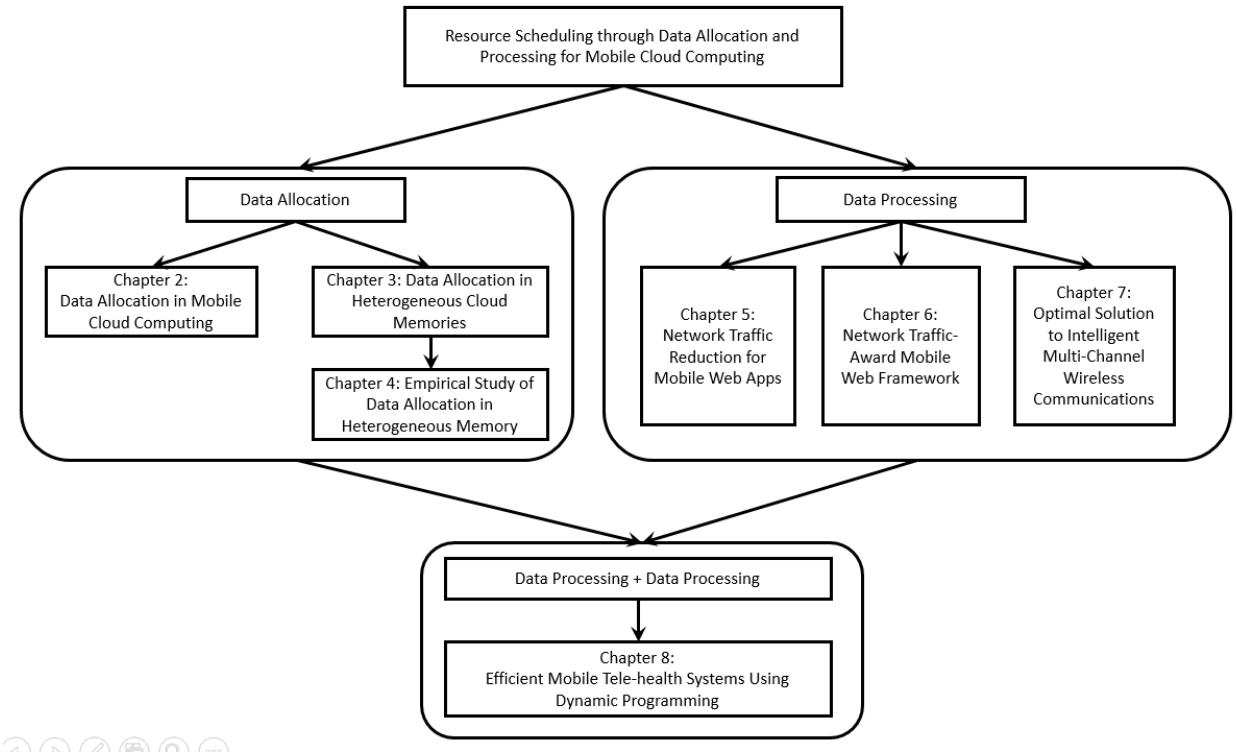


Figure 1.2: Structure of Research Project

on cloud servers and embedded systems. Allocate data into different memories of SPMs appropriate can improve data processing performance of mobile cloud servers. The main contribution of this chapter are threefold. First, we propose a novel approach for solving the data allocation problem with multiple dimensional constraints for heterogeneous memories in SPMs. This approach supports the designs of the complex SPM systems considering all influencing elements. Second, we propose a novel algorithm to solve the problem of data allocations in SPMs, which can be applied in solving other problems having the similar big data scenarios. Finally, the proposed algorithm offers a method that can produce global optimal solutions that are executed by mapping local optimal solutions and using dynamic programming. In summary, we can improve the performance of cloud servers that uses SPMs by optimal data allocations using dynamic programming.

In Chapter 3, we discuss the problem "how to schedule data resources through data allocation on heterogenous mobile cloud memories" and give the solution of this problem by using dynamic programming. Current mobile cloud systems usually have cloud server pools, which have a group of cloud server to provide service. The performance of cloud memories usually are varied. Finding

out an optimal data allocation plan for these heterogenous cloud memories is an important problem. We give an algorithm to find the optimal solution. Major work includes developing models, designing algorithms and evaluating the proposed methods.

In Chapter 4, we present an empirical study focusing on data allocations in heterogeneous memory. In this chapter, our study has evaluated a few evaluated a few typical data allocation approaches for heterogeneous memory. The work also provides a review on the evaluated approaches, which include heuristic algorithm, dynamic programming, and a few active resource scheduling algorithms. The main findings of this research can provide memory producers and heterogeneous memory researchers with a quantitative support.

In Chapter 5 and Chapter 6, we discuss "how to schedule network traffic resources for mobile web app" and give the solution of the problem at the mobile web app level and mobile web platform level, respectively. Main contributions of this part are threefold. First, we propose a novel development model for high-performance mobile systems using pre-cache. That is a situation in which receiver actively starts a transmission. Second, we propose an algorithm to solve the problem of network traffics reduction for mobile web apps. Using our proposed algorithm, PATAS, can efficiently save networking traffics. Third, we propose a model to provide a higher-level upgradable capability increasing the networking communications efficiencies and reducing network traffics. Finally, we propose an approach that is flexible for serving new computing service needs.

In Chapter 7, we discuss "how to schedule network channel to save energy consumption for efficient mobile devices". Multi-channel wireless communications are used on more and more intelligent mobile devices, such as smart mobile phones, smart watchs, and tele-health devices. These multiple wireless communications have different energy costs and successful probability when they transfer data packets. The main contribution of this part are summarized as follows: First, we present an optimal solution to data transfers using the multi-channel method. The creation of the optimal solution considers the energy cost, execution time, and success rate and the output is generated by implementing our proposed dynamic programming. The objective is to minimize the total energy cost with a relative high success rate when the total execution time length is fixed. Second, our work focuses on saving energy and formulates the problem by involving a number of

crucial elements. The proposed problem is an NP-hard problem. Our approach can outcome an optimal solution by partially solving the proposed NP-hard problem.

Finally, Chapter 8 discusses how to schedule data resource and network traffic resources for efficient mobile tele-health Systems. Currently, we propose a novel differential schema for high performance big data tele-health systems using pre-cache. That is a situation that sender start transmission actively. Using our proposed algorithms, DFA and DASA, can efficiently save the networking traffics.

In summary, this work proposes a series of models and algorithms to reduce the computing resource costs. The proposed approaches covers a few aspects, including networking traffics, computational efficiency, communication workloads, and energy consumptions. The crucial method proposed by this work was using resource scheduling algorithms to increase the efficiency of data allocations and enhance the entire data processing performance. It has been proved that the network traffics could be dramatically lowered down when the proposed approach is applied, which matched the expected design goal.

Chapter 2

Data Allocation in Mobile Cloud Computing

The data processing performance is an important factor for Mobile Cloud Computing. This chapter focuses on finding out the method of improving the data processing performance of mobile cloud servers. The mechanism employs SPMs and applies dynamic programming.

Multiple constraints in SPMs are considered a problem that can be solved in a nondeterministic polynomial time. In this chapter, we propose a novel approach solving the data allocations in multiple dimensional constraints. For supporting the approach, we propose a novel algorithm, which is designed to solve the data allocations under multiple constraints in a polynomial time. Our proposed approach is a novel scheme of minimizing the total costs when executing SPM under multiple dimensional constraints. Our experimental evaluations have proved the adaption of the proposed model that could be an efficient approach of solving data allocation problems for SPMs.

2.1 Problem Description

The dramatical booming requirements of high performance computing systems have been driving multi-core system designs for cloud computing in recent years [1, 2, 3]. As one of the popular approaches supporting software-controlled on-chip memories, SPM has been broadly implemented

in a variety of industries while the workload of cloud servers larger and larger. An important benefit of using SPMs is reducing the total operating cost by allocating data for having less hardware overhead and more data controls [4, 5]. The data accesses are controlled by a program stored in SPMs, which can process data by dynamic manipulations. For gaining an efficient data processing, the critical issue of the cost optimizations is designing an approach of achieving optimal data allocations. Focusing on this issue, this chapter proposes a novel approach named *Optimal Data Allocation in Heterogeneous Memories* (ODAHM) model, which sights at minimizing the total costs of SPMs by organizing and controlling the data allocations.

The recent development of the heterogeneous memories mainly addressed the minimizations of the costs by balancing different dimensional consumptions, such as energy, performance, and time constraints [6, 7, 8]. The operating principle is that the input data are mapped onto difference spaces in SPMs. The challenging aspect of using this mechanism is that it is hard to guarantee real-time performance within the desired cost scope due to the complicated allocation processes [9, 10]. The performance of heterogeneous memories can hardly reach the full-performance unless the efficient data allocation approach is applied. This restriction will become even more challenging when the data size turns into larger and the amount of the cost dimensions gets greater. Therefore, we consider this restriction a critical issue in improving heterogeneous memories in SPM and propose our approach to solve the data allocation problems with multiple cost dimensions.

The proposed model, ODAHM, defines the main operating procedure and manipulative principle, which deems multiple constraints influencing the costs while the data are allocated to memories. We modelize the operation of data allocations into a few steps, which include defining constraint dimensions, mapping the costs for each constraint, and producing optimal data allocation scheme according to the inputs. Implementing ODAHM can enable an advanced data allocation strategy for SPM since more impact factors can be involved for saving costs.

For reaching this goal, we propose an algorithm, *Optimal Multiple Dimensional Data Allocation* (OMDDA), which is designed to solve multiple dimensional constraints data allocation problem. This algorithm uses dynamic programming and produces the optimal solution synchronously under a few constraints. The operation principle of this algorithm is using a deterministic approach

to point at the cost caused by the corresponding constraints, which are mapped in the table. We produce local optimal solutions for each constraint and generate a global optimal solution deriving from the outcomes of the local optimal solutions.

Main contributions of this chapter are threefold:

1. We propose a novel approach for solving the data allocation problem with multiple dimensional constraints for heterogeneous memories in SPMs. This approach supports the designs of the complex SPM systems considering all influencing elements.

2. This chapter proposes a novel algorithm to solve the problem of data allocations in SPMs, which can be applied in solving other problems having the similar big data scenarios.

3. The proposed algorithm offers a method that can produce global optimal solutions that are executed by mapping local optimal solutions and using dynamic programming.

2.2 Related Work

We have reviewed recent research work of data allocations in heterogeneous memories from different perspectives in this section, such as reducing energy costs and increasing working efficiency. The increasing demands of the Internet-based applications have driven a higher-level of memory requirements [11, 12, 13, 14]. First, it has been proved that the power leakage consumption is a critical issue for heterogeneous memories with large scaled transistors due to the different memory capacities [15, 1]. This has resulted in obstacles in applying heterogeneous memories since allocating data to different memories needs to assess various costs taken place during a few phases, such as data processing and data moves.

Recent research has been focusing on reducing costs using different techniques in various fields. Addressing the physical operating environment, a proposed approach was using temperature-aware data allocations in order to adjust the workload distributions between cache and SPM [16]. The optimization method was applying an energy-aware loop scheduling algorithm. The energy could be saved when the loop scheduling was improved by retiming-based methods. Meanwhile,

considering the operating system environment, an approach was proposed to have an integrated real-time/non-real-time task execution environment for separately running the real-time or non-real-time nodes on OS and Linux [17, 18]. This approach could reduce the total costs through a selective execution. However, the proposed approaches above mainly focused on software level optimizations, even though the energy consumptions and other costs can be managed or controlled by a software-based solution. There is little relationship with the manner of data allocations to memories.

Moreover, the volatile-related features of the memories provide optimizations with optional choices. One advantage of using heterogeneous memories is that the volatile memories can be combined with non-volatile memories, which can lead to saving energy [19]. The benefits of using non-volatile memories include low power leakage and high density, but the unbalanced usages and inefficient tasks scheduling [20]. A solution [21] was produced to being an efficient scheduling method by optimizing the global data allocations that are based on the regional data allocation optimizations. This approach can be further improved when more elements are considered. Our proposed approach can solve the problem covering multiple elements.

Next, for further optimizations, other constraints are also considered, such as reducing latency, increasing success probabilities, and hardware capacities. An approach [22] solving multidimensional resource allocations has been proposed, which combined a few low-complexity sub-optimal algorithms. This mechanism used the principle of integrating a few suboptimal solutions to form an adoptable solution. Our proposed scheme distinguishes from this approach since we generate global optimal solutions deriving from the local optimal solutions in different dimensions.

In addition, a genetic algorithm was proposed for data allocations of hybrid memories by configuring SRAM, MRAM, and Z-RAM [23]. This algorithm was designed to enhance the performance of SPM by dynamically allocate data to different memories having various usage constraints. Combing this genetic algorithm with dynamic programming, the memory costs can be reduced in terms of power consumption and latency [6]. Despite the memory performance can be increased, the implementations can only process the limited constraint dimensions due to the time complexity restrictions.

Another research direction focused on reducing costs on *Phase-Change Memory* (PCM) that could consist of multiple memories on different levels of the memory hierarchy [24, 25]. The methods used in PCM are similar to heterogeneous memories in SPM. One approach was partitioning and allocating data to the multitasking systems depending on the memory types [26, 27]. However, the solutions based on this research direction could hardly solve the problem of the computation complexity when aiming gain an optimal solution.

Furthermore, allocation optimizations have also been applied in saving energy consumptions, such as energy storage within the distributed networks [28]. Another approach of reducing the complexity was using the weighted sum of the usage from multiple channels [29]. Nonetheless, most optimizations could solve partial of the complexity problem within the constraints.

In summary, our approach is different from most prior research achievements. The proposed approach is a scheme of producing optimal solutions for multiple dimensional heterogeneous memories. We further expand the condition constraint from a limited amount to multiple dimensions.

2.3 Motivational Example of ODAHM

In this section, we give a simple example of using our proposed approach to process data allocations to heterogeneous memories. In this example, assume that there are 4 ZRAM and 2 SRAM available. Main memory is always available for data. There are 7 input data that required different read and write accesses. The output is a data allocation plan that requires the minimum total cost. Table 2.1 represents the cost differences of data allocations to each memory.

Operation		SRAM	ZRAM	Main
Read		2	3	75
Write		2	7	75
Move	SRAM	0	10	72
	Z-RAM	6	0	72
	Main	70	76	0

Table 2.1: Cost differences for data allocations to heterogeneous memories.

Data	Read	Writes
A	5	2
B	10	8
C	8	5
D	4	4
E	2	10
F	3	15
G	15	4

Table 2.2: The number of the memory accesses.

Table 2.2 displays the number of the memory accesses, including *Read* and *Write*. 7 input data are A, B, C, D, E, F, and G. For instance, data A require 5 reads and 2 writes, according to the table. Our example's initial status is to allocate $A \rightarrow Main$, $B \rightarrow SRAM$, $C \rightarrow Main$, $D \rightarrow ZRAM$, $E \rightarrow Main$, $F \rightarrow Main$, and $G \rightarrow Main$.

Table 2.3: Mapping the costs for heterogeneous memories.

Data	SRAM	ZRAM	Main
A	84	105	525
B	36	96	1422
C	96	135	975
D	22	40	672
E	94	152	900
F	106	190	1350
G	108	149	1425

Based on the conditions given by Table 2.1 and 2.2, we map the costs of data allocations to heterogeneous memories in Table 2.3. For example, as shown in the table, data A costs 84 when it is allocated to SRAM, switched from Main memory. We are going to use Table 2.3 to calculate the total cost after the allocations. For obtaining the final optimal data allocation plan, we will produce a *D Table* showing the optimal data allocation plan. The generation process is given as follows, which consists of a few steps.

525 (Main)	105 (ZRAM)
84 (SRAM	

Table 2.4: Allocation cost for data A.

A		0	1	2	3	4	ZRAM
SRAM	0	525	105				
	1	84					
	2						

Table 2.5: A partial D Table.

First, we only consider one input data A. Deriving from Table 2.3, we produce a four-cell grid that is shown Table 2.4. As shown in this grid, we mark the memory to the corresponding cost. Meanwhile, the D Table generation starts from adding Table 2.4 into the D Table. Table 2.5 represents the manner of the data insertions in D Table. As displayed in the table, A refers to the new added data A. In the first row, 0-4 means the availability of ZRAM. Correspondingly, 0-2

means the availability of SRAM in the second column. For instance, 525 is associated to (0, 0), which means neither ZRAM nor SRAM is available for data A. The cost of data A is 525 in this situation, which is allocated to Main memory.

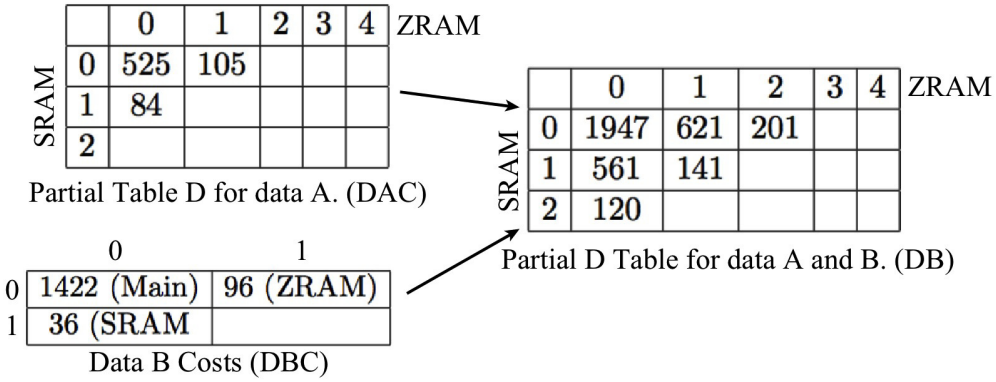


Figure 2.1: Generating partial D Table for data A and B deriving from Table 2.3 and 2.5.

Next, we consider adding data B to continue generating D Table. Fig. 2.1 represents the process of adding Data B into D Table. We mark the table showing the partial Table D with data A cost as DAC, table showing data B costs as DBC, and partial D Table for data A and B as DB. Since the maximum cases for each cell is 3, we alternative the allocation plan with the lowest cost in each cell. For calculating the values, we use the memory availability values as coordinates. For example, DB(1, 1) refers to 141 in DB; DBC(1, 0) refers to 96 (ZRAM) in DBC. Therefore, the minimum cost for each cell in DB, DB(i, j) can be gained by $min\langle$[DBC(0, 0)+DAC(i, j)], [DBC(0, 1), DAC(i, j-1)], [DBC(1, 0), DAC(i-1, j)]$\rangle$. For instance, DB(1, 1) = $min\langle$[DBC(0, 0)+DAC(1, 1)], [DBC(0, 1)+DAC(1, 0)], [DBC(1, 0)+DAC(0, 1)]$\rangle$=$min\langle$[1422+0], [36+105], [96+84]$\rangle$=$min\langle$1422, 141, 180$\rangle$=141.

Moreover, we add all other data by using the same method. A completed D Table is shown in Table 2.6. A, B, C, D, E, F, and G refer to the order of adding data. We bold the number that is corresponding to the optimal solution in this example. According to the table, the lowest data allocation cost is 1143 after data G are added to the table. Based on this finding, we move backward obtain all data allocations, which are 1143→994→888→736→696→561→525. We can gain the data allocation plan for each data from the calculation method mentioned above. Therefore, the

Table 2.6: D Table. The optimal data allocations are bolded.

A		0	1	2	3	4	ZRAM
SRAM	0	**525**	105				
	1	84					
	2						

B		0	1	2	3	4	ZRAM
SRAM	0	1947	621	201			
	1	**561**	141				
	2	120					

C		0	1	2	3	4	ZRAM
SRAM	0	2292	1596	756	336		
	1	1536	**696**	276			
	2	657	237				

D		0	1	2	3	4	ZRAM
SRAM	0	3594	2268	1428	796	376	
	1	2208	1368	**736**	316		
	2	1329	697	277			

E		0	1	2	3	4	ZRAM
SRAM	0	4494	3168	2328	1580	948	
	1	3108	2268	1520	**888**	468	
	2	2229	1462	830	410		

F		0	1	2	3	4	ZRAM
SRAM	0	5844	4518	3358	2518	1770	
	1	4458	3274	2434	1686	1054	
	2	3214	2374	1626	**994**	574	

G		0	1	2	3	4	ZRAM
SRAM	0	7269	5943	4667	3507	2667	
	1	5883	4607	3423	2583	1835	
	2	4566	3363	2523	1775	**1143**	

data allocation plan is A→Main, B→SRAM, C→ZRAM, D→ZRAM, E→ZRAM, F→SRAM, G→ZRAM.

The detailed mechanism of our approach is given in Sections 2.4 and 2.5.

2.4 Concepts and the Proposed Model

We illustrate our proposed model by defining the main target problem and concepts used in the model.

2.4.1 System Definition

We assume that the data partition process of the input data completes before the data allocations. For generating the allocation plan, the target cost for saving purpose is determined and the information is available, such as computing cost per Read or per Write. The research problem is defined by the following Definition 1.

Definition 1 Minimizing Cost of Data Allocation on Heterogeneous Memories (MC-DAHM)

Problem: *Inputs include a chain of data, initial data locations, the number of available memories for each type, the number of the memory access for each data, the cost of each data for Read, Write, and Move. The output is a data allocation plan. The research problem is finding out the data allocation plan that can minimize the total cost.*

In this problem, inputs include the number of data, initial data locations, the number of memory types, the number of available memories for each type, the number of memory accesses for each data, the cost of data operations, including Read, Write, and Move. Our aim is to produce a data allocation plan that allocate the input data to the available memories by using the minimum total cost. The total cost calculation method is given in Equation 2.1.

$$Cost_{total} = \sum_{i=1}^{n} (D_i \times RC_j + D_i \times WC_j + D_i \times MV_j) \tag{2.1}$$

Assume that there are n input data D. As shown in Equation 2.1, $Cost_{total}$ refers to the total cost of data allocations. $D_i \times RC_j$ means the Read cost for a certain data and RC_j is the Read cost for data D_i each time. $D_i \times WC_j$ means the Write cost of each data and WC_j is the Write cost of

D_i at each time. Similarly, $D_i \times MV_j$ is the Move cost and MV_j is the Move cost of D_i at each time. The total cost is summing up all n input data' costs.

2.4.2 Optimal Data Allocation in Heterogeneous Memories (ODAHM)

In our proposed model, there are mainly three steps for generating the data allocation plan. First, when the data were loaded, the cost of each data at each memory will be calculated. Three cost operations will be calculated, including Read, Write, and Move. The results of the costs will be mapped into a table. A sample is given in Section 2.3 as Table 2.3. Second, a D Table will be generated for gaining the optimal data allocation plans by using dynamic programming. A sample of D Table is given in Table 2.6. The creation process of D Table is the critical component of our model. We give the mathematical formulations about D Table generations in the late part of this section and describe computing algorithm in Section 2.5. Finally, we find out the optimal data allocation plan once the D Table is created.

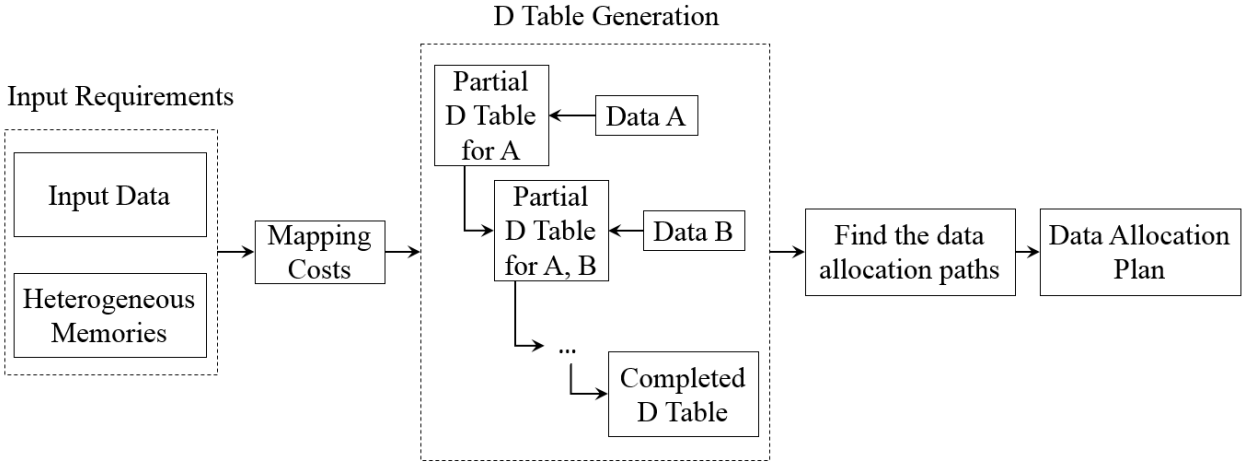


Figure 2.2: Operating process of the proposed ODAHM model

Fig. 2.2 represents the operating process of the proposed ODAHM model that shows three main steps. As shown in the figure, input requirements include the information about both input data and memories, which will be used for mapping each data cost at each memory. At the phase of D Table generation, we create the table by adding data in a succession manner. Once the D

Table is accomplished, we can find the data allocation paths to determine the allocation plan. The detailed method description is given in Section 2.5.

2.5 Algorithms

In this section, we propose the algorithm to generate the optimal data allocation using dynamic programming. We define a few definitions used in our algorithm, including *B Table* defined by Definition 2 and *D Table* defined by Definition 3.

Definition 2 B Table: *we use a multiple-dimensional array to describe a table that stores the cost of each data at each memory. Inputs include each data's cost when it is assigned to a memory. The output will be a multiple-dimensional array. BTable[i][j][k] that represents the cost when* $data_i$ *use* j *memories and* k *memories.*

Definition 3 D Table: *we use a multiple-dimensional array to describe a table that shows the total costs of data allocations by storing all optimal task assignment plans under all possible constraint. Input is the B Table. The output will be a multiple-dimensional array storing all optimal task assignments as well as their optimal costs. DTable[i][j][k] that represents the total minimum cost when allocating* $data_1$, $data_2$, ..., $data_i$ *into* j *memories and* k *memories.*

Definition 4 Supplemental cell: *we use a multiple-dimensional array to describe a table that shows the total costs of data allocations by storing all optimal task assignment plans under all possible constraint. Input is the B Table. The output will be a multiple-dimensional array storing all optimal task assignments as well as their optimal costs.*

DTable[i][j][k] that represents the total minimum cost when allocating $data_1$, $data_2$, ..., $data_i$ *into* j *memories and* k *memories.*

Algorithm 1 Optimal Multiple Dimensional Data Allocation (OMDDA) Algorithm

Require: The B Table $BTab$

Ensure: The optimal data allocation

```
Input the B Table
DTab[0] ← BTab[0]
for i ← 1 to Count(D) − 1 do
    /*Count(D) represents the count of data. */
    for j ← 1 to Size(S) do
        /*Size(S) represents the size of SRAM. */
        for j ← 1 to Size(Z) do
            /*Size(Z) represents the size of ZRAM. */
            minCost ← ∞
            for ∀cell BTab[i][b_s][b_z] in BTab[i] do
                if ∃DTab[i-1][j − b_s][k − b_z] then
                    cost ← BTab[i][b_s][b_z] in BTab[i] + DTab[i-1][j − b_s][k − b_z]
                    if cost < minCost then
                        minCost ← cost
                        update the data allocation plan
                    end if
                end if
            end for
        end for
    end for
end for
RETURN the optimal data allocation by searching the D Table according to the memory condition.
```

The algorithm 1 shows our proposed algorithm and the main phases of our algorithm include:

1. Input *B Table*.
2. Start generating D Table. We copy BTab[0] to DTab[0] to produce the first partial D Table, which represents add data[0] to D Table.
3. We add data[1] to the D Table by calculating the data allocation costs based on the partial D Table generated by the last Step 2 and partial B Table BTab[1]. The method is to find the minimal cost from the sums of BTab[1] cells and their supplemental cells in DTab[0].
4.
5. Update data allocation plan according to the minimal costs.
6. Add all data by applying the same method used from Step 3 to Step 5 until the D Table is generated.
7. Find the optimal data allocation by searching the lowest cost in D Table according to the memory condition. Output the task assignment plan.

2.6 Experiments and the Results

The multiple dimensional dynamic programming algorithm, OMDDA, is designed to find the optimal data allocation for multiple dimensional heterogeneous memories. Our experiments accomplished comparisons among Random algorithm, Greedy algorithm, DAHS algorithm [15], and our proposed algorithm. The comparisons focused on comparing four algorithms' data allocation costs and time consumptions. We implemented our experiments on a Host that ran Windows 8.1 64 bit OS. The main hardware configuration of the Host was: Intel Core i5-4210U CPU, 8.0 GB RAM.

We configured three experimental settings for assessing the performance of the proposed algorithm. There were three mainly variables in our experiments: the number of the data kind, the data size of every kind data, and the number of the available memories.

Three experimental settings are:

- Setting 1: We configured 7 kinds of data, each kind data has 1M size, 2M available SRAM and 4M available ZRAM.
- Setting 2: We configured 8 kinds of data, each kind data has 1G size, 1G available SRAM and 5G available ZRAM.
- Setting 3: We configured 10 kinds of data, each kind data has 1G size, 1G available SRAM and 6G available ZRAM.

2.6.1 Experimental Results

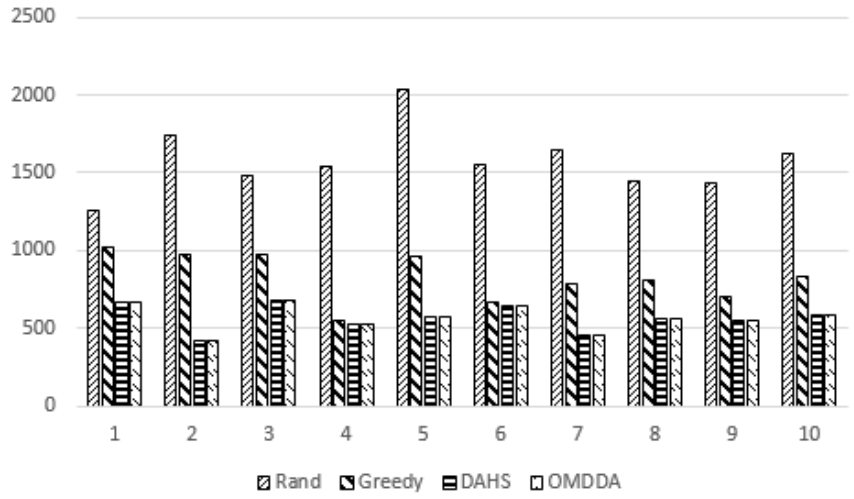


Figure 2.3: Comparisons of costs under Setting 1

The Fig. 2.3 showed that the comparison among Random algorithm, Greedy algorithm, DAHS algorithm and OMDDA algorithm under setting 1. As shown in the figure, our proposed algorithm could accomplish all tasks with requiring the lowest cost. The Random algorithm demanded the largest cost because there were great chances to allocate data to those memories inquiring bigger costs. Greedy algorithm usually needed more costs than DAHS and OMDDA but there were some chances for Greedy to produce optimal solutions. The DAHS and OMDDA algorithm have the same solution since they both are optimal data allocation algorithm.

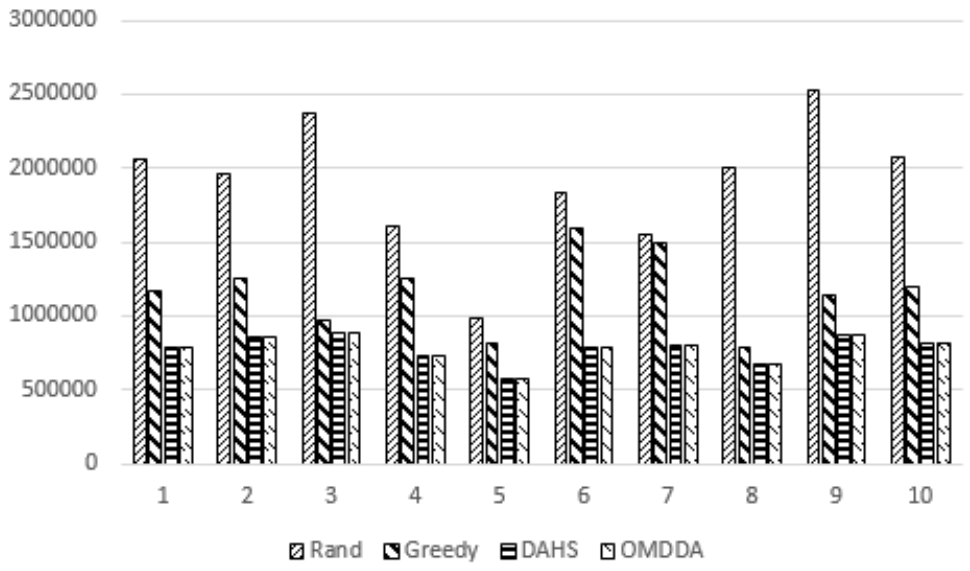


Figure 2.4: Comparisons of costs under Setting 2

Moreover, Fig. 2.4 showed that the comparison among Random algorithm, Greedy algorithm, DAHS algorithm and OMDDA algorithm under setting 2. Our proposed scheme had a better performance than both Random and Greedy algorithms under this setting.

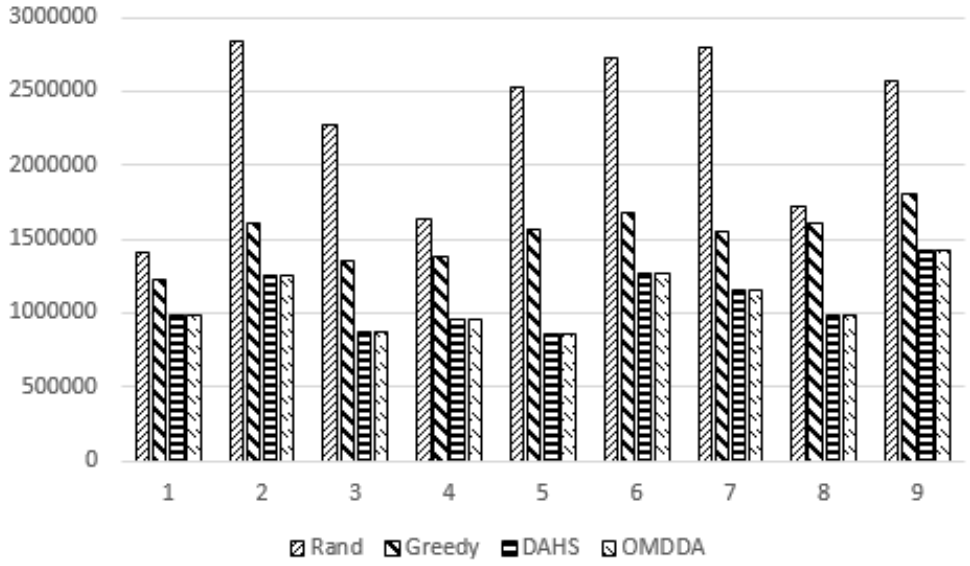


Figure 2.5: Comparisons of costs under Setting 3

Fig. 2.5 showed the comparisons among Random, Greedy, DAHS and OMDDA algorithms under setting 3. Our proposed algorithm had an advantage in saving costs, according to the display

of the figure. The vertical axis showed the cost levels for each algorithm. The advantage was obvious under this setting.

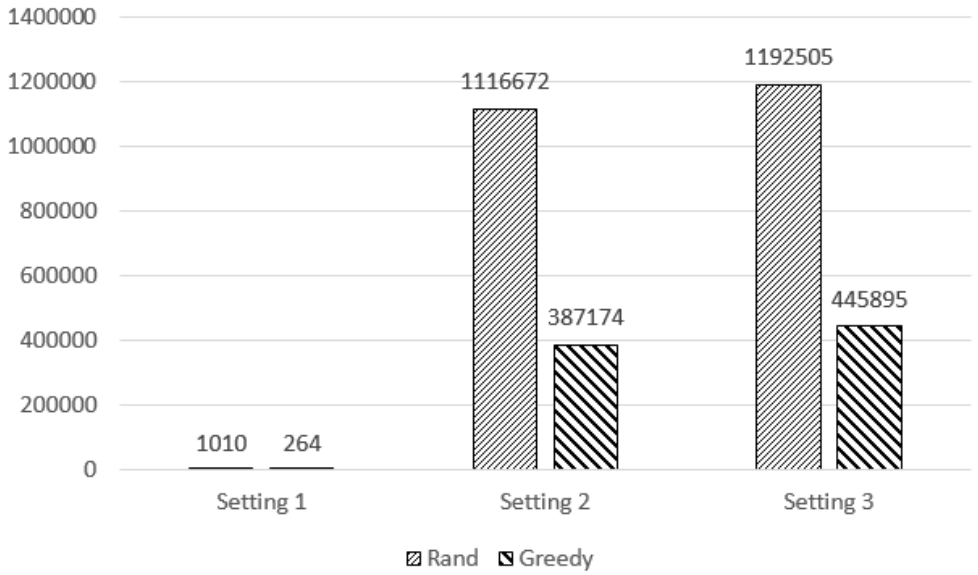


Figure 2.6: Comparisons of the saved costs showing the difference between OMDDA and Random or Greedy algorithms

Fig. 2.6 showed how many costs OMDDA saved comparing with Random algorithm and Greedy algorithm under every setting. As shown in Fig. 2.6, we could obtain the finding that the data size had a nearly positive relationship with the saved cost. It proved that our proposed scheme was suitable for the data processing in the big data context. The volume of the saved cost will become greater while the data volume becomes larger.

Fig. 2.7 showed the comparisons of execution time among Random algorithm, Greedy algorithm, DAHS algorithm and OMDDA algorithm. As shown in Fig. 2.7, we can see our proposed algorithm has less execution time than DAHS even though they have same optimal solution.

In summary, our experiments prove that the proposed algorithm, OMDDA performed better than Random algorithm and Greedy algorithm under all configured settings. Compare to DAHS, OMDDA algorithm has less consumption time to generate the optimal solution. Our findings also showed that the OMDDA had a better performance when the data volume is in a bigger size.

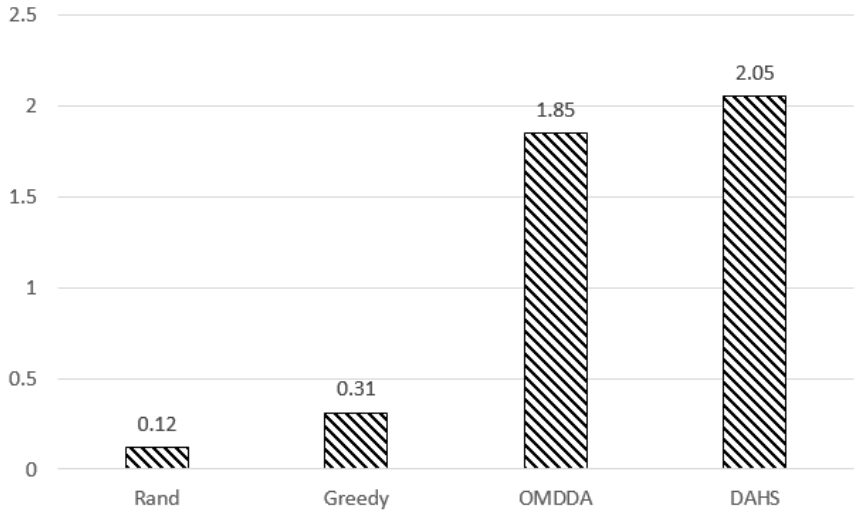


Figure 2.7: Comparisons of execution time among Random, Greedy, DAHS and OMDDA algorithms

2.7 Conclusions

This chapter addressed the issues of using heterogeneous memories to enable high performance big data processing and proposes an optimal solution that could generate the data allocation plans with the lowest cost. The proposed model was named as *Optimal Data Allocation in Heterogeneous Memories* (ODAHM) and the main algorithm in the model was *Optimal Multiple Dimensional Data Allocation* (OMDDA) algorithm. We had implemented experimental evaluations to examine our proposed model's performance and the collected results showed that our model had a great advantage in saving cost due to the optimal data allocations.

Chapter 3

Data Allocation in Heterogeneous Cloud Memories.

Our approach that is mentioned in above chapter not only can be used to improve performance of mobile cloud server that employ SPMs, but also can be used on more complex situations, such as heterogeneous cloud memories. In this chapter, we implement the idea on heterogeneous cloud memories.

3.1 Problem Description

Cloud computing is used broadly in recent years. Heterogeneous clouds are helpful for improving performance when the data processing task becomes a challenge in big data within different operating environment [30, 31]. Combining heterogeneous embedded systems and could computing can receive lots of benefits within big data environments. Currently, cloud-based memories usually deploy non-distributive manner on cloud side. Non-distributive manner has some limitation, such as overload energy and low performance resource allocation mechanism [32, 33]. These limitation restrict the implementation of cloud-based heterogeneous memories. This chapter address on this

issue and propose an approach to find out the optimal data allocation plan for minimizing total costs of the distributed heterogeneous cloud memories in mobile cloud systems.

Processors and memories that provide processing services are hosted by individual cloud servers in current cloud infrastructure. The challenge of this type of deployment is that deploying distributed memories in clouds faces a few restrictions [34]. Allocating data to multiple cloud-based memories will meet obstacles because of the various impact factors and parameters [35]. The various configurations and capabilities can limit the performance of the whole systems because the naive task allocation is inefficient to operate the entire heterogeneous cloud memories. This means that optimizing the cost of using heterogeneous cloud memories is restricted by the multiple dimensional constraint conditions.

Addressing on this issue, we propose a novel approach that named *Cost-aware Multi Dimension Data Allocation Heterogeneous Cloud Memories* (CM2DAH). The goal of CM2DAH is to find out the optimal data allocation plan to reduce data processing cost through heterogeneous cloud memories for efficient *Memory-as-a-Service* (Maas). The cost can be any resource that spend in the operation of MaaS, such as execution time, power consumption, etc.. The basic idea of the proposed approach is using dynamic programming to minimize the processing costs via mapping data processing capabilities for different datasets inputs. For supporting the proposed approach, we propose an algorithm, which named *Cost-aware Multi Dimension Dynamic Programming* (CM2DP) Algorithm, This algorithm is designed to find out the optimal total costs by using dynamic programming.

The main contribution of this chapter are twofold:

1. We propose a novel approach for solving the data allocation problem with multiple dimensional constraints for heterogeneous cloud memories. The proposed approach is an attempt in using heterogeneous cloud memories to minimize total cost by dynamically allocate data to various cloud resources.

2. We propose an algorithm to solve the problem of data allocations in heterogeneous cloud memories. The proposed algorithm can produce global optimal solutions that are executed by mapping local optimal solutions and using dynamic programming.

3.2 Related Work

Some prior researches have addressed the improvement of the entire cloud systems' performances. The basic working manner of cloud computing is using distributed computing. Thus, exploring the optimization of cloud resources has been worked in a few different aspects.

First, interconnecting different cloud resources is a popular research topic in cloud computing. The relationships among cloud nodes are considered the crucial parts of the performance improvements in cloud systems. An example is *Internet-of-Things* (IoT) that requires intercommunications and interconnections among different infrastructure. For improving the system's performance, previous researches have attempted on a variety of dimensions. For instance, cost-aware cloud computing is a design target that saves energies by implementing scheduling algorithms on Virtual Machine (VM). The outcomes are highly associated with the cost requirements or other configured parameters.

Moreover, some other researches concentrated on integrating heterogeneous cloud computing with mobile wireless networks. For example, Huajian et al. [36] proposed an optimizing approach of the file management system that used mobile heterogeneous mobile cloud computing. This system applied transparent integrations to increase the efficiency of cloud computing. Next, Kostas Katsalis et al. [37] concentrated on wireless networking channels and proposed an approach using semantic Web for strengthening communications among multiple vendors.

Furthermore, consider the real-time services' requirements, Mahadev Satyanarayanan et al. [38] proposed a Cloudlet-based solution that was designed to reduce execution time by establishing a pool of VMs that are close to users. This design was proved that it was an efficient way to reduce the response time. Another research considered high-density computing resources and developed a cognitive management solution that was based on the mobile applications [39]. This solution can be used in the dynamic networking environment, which optimized the operations of mobile ends, Cloudlet, and computing resources in clouds. Next, geographical distributions can also impact on the performance of cloud systems. Hongbin Liang et al. [40] proposed an approach using Semi-Markov Decision Process (SMDP) to determine the service migrations. This approach

could efficiently reduce the interrupts of services. Similarly, it has been assessed that a prediction-based approach could be helpful for planning the computing resource assignments based on the predictions [41]. However, most prior researches did not consider implementing heterogeneous cloud computing such that the potential optimizations were ignored.

3.3 Motivational Example of CM2DAH

In this section, We give a motivational example to describe the operational processes of CM2DP in this scenario. Assume that there are four cloud providers offering MaaS with different performances, namely M1, M2, M3, and M4. Table 3.1 shows the costs for different cloud memory operations.

Table 3.1: Cost for different cloud memories. Four types of memories are M1, M2, M3, and M4; main costs derive from Read, Write, Communications (Com.), and Move.

Operation		M1	M2	M3	M4
Read (R)		4	2	1	50
Write (W)		6	4	2	50
Comm. (C)		10	10	10	10
Move (MV)	M1	0	4	6	40
	M2	3	0	5	40
	M3	2	3	0	40
	M4	40	40	40	0

As shown in Table 3.1, we consider four main costs that include Read (R), Write (W), Communications (C), and Move (MV). R and W refer to the operation costs of reading and writing data. C refer to the costs happened to communication processes through the Internet. MV refer to the costs happened at the occasions when switching from one memory provider to another set.

Table 3.2 shows the number of the memory accesses, including *Read* and *Write*. We assume there are seven input data that are A, B, C, D, E, F, and G. For example, data A require 7 reads and 6 writes according to the table.

Our example's initial status is to allocate $A \rightarrow M2$, $B \rightarrow M2$, $C \rightarrow M4$, $D \rightarrow M2$, $E \rightarrow M3$,

Table 3.2: The number of the memory accesses.

Data	Read	Writes
A	7	6
B	6	3
C	8	6
D	1	1
E	3	1
F	6	6
G	2	3

$F \rightarrow M4$, and $G \rightarrow M3$. Moreover, The example configuration is that there are 2 M1, 2 M2, and 2 M3. We assume M4 is always available for data.

Based on the conditions given by Table 3.1 and 3.2, we map the costs of data allocations to could memories in Table 3.3.

Table 3.3: Mapping the costs for cloud memories.

Data	M1	M2	M3	M4
A	77	48	34	700
B	55	34	27	500
C	118	90	70	710
D	23	16	18	150
E	30	23	15	250
F	110	86	68	610
G	38	29	18	300

As shown in the table 3.3, data A costs 7*4+6*6+10+4=77 when it is allocated to M1, switched from M2. We call the table 3.3 as a *B Table (BTab)*. Actually, we can use a 3-dimension array to refer to every row in table 3.3. For instance, We use BTab_a[x][y][z] to refer to the costs that data A allocated to M1, M2, M3, and M4. X, y, z shows the used number of cloud memories of M1, M2, and M3 respectively. In this case, BTab_a[1][0][0]=77 indicate the costs of data A is allocated to M1, since we use 1 M1 here. BTab_a[0][1][0]=48 indicate the costs of data A is allocated to M2. BTab_a[0][0][1]=34 indicate the costs of data A is allocated to M3. BTab_a[0][0][0]=700 indicate the costs of data A is allocated to M4 because either M1, M2, and M3 are not available here.

We are going to use Table 3.3 to calculate the total cost after the allocations and the optimal allocation plan. For obtaining the final optimal data allocation plan, we will produce a *D Table (DTab)* showing the optimal data allocation plan. The generation process is given as follows, which consists of a few steps.

First, we only consider one input data A. In this case, we just copy BTab_a as DTab_a. Table 3.4 displays the costs of DTab_a.

Table 3.4: The costs of DTab_a.

DTab cell	Cost
DTab_a[0][0][0]	700
DTab_a[0][0][1]	34
DTab_a[0][1][0]	48
DTab_a[1][0][0]	77

As shown in table 3.4, we mark the cloud memory to the corresponding cost. Then, we add data B into our sight and generate the DTab_{ab}. We calculate every cell of DTab_{ab} to get the optimal costs and the optimal allocation plan of data A and data B, according to BTab_b and DTab_a.

Table 3.5: The costs of DTab_{ab}.

DTab cell	Cost
DTab_{ab}[0][0][0]	1200
DTab_{ab}[0][0][1]	534
DTab_{ab}[0][0][2]	61
DTab_{ab}[0][1][0]	548
DTab_{ab}[0][1][1]	68
DTab_{ab}[0][2][0]	82
DTab_{ab}[1][0][0]	577
DTab_{ab}[1][0][1]	89
DTab_{ab}[1][1][0]	103
DTab_{ab}[2][0][0]	132

For example, DTab_{ab}[0][1][1] = min(BTab_b[0][0][0]+DTab_a[0][1][1], BTab_b[0][0][1])+DTab_a[0][1][0], BTab_b[0][1][0])+DTab_a[0][0][1], BTab_b[1][0][0])+DTab_a[-1][1][1]). We just drop two elements,

$BTab_b$[0][0][0])+$DTab_a$[0][1][1] and $BTab_b$[1][0][0])+$DTab_a$[-1][1][1], since $DTab_a$[0][1][1] and $DTab_a$[-1][1][1] are not valid. Thus $DTab_{ab}$[0][1][1] = min($BTab_b$[0][0][1])+$DTab_a$[0][1][0], $BTab_b$[0][1][0])+DTa = min(27+48, 34+34) = min(75, 68) = 68. That means the minimum cost is 68 when we have 1 M2 and 1 M3 available. The optimal plan is allocate data A to M2 and data B to M3 since 68 is generated by $BTab_b$[0][1][0])+$DTab_a$[0][0][1].

We can gain the data allocation plan for each data from the calculation method mentioned above. Finally, we get the last result: $DTab_{abcdefg}$[2][2][2] = 438 and the optimal allocation plan is A→M2, B→M2, C→M3, D→M4, E→M1, F→M3, G→M1.

3.4 Algorithm of CM2DP

In this section, we propose the algorithm of CM2DP, which is designed to find the optimal data allocation plan for N-dimension heterogeneous cloud memories by using N-dimension dynamic programming. Assume that there are n types of memories available. We define one type of the memory as one dimension. The definition of *N-Dimensional Heterogeneous Cloud Memories* is given by Definition 5.

Definition 5 N-Dimensional Heterogeneous Cloud Memories: $\exists$ *n types of the memory available for alternatives, we define "n" as the number of dimensions and the available memory set is called N-Dimensional heterogeneous Memories. Each memory type can have different number of memories.*

We define a few definitions used in our algorithm, including *B Table* defined by Definition 6, *D Table* defined by Definition 7 and *Plan* by Definition 8.

Definition 6 B Table: *we use a multiple-dimensional array to describe a table that stores the cost of each data at each memory. Inputs include each data's cost when it is assigned to a memory. The output will be a multiple-dimensional array.* $\exists$ j *types of memory,* $\{M_j\}$*, and each* M_j *has*

n_j memories available. The mathematical expression is BTab[i]⟨(M_1, nb_1), (M_2, nb_2), ..., (M_j, nb_j)⟩, which represents the cost when $data_i$ use j types of memory and the number of each memory type.

Definition 7 D Table: *we use a multiple-dimensional array to describe a table that shows the total cost of data allocations by storing all task assignment plans. Input is the B Table. The output will be a multiple-dimensional array storing all task assignments as well as their costs. ∃ j types of memory, $\{M_j\}$, and each M_j has n_j memories available. The mathematical expression is DTab[d]⟨(M_1, nd_1), (M_2, nd_2), ..., (M_j, nd_j)⟩, which represents the cost of all d $data$ when using j types of memory. The tuple shows the assignment plan.*

Definition 8 Plan: *we use a multiple-dimensional array to describe a table that shows the optimal data allocation plan. Input is the B Table. The output will be a multiple-dimensional array storing all task assignments as well as their costs. ∃ j types of memory, $\{M_j\}$, and each M_j has n_j memories available. The mathematical expression is plan[d]⟨(M_1, nd_1), (M_2, nd_2), ..., (M_j, nd_j)⟩, which represents the allocation plan of all d $data$ when using j types of memory.*

The algorithm 2 shows the proposed algorithm and the main phases of our algorithm include:

1. Input *B Table*.
2. Copy BTab[0] to DTab[0] to produce the first partial D Table, which represents add data[0] to D Table.
3. Find the minimal cost from the sums of BTab[1] cells and their supplemental cells in DTab[0] that generated by the last Step 2.
4. Assigning optimal data allocation plan for data 0 and data 1 according to the minimal costs.

Algorithm 2 Cost-aware Multi Dimension Dynamic Programming (CM2DP) Algorithm

Require: The B Table $BTab$
Ensure: The optimal data allocation plan $PlanD$

input the B Table
initialize Plan
DTab[0] $\leftarrow$ BTab[0]
FOR $\forall$ rest cell in DTab,
/* DTab[i]$\langle(M_1, nd_1), (M_2, nd_2), \ldots, (M_j, nd_j)\rangle$ */
    minCost $\leftarrow \infty$
    minCostIndex $\leftarrow null$
    FOR $\forall$ cell in BTab[i],
        /* BTab[i]$\langle(M_1, nb_1), (M_2, nb_2), \ldots, (M_j, nb_j)\rangle$ */
        IF $\exists$ DTab[i-1]$\langle(M_1, nd_1 - nb_1), (M_2, nd_2 - nb_2), \ldots, (M_j, nd_j - nb_j)\rangle$
            sum $\leftarrow$ BTab[i]$\langle(M_1, nb_1), (M_2, nb_2), \ldots, (M_j, nb_j)\rangle$ + DTab[i-1]$\langle(M_1, nd_1 - nb_1), (M_2, nd_2 - nb_2), \ldots, (M_j, nd_j - nb_j)\rangle$
            IF sum<minCost
                minCost $\leftarrow$ sum
                minCostIndex $\leftarrow \langle(M_1, nb_1), (M_2, nb_2), \ldots, (M_j, nb_j)\rangle$
            ENDIF
        ENDIF
        IF minCostIndex != null
            plan[i]$\langle(M_1, nd_1), (M_2, nd_2), \ldots, (M_j, nd_j)\rangle$. add(Plan[i-1]$\langle(M_1, nd_1 - nb_1), (M_2, nd_2 - nb_2), \ldots, (M_j, nd_j - nb_j)\rangle$)
            plan[i]$\langle(M_1, nd_1), (M_2, nd_2), \ldots, (M_j, nd_j)\rangle$. add(allocate $data_i$ to $\langle(M_1, nb_1), (M_2, nb_2), \ldots, (M_j, nb_j)\rangle$)
    ENDFOR
ENDFOR
RETURN Plan.

5. Add all data by applying the same method used from Step 3 to Step 4 until the D Table is generated.

6. Find the optimal data allocation by searching the lowest cost in D Table according to the memory condition. Output the task assignment plan.

3.5 Experiments and the Results

In this section, We illustrated our experimental evaluations. Section 3.5.1 represented our experimental settings. Section 3.5.2 provided partial experimental results.

3.5.1 Experiments settings

We use experimental evaluations to assess the performance of the proposed scheme. The evaluation is based on a simulation that compare CM2DP algorithm and FIFO algorithm, greedy algorithm, and MDPDA algorithm [42]. We implemented our experiments on a Host that ran Windows 8.1 64 bit OS. The main hardware configuration of the Host was: Intel Core i5-4210U CPU, 8.0 GB RAM. we have three main experimental settings that are designed to assess the proposed approachs performance. Three experiments settings are:

1. We configured 3 kinds of memories and 7 kinds of data, 3 available M1, 3 available M2, and 2 available M3.

2. We configured 4 kinds of memories and 8 kinds of data, 2 available M1, 3 available M2, and 2 available M3. M4 is always available for data.

3. We configured 4 kinds of memories and 10 kinds of data, 2 available M1, 3 available M2, and 3 available M3. M4 is always available.

3.5.2 Experiments results

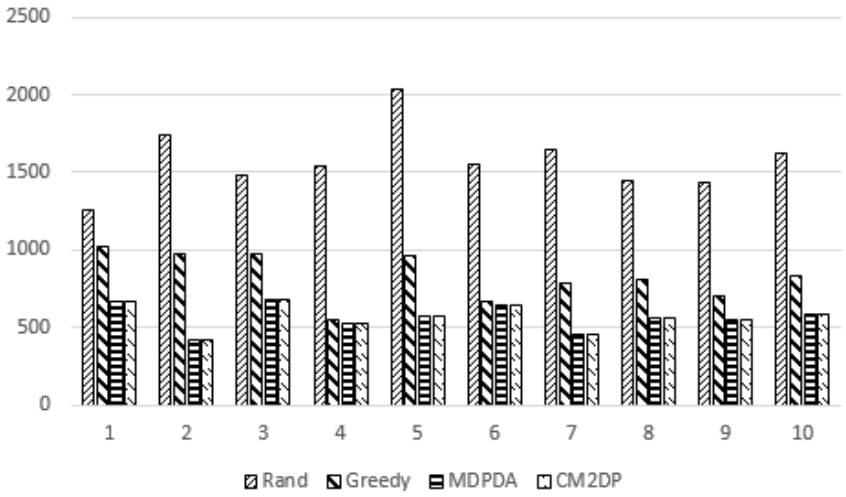


Figure 3.1: Comparisons of costs under Setting 1

As shown in Fig. 3.1, our proposed approach has the less total cost than FIFO algorithm and greedy algorithm under setting 1. The FIFO algorithm has the biggest total cost because there is no technical choose to allocate the data to the memories that has less cost. The greedy algorithm usually has the larger total cost than CM2DP and MDPDA algorithm. But sometimes greedy algorithm can get the optimal solution, too. The MDPDA algorithm and our proposed algorithm have the same total cost result since they both can get optimal solution.

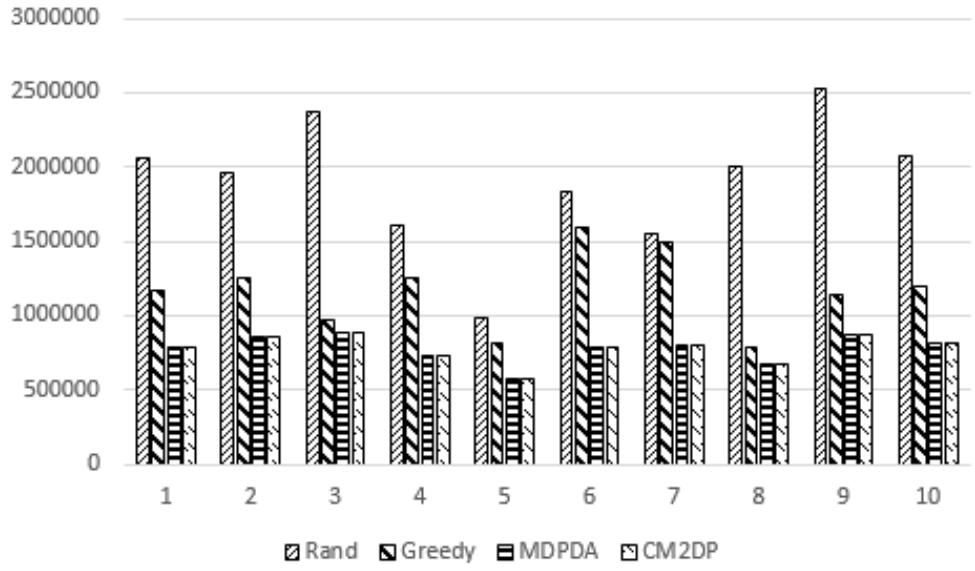


Figure 3.2: Comparisons of costs under Setting 2

Fig. 3.2 shows the comparison among FIFO algorithm, greedy algorithm, CM2DP algorithm, and MDPDA algorithm under setting 2. DM2DP algorithm and MDPDA algorithm always have better performance than FIFO and greedy algorithm under this setting.

Moreover, Fig. 3.3 shows another group of comparison results of FIFO algorithm, greedy algorithm, CM2DP algorithm, and MDPDA algorithm under setting 3. The figure represents that our proposed algorithm has the same optimal result with MDPDA algorithm, which is better than FIFO algorithm and greedy algorithm. As shown in the above three figures, we can find that usually the more data and memories we have, the more cost can be saved by using our proposed algorithm.

The Fig. 3.4, Fig. 3.5, and Fig. 3.6 show the comparison of execution time consumption of FIFO algorithm, greedy algorithm, CM2DP algorithm, and MDPDA algorithm under three settings respectively. From these figures we can find that our proposed algorithm has the less execution time

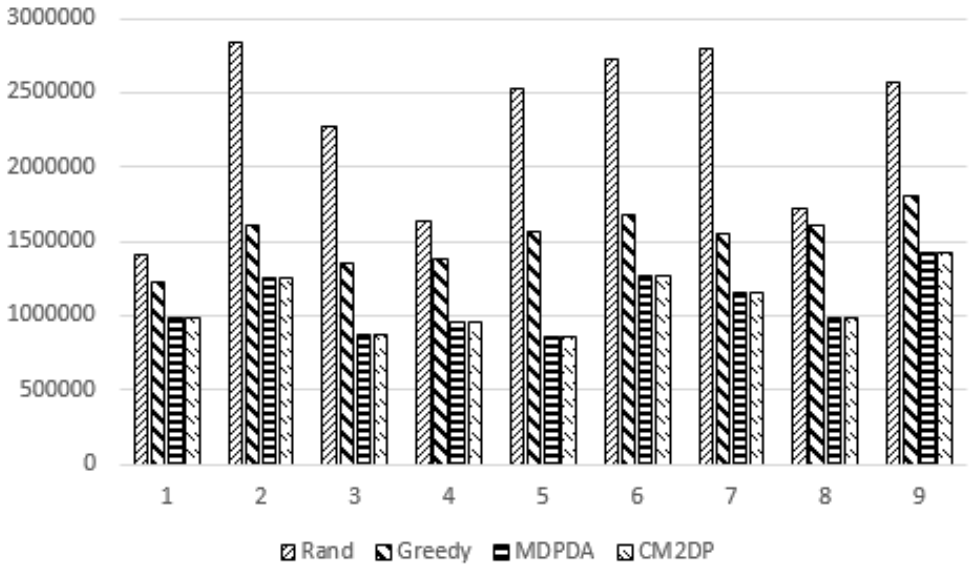


Figure 3.3: Comparisons of costs under Setting 3

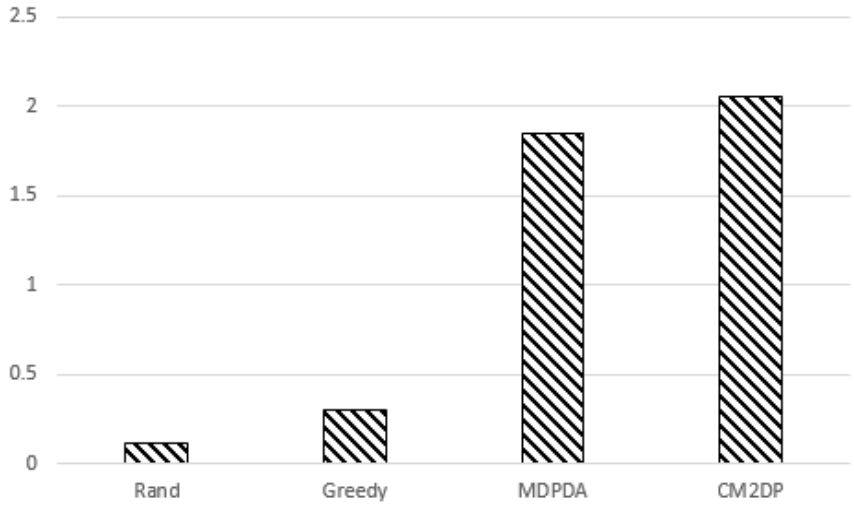


Figure 3.4: Comparisons of execution time among Random, Greedy, CM2DP and MDPDA algorithms under setting 1

than the MDPDA algorithm when they both can get the optimal solution under every setting.

In summary, our approach always can get the optimal data allocation plan, which is better than the results of FIFO algorithm and the greedy algorithm. Moreover, CM2DP has the better execution performance than MDPDA, another optimal algorithm.

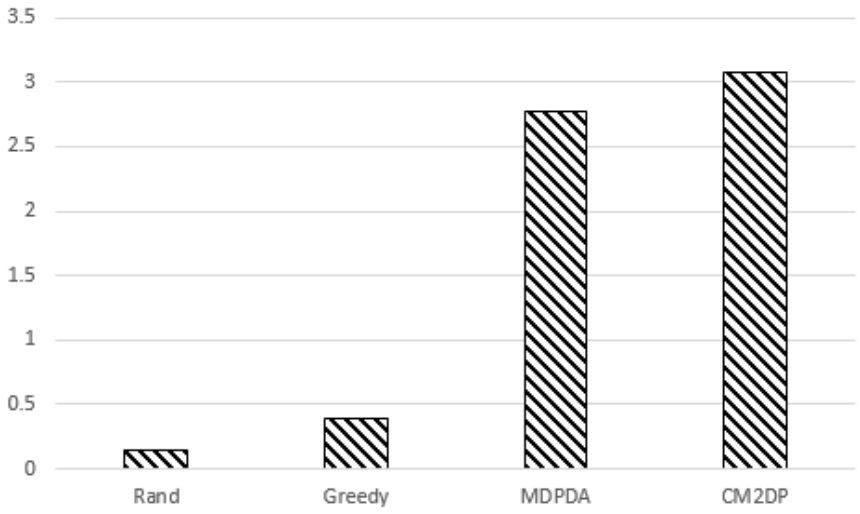


Figure 3.5: Comparisons of execution time among Random, Greedy, CM2DP and MDPDA algorithms under setting 2

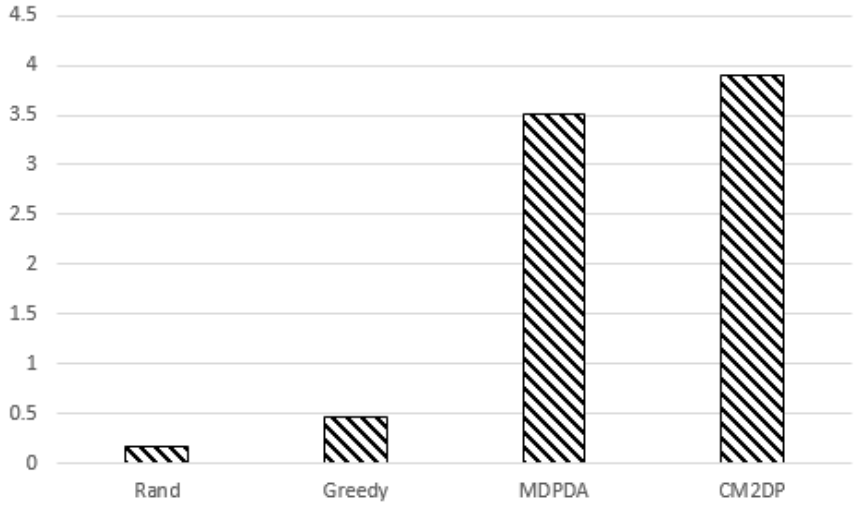


Figure 3.6: Comparisons of execution time among Random, Greedy, CM2DP and MDPDA algorithms under setting 3

3.6 Conclusion

In this chapter, we proposed an approach that named *Cost-aware Multi Dimension Data Allocation Heterogeneous Cloud Memories* (CM2DAH) to find out the optimal data allocation plan in heterogeneous cloud memories. To support the proposed approach, a N-dimension dynamic programming algorithm was designed. This algorithm is named *Cost-aware Multi Dimension Dynamic Programming* (CM2DP) Algorithm. The experimental results proved that our approach is

an effective mechanism compare with FIFO algorithm, greedy algorithm, and MDPDA algorithm.

Chapter 4

Empirical Study of Data Allocation in Heterogeneous Memory.

4.1 Introduction

With the swift development of the memory design and production, the financial cost of the memory has been cutting down while the hardware performance is continuously increasing. This trend has enabled a flexible deployment of the memory for fitting in various users' demands [43][44][45]. Meanwhile, the blooming adoption of MCC requires enhancements in both data processing and data storage. The connected environment further grows the workload of the data exchange and task distributions [46][47][14][48]. As a core component of data processing, increasing the usage and efficiency of the memory is urgent needs. Thus, designing heterogeneous memories has become a popular research domain that attracts numerous researchers over years. A variety of data allocation algorithms have been proposed.

Moreover, heterogeneous memory also has a potential value for in-memory analytics. The study [49] applied the technique of the file virtual address to the in-memory file system in order to optimize memory usage and the approach was called SIMFS. Another study [50] also addressed

similar technique and proposed a data storage approach, called SIM-DAM, which enhanced the efficiency of the memory space by optimally distributing data storage between in-memory storage and disk storage.

Despite many prior studies addressing the design of the heterogeneous memory, it is hard to discern the performance of various data allocation algorithms. Most optimization algorithms are developed to specific implementations. Inputs and outputs maybe varied due to different application assumptions and system configurations. The availability of memory types also introduces difficulties in understanding the variety of different data allocation algorithms. Lack of unified configuration standard results in the situation that memory producers can hardly understand the differences between distinct algorithms, which restricts the adoption of the heterogeneous memory in practice. Therefore, there is an urgent demand to perceive the prior data allocation approaches' performance throughout a comprehensive intercross evaluation.

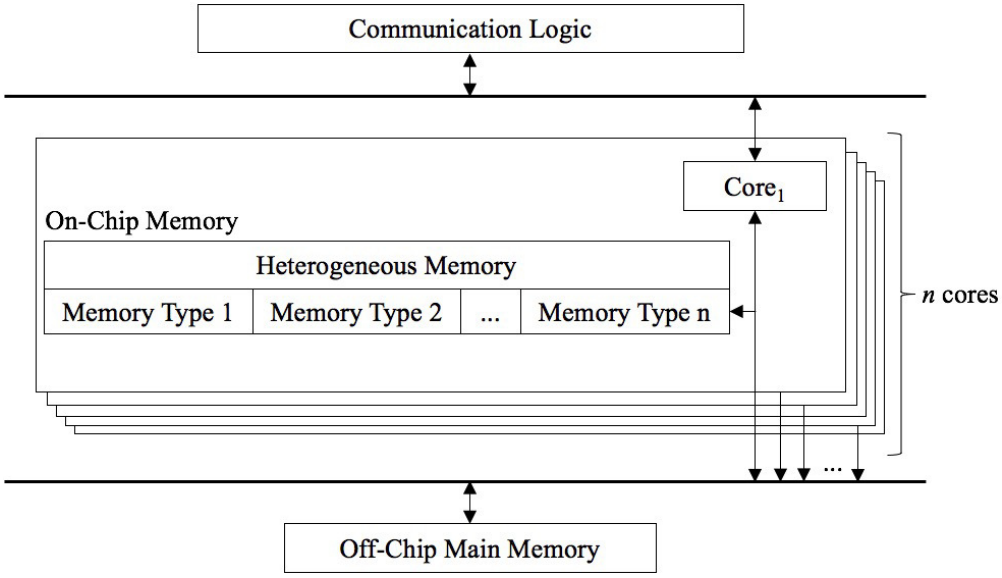


Figure 4.1: The typical architecture of the heterogeneous memory.

This work concentrates on evaluating the performance of the data allocation algorithms for heterogeneous memory, which mainly covers the application scenarios, input and output configurations, computing resource consumptions, energy costs, and strategy generation time. Fig. 4.1 illustrates the typical architecture of the heterogeneous memory. A data allocation scheduling

mainly address the task distribution to different memory types including both on-chip memory and off-chip main memory. An empirical study has been accomplished, which are aligned with the concerns mentioned above. The significance of this study is observable due to both urgent demands in practice and contributions to the body of knowledge.

The main contributions of this work include:

1. This empirical study has evaluated a few typical data allocation approaches for heterogeneous memory. The study provides a review on the evaluated approaches, which include heuristic algorithm, dynamic programming, and a few active resource scheduling algorithms.

2. The synthesis of main findings can provide memory producers and heterogeneous memory researchers with a quantitative support. We also provide recommendations for distinct algorithms' implementations.

4.2 Preliminary

We reviewed a number of prior characteristic data allocation algorithms throughout a qualitative study in this section. The number of the reviewed approaches was limited due to the page length restriction so that only representative data allocation mechanisms were selected.

First, contemporarily active approaches generally required a dramatic short time for generating data allocation strategies. *Round-Robin* (RR) algorithm was a classic scheduling algorithm that was widely adopted in practice due to its easy deployment [51]. The disadvantage of RR was that the optimization level of the output was low. The other commonly deployed algorithm is a greedy algorithm [52]. This type of the algorithm had a priority setting so that the output generally was a sub-optimal solution while ensuring a short strategy generation time. The main reason for broadly deploying these two types of algorithms was its extremely short time period for creating data allocation plans. This phenomenon also was related to the implementation characteristics of the memory, since most in-memory data processing tasks take a short moment. A long execution time for creating a data allocation strategy could result in a latency.

Moreover, the heuristic algorithm was a typical approach in data allocations. The main advantage of a heuristic algorithm was that the data allocation strategy generation was relatively efficient. However, most heuristic algorithms could not produce optimal solutions. An expected output of the heuristic algorithm was a near-optimal solution. For example, Qiu et al. [24] proposed a genetic algorithm to optimize SLC/MLC PCM memory for the purpose of green computing. In addition, SDAM [53] was a heuristic algorithm that was designed for a multi-dimensional heterogeneous memory setting. The algorithm used a cliff-climbing mechanism that transferred a sub-optimal solution to a near-optimal or an optimal solution. The crucial component of this algorithm was called a *Smart Switch*, in which the enhancement of the strategy took place. Meanwhile, considering the connected environment, CAHCM is another heuristic algorithm [35] that emphasized the implementation of cloud computing. The number of the reads and writes for the input data was used as the priority for creating an original strategy. An optimization was operated over the original strategy such that the output was a near-optimal solution. This algorithm also considered the performance cliff of the memory.

Furthermore, dynamic programming was an approach that could produce optimal solutions that was a major advantage. In general, the main weakness of dynamic programming was that it required a long execution time period for creating outputs. Hu et al. [15] proposed DAHS that was an optimization solution to hybrid scratch pad memory combining SRAM and nonvolatile memory. Qiu et al. [6] further developed a dynamic programming (MDPDA) that could allocate data to three types of memory for hybrid scratch-pad memory. This algorithm was a representative dynamic programming designed for heterogeneous memory. The number of the memory type was fixed according to the algorithm setting. Zhao et al. [54] further proposed an approach entitled OMDDA for multi-dimensional heterogeneous memory. The number of the memory type was unfixed so that it could be applied to various heterogeneous memory. Its back-forward table was also improved such that the strategy generation time could be shortened, considering the perspective of the computation complexity.

The next section will illustrate some experiment results collected from the evaluation of the selected algorithms.

4.3 Empirical Studies

4.3.1 Experiment Configuration

We illustrated partial experiment results and main findings that are associated with Section 4.2. The evaluations were completed on our simulator that offered the standard and unified input settings. Comparisons were completed based on a number of representative scheduling algorithms, which included RR [51], Greedy [52], SDAM [53], CAHCM [35], MDPDA [6], DAHS [15], and OMDDA [54]. These algorithms involved greedy, genetic, heuristic, and dynamic programming algorithms, which could embody the mainstream of techniques in data allocations for heterogeneous memory. The number of evaluation rounds was between 1×10^3 - 1×10^4, which could avoid noises and ensure the correctness of the assessment.

Moreover, the comparisons mainly concerned two aspects, which were costs and strategy generation time. The first aspect would consider all possible computing resources utilized or required in data allocation manipulations the cost, such as the energy consumption, latency, memory space, hardware availability, and probability. Therefore, the cost was counted in units for the purpose of comparisons in our evaluation. The other aspect examined the execution time of the algorithm, which could assess whether the algorithm was adoptable in practice. In most situations, an optimal algorithm required a longer execution time than sub-optimal/near-optimal algorithms. However, in some situations, an optimal algorithm might be adoptable due to the specific configuration, such as a low-dimensional memory setting or less input task amount. In our evaluations, the time length was counted in milliseconds.

Table 4.1 displayed a series of experiment configuration used in the evaluations. There were five settings given by the table, in which the configuration depended on the number of the memory type. This configuration fitted in various data allocation algorithms in distinct memory settings. For example, algorithm DAHS [15] mainly supported two heterogeneous memories, while the algorithm MDPDA [6] was designed for a three-heterogeneous memory setting. Thus, it implied that not all algorithms participated in all comparison evaluations. For instance, the DAHS algorithm might be involved in Setting 1 only; the OMDDA algorithm could be examined throughout all

Table 4.1: Input Configuration

	Memory #	Input Data Number						
		(a)	(b)	(c)	(d)	(e)	(f)	(g)
Setting 1	2	10	20	30	40	50	75	100
Setting 2	3	10	20	30	40	50	75	100
Setting 3	4	10	20	30	40	50	75	100
Setting 4	5	10	20	30	40	50	75	100
Setting 5	6	10	20	30	40	50	75	100

settings due to its multi-dimensional design.

In addition, under each setting, there were a few sub-settings configured by the number of input data packages/tasks. As shown in the table, sub-settings were given from (a) to (g), in which provided distinct input amounts as 10, 20, 30, 40, 50, 75, and 100, respectively. The reason for having these settings was to further simulate diverse application scenarios. In line with the comparison aspects, the condition of the inputs could impact on the performance of the algorithm. For instance, we assumed some near-optimal solutions could have a better performance within a specific scope of input task volume than other scopes, since the rate of obtaining optimal solutions might be higher. Another assumption could be that a dynamic programming could be more adoptable in a range of input tasks than other ranges due to the balance between the latency caused by the strategy generations and the cost saving due to the optimal solution. All these issues would be addressed in the empirical study.

In summary, the experiment configuration in this empirical study highlighted the adaptabilities for various algorithms. The investigations would focus on a few featured algorithms throughout a series of comparisons and syntheses. The next section would display partial results and present our main findings derived from our experiments.

4.3.2 Experiment Results

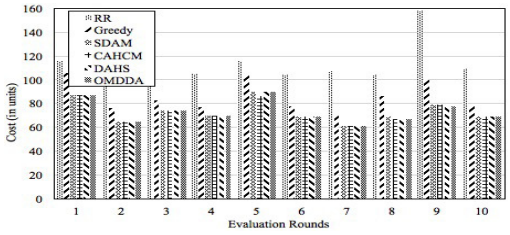
We presented partial experiment results in this section, which aligned with settings given in Section 4.3.1. Fig. 4.2 illustrated a number of figures that depicted partial results collected from settings 1-

(a) and 1-(b). Under this setting, heterogeneous memory consisted of two memories. Fig. 4.2a and Fig. 4.2b depicted cost comparisons based on ten-round evaluations that were randomly picked from 1×10^3 rounds. DAHS and DMDDA were two dynamic programming algorithms, which could output optimal solutions. SDAM had a high rate of obtaining optimal solutions under Setting 1. CAHCM had around 80% percentage of gaining optimal solutions.

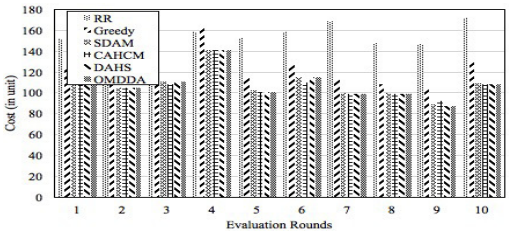
Fig. 4.2c and Fig. 4.2d showed comparisons of the strategy generation time using logarithmic scale at 2. The results provided an observable difference between dynamic programming and other algorithms. Under Setting 1-(a), The average generation time lengths for DAHS and OMDDA were 67.53 and 37.99 milliseconds, which were much longer than two heuristic algorithms, SDAM and CAHCM. The average generation time of SDAM and CAHCM was 5.56 milliseconds and 2.64 milliseconds, respectively. With the increase of the input data packages, the generation time of dynamic programming would be highly impacted while other algorithms had a limited impact. Under Setting 1-(b), DAHS and OMDDA's average strategy generation time became 93.57 milliseconds and 62.68 milliseconds. Increase rates were 38.59% and 64.97%, respectively. Having this phenomenon was that the computation complexity could be remarkably increased for dynamic programming but other algorithms had few impacts when the memory-dimension was two.

Next, Fig. 4.3 depicted some experiment results collected from Setting 2. Similar to result displays of Setting 1, we provided some results from two sub-settings to show the assessment trends. Two heuristic algorithms had a stable performance under this setting. We also observed that there was a great impact on the generation time of dynamic programming due to the increase of the memory types. There were two dynamic programming algorithms involved in this setting, namely, MDPDA and OMDDA. According to our statistics, MDPDA's average generation time was 90.50 milliseconds and 201.99 milliseconds for Setting 2-(c) and Setting 2-(d), respectively. OMDDA's average generation time was 62.62 milliseconds and 133.13 milliseconds under the same settings.

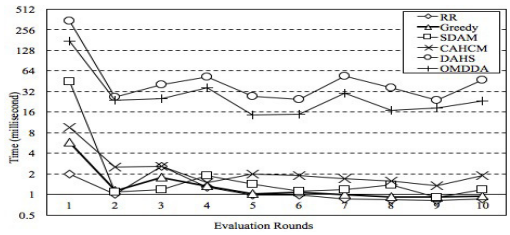
Furthermore, Fig. 4.4 provided some results obtained from Setting 3. Two sub-settings were settings 3-(e) and 3-(g), which respectively had 75 and 100 input data packages. There was only one dynamic programming (OMDDA) due to its four-memory setting. The main finding under



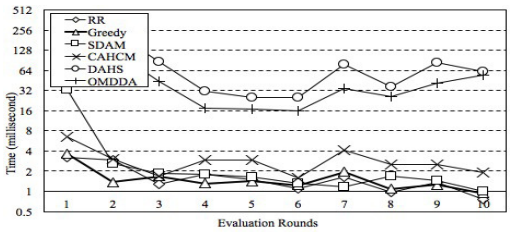
(a) Partial experiment results comparing cost levels collected from Setting 1-(a).



(b) Partial experiment results comparing cost levels collected from Setting 1-(b).

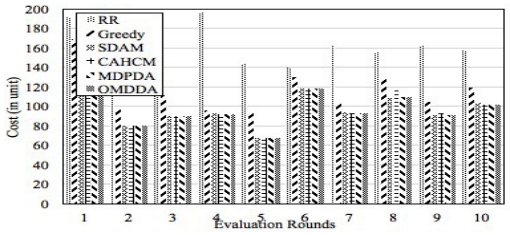


(c) Partial experiment results comparing strategy generation time collected from Setting 1-(a). (Logarithmic scale: 2)

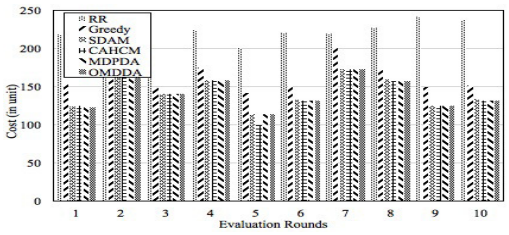


(d) Partial experiment results comparing strategy generation time collected from Setting 1-(b). (Logarithmic scale: 2)

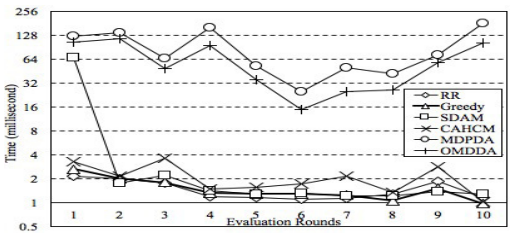
Figure 4.2: Partial experiment results collected from Setting 1.



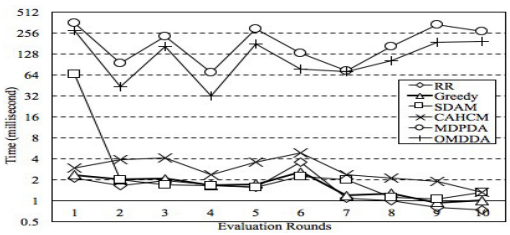
(a) Partial experiment results comparing cost levels collected from Setting 2-(c).



(b) Partial experiment results comparing cost levels collected from Setting 2-(d).

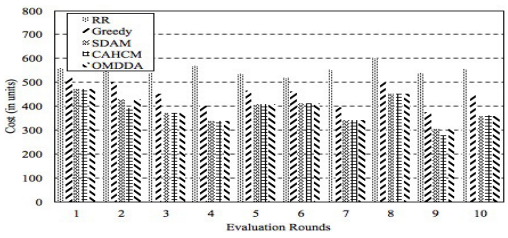


(c) Partial experiment results comparing strategy generation time collected from Setting 2-(c). (Logarithmic scale: 2)

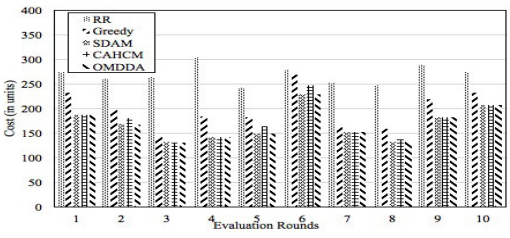


(d) Partial experiment results comparing strategy generation time collected from Setting 2-(d). (Logarithmic scale: 2)

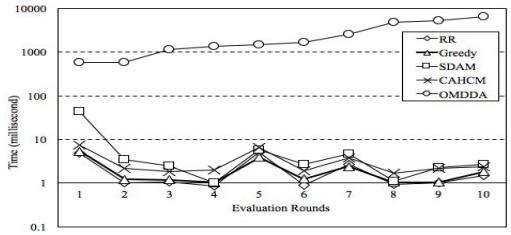
Figure 4.3: Partial experiment results collected from Setting 2.



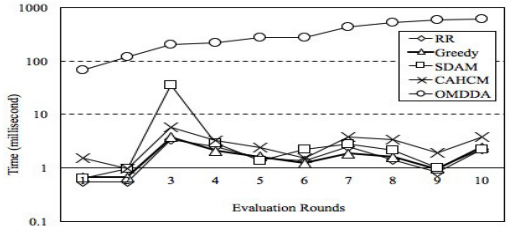
(a) Partial experiment results comparing cost levels collected from Setting 3-(e).



(b) Partial experiment results comparing cost levels collected from Setting 3-(g).



(c) Partial experiment results comparing strategy generation time collected from Setting 3-(e). (Logarithmic scale: 10)



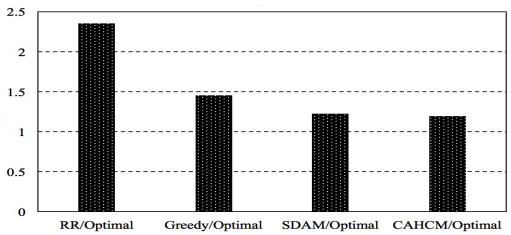
(d) Partial experiment results comparing strategy generation time collected from Setting 3-(g). (Logarithmic scale: 10)

Figure 4.4: Partial experiment results collected from Setting 3.

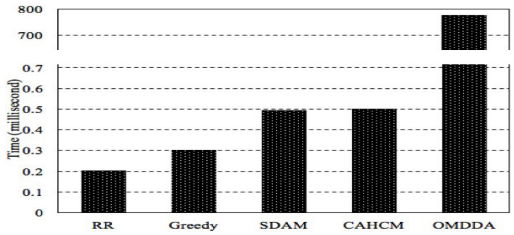
this setting was that implementing dynamic programming could result in a great latency. We observed that the average strategy generation time lengths for OMDDA were 0.33 seconds and 2.60 seconds under Setting 3-(e) and Setting 3-(g), respectively. This result implied that using dynamic programming would have a great cost in creating strategies.

Moreover, Fig. 4.5 illustrated partial results that were based on a 1×10^4-round evaluation under Setting 4-(e). We used a rate deriving from "X/optimal" to examine the performance of saving costs by comparing a sub-/near-optimal solution to an optimal solution. Fig. 4.5a depicted rate values on average for four algorithms. It was observable that two heuristic algorithms had a similar performance. The average of SDAM/Optimal was 1.23 and the average of CAHCM/Optimal was 1.20. Fig. 4.5b depicted the strategy generation time for different algorithms on average. It showed that OMDDA was not suitable for small-sized data package due to its big cost in efficiency. The generation time for SDAM, CAHCM, and OMDDA on average were 0.496 milliseconds, 0.500 milliseconds, and 777.648 milliseconds, according to our statistics. Figures 4.5c - 4.5f showed trend lines of the strategy generation time for four algorithms.

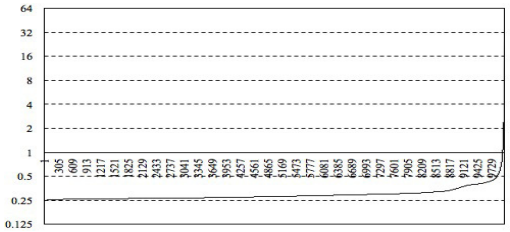
In addition, Fig. 4.6 depicted some results that focused on the strategy generation time, which



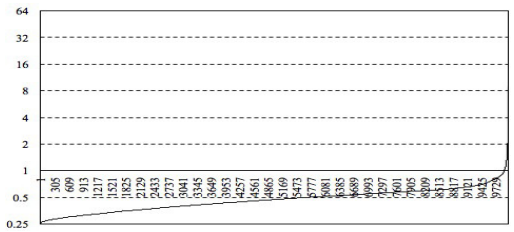
(a) Average comparisons between sub-optimal/near-optimal and optimal solutions.



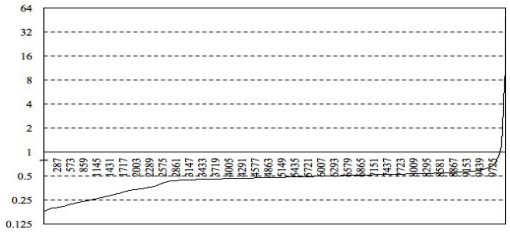
(b) Average strategy generation time comparisons.



(c) Trend line of the strategy generation time for greedy algorithm, counted in milliseconds. (Logarithmic scale: 2)



(d) Trend line of the strategy generation time for CAHCM algorithm, counted in milliseconds. (Logarithmic scale: 2)



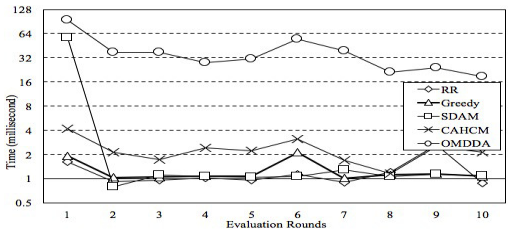
(e) Trend line of the strategy generation time for SDAM algorithm, counted in milliseconds. (Logarithmic scale: 2)

(f) Trend line of the strategy generation time for OMDDA algorithm, counted in milliseconds. (Logarithmic scale: 2)

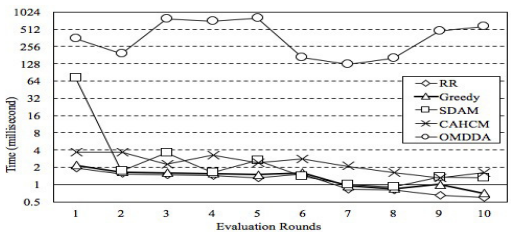
Figure 4.5: Partial experiment results from Setting 4-(e) with 1×10^4 rounds evaluations.

were collected from settings 5(a), 5(c), 5(f), and 5(g). SDAM's generation time lengths on average for these settings were 6.72, 8.73, 45.09, and 51.78 milliseconds, according to these 10-round evaluations. Under the same setting, CAHCM's average time lengths were 2.33, 2.46, 17.14, and 18.73 milliseconds; OMDDA's were 38.49, 431.91, 10979.34, and 7141.49 milliseconds. The reason for these results, based on our analysis, was that input data packages also had impact on individual strategy generation.

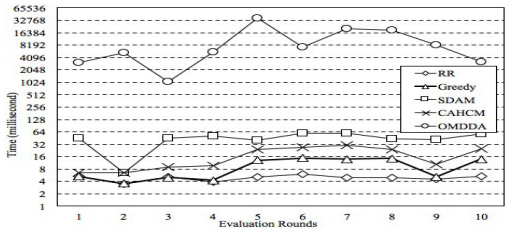
In summary, our findings included :(1) dynamic programming would be restricted when the number of input data packages was more than 50, based on our evaluation configuration; (2) both heuristic algorithms tested in our empirical study had stable performance in cost saving and required an acceptable strategy generation time; (3) dynamic programming was more suitable for



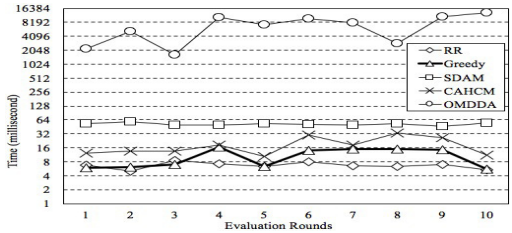
(a) Comparisons of partial evaluations' strategy generation time under Setting 5-(a). (Logarithmic scale: 2)



(b) Comparisons of partial evaluations' strategy generation time under Setting 5-(c). (Logarithmic scale: 2)



(c) Comparisons of partial evaluations' strategy generation time under Setting 5-(f). (Logarithmic scale: 2).



(d) Comparisons of partial evaluations' strategy generation time under Setting 5-(g). (Logarithmic scale: 2)

Figure 4.6: Partial experiment results collected from Setting 5.

those large-sized input tasks when considering the trade-off between the cost-saving advantage and the latency caused by the strategy generation.

4.4 Conclusions

In this work, we reviewed a few representative data allocation approaches for heterogeneous memory and completed an empirical study via a series of comparisons. The study gave future research a theoretical guidance about the adaptability. Our main findings showed that dynamic programming was restricted by the volume of input tasks as well as the amount of memory types. Two heuristic algorithms in our investigations had a stable performance, which had a higher-level adaptability.

Chapter 5

Network Traffic Reduction for Mobile Web Apps

Wireless network resource is another important resource in Mobile Cloud Computing environment. There are many repeated data in the wireless network flow where the mobile devices communicate with the cloud servers.

Lots of wireless network traffic will be saved if we can reduce these repeated data in network flow. Reducing network traffic can shorten wireless network response time efficiently. Scheduling resource through data processing to improve the performance of MCC is our research goal in network aspect. This include two different situation. One is that the receivers start the transmission. The other is that the senders start the transmission actively. We use Android web app as an example to discuss how to reduce network traffic in active receiver situation.

5.1 Problem Description

With the rapid development of the mobile technologies, mobile applications have been broadly used in multiple domains. The implementations of the mobile apps have brought mobile users a

variety of benefits, such as entertainments, communications, and personal information management. Mobile devices have become a platform supporting mobile apps and wireless communications [55]. The high-performance of wireless networks are also beneficial for the expansion of the mobile apps, such as distributed computing and mobile cloud computing [56, 57]. The openness of Android systems brings a higher-level flexibility for implementing multiple wireless technologies [58]. However, current Android apps are facing a great challenge in efficiency and fast response to the user demands due to the inefficient execution models [59].

As a popular framework, Android is established on the basis of the Linux kernel. Considering a complete open mobile platform, the emergence of Android has brought a large number of opportunities to smart phones as well as challenges. Android system has become a popular platform on mobile embedded systems. Android systems have been continuously increasing as the improvements of computation capability, wireless networks, and distributed deployments [60, 61, 62]. Cloud computing provides Android systems with a broad service delivery platform with a strengthened communication capability [63]. Content caching is an approach for optimizing performances of Android systems in distributed computing systems[64, 65].

Nevertheless, increasing the efficiency of the Android apps is a challenge issue for current mobile developers. There are two traditional app approaches for Android, including *Activity + XML Layout Files* (AXLF) and *HTML + WebKit* (HWK). First, as an official method recommended by Google, AXLF performs a lower-level response to user demands, even though it supports all great features of Android system. The other approach is HWK, which uses HTML technology to develop mobile web apps. Despite the efficiency of HWK is better than AXLF, the drawback of using this model is that a lot of unnecessary documents will be generated by repeatly downloading from remote servers.

When using HWK model in Android applications, HTML technology is applied to develop webpage so as to achieve the applications user interface and response functions of each component. Then WebKit components of Android are used to load the page to display complex user interface and useful information in the form of the page.

When the programmers develop the application use this model, they write web pages, even

can directly use the pages existing in the website as the mobile user interface, and load the page as the application user interface to achieve the human-computer interaction. Currently, this model has gradually become one of the mainstream models for Information-as-a-Service mobile apps. However, implementing HWK model usually need a lot of extra execution time and can cause heavy networking traffic. Since lots of web page's contents ,such as the page layout, page title, and background picture, etc., do not change frequently, these page contents are usually repeated data in network flow when user load a same page. Due to the workload of updating data, users have to download many extra contents. As all know, the receivers start a network communication when users download a web page. Therefore, this is a typical problem for reducing repeated data of network traffic when receivers is active.

Addressing this issue, we propose a novel method named *PrecAche Technology of Active Receiver* (PATAR). The proposed paradigm mainly consists of three algorithms, namely *Server-Side Execution Procedure Algorithm* (2SEPA), *On-Premise Load Data Algorithm* (OPLDA), and *Data Synchronization Algorithm* (DSycA). Implementing the proposed schema aims to prevent from unnecessary downloading data base on the high efficiency performance. The main contributions of this paper are twofold:

1. We proposed a novel model for high-performance Mobile Cloud Computing Systems using the pre-cache-based technology.

2. Using our proposed schema can efficiently save networking traffics.

5.2 Related Work

Prior research has addressed Android research in multiple perspectives. First, we reviews the related work concerning Android models by using AXLF and HWK. Being aware of the Android systems, the *Open Handset Alliance* (OHA) was established by Google and its partners in November 2007 when released the first Beta version of the Android operating system, *Software Development Kit* (SDK). More than 80 firms in mobile domains attach to the organization and most mobile

apps follow the standard formed by OHA.

Due to the restrictions of the computing capabilities for mobile devices, current mobile apps are required to performance low energy costs [66]. Many previous research has addressed the issue of energy consumption. Leveraging mobile cloud-based solutions is an option for mobile apps to achieve low energy costs, high performance, and optimal task scheduling. Using heterogeneous distributed computing has a positive impact on mobile computation offloading and connections across multiple platforms [67]. The benefits of heterogeneity in mobile distributed computing enable many mobile systems to migrate the focuses into interface design and data display with implementing human-computer interactions.

AXLF is one of the most traditional and classical methods supported by Android. The applications use this model developed can straight use the components provide by Android and can maximize the use of the Android system resources to achieve a dazzling visual effect. Multiple functionality-aimed apps can be executed by this approach, such as games, multimedia, and online communications. Two crucial parts in this model include *Activity Manager* (AM) and *View System* (VS). Using XML layer files to describe user interface layout is able to reduce the complexity of the system [68]. The interconnections between platforms can enable the apps communications based on the cross-platform tools.

However, those mobile apps providing *Information-as-a-Service* (IaaS) do not require dazzling but the system structures are complex, such as e-dictionary, *Office Automation* (OA) systems, and E-commerce-related apps. Using AXLF approach can cause result in heavy workload and interface executions. In addition, the default component provided by Android needs users to have a strong background in computing, which results in the difficulties in workload controls. There are some system which need to be provided traditional computing network access and mobile access both, it will take twice as much time to developed respectively webpage design and mobile applications, and these two works cannot be shared each other, this also greatly increased the workload of developers.

HWK is the other optional approach because the AXLF model is not efficient in running apps delivering information services. When using this model in Android applications, HTML technol-

ogy is applied to develop webpage so as to achieve the applications user interface and response functions of each component [69, 70, 71, 72, 73, 74]. Then WebKit components of Android are used to load the page to display complex user interface and useful information in the form of the page [75, 76].

Those applications providing information services can store the data on-premise when the information is not needed to be changed, such as e-dictionaries. The wireless networks are not required if the source is stored locally. Using this model can solve the problem we mentioned before. However, implementing HWK model usually need a lot of extra execution time and can cause heavy networking traffic. Due to the workload of updating data, users have to download many extra contents.

Ajax is a solution to the problem of repeating downloads. Nevertheless, Ajax is not a good choice in this scenario. There are several reasons. First, using Ajax can result in security problems [77, 78]. Ajax uses Javascript code that is included in the web page to indicate what should do. Web pages that include script code are *Active Pages*. Using active pages maybe dangerous. Customers do not know weather the malicious codes are included in the active pages. In our approach, the web pages always are *Static Pages*. The pages do not include any code, which ensures the security from the perspective of codes. Second, using Ajax needs the browser or webkit to support Javascript technique. A browser or webkit will be complex when they support Javascript. This will make the browser or webkit inefficient. Thus, a possible solution is using a basic webkit that only can read static web pages and show UI to customers. Moreover, using basic webkit can guarantee a high level safety even though the web pages may include malicious code in them. Finally, Ajax was not fully supported by the earlier versions of mobile platforms.

5.3 Models and Concepts of PATAR

Addressing this problem, we propose a novel method named *PrecAche Technology of Active Receiver* (PATAR). Implementing the proposed schema aims to prevent from unnecessary downloading data base on the high efficiency performance.

Two crucial aspects of PATAR are covered in this section, including HTML page separation and execution procedure re-design. The whole HTML page is separated into two components, including *Page Framework* (PF) and *Page Data* (PD).

Page Framework and Page Data: PF refers to the framework of web pages, such as page layout, page title, and background picture in mobile web apps. PF usually is repeated data when the app download the whole page. PD represents the real-time data that need load into the page, such as prices and images in e-commerce, which usually is the necessary data when downloading the whole page. Using pre-caching technology can allow applications to avoid downloading the re-stored PF and PD. The Figure 5.1 shows the architecture of PATAR.

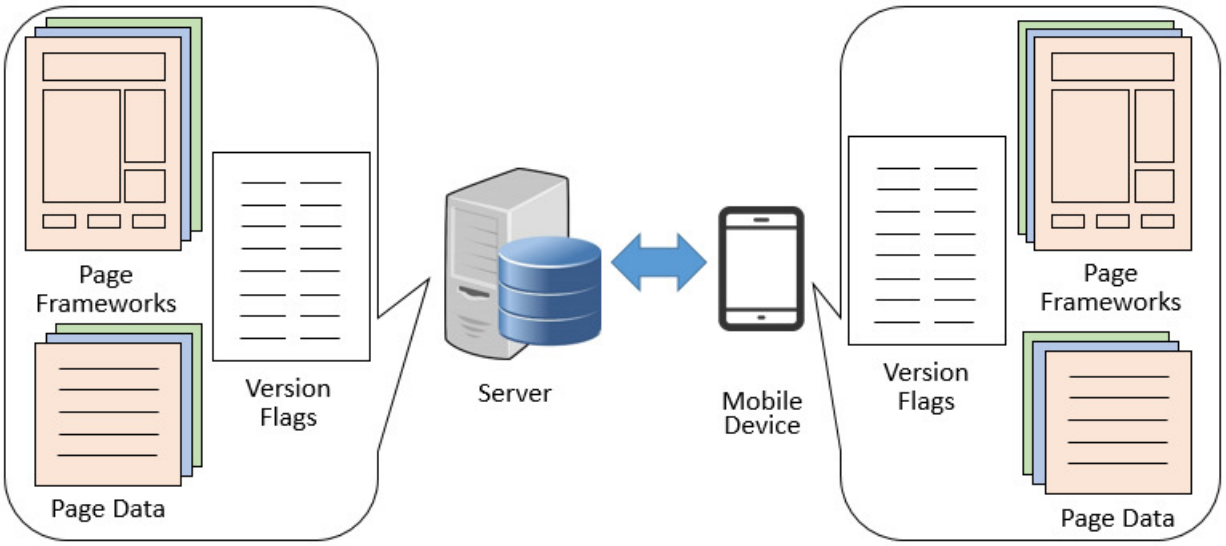


Figure 5.1: Architecture of PATAR

Page Separation: We separate the page contents into two parts, PF and PD, while developing applications. The purpose of page separation is that the mobile apps can respectively download and cache the two parts to the local when running apps. Two parts are combed into a complete page and the apps use WebKit to load and display the page.

Version Flags: Both servers and mobile devices maintain a set of version flags. Since the data receiver is the initiator of the network communication, the receivers cannot know which data need download from cloud side and which data do not change so they do not need download them

again. We can use version flags to indicate which data need update. Mobile apps download this set of version flags from server while mobile apps start. Comparing version flags local cached with version flags downloaded, the apps can select to just download the contents that need updating from servers.

Data Format: We select *JavaScript Object Notation* (JSON) as the data interchange format in this case. JSON is a text format that is completely language independent. Version flags can define flexibly according to the needs of applications. For example, we define version flag format of a page in our system coarse-grained like this:

{

“frame”:“12”,

“data”:“123”,

}

Requests and responds of data is also encapsulated to JSON string while they are uploaded and downloaded. The data format can define flexibly according to the needs of apps.

Compression: The servers compress the page components that need to be downloaded to further reduce network traffics. As the data which is transported between server and mobile terminal is basically character string, there is enough compression space. The compression can reduce the networking traffic, even though it increases the workload of compression and decompression, Compressing data increases the performance in wireless communications. We will design algorithms and make experiments to prove the efficient of our proposed schema.

5.4 Motivational Example of PATAR

We give a motivational example to explain PATAR algorithm.

In our algorithms, we use a map $Map\langle pName, vMapCached\rangle flags$ to store version flags, which are information pairs $pName$ and $vMapCached$. Here $pName$ refers to a $PageName$. $vMapCached$ refers to another map $Map < vName, vValue >$. In $vMapCached$, $vName$ refers to a version flag name such as “framVersion” and “dataVersion”. $vValue$ refers to a version

flag value that is an integer.

We assume there is an app using PATAR algorithm runing on a mobile device. The server is ready and the app has been installed into a mobile device at first.

The app's cache is empty at that time. When the app try to display a page as user interface, it search the version flags $vMapCached$ in version flags cache $flage$. It cannot find version flags corresponding the page now, of course. Thus, the app send the request to the server and download all components of the page, including Page Framework, Page Data, and version flags as $vMap$, which has the same data type with $vMapCached$. Then, the app store all these components into local cache. This is the process of page initialized. After this, the app combine Page Framework and Page Data into a HTML file and cache it, too. Finally, the app call Webkit to load and show the page to users.

Several minutes ago, the app try to show the page that is initialized again. It search the version flags $vMapCached$ in local cache $flags$ and get them at first. Then, the app send the request to ask for newest version flags $vMap$ of the page from the server. After download the newest version flags, the app compare the version flags from server $vMap$ and the version flags stored in local cache $vMapCached$. We assume they are all same at this time. This means all the page components have been ready. Thus, the app just call Webkit to load HTML file in the cache and show it to users.

A few days ago, the app try to show this page once again. It search the version flags in local cache $vMapCached$ and get them at first. After this, the app send the request to ask for newest version flags of the page $vMap$ from the server. Then the app compare the version flags from server $vMap$ and the version flags stored in local cache $vMapCached$. We assume there are some version flags are different at this time. This means some page components was expired. In this case, the app download expired page components that the version flags indicated and update this part components in local cache. Then, the app update the $vMapCached$. After doing this, the app combine Page Framework and Page Data into a HTML file and update it in local chache. Thus, all components of the page store in local cache are up to date now. Finally, the app call Webkit to load and show the page to users.

The above is the process of app using PATAR algorithm. It can reduce network traffic effectively normally.

5.5 Algorithms of 2SEPA, OPLDA, and DSycA

In this section, we propose a high efficient algorithm to save the total network traffic cost and power cost using a pre-cache technique.

Our algorithms run on the server end and mobile device respectively. The algorithm 3 runs on server end. The algorithm3 maintain all materials of the pages, such as version flags, Page Frameworks, and Page Data, and provide them to mobile devices.

Algorithm 3 Server-Side Execution Procedure Algorithm (2SEPA)

Require: Information pairs $Map < vPName, vVMap > flags$,
the Page Data, and the Page Frameworks
Ensure: The newest Page Components

```
start server, listen to a specific port
IF received the request of the version flags THEN
    encapsulates the version flags to JSON string;
    compress the JSON string;
    send back the compressed JSON string;
ENDIF
IF received the request of the Page Data THEN
    encapsulates the Page Data to JSON string;
    compress the JSON string;
    send back the compressed JSON string;
ENDIF
IF received the request of the Page Framework THEN
    compress the Page Framework;
    send back the compressed JSON string;
ENDIF
IF received the request of the Page Data update THEN
    updates the Page Data;
    adds 1 to the data version flag on server;
ENDIF
IF received the request of the Page Framework update THEN
    updates the Page Framework;
    adds 1 to the framework version flag on server;
ENDIF
```

On the server side, we start server program at first. Let sever program listen to a specific port, and wait for the access of the mobile devices. The server encapsulates the version flags to JSON string, compress and send it back to mobile device after receiving the request of the version flags from the mobile device. According to the data flags provided by the mobile device, the server encapsulates the recorded data to JSON string, compress and send it back after receiving the request of the data from the mobile terminal. The server compresses the stored Page Framework contents, compress them and then send them back after receiving the request of the page framework from the mobile device. The server updates the data, and then adds 1 to the Page Data version flag on the server after receiving the request of the data update from the mobile device. The server updates the page framework stored by the server, and then adds 1 to the Page Framework version flag on the server after receiving the request of the page framework update. This is the execution process on Server.

The mobile device also maintain a set of version flags $Map < pName, vMap > flags$ at local. The mobile app use these version flags to decide if it need to download page components from server when the app try to load a page. Version flags also indicate which components need to download.

The Algorithm 4 describe the algorithm of loading page.

On mobile devices, a mobile app searches the version flags of a page when the app try to show the page. The page will be initialized if the version flags of this page cannot find. When a page is initialized, All materials of this page are downloaded and cached to mobile device, such as version flags, Page Frameworks, and Page Data. Then the mobile app combines Data Cache and Framework Cache to generate a new HTML page file, and store the HTML page in local cache. If the version flags of this page are found, the mobile app synchronize the page using version flags. Finally, the app load the page when the page is ready.

For every initialized page, it must be synchronized with server before mobile app load it. The Algorithm 5 describe the algorithm of synchronize a page.

In algorithm 5, the mobile app download and uncompress the version flags of the page try to show firstly. Then the downloaded version flags are compared with corresponding version flags

Algorithm 4 On-Premise Load Data Algorithm (OPLDA)

Require: Information pairs $Map < vPName, vVMap > flags$,
the name of page to be shown $StringpName$
Ensure: The HTML page file web

IF $flags.get < pName >== NULL$ THEN
    download JSON string of version flags of the page from server;
    parse the JSON string into a map $Map < vName, vValue > vMap$
    $versionFlags.put(pName, vVMap)$;
    download Page Framework from server;
    download Page Data from server;
    combine Data Cache and Framework Cache;
    generate a new HTML page file web;
    cache the HTML page;
ELSE
    synchronize the page;
ENDIF
call WebKit to load the HTML page;

Algorithm 5 Page Synchronization Algorithm (DSycA)

Require: Information pairs $Map < vPName, vVMap > flags$,
the name of page to be shown $StringpName$
Ensure: The HTML page file web

$changed \leftarrow false$
download compressed JSON string of version flags of the page from server;
uncompress JSON string;
parse the JSON string into a map $Map < vName, vValue > vMap$
$Map < vName, vValue > vMapCached \leftarrow flags.get < pName >$
For EACH pair $< vName, vValue >$ IN $vMap$
    IF $vValue! = vMapCached.get < vName >$ THEN
        download compressed page component that $vName$ refer to.
        uncompress downloaded component and update cache.
        $vMapCached.put < vName, vValue >$
        $changed \leftarrow true$
    ENDIF
ENDFOR
IF changed == true THEN
    combine Data Cache and Framework Cache;
    generate a new HTML page file web;
    cache the HTML page;
ENDIF

stored in local cache. If downloaded version flags are not same with local cached version flags, the app download the components that version flags indicate and update page component cache and version flags. After this, the mobile app combines Data Cache and Framework Cache to generate a new HTML page file, and store the HTML page in local cache if any page component is updated. Now the page that app try to load is ready.

5.6 Experiment

The PATAR algorithm is the improvement of the HWK model. Using our algorithm, the mobile app can avoid download lots of components that already is stored in local cache. That can save many network traffics. Thus, the energy consumption can discount. The latency of load page can reduce at the same time. This can improve the user experience of apps and can increase the duration of mobile devices.

Our experiment accomplished comparisons between HWK model and our proposed schemas. The comparisons focused on comparing the energy consumptions and execution latency for each app. And the outcomes of the experiment indicated that our algorithm has a better performance than HWK model. The examined apps are listed in Table 5.1. We compare a variety of parameters, including functions, *Data Update Frequency* (DUF), PF, and PD.

Table 5.1: The attributes of apps.

App Name	Function	DUF	PF	PD	Total
App1	Email	Medium	1.25MB	42KB	1.29MB
App2	MSG	Medium	832KB	30KB	862KB
App3	Notification	Low	1.3MB	1KB	1.3MB
App4	Social Networks	High	523KB	1.6MB	2.11MB
App5	Real-Time Price	High	3.4MB	1KB	3.4MB

We use three devices to run our experiments, as shown in Table 5.2.

The first experimental device is HUAWEI U8860 Smartphone with Android OS v2.3 (Gingerbread), Qualcomm MSM8255T Snapdragon Chipset, 1.4 GHz Scorpion (one core) CPU, Adreno 205 GPU, 512 MB RAM, 1 GB Storage. The second experimental device is Samsung Galaxy S5

Table 5.2: Experiment Devices

Device	HUAWEI U8860	Samsung Galaxy S5	AVD
OS	Android OS v2.3 (Gingerbread)	Android OS v4.4.2 (KitKat)	Android OS v5.0.1 (Lollipop)
Chipset	Qualcomm MSM8255T Snapdragon	Qualcomm MSM8974AC Snapdragon 801	N/A
CPU	1.4 GHz Scorpion (one core)	Quad-core 2.5 GHz Krait 400	armeabi-v7a (Intel Core i5-4210U Host CPU)
GPU	Adreno 205	Adreno 330	Intel HD Graphics Family Host GPU
RAM	512 MB	2 GB	343MB (8.0GB Host RAM)
Storage	1 GB	32 GB	512MB

Smartphone with Android OS v4.4.2 (KitKat), Qualcomm MSM8974AC Snapdragon 801 Chipset, Quad-core 2.5 GHz Krait 400 CPU, Adreno 330 GPU, 2 GB RAM, 32 GB Storage. The Third experimental device is *Android Virtual Device* (AVD) in *Android Software Development Kit* (Android SDK) with Android OS v5.0.1 (Lollipop), armeabi-v7a CPU, 343MB RAM, 512MB Storage. This AVD running on a Host with Windows 8 64bit OS, Intel Core i5-4210U CPU, 8.0GB RAM, Intel HD Graphics Family GPU.

5.6.1 Energy Consumption

There is no a controvertible method to qualify the energy consumption. Energy consumption is related to many factors, such as execution time, signal frequency of channel, compressing strategy, data amount, network traffic, etc. Normally, there is some kind of proportionality between network traffic and energy consumption. In our experiments environment, most factors are constant except the network traffics. Thus, We use the network traffic to analyse the energy consumption. The figure 5.2 shows the comparing result of energy consumption of two apps developed by two corresponding algorithm.

According figure 5.2, we see that our algorithm can discount network traffic effectively, especial the apps that have less *Page Data*, such as notification apps and real-time price apps. Therefore, our algorithm can increase the duration of mobile devices.

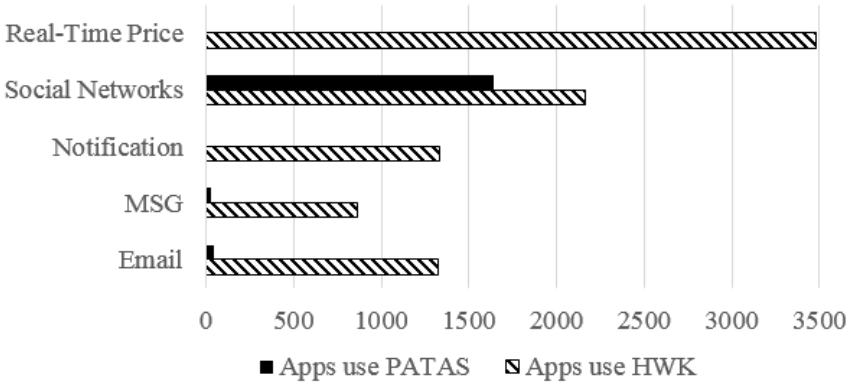


Figure 5.2: Comparisons of Network Traffic

5.6.2 Execution Latency

Execution latency is related to many factors, too, such as the CPU frequency, memory usage, communication technology, network traffic, network bandwidth etc. In our experiments, we restart the devices before every app running to reduce environmental interference. Thus, we do our best to guarantee that every app runs under the same device environment.

We run the apps on HUAWEI U8860 Smartphone at first. The figure 5.3 shows the comparing result of execution latency of two apps developed by two corresponding algorithm.

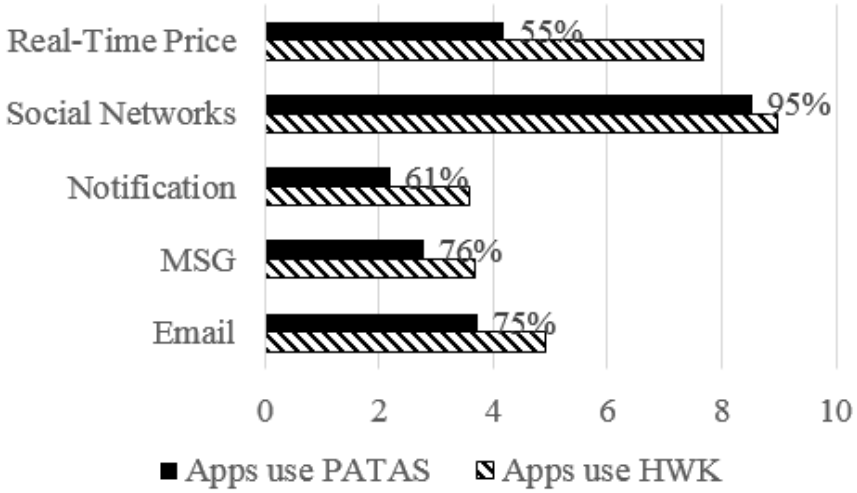


Figure 5.3: Comparison of Execution Latency on HUAWEI U8860

Next, we run the apps on Samsung Galaxy S5 Smartphone. The figure 5.4 shows the comparing result of execution latency of two apps developed by two corresponding algorithm.

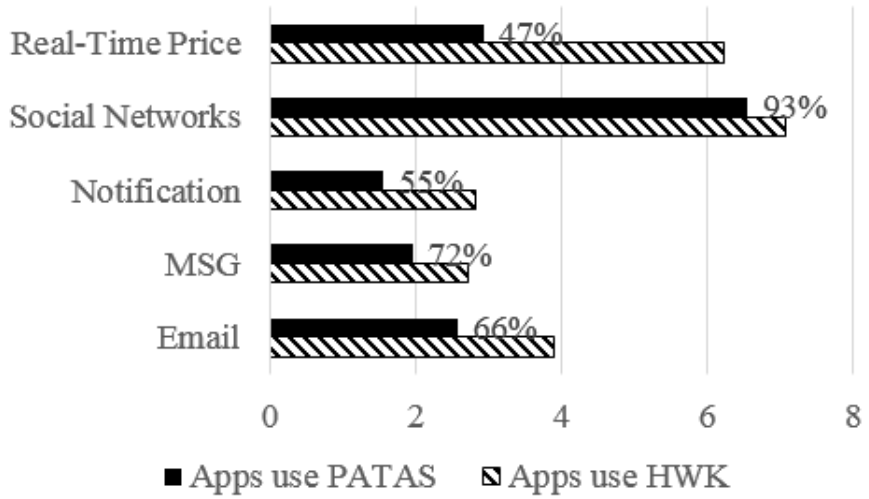


Figure 5.4: Comparison of Execution Latency on Samsung Galaxy S5

Finally, we run the apps on AVD. The figure 5.5 shows the comparing result of execution latency of two apps developed by two corresponding algorithm.

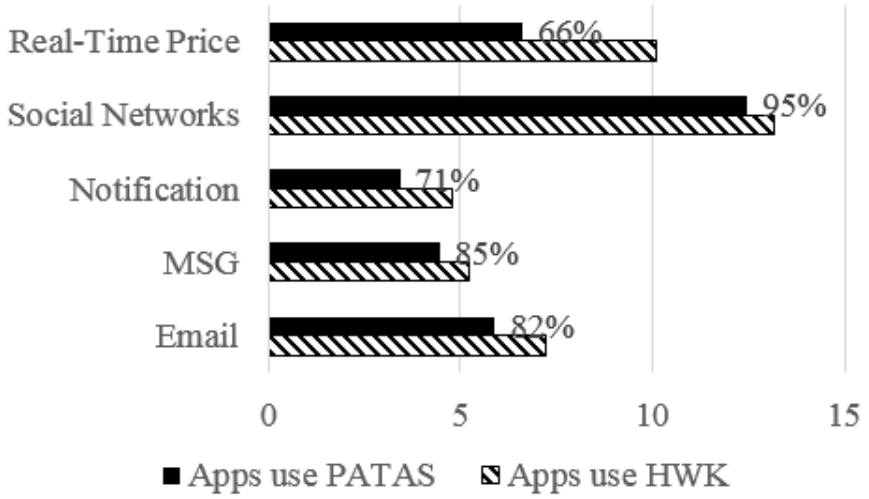


Figure 5.5: Comparison of Execution Latency on AVD

According figure 5.3, figure 5.4, and figure 5.5, we see that the apps using PATAR algorithm have lower execution latency than the others on every device. Better specifications devices have, higher the declining proportion of execution latency is. Thus, our algorithm can provide better user experience.

Our algorithm is the improvement of the HWK model. This algorithm allows programmer

use the HTML to develop apps user interface. It can make traditional PC access and mobile access efficiently integrated into one system. It also can make the development efficient. The experimental results show that our proposed mechanism can enhance the performances in network traffic control, power management, and executive latency.

5.7 Conclusions

There are many repeated data in the wireless network flow where the mobile devices communicate with the cloud servers. This chapter aim to reduce repeated network traffics in MCC when the data receivers are the initiators of network communication. We proposed a novel model, PATAR, to solve this problem. This model use version flags to indicate which data need update. Thus, receivers can avoid download unnecessary data from cloud side. The experimental results show that our proposed mechanism can enhance the performances in network traffic control, power management, and executive latency.

Chapter 6

Network Traffic-Aware Mobile Web Framework

Based on the above problem, we can make page frame fine grit as app elements and share the app elements with the web pages and apps that use the same app elements. In this chapter, we proposed a new mobile app model using precache technology to overcome the current existing obstacles. For supporting the model, we propose two algorithms, which are designed to reduce data redundancy and save total network traffic costs with using pre-cache technology. The results of experiments show that our model can reduce network traffic of Android apps efficiently.

6.1 Problem Description

Recent increasing demands of the Internet-related services have driven great needs of the Web services and interconnections between different service parties. The advance of the Web technologies brings a great potential of shortening the time consumptions when a new service deployment is introduced. This trend requires a Web interface formulation mechanism offering higher-level efficiency in order to satisfy the dynamic requirements and continuous changing circumstances.

Moreover, in the perspective of the maintenance, adding news services or functionalities may cause a series of changes [79, 80].

Lots of app elements of different page are same in one app. For example, almost every page has the same header and footer in one app generally. Moreover, there are some app elements, such as exiting alert dialog is used by lots of different apps.

However, the traditional mechanism of web apps is that servers provide web pages and the mobile devices with downloading pages and loading them using WebKit. As a platform, this mechanism has several problems because there are many data redundancies. This increases the difficult of maintaining. Moreover, redundancy data will increase unnecessary network traffics.

Now the challenge is how to design a high-performance-oriented maintainable framework for Android by reducing network traffics. The main contributions of our proposed approach are twofold:

1. Our proposed model provides a higher-level upgradable capability increasing the networking communications efficiencies and reducing network traffics.

2. We propose an approach that is flexible for serving new computing service needs.

6.2 Related Work

As an implementation of embedded systems, high performance of the embeded systems has been addressed by researchers in recent years [81]. The recent achievements related to the improvements of the embeded systems have covered multiple aspects [82]. The first hemisphere is to achieve high reliable real-time performance. Implementing loop fusion and multi-functional-unit scheduling for multi-dimensional DSP was examined as an efficient approach for minimizing energy consumptions [83]. Another concern is that the optimizations need to be achieved within a hard of software timing constraints in heterogeneous embedded systems. Some approaches have been developed with using dynamic programming in heterogeneous embedded systems [84].

Next, saving energy consumptions is always a big concern in increasing the adaptions of

the embedded systems [85]. Previous research has done a few attempts in various dimensions in energy-saving fields. For instance, an energy minimization solution was developed by using three-phase time-aware approaches with using DVFS and unrolling for chip multiprocessors [86]. Adopting loop parallelism maximization for multi-core DSP architecture was proved to be an energy-aware approach [87]. In order to deal with multi-tasks, an energy-aware algorithm was developed to guarantee the probability satisfying timing constraints [88].

Furthermore, prior research explorations also considered using advanced algorithms to improve the performances of the existing infrastructure [89]. For instance, a novel schema with using pre-cache technology was proposed to reduce network traffic [90]. A solution with using scratch-pad memory was developed to optimize data allocations on embedded multi-core systems [91]. Further explorations have achieved low-power as well as low-latency with using hybrid scratch-pad memory [6]. Another example was minimizing transferred data for code update on wireless sensor networks for the purpose of performance improvements [92].

Finally, cloud computing has become an important approach to increase the performances or functionalities of tele-health systems [93]. Various dimensions within cloud computing have been attempted by prior research [42]. One of the dimensions was using online optimizations for scheduling preemptable tasks on *Infrastructure-as-a-Service* (IaaS) cloud systems.

In summary, very few prior research addressed the issue of network traffics in implementing mobile embedded systems. This work focuses on optimizing the performances of the mobile systems from the perspective of reducing network traffics.

6.3 Models and Concepts of APWI

Focusing on the task *High-Performance Frameworks*, we propose an approach using pre-caching techniques to formulate an efficiency-aware Web interface schema. The proposed schema is named as *Adaptive Pre-caching Webpage Incubator* (APWI) model, which is designed to efficiently meet the requirements of the interfaces changes according to the constant changing demands. The proposed model aims to reduce data redundancy and improve the maintainability of Android apps. At

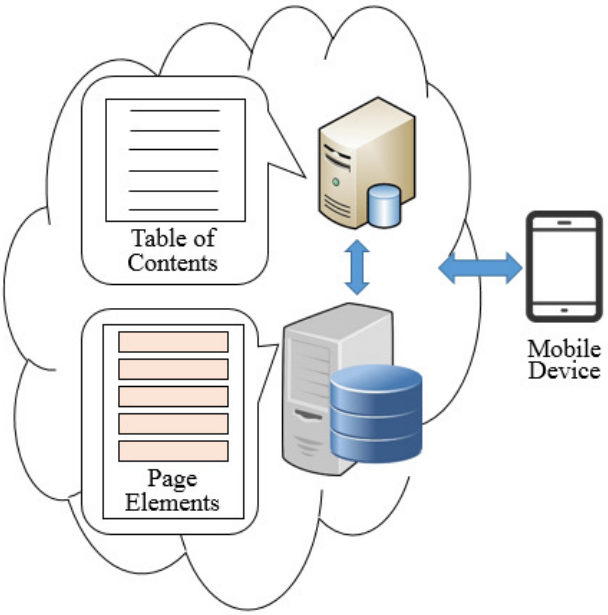


Figure 6.1: Architecture of the proposed APWI Model

the same time, the proposed model can reduce network traffic effectively.

The significance of our proposed mechanism is that we consider the execution efficiency as well as the maintenance capacity. The proposed schema can be used for reaching a high performance Web service meeting dynamic service demands. Two crucial aspects of our algorithms are covered in this section, including *Pre-cached* technology and *Version maintain* Technology.

The basic system architecture of the APWI model is shown in Fig. 6.1. In a APWI model, there are three main roles that include *Mobile Devices*, *Table of Contents Server*, and *Contents Servers*. A *Mobile Device* is a end device that the apps using APWI model running on it. *Table of Contens Server* (TS) is a server that stores and maintains a table of contents of app elements in APWI model. *Contents Servers* (CS) are a group of cloud servers that store and maintain a part of the app elements in APWI model.

App Elements

In APWI model, a *App Element* refers a part of page content, such as menus, page footers, page contents, and page data etc. *App Elements* are basic units of web pages in APWI model. Apps can combine *App Elements* into web pages by different ways. Each *App Element* is stored and maintained by *Contents Servers* individually. In this case, A description of a web page is a list

of *App Elements*

Page Speration

In APWI model, we seperate the web pages into several *App Elements*. How to sperate a part of page content as a *App Element* is very flexible. A good page speration policy can enhance reutilization rate of *App Elements*. *Mobile devices* can just request a part of *App Elements* in a page when theses *App Elements* are expired and load the other parts pre-cached *App Elements* from local storage. Then, *Mobile devices* combine all these *Page Elemnets* into a whole web page and show the web page to user using Webkit. This can avoid unnecessary network traffic and download waiting time effectually.

Table of Contents and Version Flag

Each *Mobile Device*, *Table of Contents Server*, and *Contents Server* maintain a *Table of Contens* respectively. The *Table of Contens* consist of a group of records. Each record has ID of *App Element* and *Version Flag* of *App Element*. The *Table of Contens* records the newest version of *App Elements*. Apps can use *Table of Contens* to judge wether the *App Elements* local cached is expired or not. When apps find that a *App Element* local cached is expired, it will request and download the newest version *App Element* from cloud servers.

Compression

There are different type of *App Elements*. For some kind of *App Elements*, such as text elements, compression can reduce the size of data effectually. In this situation, compression is helpful for data transmission. For some other kind of *App Elements*, such as pictures, videos, and audio, additional compression cannot reduce the size of data clearly because these type of data are already compressed. In this situation, compression is unnecessary. It will only waste processing time of CPU. Therefore, we have to decide wether a *App Element* should be compressed or not. When we upload a *App Element* to a *Contents Server*, the server compress this *App Element* and compare the compressed *App Element* with the original *App Element*. If the rate of compression is lager than a threshold value, the *Contents Server* store the compressed version and record this *App Element* is compressed. Otherwise, the *Contents Server* just store the original *App Element* and record it is uncompressed.

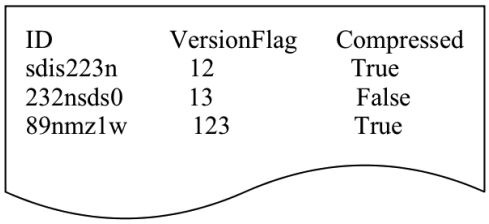


Figure 6.2: Data Format of Table of Contents

ID	VersionFlag	Compressed
A001	12	true
A002	20	true
A003	8	true
B001	234	true

Figure 6.3: Table of contents of page A

Data Format A whole record in *Table of Contents* has three fields including *ID*, *VersionFlag*, and *Compressed*. *ID* is a string label assigned to each *App Element*. All *IDs* of *App Elements* are different each other. *VersionFlag* is a integer that indicate the version of corresponding *App Element* local cached. *Compressed* is a boolean value that indicate wether the corresponding *App Element* is compressed or not. A part of *Table of Contents* may look like Fig. 6.2.

6.4 Motivation Example of APWI

In this section, we give a simple example of using our proposed approach to process data processing by using pre-cache technology. Assume that there is an app using APWI model, which is installed on a mobile device. In this app, there are two pages named page A and page B respectively. Page A is separated into four parts, including page tittle, which id is A001, page body frame , which id is A002, page footer, which id is A003, and page data, which id is B001. Page B also has four parts: page tittle, which id is A001, page body frame , which id is A004, page footer, which id is A003, and page data, which id is B002.

At first, this app try to show page A to the user. The app download table of contents of page A from mobile cloud server.

As displayed in Fig. 6.3, The table of contents shows that page A include four page elements,

ID	VersionFlag	Compressed
A001	12	true
A004	14	true
A003	8	true
B002	210	true

Figure 6.4: Table of contents of page B

which are A001, A002, A003, and B001. The table of contents also shows that the current version flags of these page elements are A001: 12, A002: 20, A003: 8, and B001: 234. Then the app search version flags of these page elements in local cache. It cannot find anything since there is nothing cached now. Thus, the app send a request to mobile cloud server to download page elements A001, A002, A003, and B001. After download these page elements successfully, the app cache these version flags and page elements on local cache. Next, the app combine these page elements as a whole page and cache this combined page. Finally, the app shows this page using webkit.

Then, the user ask the app shows page B. The app download table of contents of page B from mobile cloud server.

As displayed in Fig. 6.4, The table of contents shows that page A include A001, which version is 12, A004, which version is 14, A003, which version is 8, and B001, which version is 210. Next, the app search version flags of these page elements in local cache. It find that A001 and A003 are updated. A004 and B002 need download. Therefore, the app request A004 and B002 from mobile cloud server. When download these page elements, the app save the version flags and the page elements into local cache. Then the app combine the page B using page elements A001, A004, A003, and B002 and cache page B. Finally, page B is shown by the app.

After a while, the app try to show page A again. It request table of contents of page A. We assume table of contents of page A that download form cloud server is same as Fig. 6.3 After compare the version flags download from cloud server and the version flags local cached, the app knows that the page A local cached are updated. Thus, the app does not download anything. It just shows the local cached page A to the user.

6.5 Algorithms of PEIS and PEIL

In this section, we represent our proposed algorithms that are designed to reduce data redundancy and save total network traffic costs with using pre-cache technology. The algorithms support activities on both cloud servers and mobile devices.

Notations used in this chapter are listed in Table 6.1.

Table 6.1: Main notations and definitions

Notations	Definitions
©	An operator for accomplishing a "compress" activity
◯	An operator for accomplishing an "uncompress" activity
TC	Table of Contents, a database of Record
TCC	A part of Table of Contents Cached locally
$Record$	a record in TC
Req_r	Request of records
RC	Record Cached locally
RD	Record Download from cloud server
$List_r$	Records List
PR	Page Element request
PE	Page Element
PEC	Page Element Cached locally
$PE_{©}$	Compressed Page Element
$PE_{◯}$	Uncompressed Page Element
$List_p$	Page Elements List
Req_p	Request of page elements
Req_pu	Request of page elements update
PD	Page description

The first algorithm is PEIS that is shown in 6

The algorithm runs on cloud servers, which is designed to maintain all materials of the pages, such as Table of Contents and Page elements. As shown in the algorithm, Algorithm 6, there are a number of phrases that are given as follows.

1. On the cloud servers, we start server program. The sever program listen to a specific port and wait for the access request of the mobile devices.

Algorithm 6 Page Element Implementations on Server(PEIS)

Require: TC, Req_r, Req_p, and Req_pu
Ensure: Updated TC

start server, listen to a specific port
IF received Req_r THEN
    new $List_r$
    FOR$\forall$ $Req_r.ID$
        select $Record$ from TC
        $List_r$.add($Record$)
    ENDFOR
ⓒ $List_r$
    send back the compressed $List_r$
ENDIF
IF received Req_p THEN
    new $List_p$
    FOR$\forall$ $Req_p.ID$
        select PE
        $List_p$.add(PE)
    ENDFOR
    send back the $List_p$
ENDIF
IF received Req_pu THEN
    select $Record$ from TC
    IF $Record$ == NULL
        new $Record$ using {new ID, 0, false}
    ENDIF
    ⓒ received PE
    IF $PE_{\bigcirc}$.size/$PE_{©}$.size $>$ threshold value
        update PE using $PE_{©}$
        $Record.compressed \leftarrow true$
    ELSE
        update PE using $PE_{\bigcirc}$
        $Record.compressed \leftarrow false$
    ENDIF
add 1 to $Record.versionFlag$
store $Record$ into TC
ENDIF

2. The cloud server select correspond records as a records list from Table of Contents, compress the records list and send the records list back after receiving records request.

3. The cloud server select correspond page elements as a page elements list and send the page

elements list back after receiving page elements request.

4. After receiving the request of page elements update, the cloud server compress the page elements, update compressed page elements or uncompressed page elements according to the rate of compression, update the "compressed" field in the record, and add 1 to the "version flag" field in the record.

On the other end, the mobile devices maintain a part of Table of Contents TCC and some Page Elements PEC, which the mobile apps using. When a mobile app is going to load a web page, the algorithm 7 will be processed.

Algorithm 7 Page Element Implementations Local (PEIL)

Require: PD, TCC, and PEC
Ensure: The HTML page file web

```
download PD
◯ PD
generate List_r base on PD
new Req_p
FOR ∀RD in List_r
    select RC from TCC
    IF RC == NULL or RC.versionFlag != RD.versionFlag THEN
        Req_r.add(RD.ID)
    ENDIF
ENDFOR
IF Req_r != NULL THEN
    request List_p using Req_r
    FOR ∀ PE in List_p
        IF PE.compressed == ture
            ◯ PE
        ENDIF
        PEC ← PE
    ENDFOR
ENDIF
Compose web using PECs
call WebKit to load web
```

As shown in Algorithm 7, there are a several steps that are given as follows.

1. The mobile app download Page description from cloud server.

2. Record request list is generated using page elements IDs in page description.

3. Download Record List using record request list.

4. Uncompress Page Elements if necessary.

5. Update expired Page Elements according to the Downloaded Record List and Cached Record List.

6. Compose HTML web page using updated Page Elements

7. Load generated HTML web page using WebKit.

6.6 Experiment and the Results

In our experiment, we use serval Android apps running on 2 Smart Phones and Android Emulator to test our proposed model. The apps are implemented using our proposed model and traditional Android develop mechanism. We compare the apps implemented by the 2 ways to test the effectiveness.

6.6.1 Experimental Configuration

The examined apps are listed in Table 6.2

Table 6.2: The attributes of apps

App Name	**Function**
App1	Email
App2	MSG
App3	Notification
App4	Social Networks
App5	Real-Time Price

Fig. 6.5 shows that the comparatione of apps developed by two mechanism respectively.

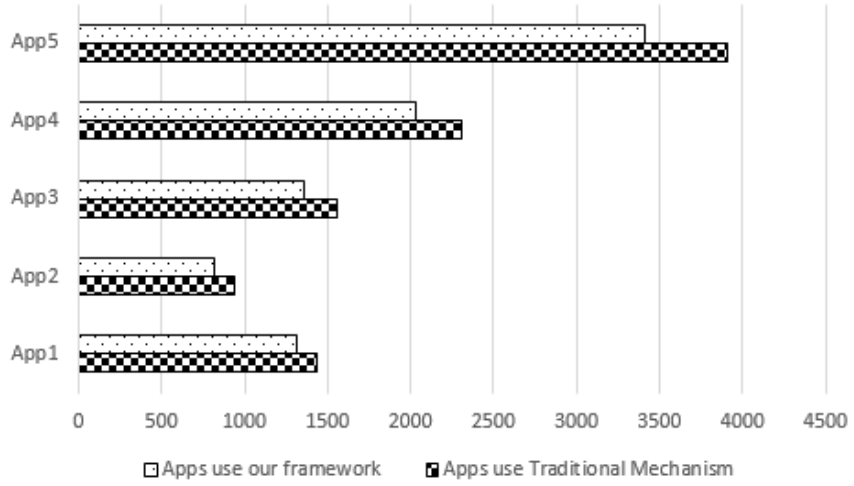


Figure 6.5: Comparisons of Size

From Fig. 6.5, we can see that apps using our proposed model have less size because they can share the same page elements. This reduce data redundancy and improve maintainability of Android apps.

Then, We used 3 Android devices to run these apps. The first device is HUAWEI U8860 Smartphone with Android OS v2.3 (Gingerbread), Qualcomm MSM8255T Snapdragon Chipset, 1.4 GHz Scorpion (one core) CPU, Adreno 205 GPU, 512 MB RAM, 1 GB Storage. The second device is Samsung Galaxy S5 Smartphone with Android OS v4.4.2 (KitKat), Qualcomm MSM8974AC Snapdragon 801 Chipset, Quad-core 2.5 GHz Krait 400 CPU, Adreno 330 GPU, 2 GB RAM, 32 GB Storage. The Third device is *Android Virtual Device* (AVD) in Android SDK with Android OS v5.0.1 (Lollipop), armeabi-v7a CPU, 343 MB RAM, 512 MB Storage. This AVD running on a Host with Windows 8 64 bit OS, Intel Core i5-4210U CPU, 8.0 GB RAM, Intel HD Graphics Family GPU.

6.6.2 Experimental Results

We run the apps implemented by traditional mechanism and our model thirty times and calculate the average network traffic respectively. Fig. 6.6 showed that the comparisons of network traffic between traditional mechanism and our proposed model. According to the figure, the network

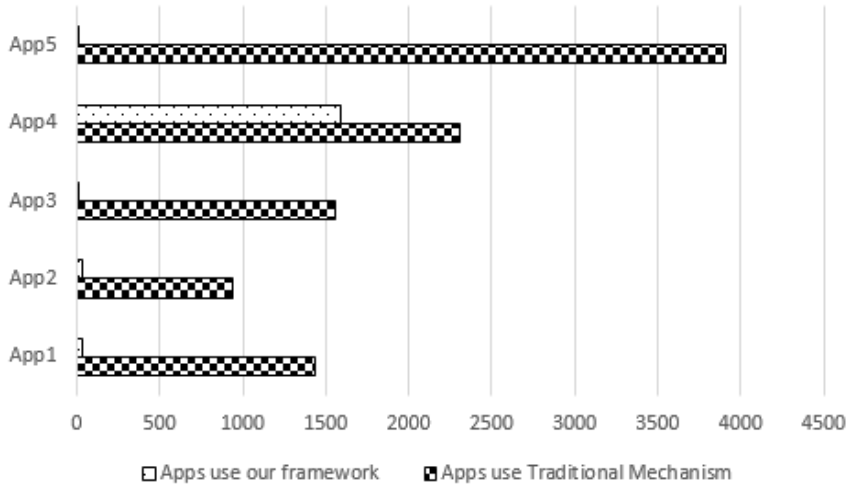


Figure 6.6: Comparisons of network traffic between traditional mechanism and proposed model

traffic consumption of apps designed by our model was reduced clearly. Using our model can save network traffic effectively.

Next, we compared the time consumption of apps designed by traditional mechanism and our model. We restart the devices every time before run the test apps to minimize the disturb of system. The Fig. 6.7, Fig. 6.8, and Fig. 6.8 shows the result of comparatione using HUAWEI U8860, Sumsungas Galaxy S5, and AVD as test device respectively.

According to the The Fig. 6.7, Fig. 6.8, and Fig. 6.9, we can see that our proposed model can reduce the time consumption of apps. Moreover, it shows that the level of device specifications has a positive relationship with the declining proportion of the time consumption because the time proportion of network traffic goes up when the devices has a fast process speed. For example, Samsung Galaxy S5 has the highest computation capability among the examined mobile devices. Our approach has the greatest advantage by using this Samsung device.

In summary, the results of experiments show that our model can reduce network traffic of Android apps efficiently. At the same time, using our proposed model can reduce data redundancy and improve maintainability of Android apps.

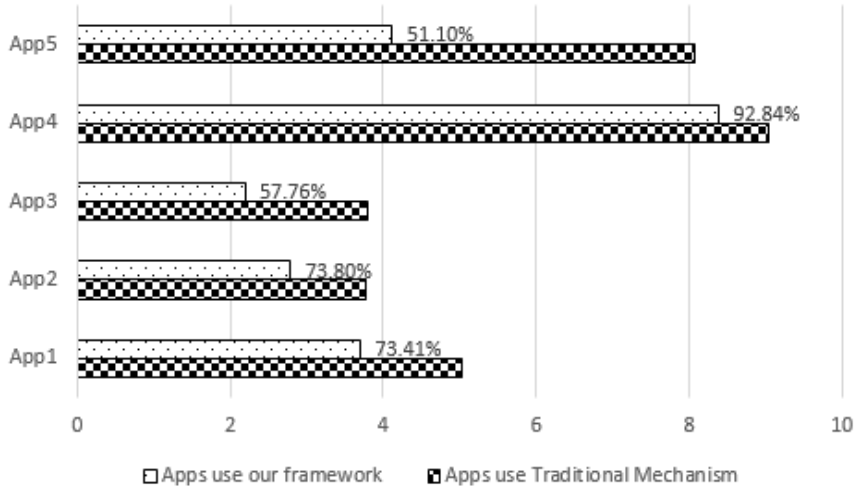


Figure 6.7: Comparison of Time Consumption on HUAWEI U8860

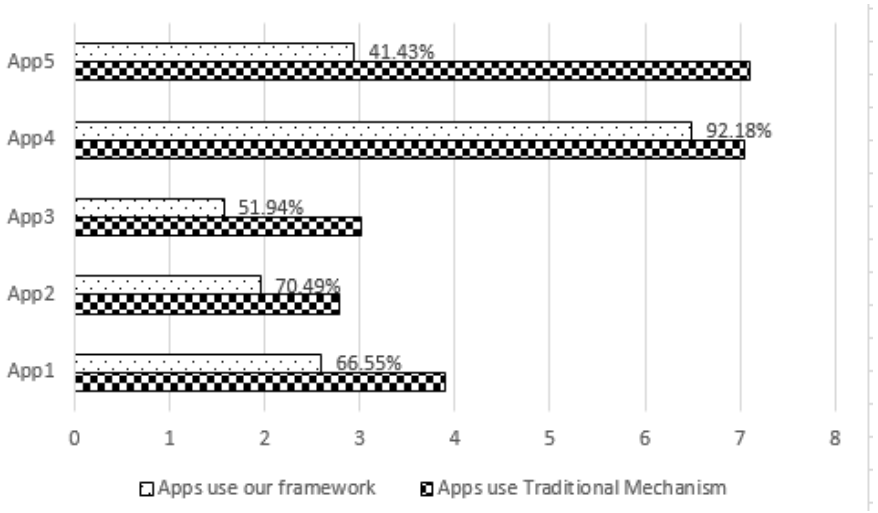


Figure 6.8: Comparison of Time Consumption on Samsung Galaxy S5

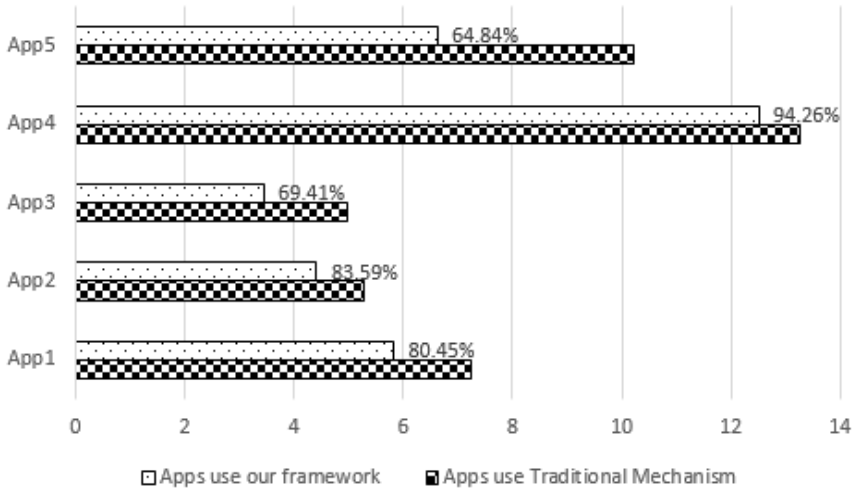


Figure 6.9: Comparison of Time Consumption on AVD

6.7 Conclusions

This paper proposed a novel mobile app model based on pre-cache technology, APWI, for reducing the data redundancy and improving maintainability of Android apps. At the same time, using APWI can reduce network traffic efficiently The crucial algorithms in the proposed paradigm include PEIS and PEIL. The efficiency of our solution was proved by the experimental evaluation, which showed that our approach could reduce size of Android apps and network traffics.

Chapter 7

Optimal Solution to Intelligent Multi-Channel Wireless Communications.

Nowadays, multi-channel wireless communications are used on more and more intelligent mobile devices, such as smart mobile phones, smart watches, and tele-health devices [94, 95]. These multiple wireless communications have different energy costs and successful probability when they transfer data packets. In this chapter, we focused on the issue of the energy-saving for mobile devices in the context of MCC and designed an intelligent energy-aware communication strategy by using dynamic programming.

7.1 Problem Description

Currently, the implementation of networking-oriented technologies is turning into a strong driven force in the deployment of the intelligent solutions, such as embedded intelligent systems, *Internet-of-Things* (IoT) [96, 97, 98], cloud computing [99, 100][24, 101, 102], and edge computing [103, 104]. It is a desired objective for many enterprises or organizations to operate these emerging technologies for the purpose of gaining values. The crucial component of the value retrieval is gathering

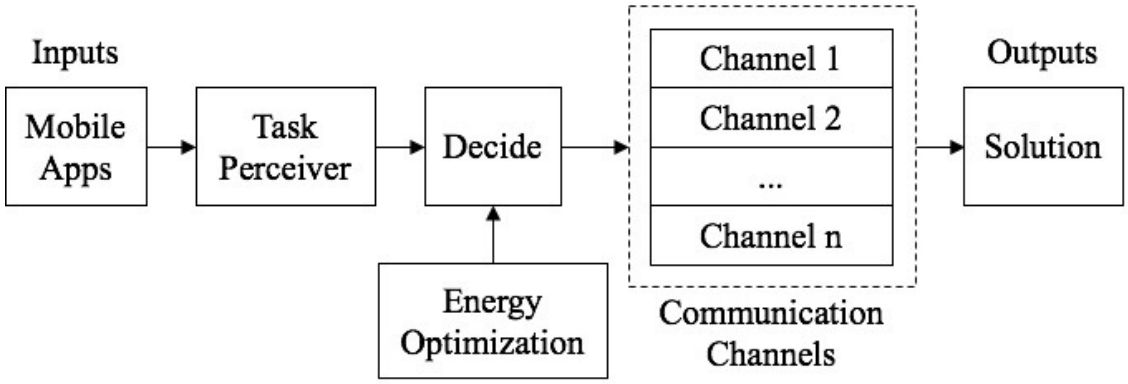


Figure 7.1: High level architecture of the proposed model.

data and acquiring valuable information from a large pool of data [105, 45, 48, 106, 107]. Contemporary techniques addressing information acquisitions are varied in order to meet various demands of data mining. A customized approach for achieving targeted functionality within a networked system is known as a specific-designed intelligent solution. There are numerous successful cases of implementing specific-designed intelligent solutions that depend on the large amount of data in practice, such as in tele-health, e-commerce, and the financial industry.

Despite great demands in reality and high return values, most current intelligent systems are facing a problem raised the heavy workload of data transmissions. Mobile devices generally are restricted by the limited energy supply capacity [108, 109, 110]. As a major component of the energy consumption, the increasing volume of data transmissions is challenging the consistent real-time services and large size data collections. Most mobile real-time services require consistent data transmissions; nonetheless, data transfers consume a major part of the energy on mobile batteries [111, 112, 113]. The phenomenon of heavy data transfers has raised a great concern about the enduring energy supplies in mobile computing. To address this concern, we formulate the energy-saving problem considering multiple constraints and present an optimal solution to minimizing the energy cost in this chapter.

Our approach is called *Intelligent Multi-Channel Communication* (IMCC) model, which considers multiple communication channels using various data transfer protocols. We standardize a number of parameters for each protocol, including the energy consumption, success transfer like-

lihood, and execution time. Fig. 7.1 illustrates the high level architecture of our approach. The mechanism of our approach is to deploy a *Task Perceiver* (TP) that perceives parameters of the input task. As shown in the figure, the input tasks are those data that need to be sent to the remote from mobile devices. It generally attaches to a certain mobile app. Thus, TP can be specifically designed for each objective application and it also can capture the parameters of the input data packages. We implement our energy optimization algorithm once receiving the parameters of the data package. Our optimization method considers the diversity of the energy cost as well as execution time for various transfer protocols. The output is a determination on the channel selections.

The significance of this work is relevant with data-driven applications in the MCC context. The main contributions of our work are summarized as follows:

1. This work presents an optimal solution to data transfers using the multi-channel method. The creation of the optimal solution considers the energy cost, execution time, and success rate and the output is generated by implementing our proposed dynamic programming. The objective is to minimize the total energy cost with a relative high success rate when the total execution time length is fixed.

2. Our work focuses on saving energy and formulates the problem by involving a number of crucial elements. The proposed problem is a NP-hard problem. Our approach can outcome an optimal solution by partially solving the proposed NP-hard problem.

7.2 Related Work

This section synthesized recent academic work related to our research. The review concentrated on the fields of communication protocols, multi-channel applications, and optimizations.

First, the feasibility and adoptability of implementing multi-channel communications had been demonstrated by many previous studies [114, 115]. The operating principle of multi-channel communications was that parameters of protocols were distinctive from one another. There were many application layer protocols deployed in current networks. Besides common communication proto-

cols, some other alternatives included *Constrained Application Protocol* (CoAP), *Message Queue Telemetry Transport* (MQTT), *Extensible Messaging and Presence Protocol* (XMPP), *Representational State Transfer* (RESTFUL Services), *Advanced Message Queuing Protocol* (AMQP), and Websockets [116][117][100].

In addition, some of protocols mentioned above were suitable for deploying multi-channel communications comparing with non-multi-channel protocols, such as TCP. For example, CoAP was designed for both unicast and multicasts [118]. Meanwhile, MQTT was a protocol supporting control services, which needed less energy and lower bandwidth than that of HTTP, even though it might cause high latency [119][120]. Prior studies showed that various protocols carried different functional features in energy cost, bandwidth, latency, availability, and success transfer rate [121][119][122]; therefore, it was adoptable to standardize input data packages with providing parameters.

Moreover, many recent researches explored the implementation of multi-channel methods in the practical realm [89, 123]. Wood et al.'s [124] investigation depicted that parallel multi-channel communication using virtualization techniques could improve the usage of data. This work was based on a large real-world dataset in the context of IoT. Our work was different from this research because we focused on the multi-channel communications rather than multi-channel data source for virtualized displays. Another work completed by EISawy and Hossain [125] was proposing a framework for establishing uplink transmissions in either a single or multi-tier cellular networks. This work considered the power control of each user equipment and discussed the tradeoffs risen by the cutoff threshold and the transmit power. Instead, our work had a distinct focus that intended to have multi-channel connections between users rather than equipment power controls within a network.

Furthermore, some other research attempts also had similar research attentions to ours. Mo et al. [126] presented a method for designing cooperative communication protocols using the data of channel statistics. This method did not require data collections on channel states from the transmitter side and it also considered power and time. But the information retrieves depended on the source broadcast within pre-defined time intervals and optimization used Gaussian distributions.

However, the complexity of this method was high due to multiple relays and complex mathematical computations. Additionally, Gai et al. [89] proposed a privacy-preserving approach that used multi-channel communications. This approach considered the privacy protection the priority in the system design. Unlike Gai et al.'s [89] approach, our work emphasized energy-saving and mainly concerned the energy on the user end side.

Meanwhile, addressing the multi-channel scheduling problems, Yigit et al. [127] argued that multi-channel optimizations could be formulated into a scheduling optimization problem. Their study proved the argument and applied a quality-aware scheduling optimization algorithm in the multi-channel communications of the wireless smart grid. The mechanism used a *Log-Normal Shadowing* (LNS) model to model inputs and outputs were sub-optimal solutions. Our research had a similar intention to Yigit et al.'s work; however, we aimed to produce optimal solutions to minimizing the energy cost through an optimization scheduling algorithm.

Next, in another study, Almotiri et al. [128] claimed that they developed an optimal solution to maximizing the networking performance by proposing a distributed multi-channel MAC protocol for Ad Hoc wireless networks. It represented a group of academic attempts having similar research topics on networking capacities [1][129]. Almotiri et al.'s protocol used independent frequency hopping and ensured the maximum saturation for each channel. The maximum value relied on the optimal transmission probability. As our observation, the proposed problem was not a NP-hard problem due to the limited condition constraints. The focus of this work also was different from ours since it concentrated on the networking capacity optimizations.

In summary, our research problem was researchable that could be supported by the prior academic achievements. Many attempts on multi-channel problems had been made by the prior studies, even though rare research addressed the energy-saving issues. Thus, this proposed research had an observable difference from prior studies and could contribute to the body of knowledge. The next section showed an example about the implementation of the proposed model.

7.3 Motivational Example of IMCC

In this section, we give a simple example to explain our approach to find the optimal solution to deploying multi-channel connections with minimum energy costs. We assume that there are four channels and five data packets required for transmissions. Two working modes are available for each data packet at each channel. Distinct working modes are different from one another in success rates and energy costs. Details are given in Table 7.1. As shown in the table, T1 refers to the time consumption of the data packet using channel 1; P1 refers to the success rate of the data packet using channel 1; C1 refers to the cost of the data packet using channel 1.

Table 7.1: Raw Table for Data Packets.

Data	Channel 1			Channel 2			Channel 3			Channel 4		
	T	P	C	T	P	C	T	P	C	T	P	C
D1	1	0.8	9	2	0.8	5	4	0.4	1	3	0.7	7
	2	1	9	3	1	5	6	1	1	5	1	7
D2	4	0.7	6	4	0.7	5	5	0.5	2	4	0.7	9
	5	1	6	7	1	5	8	1	2	6	1	9
D3	3	0.6	4	2	0.6	2	4	0.4	1	3	0.8	5
	6	1	4	5	1	2	7	1	1	5	1	5
D4	2	0.6	7	1	0.8	6	3	0.3	3	2	0.9	10
	4	1	7	3	1	6	6	1	3	5	1	10
D5	5	0.8	8	4	0.9	6	5	0.4	4	3	0.5	12
	7	1	8	6	1	6	7	1	4	5	1	12

In our proposed model, we call Table 7.1 a *Raw Table* (R-Table). We use R-Table to further obtain a *Standard Table* (S-Table). Table 7.2 displays an S-Table that is corresponding with Table 7.1. In the table, T refers to the time consumption and the tuple at each cell is (p, c). P refers to the success rate and C refers to the cost.

Furthermore, we create a *Plan Table* (P-Table) to find out an optimal solution. First, we consider the situation that only has Data 1, such that the first row of S-Table is added into the first row of P-Table. Next, we consider adding Data 2. We obtain the second row of P-Table from the second row of S-Table and the first row of P-Table. For example, the sixth cell at the second row of P-Table $PTab\_(2,6)$ can be obtained from the following calculation: $PTab\_(2,6)$ = $\{STab\_4$

Table 7.2: Standard Table for Data Packets.

T	1	2	3	4	5	6	7	8
D1	(0.8, 9)	(1.0, 9) (0.8, 5)	(1.0, 5) (0.4, 1)	(1.0, 5) (0.4, 1)	(1.0, 5) (0.4, 1)	(1.0, 1)		
D2	-	-	-	(0.7, 5)	(1.0, 6) (0.7, 5) (0.5, 2)	(1.0, 6) (0.7, 5) (0.5, 2)	(1.0, 5) (0.5, 2)	(1.0, 2)
D3	-	(0.6, 2)	(0.8, 5) (0.6, 2)	(0.8, 5) (0.6, 2) (0.4, 1)	(1.0, 2) (0.4, 1)	(1.0, 2) (0.4, 2)	(1.0, 1)	
D4	(0.8, 6)	(0.9, 10) (0.8, 6)	(1.0, 6) (0.3, 3)	(1.0, 6) (0.3, 3)	(1.0, 3)			
D5	-	-	(0.5, 12)	(0.9, 6)	(1.0, 12) (0.9, 6) (0.4, 4)	(1.0, 6) (0.4, 4)	(1.0, 4)	

$\bigotimes$ $PTab\_2$, $STab\_5 \bigotimes PTab\_1\}$. In this operation, a $\bigotimes$ b = (a.probability * b.probability, a.cost + b.cost). Therefore, $PTab\_(2, 6)$ = $\{$(0.7, 5) $\bigotimes$ (1.0, 9), (0.7, 5) $\bigotimes$ (0.8, 5), (1.0, 6) $\bigotimes$ (0.8, 9), (0.7, 5) $\bigotimes$ (0.8, 9), (0.5, 2) $\bigotimes$ (0.8, 9)$\}$ = $\{$(0.7, 14), (0.56, 10), (0.8, 15), (0.56, 14), (0.4, 11)$\}$. We remove pairs (0.56, 14) and (0.4, 11) after comparisons and the eventual result is $PTab\_(2, 6)$ = $\{$(0.7, 14), (0.56, 10), (0.8, 15)$\}$. The same method is applied to add the rest of the data and the operation will not be stopped until all data are added into the P-Table. Table 7.3 illustrates the final P-Table. As shown in the table, when timing constraint is 14, one of the optimal solutions is that the cost is 32 with the success rate at 0.4608.

7.4 Concepts and the Proposed Model

In this section, we introduce the basic models and concepts of our proposed schema, IMCC.

Table 7.3: Plan Table

T	Optimal Solution
11	(0.1344, 34)
12	(0.24192, 28)
13	(0.3456, 29), (0.3024, 28), (0.24192, 24)
14	(0.4608, 32), (0.432, 29), (0.3456, 25), (0.3024, 24), (0.1728, 21), (0.12096, 20)
15	(0.576, 32), (0.4608, 28), (0.432, 25), (0.3024, 24), (0.216, 21), (0.12096, 20), (0.0864, 17)
16	(0.648, 36), (0.576, 28), (0.432, 25), (0.4032, 24), (0.2688, 22), (0.216, 21), (0.1512, 20), (0.096, 19), (0.0864, 17), (0.0384, 15)
17	(0.72, 29), (0.576, 25), (0.504, 24), (0.384, 23), (0.3456, 21), (0.3024, 20), (0.192, 19), (0.1344, 18), (0.108, 17), (0.0576, 16), (0.0384, 15), (0.0324, 14)
18	(0.81, 33), (0.72, 25), (0.504, 24), (0.48, 23), (0.432, 21), (0.3024, 20), (0.24, 19), (0.216, 17), (0.096, 15), (0.0324, 14), (0.0144, 12)
19	(0.9, 29), (0.72, 25), (0.63, 24), (0.504, 23), (0.448, 22), (0.432, 21), (0.378, 20), (0.24, 19), (0.216, 17), (0.096, 15), (0.036, 14), (0.0216, 13), (0.0144, 12)
20	(1.0, 35), (0.9, 25), (0.72, 24), (0.64, 23), (0.576, 21), (0.504, 20), (0.384, 19), (0.336, 18), (0.27, 17), (0.144, 16), (0.12, 15), (0.108, 14), (0.036, 12), (0.0096, 11)
21	(1.0, 29), (0.9, 25), (0.8, 23), (0.72, 21), (0.504, 20), (0.48, 19), (0.432, 17), (0.24, 15), (0.108, 14), (0.048, 12), (0.0096, 11)
22	(1.0, 25), (0.9, 24), (0.8, 23), (0.72, 21), (0.63, 20), (0.504, 19), (0.432, 17), (0.24, 15), (0.12, 14), (0.072, 13), (0.048, 12), (0.024, 11)
23	(1.0, 23), (0.9, 21), (0.72, 20), (0.64, 19), (0.56, 18), (0.54, 17), (0.36, 16), (0.3, 15), (0.27, 14), (0.12, 12), (0.032, 11)
24	(1.0, 23), (0.9, 21), (0.8, 19), (0.72, 17), (0.48, 15), (0.27, 14), (0.12, 12), (0.032, 11)
25	(1.0, 21), (0.9, 20), (0.8, 19), (0.72, 17), (0.48, 15), (0.3, 14), (0.18, 13), (0.12, 12), (0.08, 11)
26	(1.0, 19), (0.9, 17), (0.72, 16), (0.6, 15), (0.54, 14), (0.3, 12), (0.08, 11)
27	(1.0, 19), (0.9, 17), (0.8, 15), (0.54, 14), (0.3, 12), (0.08, 11)
28	(1.0, 17), (0.9, 16), (0.8, 15), (0.6, 14), (0.45, 13), (0.3, 12), (0.2, 11)
29	(1.0, 15), (0.9, 14), (0.6, 12), (0.2, 11)
30	(1.0, 15), (0.9, 14), (0.6, 12), (0.2, 11)
31	(1.0, 14), (0.9, 13), (0.6, 12), (0.5, 11)
32	(1.0, 12), (0.5, 11)
33	(1.0, 12), (0.5, 11)
34	(1.0, 11)

7.4.1 System Definition and Problem Statement

We assume that there are several transfer channels in mobile devices. Data packages can be transferred through these channels. Using different channels for transferring data packages requires

various time consumption, energy consumption, and success rate. The research problem is defined by the following Definition 9.

Definition 9 Minimizing Cost of Data Transmission on Multi Channel (MC-DTMC) Problem: *Inputs include a chain of data packages; the time consumption for each data package at a channel; and success rate for each data package attached to a channel. The output is a optimal data transfer plan. The research problem is finding out the data transfer plan that can minimize the total cost under acceptable success rate.*

In this problem, input include the number of data packages, the number of transfer channels, the time consumptions of each data package transfer through every channel in each mode, the success rate of each data package transfer through every channel in each mode. Our aim is to find an optimal plan to transfer every data packages through properly channel by using minimum total energy cost under success rate.

7.4.2 System Design

During solve problem MC-DTMC, we use three tables include the Raw Table, the Standard Table, and the Plan Table. **Raw Table** refers to the original data inputs. It includes all data mentioned in Definition 9. The transmission time can be guaranteed by various approaches, such as observation from historical records using time series predications or statistical methods. **Standard Table** is generated from the raw table. Actually, it just is the raw table in another format. In this table, every raw refer a data packages transmission parameters by using various channel. For every data package, we fill the (probability, cost) pairs into corresponding cell according the transfer time. **Plan Table** is the optimal solution under every appropriate time limitation. The algorithms to generate Plan Table are given in next section.

In our calculation, we use an operator $\bigotimes$ to calculate two (probability, cost) pairs. The result of this operation refer to the (probability, cost) pair when we transfer two data packages by using

two original pairs. Assume we have two (probability, cost) pairs, A and B. A: ($prob_a$, $cost_a$) B: ($prob_b$, $cost_b$) That means we transfer one data package by consuming energy $cost_a$, the success rate is $prob_a$ and then we transfer another data package by consuming energy $cost_b$, the success rate is $prob_b$ We define that

$$A \bigotimes B = (prob_a \times prob_b, cost_a + cost_b) \tag{7.1}$$

The result is the energy consumption and success rate when we transfer the two data packages.

Furthermore, we define a term **Unnecessary Pair** to refer a (probability, cost) pair that can be eliminated. An unnecessary Pair means a (probability, cost) pair has more energy consumption and lower success rate than another pair under the same time limitation. For example, there are two (probability, cost) pairs under time limitation 10, A and B. A is (1.0, 15); B is (0.9, 20). Obviously, pair A is better than pair B on both success rate and energy cost. Thus, pair B is an unnecessary pair and it can be eliminated.

7.5 Algorithms

7.5.1 Optimal Energy Cost Algorithm

In this section, we present our main algorithm, entitled *Optimal Energy Cost* (OEC) algorithm, which is designed to produce the minimum energy consumption based on the information retrieved from the implementation of ECD algorithm. Inputs of this algorithm include S-Table and time restriction. The output of this algorithm is P-Table

Algorithm 8 shows our proposed algorithm and the main phases of our algorithm include:

1. Input *S-Table*.

2. Start generating P Table. We copy $STab_0$ to $PTab_0$ to produce the first line of P-Table.

Algorithm 8 Optimal Energy Cost (OEC) algorithm

Require: The S-Table $STab$
Ensure: The P-Table $PTab$
1: input the S-Table
2: $PTab\_1 \leftarrow STab\_1$
3: FOR $\forall$ data i
4: FOR $\forall$ channel j
5: FOR $\forall$ k in $STab_i$
6: IF $PTab_{(i-1,j-k)}$!= null
7: $PTab_{(i,j)}$ = $PTab_{(i-1,j-k)}$ @ $STab_{(i,k)}$
8: /* @: denotes that executing $\bigotimes$ operations on all (p, c) pairs from $PTab_{(i-1,j-k)}$ with all (p, c) pairs from $STab_{(i,k)}$ */
9: END IF
10: END FOR
11: remove unnecessary pairs
12: END FOR
13: END FOR
14: RETURN $PTab$

3. We calculate $data_1$ of the P-Table by using the partial P-Table generated by the last Step 2 and partial S-Table $STab_1$.

4. Add all data by using the same method used at Step 3 until the whole P-Table is generated.

7.5.2 Energy Cost Detection Algorithm

We present an *Energy Cost Detection* (ECD) algorithm in order to standardize the parameters of the input data package. The input of algorithm ECD is the R-Table. The output of this algorithm is the S-Table.

Algorithm 9 shows ECD algorithm and the main phases of our algorithm include:

1. Input *R-Table*.

2. Start generating S-Table. Take each (p, c) pair from a triple (t, p, c) and add them into the time consumption-oriented list.

3. Remove unnecessary (p, c) pairs from the time consumption-oriented list.

Algorithm 9 Energy Cost Detection (ECD) algorithm

Require: The R-Table $RTab$
Ensure: The S-Table $STab$

```
input the R-Table
FOR ∀ data
    FOR ∀ channel
        FOR ∀ mode
            add (p, c) into list_t
/*Here p refer probability, c refer cost, list_t refer a list that include every (p, c) pair if time consumption is t */
        END FOR
    END FOR
    FOR ∀ list
        remove unnecessary pairs
    END FOR
END FOR
RETURN S-Table
```

7.5.3 Remove Obsolete Pairs Algorithm

We use a "remove unnecessary pairs" operate in algorithm 9 and algorithm 8. Now we present *Remove Obsolete Pairs* (ROP) algorithm. The input of algorithm ROP is a list of (p, c) pairs. The output of this algorithm is the list without unnecessary pairs.

Algorithm 10 Remove Obsolete Pairs (ROP) algorithm

Require: A list of (p, c) pairs
Ensure: A list without unnecessary pairs

```
Sort list by probability ascending
FOR ∀ pair i in list
    IF pair_i.cost > pair_(i−1).cost
        remove pair_i
    ELSE
        IF pair_i.probability > pair_(i−1).probability
            remove pair_(i−1)
    END IF
END FOR
RETURN list
```

The algorithm 9 shows ECD algorithm and the main phases of our algorithm include:

1. Sort pairs in the list by probability ascending.

2. Compare two neighboring pairs and remove one pair if it is unnecessary.

7.6 Experiments and the Results

7.6.1 Experiment Configuration

The *Intelligent Multi-Channel Communication* (IMCC) model and the *Optimal Energy Cost* (OEC) algorithm were designed to minimize the total energy cost when ensuring the performance meets efficiency demands. The purposes of implementing experiment evaluations were threefold: (1) evaluating whether our approach could always create optimal solutions; (2) evaluating the performances of our approach in distinct implementation scenarios; (3) evaluating the advantage of the optimal solutions comparing with other non-optimal solutions. In addition, in our evaluations, we implemented our algorithm on a host that ran Ubuntu Linux 16.04 LTS OS. The main hardware configuration of this host was: Intel Core i54210U CPU, 8.0G RAM.

We configured three experiment settings to evaluate our algorithms' performance under different settings. There are two main configuration variables in our experiment settings: the number of data and the number of channels. These two variables were crucial impact elements for implementing our approach, since the number of the data packages and the number of available channels could remarkably influence the complexity of the computation. The implementation could become more complicated when the number of the data packages was growing as well as that of the number of channels; thus, the execution time for creating optimal solution could be longer along with the increasing size of the input. Additionally, the variety of the input could influence the differences between optimal and non-optimal solutions. We assumed that a bigger sized input could result in a greater advantage of the optimal solution. The experimental settings were:

1. Setting 1: We configured 4 data and 2 channels.

2. Setting 2: We configured 6 data and 3 channels.

3. Setting 3: We configured 10 data and 4 channels.

Moreover, we accomplished a series of comparisons between our approach and a few active methods, including greedy and *Brute Force* (BF) algorithms. The implementation of the greedy algorithm was always giving the energy cost the priority and selecting the smallest value from a group of energy costs. BF algorithm found the optimal solutions by traversing all possible pairs. We used BF algorithm to assess whether our algorithm could create optimal solutions at all times.

The next section illustrated experiment results collected from the evaluation.

7.6.2 Experiment Results

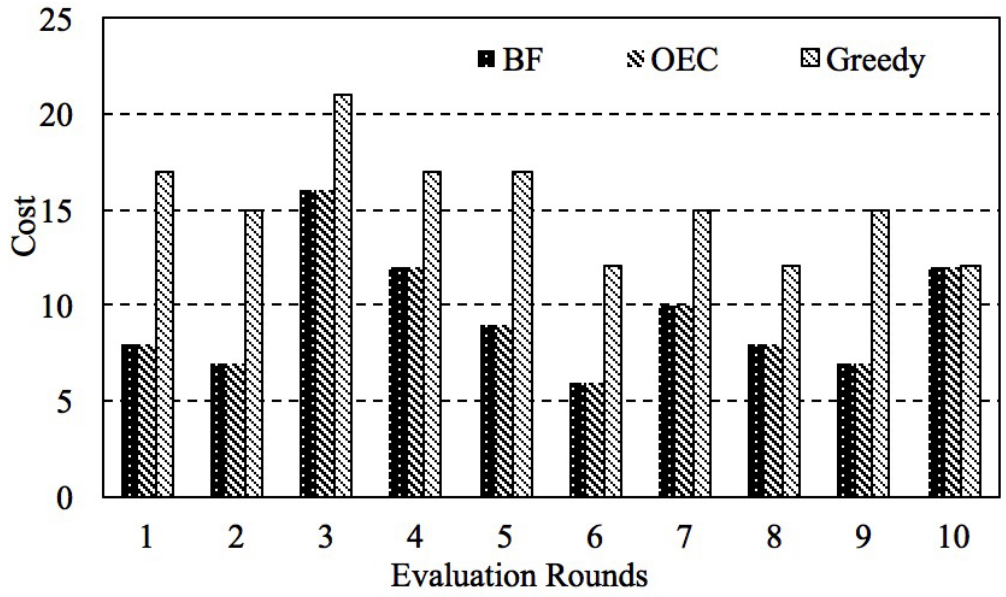


Figure 7.2: Comparisons of energy costs under Setting 1

Fig. 7.2 showed the comparison among Greedy algorithm, OEC algorithm, and Brute-force algorithm under setting 1. As shown in this figure, we could see that the result of our algorithm was better than Greedy algorithm and our algorithm could always obtain the same results as that of Brute-force algorithm. Comparing with greedy algorithm, our approach could reduce 37.9% energy cost.

Next, Fig. 7.3 showed the comparison among Greedy algorithm, OEC algorithm, and Brute-force algorithm under setting 2. We observed that our algorithm was superior to Greedy algorithm and could obtain the same outcomes as the Brute-force algorithm. According to our statistics,

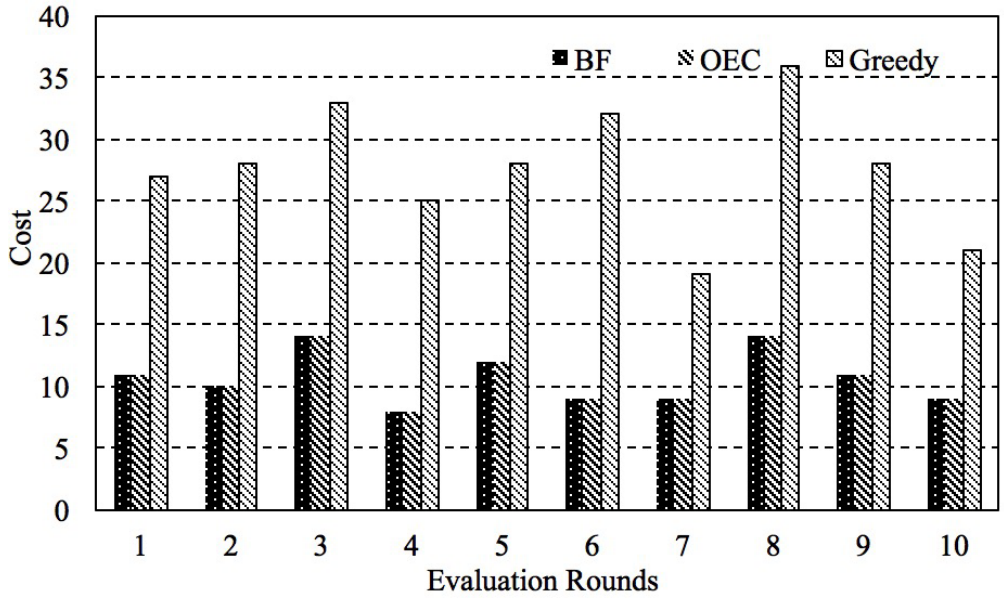


Figure 7.3: Comparisons of energy costs under Setting 2

our approach could lower down 61.3% energy cost comparing with greedy algorithm under this setting.

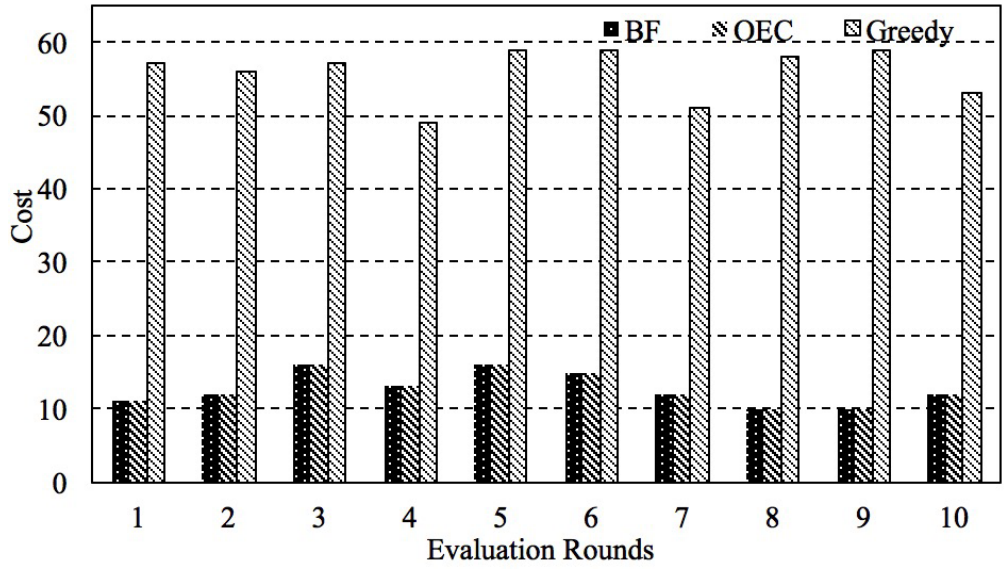


Figure 7.4: Comparisons of energy costs under Setting 3

In addition, Fig. 7.4 showed the comparisons among Greedy algorithm, OEC algorithm, and Brute-force algorithm under setting 3. Based on the collected data, we saw that our algorithm had a better performance than the Greedy algorithm under Setting 3. Our results depicted that

the energy cost was reduced 77.2% on average, comparing with greedy algorithm. Our algorithm could obtain the optimal solutions since Brute-force algorithm traversed all situations, which also proved that our approach could produce optimal solutions. The results above also proved our assumption, which was that a bigger sized input could lead to a larger advantage by implementing our approach.

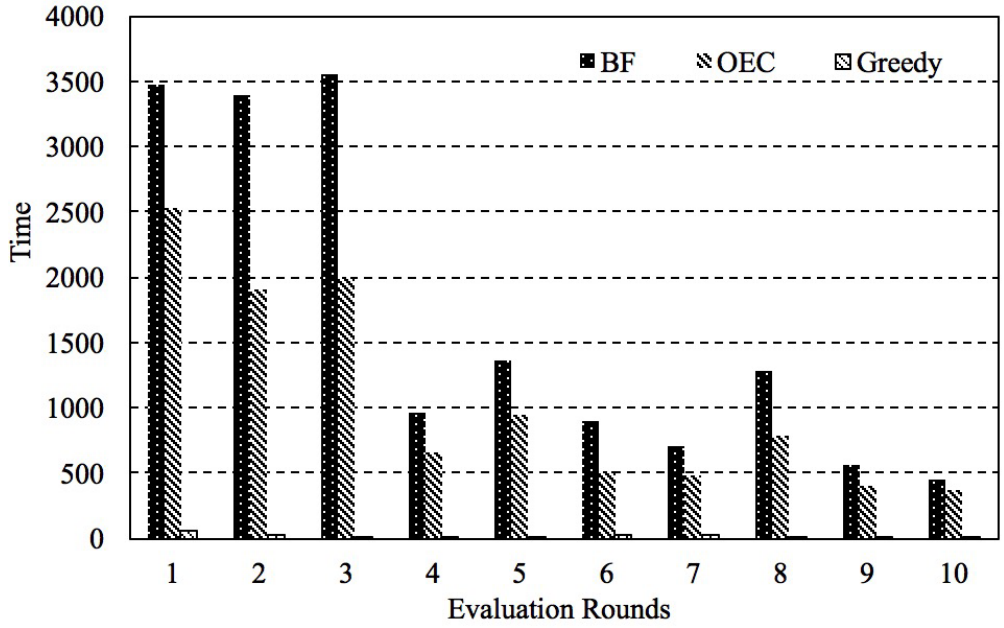


Figure 7.5: Comparisons of execution time under Setting 1

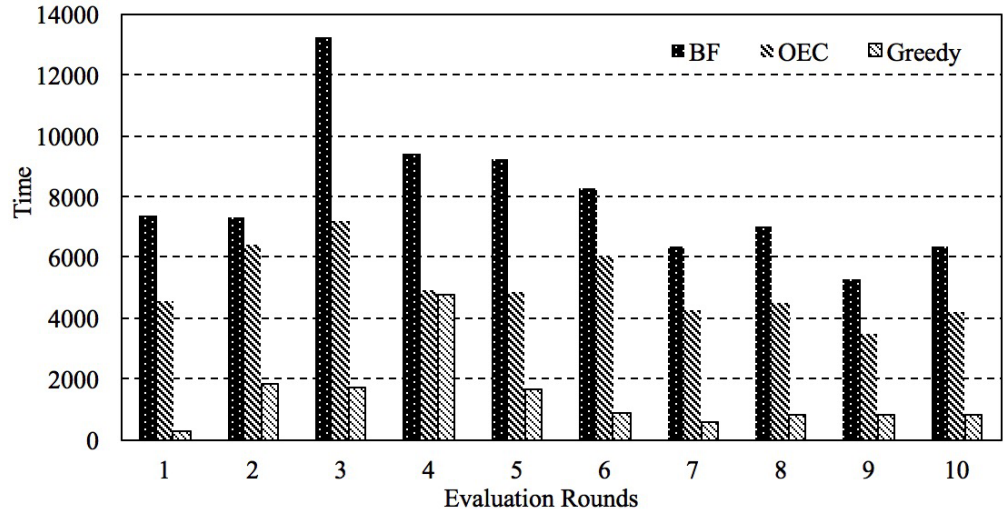


Figure 7.6: Comparisons of execution time under Setting 2

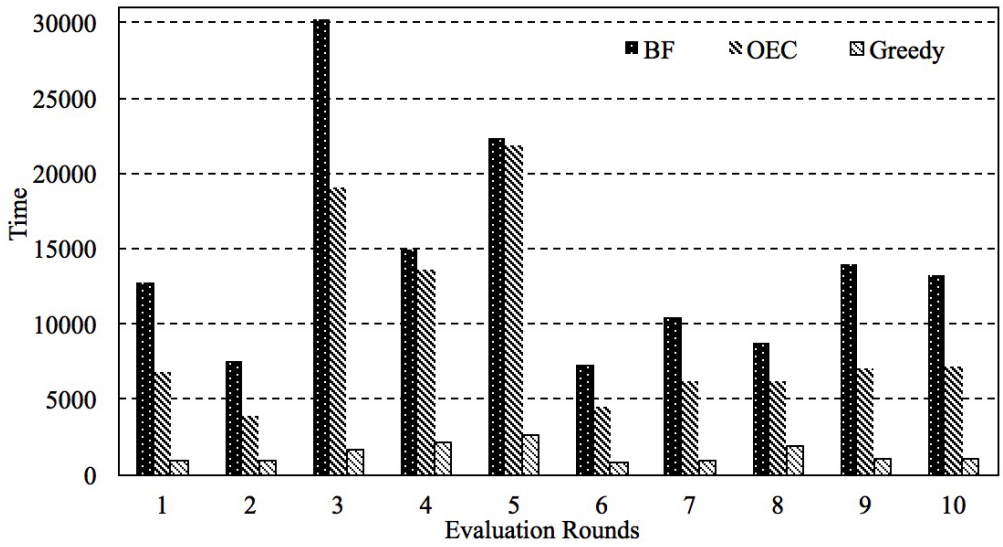


Figure 7.7: Comparisons of execution time under Setting 3

Furthermore, Fig. 7.5, Fig. 7.6, and Fig. 7.7 showed the comparison of execution time among Greedy algorithm, OEC algorithm, and Brute-force algorithm under Setting 1, Setting 2, and Setting 3, respectively. Throughout these three figures, our approach could outcome optimal solutions and reduce the execution time comparing with the Brute-force algorithm. These figures showed that our approach could save strategy generation time at 36.3%, 36.8%, and 31.8% under Setting 1, 2, and 3, respectively. The advantage of using our approach became greater when the number of the channels grew.

In summary, the experimental results have demonstrated that our approach could output optimal solutions as well as reduce the energy consumptions during the data transmissions. We compared our approach with the Brute-force algorithm and found that our approach required a remarkably shorter execution time than Brute-force.

7.7 Conclusions

This chapter focused on the issue of the energy-saving for mobile devices in the context of MCC and designed an intelligent energy-aware communication strategy. The proposed solution considered the energy-saving the priority and the outcome was a deterministic plan for selecting protocols

by which a minimum energy cost was required under a timing constraint and an expected delivery success rate. We exmined our approach though a series of experimental evaluations and demonstrated the advantage of our approach.

Chapter 8

Efficient Mobile Tele-health Systems Using Dynamic Programming

The first focus of this work is solving the problem of data allocations in heterogeneous cloud memories, which is one of the critical aspects of restricting cloud-based infrastructure's deployment. The other focus of this work is developing an efficient method of using pre-cache techniques and dynamic programming to reduce network traffics. That is how to reduce network traffic when the senders are the initiators of mobile network communications. We use mobile tele-health systems as an example to discuss this problem. In this case, the sensors of mobile tele-health systems transfer patients' *Personal Health Records* (PHRs) to the cloud server actively.

8.1 Problem Description

First, deploying cloud-based memories is facing a limitation that is caused by the heterogeneous cloud resources. The Data allocations to heterogeneous cloud memories is a challenging issue due to the high time complexity [45, 130]. To generate an optimal data allocation plan, a number of parameters and variables need to be considered, such as the number of data, the number of cloud

memories, the number of reads and writes for each data, the cost of read and write for each data on every cloud memory, and the cost of movement between cloud memories. The proposed problem is to minimize the total cost by generating an algorithm by which the data allocations are operated.

Moreover, overcoming the limited bandwidth, as well as gaining continuous high performances is an important problem for mobile Tele-health systems. Current mobile Tele-health systems are facing serious problems caused by the networking traffics. The performances of the real-time responses cannot satisfy the requirements of the medical diagnosis when the networking capability is functioning lowly. This work concentrates on the problem of the real-time performances restricted by the networking conditions. Based on the implementations of mobile Tele-health systems, we aim to explore the improvement of the function presentation efficiency. Current Tele-health systems provide a foundation for system adoption decisions and serve as a starting point for the requirement analysis for more complicated Tele-health systems, such as *Microsoft HealthVault* and *Google Health*. Therefore, the critical challenge is overcoming the limited bandwidth as well as gaining continuous high performances.

8.2 Related Work

As an implementation of embedded systems, high performance of the tele-health systems has been addressed by researchers in recent years [81, 131, 132]. The recent achievements related to the improvements of the tele-health systems have covered multiple aspects [82, 133, 134]. The first hemisphere is to achieve high reliable real-time performance [135, 136, 137]. One of the proved solutions was adapting virtual machines with hidden Markov predicting approaches in tele-health systems [138]. Another approach was attempted by applying a hierarchical reliability-driven scheduling algorithm in grid systems [139, 140].

Meanwhile, the performance improvement can be also realized by optimizing task scheduling [141]. The tasks consisting of multiple phases can be improved by applying phase-change memory optimizations with genetic algorithms [24]. Implementing loop fusion and multi-functional-unit scheduling for multi-dimensional DSP was examined as an efficient approach for minimizing en-

ergy consumptions [83]. Another concern is that the optimizations need to be achieved within a hard of software timing constraints in heterogeneous embedded systems [142, 84]. Some approaches have been developed with using dynamic programming in heterogeneous embedded systems [84].

Next, saving energy consumptions is always a big concern in increasing the adaptions of the embedded systems [85, 143, 144]. Previous research has done a few attempts in various dimensions in energy-saving fields [145, 146]. For instance, an energy minimization solution was developed by using three-phase time-aware approaches with using DVFS and unrolling for chip multiprocessors [86]. Next, adopting loop parallelism maximization for multi-core DSP architecture was proved to be an energy-aware approach [87]. In order to deal with multi-tasks, an energy-aware algorithm was developed to guarantee the probability satisfying timing constraints [88].

Moreover, considering the requirements of the security, embedded systems have been explored by the prior research from different perspectives [147]. The balance of security strength and energy is always a challenge in performance improvements for embedded systems [148]. Security-aware optimization was one of the directions of improving the performances of embedded systems [149]. One solution was tested with using an energy-aware security algorithm in power grid wide ware monitoring systems [150].

Furthermore, prior research explorations also considered using advanced algorithms to improve the performances of the existing infrastructure. For instance, a solution with using scratch-pad memory was developed to optimize data allocations on embedded multi-core systems [151, 91]. Further explorations have achieved low-power as well as low-latency with using hybrid scratch-pad memory [6]. Another example was minimizing transferred data for code update on wireless sensor networks for the purpose of performance improvements [152].

Finally, with the development of Web-based technologies, cloud computing has become an important approach to increase the performances or functionalities of tele-health systems [153, 154, 53]. Various dimensions within cloud computing have been attempted by prior research [46, 155, 156]. One of the dimensions was using online optimizations for scheduling preemptable tasks on *Infrastructure-as-a-Service* (IaaS) cloud systems.

In summary, very few prior research addressed the issue of network traffics in implementing tele-health embedded systems. This chapter focuses on optimizing the performances of the tele-health systems from the perspective of reducing network traffics.

8.3 Models and Concepts of RPOM

For this problem, in order to prevent the systems from the restrictions of networks, we propose a novel model named *Reusable Pre-cache Optimal Model* (RPOM). In this model, the mobile tele-health systems have two phases work.

The first work is that the *Body Senor Network* (BSN)collects patients' physical information and transfer the information to the cloud side periodically. This process is a main component consuming networking traffics. We mainly focuses on the method of reducing the consumptions of the network traffics in this phase.

The other work is that the cloud servers use the collected PHRs to judge wether the patients' health information are normal. If the PHRs are abnormal, the cloud servers will send a warning message to the medical center and the patients' mobile tele-health devices.

Telehealth System Architecture

The basic system architecture of the telehealth systems is shown in Fig. 8.1. In a typical Telehealth system, there are three basic components that include sensors, communication units, and cloud servers. A *Sensor* is a device that is used to capture health-related data. *Communication Units* are used to transfer data between sensors and cloud servers after the data are captured by sensors. *Cloud Servers* are defined as an end that can store and analyze the data as well as generate response results.

However, the accuracy of the sensors is too high for the real requirements in practice, which may cause additional network burdens. In some situations, the data captured by the sensor have few diversifications due to the high level sensibilities. The similar data records can hardly influence the outcomes since the low level of the diversifications. This means that transferring all data from sensors to cloud sever may result in the overloaded network traffics. Addressing these problems,

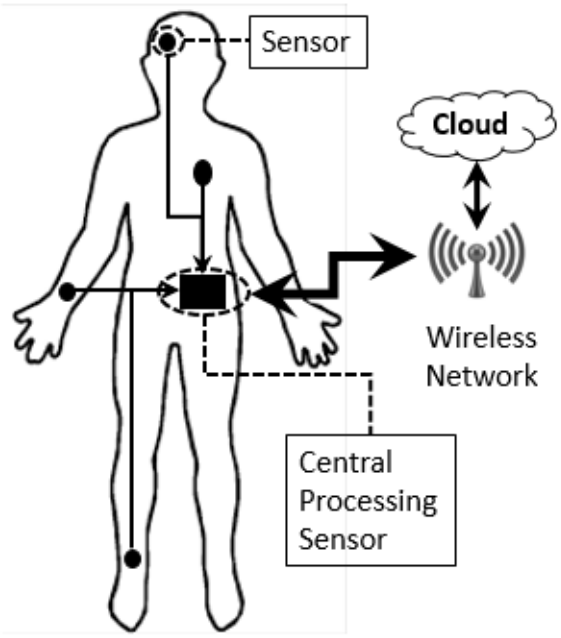


Figure 8.1: Architecture of Tele-health system

we designed a new differential algorithm to save the unnecessary network traffics.

Proposed System Model

Fig. 8.2 describes the proposed system model, RPOM. There are two main phase work in this model. One is from the sensor capture the raw data to the cloud server generate the generated data. We use pre-cache technology and dynamic programming to reduce network resource in this work. The other is the judgement server judge the patients' health information according to these generated data. We allocate the data to appropriate memory to get the minimal cost by using

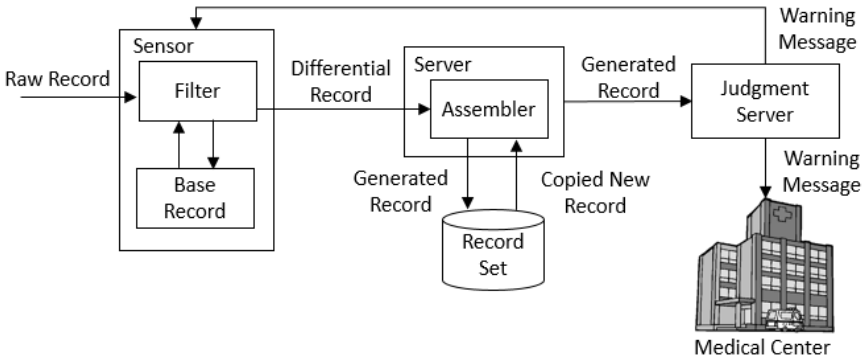


Figure 8.2: System model (RPOM) for the proposed algorithm

dynamic programming.

8.3.1 Reduce Network Traffic by Data Processing

In the first work, we defined a data set that sensors captured at one time a *Record*. Data in a Record are defined as a *Field*. The original Record sensors captured are called *Raw Record*. In our schema, we set up a *Filter* inside the sensors to filter the records that are sent from sensors to server. Sensors store the data set that has the same structure as a record and the pre-stored data are *Base Record*. For every field in a record, the filter generates an *Acceptable Scale* based on the corresponding field of the base record. The field will be dropped when the field's value is in the acceptable scale. If a captured field is not in the acceptable scale, the sensor needs to update the corresponding field in base record. The filter uses an acceptable scale to determine which field in a raw record need to be sent.

Moreover, the raw record will not be send to server directly. After a sensor captures a raw record, the filter chooses the fields that need to be sent according to the acceptable scale and combines the fields with a *Differential Record*. The sensor transfers the differential record to the server. All captured records by are stored in a *Record Set* at the server end. There is an assembler in a server to generate a full *Generated Record* in our algorithm. After the corresponding server receives a differential record from a sensor, the assembler creates the generated record base on the differential record. Then, the server puts the generated records into the *Record Set* as the official data that are ready for usage.

We can see that the more acceptable scale we set, the less accuracy data we have base on the above description. This is a tradeoff. The signal of wireless network is not good enough in some environment, such as in the mountain. In this situation, we can dynamically set the acceptable scale according to the wireless network signal level to get the maximum data accuracy when guarantee the acceptable less network traffic by using dynamic programming.

8.3.2 Reduce Judgement Cost by Data Allocation

On the cloud side, the generated data will be used to judge wether the patients' health information are abnormal when the generated data are created. A warning message will be sent to the medical center and the patients' main sensor if the abnormal health information are found. Usually it needs the judgment operation complete in a specific time. We can allocate the patients' health data to the appropriate memory to reduce the processing time consumption by using n-dimension dynamic programming we mentioned prior when the servers that do the judgement operation employed SPMs.

8.4 Algorithms

We design three algorithms to achieve our proposed model, which include *Data Filter Algorithm* (DFA), *Data Assembler on Sever Algorithm* (DASA), and *Optimal Multiple Dimensional Data Allocation* (OMDDA) Algorithm. Algorithm DFA and DASA are used to to reduce network traffic costs with using pre-cache technique and differential technique. The algorithm OMDDA is used on cloud side, which aim to find the data allocation plan to minimize the judgement operation on the servers that employed SPMs. This algorithm is mentioned in Chapter 2.

In this section, we represent our proposed algorithms, DFA and DASA, that are designed to save total network traffic costs with using a differential technique. The algorithms support activities on both server-end and sensors side. Main notations used in this section are given in Table 8.1.

The first proposed algorithm is DFA shown in Algorithm 11. The implementation of Algorithm 11 runs on sensors side. As described in Section 8.3, a sensor maintains a base record that is used by the filter in the local memory. The sensor sends the record to the filter after it captures a raw record for each time. The filter generates an acceptable scale for all fields using the base records and pre-configure threshold values for every filed.

In addition, the filter judges every field of the raw record and determine whether it is in the corresponding acceptable scale. If the field of the raw record is not in the acceptable scale, the

Table 8.1: Main notations used in this paper

Notations	Definitions
RS	Raw record captured by sensors
RS'	The updated Record Set recorded on Server
RC	Base record stored in local cache by sensors
RT	Part record that transfer from sensors to clouds
f	Fields in record
fc	A field in RC
fs	A field in RS
R.get(x)	Get a field value that named x from record *R*
R.put(x, y)	Put a field with name: x and value: y into record *R*

filter will add it into the *Part Record* and update the corresponding fields of base records with using the field. Finally, the sensor transfers the generated *Part Record* to the *Server*.

Algorithm 11 Data Filter Algorithm (DFA)

Require: *RS*, *RC*
Ensure: *RT*
1: FOR Each field f in *RS*
2: $fs \leftarrow RS.get(f)$
3: $fc \leftarrow RC.get(f)$
4: Generate acceptable scale ($aScale$) using fc
5: IF $fsnot \in aScale$
6: $fc \leftarrow fs$
7: $RT.put(f, fs)$
8: ENDIF
9: ENDFOR
10: RETURN *RT*

On the server side, the assembler is supposed to generate the missing fields and assemble a full record after receives a *Part Record* from the sensor. We create DASA algorithm to achieve data assembling, which is represented in Algorithm 12.

When the server receives a *Part Record* from sensor, the last record is copied from *Record Set* and stored as a *Model Record*. Then, the assembler gains the fields from *Part Record* and uses it to replace the corresponding fields in *Model Record*. Finally, the server stores the full record as an official record in *Record Set*.

Algorithm 12 Data Assembler on Sever Algorithm (DASA)

Require: RT, RS
Ensure: RS'
1: Copy the last record of RS as Model Record RN
2: FOR EACH field f IN RN $fs \leftarrow RT.get(f)$
3: IF fs not $null$
4: $RN.put(f, fs)$
5: ENDIF
6: END FOR
7: add RN into RS
8: return RS

8.5 Experiments

In our experiment, we use Android Emulator to generate a simulated scenario with deploying a sensor and applying our proposed algorithms.

8.5.1 Experimental Configuration

We implemented the proposed algorithm within a simulated emulation framework that were built to test the effectiveness. The developed simulator used armeabi-v7a CUP, 343MB RAM, 512M RAM and ran Android OS v5.0.1 (Lollipop). Th emulator ran on a host with Intel Core i5-4210U CPU, 8.0G RAM, Intel HD Graphics Family GPU, and ran Windows 8.1 64bit OS. Moreover, we used a computer that has i7-4810MQ 2.80 GHz CPU and 16 GB memory as a server. An app using our algorithms was developed and ran under the configuration as described above. We used a set of heart beating rates (beats/min) data of two patients to simulate the telehealth data by which the sensors captured. The app we developed read data at every second to simulate the actions of the sensor capturing.

The main goal of our algorithm is reducing the network traffics. Our experiment compared the network traffics with using different methods, including the traditional and our algorithm methods.

Three experimental settings were used to examine our proposed schema in the experiment.

1. Setting 1. We configured the *Acceptance Parameter* as a 0. Under this configuration, the data would not be transferred to the server until it was as same as the prior transferred data.

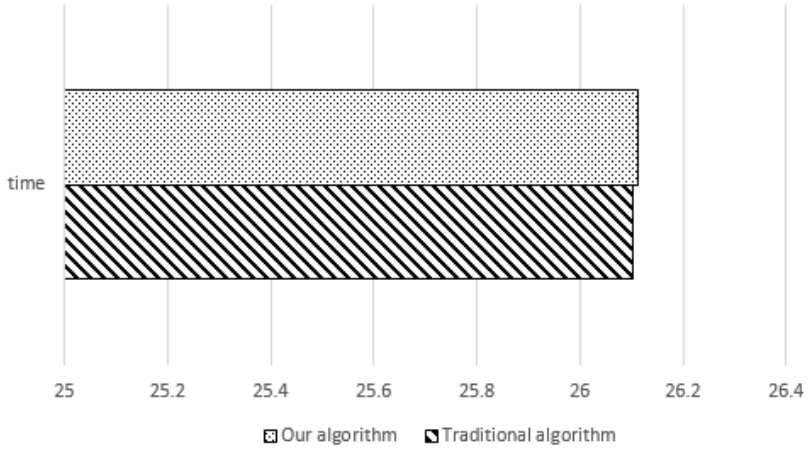


Figure 8.3: Comparisons of time consumption between traditional and proposed algorithms

2. Setting 2. We set the *Acceptance Parameter* as a 1. This configuration makes the *Acceptable Scale* scoped between (fc-1) and (fc+1).

3. Setting 3. The last experimental setting is using 5 as the *Acceptance Parameter*. This setting defined *Acceptable Scale* in the scope between (fc-5) and (fc+5).

8.5.2 Experimental Results

Fig. 8.3 showed that the comparisons of time consumptions between traditional algorithm and our proposed algorithm. According to the figure, the extra time consumptions of our algorithm was acceptable, which was similar to the performance of the traditional algorithm. This evaluation proved that our proposed schema could be applied in tele-health systems from a perspective of timing constraints.

Next, we ran our simulator under three different *Acceptance Parameters* settings to examine the performances of the network traffic saving. Fig. 8.4 represents that the comparison results, which exhibited that our proposed schema could efficiently reduce network traffics. However, the accuracy might be also cut down when reducing the network traffics. Our experiment verified the approach of determining whether the proposed schema is usable under different *Acceptance Parameters*.

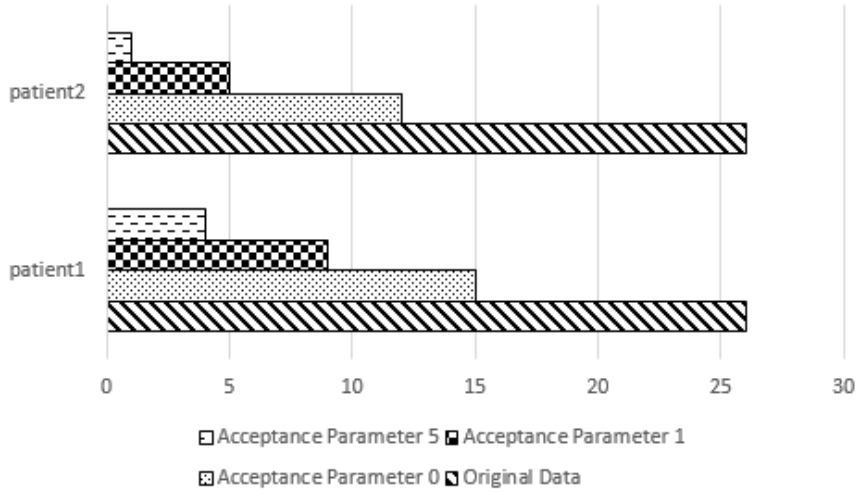


Figure 8.4: Comparisons of network traffics under different acceptance parameters

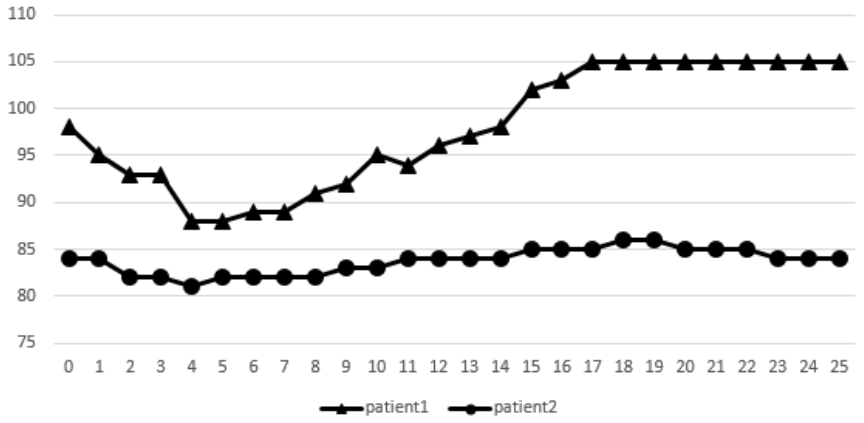


Figure 8.5: Original data captured by sensor

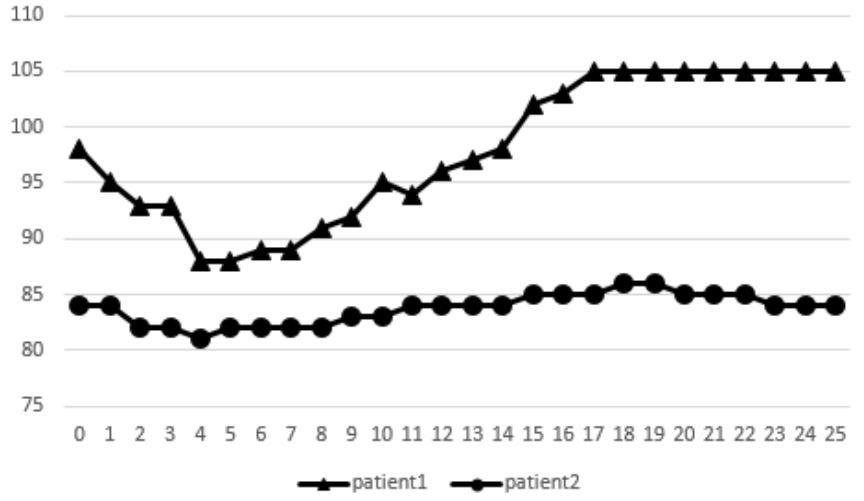


Figure 8.6: Data generated with Acceptance Parameter 0

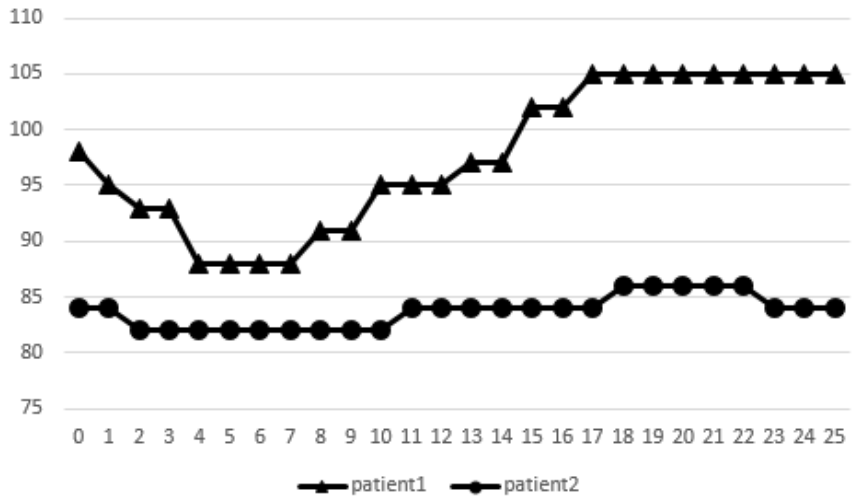


Figure 8.7: Data generated with Acceptance Parameter 1

Fig. 8.5 represents the original data captured by the sensor side. Fig. 8.6 illustrated the data generated by using our algorithm with *Acceptance Parameter* 0 at the the server end. From Fig. 8.5 to Fig. 8.6, the results showed that using different algorithms generated the similar data when implementing setting 1 with 0 value of *Acceptance Parameter*. The crucial benefit of applying setting 1 is reducing network traffics without lowering the data accuracy.

Fig. 8.7 shows that the data generated using our algorithm with $AcceptanceParameter1$ at the server end.

Fig. 8.7 represents the data generations under setting 2, which shows that the data generated using our algorithm is acceptable and adaptive when *Acceptance Parameter* is 1. This evaluation implied that the network traffics levels might have a negative relationship with the value of the*Acceptance Parameter* but have a potential positive relationship with the data accuracy level. Figure 8.8 represents the experimental results when implementing setting 3, which used 5 as the *Acceptance Parameter*.

As shown in Fig. 8.8, the implementation was not acceptable when the *Acceptance Parameter* was 5. The figures showed that the *Acceptance Parameter* had a positive relationship with the network traffic saving when the proposed algorithm is applied. However, similar to the evaluations in setting 1 and 2, the data accuracy is also low, which depicts that the *Acceptance Parameters* with large values might not be acceptable in practice.

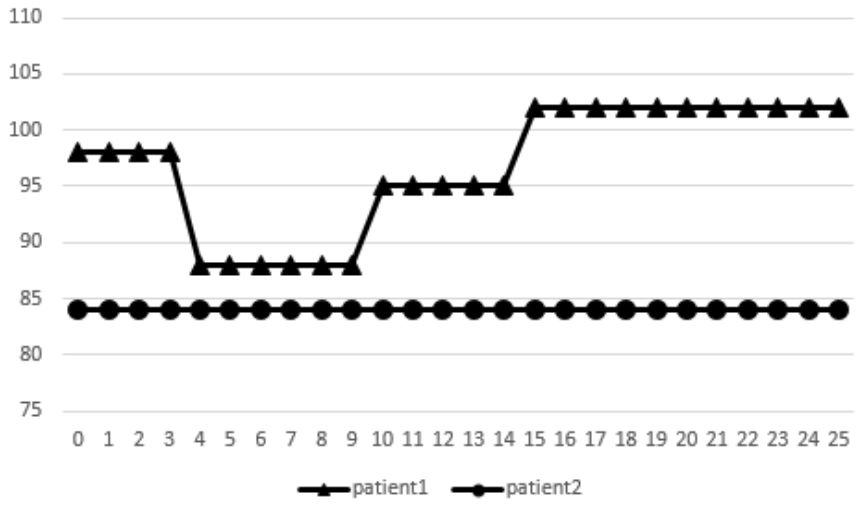


Figure 8.8: Data generated with Acceptance Parameter 5

Considering the implementation in practice, it should be careful when determining the value of the *Acceptance Parameters*. In practice, the parameter can be decided by the requirements of the medical conditions, such as recommendations from physicians.

In summary, our experiments have shown that using the proposed algorithms could efficiently reduce network traffics. The parameter is a crucial factor that can determine the network traffics saving performances and data accuracy levels.

8.6 Conclusions

This chapter aim to solve two problems of MCC. One is solving the problem of data allocations in heterogeneous cloud memories, which is one of the critical aspects of restricting cloud-based infrastructure's deployment. The other is improving wireless network performance of MCC by reducing network traffics when data senders are the initiator of network communication. We use mobile tele-health as a use case to describe our approach. In this chapter, we proposed a novel differential schema for achieving high performance big data in tele-health systems with using pre-cache technologies. The crucial algorithms in the proposed paradigm include DFA and DASA. The efficiency of our solution was proved by the experimental evaluation, which showed that our

approach could reduce network traffics in this case.

Chapter 9

Summary

In this research, our goal is to gain efficient mobile systems by scheduling resource through data allocation and processing for Mobile Cloud Computing. Our research is supported by four proposed sub research questions, The questions include:

1. How can we improve data processing performance through data allocation on mobile cloud servers by using dynamic programming (i.e., data allocation in mobile cloud)?

2. How can we improve wireless network performance of MCC by reducing network traffics?

3. How can we save wireless power consumption of MCC by finding the optimal transfer plan(i.e., data processing in mobile cloud)?

4. How can we improve system performance of MCC by data allocation and processing (i.e., data allocation and processing in mobile cloud)?

We proposed a serials models and algorithms to solve these problems, including cloud server performance and wireless network performance.

1. We proposed a model and algorithm to solve the problem of data allocation in mobile cloud. We had implemented experimental evaluations to examine our proposed model's performance and the collected results showed that our model had a great advantage in saving

cost due to the optimal data allocations to the mobile cloud servers that employ SPMs. we proposed an approach to find out the optimal data allocation plan in heterogeneous cloud memories. The experimental results proved that our approach is an effective mechanism.

2. We proposed models and algorithms to solve the problem of data processing in mobile cloud to reduce the wireless network traffics. The experimental results show that our proposed mechanism can enhance the performances in network traffic control, power management, and executive latency.

3. We discuss how to save wireless power consumption of MCC by finding the optimal transfer plan. We presents a novel optimal solution to deploying multi-channel connections with minimum energy costs.

4. We discuss how to schedule data resource and network traffic resources for efficient mobile tele-health Systems. We proposed a novel differential schema for achieving high performance big data in tele-health systems with using pre-cache technologies. The efficiency of our solution was proved by the experimental evaluation, which showed that our approach could reduce network traffics.

Bibliography

[1] K. Gai, Z. Du, M. Qiu, and H. Zhao. Efficiency-aware workload optimizations of heterogenous cloud computing for capacity planning in financial industry. In *The 2nd IEEE International Conference on Cyber Security and Cloud Computing*, pages 1–6, New York, USA, 2015. IEEE.

[2] N. Min-Allah, H. Hussain, S. Khan, and A. Zomaya. Power efficient rate monotonic scheduling for multi-core systems. *Journal of Parallel and Distributed Computing*, 72(1):48–57, 2012.

[3] K. Gai and M. Qiu. Reinforcement learning-based content-centric services in mobile sensing. *IEEE Network*, PP(99):1, 2018.

[4] G. Rodríguez, J. Touriño, and M. Kandemir. Volatile STT-RAM scratchpad design and data allocation for low energy. *ACM Transactions on Architecture and Code Optimization*, 11(4):38, 2014.

[5] K. Gai, M. Qiu, Z. Xiong, and M. Liu. Optimal resource allocation using reinforcement learning for iot content-centric services. *Applied Soft Computing*, PP(99):1, 2018.

[6] M. Qiu, Z. Chen, and M. Liu. Low-power low-latency data allocation for hybrid scratch-pad memory. *IEEE Embedded Systems Letters*, 6:69–72, 2014.

[7] D. Chang, C. Lin, Y. Chien, C. Lin, A. Su, and C. Young. CASA: Contention-aware scratchpad memory allocation for online hybrid on-chip memory management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(12):1806–1817, 2014.

[8] M. Sabry, D. Atienza, and F. Catthoor. OCEAN: An optimized hw/sw reliability mitigation approach for scratchpad memories in real-time SoCs. *ACM Transactions on Embedded Computing Systems*, 13(4s):138, 2014.

[9] B. Gaster, D. Hower, and L. Howes. HRF-relaxed: Adapting HRF to the complexities of industrial heterogeneous memory models. *ACM Transactions on Architecture and Code Optimization*, 12(1):7, 2015.

[10] H. Waidyasooriya, Y. Ohbayashi, M. Hariyama, and M. Kameyama. Memory allocation exploiting temporal locality for reducing data-transfer bottlenecks in heterogeneous multicore processors. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(10):1453–1466, 2011.

[11] D. Kliazovich, P. Bouvry, and S. Khan. GreenCloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, 62(3):1263–1283, 2012.

[12] N. Fernando, S. Loke, and W. Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84–106, 2013.

[13] K. Gai and S. Li. Towards cloud computing: a literature review on cloud computing and its development trends. In *The 4th IEEE International Conference on Multimedia Information Networking and Security*, pages 142–146, Nanjing, China, 2012. IEEE.

[14] K. Gai, M. Qiu, H. Zhao, L. Tao, and Z. Zong. Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing. *Journal of Network and Computer Applications*, 59:46–54, 2015.

[15] J. Hu, C. Xue, Q. Zhuge, W. Tseng, and E. H. Sha. Data allocation optimization for hybrid scratch pad memory with SRAM and nonvolatile memory. *IEEE Transactions on Very Large Scale Integration Systems*, 21(6):1094–1102, 2013.

[16] Z. Jia, Y. Li, Y. Wang, M. Wang, and Z. Shao. Temperature-aware data allocation for embedded systems with cache and scratchpad memory. *ACM Transactions on Embedded Computing Systems*, 14(2):30, 2015.

[17] H. Wei, Z. Shao, Z. Huang, R. Chen, Y. Guan, J. Tan, and Z. Shao. RT-ROS: A real-time ros architecture on multi-core processors. *Future Generation Computer Systems*, 2015.

[18] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, and Z. Gu. Online optimization for scheduling preemptable tasks on IaaS cloud systems. *Journal of Parallel and Distributed Computing*, 72(5):666–677, 2012.

[19] M. Qiu, L. Chen, Y. Zhu, J. Hu, and X. Qin. Online data allocation for hybrid memories on embedded tele-health systems. In *2014 IEEE International Conference on High Performance Computing and Communications, 2014*

IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst, pages 574–579. IEEE, 2014.

[20] J. Hu, Q. Zhuge, C. Xue, W. Tseng, and E. H. Sha. Management and optimization for nonvolatile memory-based hybrid scratchpad memory on multicore embedded processors. *ACM Transactions on Embedded Computing Systems*, 13(4):79, 2014.

[21] Q. Zhuge, Y. Guo, J. Hu, W. Tseng, C. Xue, and E. H. Sha. Minimizing access cost for multiple types of memory units in embedded systems through data allocation and scheduling. *IEEE Transactions on Signal Processing*, 60(6):3253–3263, 2012.

[22] B. Han, M. Peng, Z. Zhao, and W. Wang. A multidimensional resource-allocation optimization algorithm for the network-coding-based multiple-access relay channels in OFDM systems. *IEEE Transactions on Vehicular Technology*, 62(8):4069–4078, 2013.

[23] M. Qiu, Z. Chen, J. Niu, G. Quan, X. Qin, and L. Yang. Data allocation for hybrid memory with genetic algorithm. *IEEE Transactions on Emerging Topics in Computing*, pp:1–11, 2015.

[24] M. Qiu, M. Zhong, J. Li, K. Gai, and Z. Zong. Phase-change memory optimization for green cloud with genetic algorithm. *IEEE Transactions on Computers*, 64(12):3528 – 3540, 2015.

[25] Z. Wang, Z. Gu, and Z. Shao. Optimizated allocation of data variables to PCM/DRAM-based hybrid main memory for real-time embedded systems. *IEEE Embedded Systems Letters*, 6(3):61–64, 2014.

[26] P. Panda, N. Dutt, and A. Nicolau. On-chip vs. off-chip memory: the data partitioning problem in embedded processor-based systems. *ACM Transactions on Design Automations of Electronic Systems (TODAES)*, 5(3):682–704, 2000.

[27] Y. Li, W. Dai, Z. Ming, and M. Qiu. Privacy protection for preventing data over-collection in smart city. *IEEE Transactions on Computers*, PP:1, 2015.

[28] Y. Zheng, Z. Dong, F. Luo, K. Meng, J. Qiu, and K. Wong. Optimal allocation of energy storage system for risk mitigation of DISCOs with high renewable penetrations. *IEEE Transactions on Power Systems*, 29(1):212–220, 2014.

[29] R. Fan, H. Jiang, Q. Guo, and Z. Zhang. Joint optimal cooperative sensing and resource allocation in multi-channel cognitive radio networks. *IEEE Transactions on Vehicular Technology*, 60(2):722–729, 2011.

[30] F. Hao, G. Min, J. Chen, F. Wang, M. Lin, C. Luo, and L. Yang. An optimized computational model for multi-community-cloud social collaboration. *Services Computing, IEEE Transactions on*, 7(3):346–358, 2014.

[31] H. Zhao, M. Qiu, M. Chen, and K. Gai. Cost-aware optimal data allocations for multiple dimensional heterogeneous memories using dynamic programming in big data. *Journal of Computational Science*, PP(99):1, 2016.

[32] K. Gai, M. Qiu, H. Zhao, and M. Liu. Energy-aware optimal task assignment for mobile heterogeneous embedded systems in cloud computing. In *2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud)*, pages 198–203, Beijing, China, 2016. IEEE.

[33] A. Basu, J. Gandhi, J. Chang, M. Hill, and M. Swift. Efficient virtual memory for big memory servers. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 237–248. ACM, 2013.

[34] Z. Wei, G. Pierre, and C. Chi. Cloudtps: Scalable transactions for web applications in the cloud. *Services Computing, IEEE Transactions on*, 5(4):525–539, 2012.

[35] K. Gai, M. Qiu, and H. Zhao. Cost-aware multimedia data allocation for heterogeneous memory using genetic algorithm in cloud computing. *IEEE Transactions on Cloud Computing*, PP(99):1–11, 2016.

[36] H. Mao, N. Xiao, W. Shi, and Y. Lu. Wukong: A cloud-oriented file service for mobile internet devices. *Journal of Parallel and Distributed Computing*, 72(2):171–184, 2012.

[37] K. Katsalis, V. Sourlas, T. Korakis, and L. Tassiulas. A cloud-based content replication framework over multi-domain environments. In *Communications (ICC), 2014 IEEE International Conference on*, pages 2926–2931. IEEE, 2014.

[38] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14–23, 2009.

[39] M. Chen, Y. Zhang, Y. Li, S. Mao, and V. Leung. EMC: emotion-aware mobile cloud computing in 5G. *IEEE Network*, 29(2):32–38, 2015.

[40] H. Liang, L. Cai, D. Huang, X. Shen, and D. Peng. An smdp-based service model for interdomain resource allocation in mobile cloud networks. *Vehicular Technology, IEEE Transactions on*, 61(5):2222–2232, 2012.

[41] S. Sood and R. Sandhu. Matrix based proactive resource provisioning in mobile cloud environment. *Simulation Modelling Practice and Theory*, 50:83–95, 2015.

[42] M. Qiu, Z. Chen, Z. Ming, X. Qin, and J. Niu. Energy-aware data allocation with hybrid memory for mobile cloud systems. *IEEE Systems Journal*, PP:1–10, 2014.

[43] M. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. Loh. Heterogeneous memory architectures: A hw/sw approach for mixing die-stacked and off-package memories. In *IEEE 21st International Symposium on High Performance Computer Architecture*, pages 126–136, Burlingame, CA, USA, 2015. IEEE.

[44] N. Agarwal, D. Nellans, M. Stephenson, M. O'Connor, and S. Keckler. Page placement strategies for GPUs within heterogeneous memory systems. *ACM SIGPLAN Notices*, 50(4):607–618, 2015.

[45] K. Gai, M. Qiu, and X. Sun. A survey on fintech. *Journal of Network and Computer Applications*, PP(99):1, 2017.

[46] K. Gai, M. Qiu, and H. Zhao. Energy-aware task assignment for mobile cyber-enabled applications in heterogeneous cloud computing. *Journal of Parallel and Distributed Computing*, 111:126–135, 2018.

[47] K. Gai, M. Qiu, H. Zhao, and X. Sun. Resource management in sustainable cyber-physical systems using heterogeneous cloud computing. *IEEE Transactions on Sustainable Computing*, PP(99):1–13, 2017.

[48] K. Gai and M. Qiu. Blend arithmetic operations on tensor-based fully homomorphic encryption over real numbers. *IEEE Transactions on Industrial Informatics*, pp(99):1, 2018.

[49] E. Sha, X. Chen, Q. Zhuge, L. Shi, and W. Jiang. A new design of in-memory file system based on file virtual address framework. *IEEE Transactions on Computers*, 65(10):2959–2972, 2016.

[50] K. Gai, M. Qiu, M. Liu, and Z. Xiong. In-memory big data analytics under space constraints using dynamic programming. *Future Generation Computer Systems*, 83:219–227, 2018.

[51] E. Hahne. Round-robin scheduling for max-min fairness in data networks. *IEEE Journal on Selected Areas in communications*, 9(7):1024–1039, 1991.

[52] B. Cully, J. Wires, D. Meyer, K. Jamieson, K. Fraser, and et al. Strata: High-performance scalable storage on virtualized non-volatile memory. In *Proc. of the 12th USENIX Conf. on File and Storage Technol.*, pages 17–31, San Jose, CA, USA, 2014.

[53] K. Gai, M. Qiu, X. Sun, and H. Zhao. Smart data deduplication for telehealth systems in heterogeneous cloud computing. *Journal of communications and information networks*, 1(4):93–104, 2016.

[54] H. Zhao, M. Qiu, M. Chen, and K. Gai. Cost-aware optimal data allocations for multiple dimensional heterogeneous memories using dynamic programming in big data. *J. of Comp. Sci.*, PP:1, 2016.

[55] J. Araújo, M. Mazo, A. Anta, P. Tabuada, and K. Johansson. System architectures, protocols and algorithms for aperiodic wireless control systems. *IEEE Transactions on Industrial Informatics*, 10(1):175–184, 2014.

[56] L. Zhang, D. Bild, R. Dick, Z. Mao, and P. Dinda. Panappticon: event-based tracing to measure mobile application and platform performance. In *IEEE International Conference on Hardware/Software Codesign and System Synthesis*, pages 1–10, Montreal, QC, Canada, 2013.

[57] P. Rost, C. Bernardos, A. Domenico, M. Girolamo, M. Lalam, A. Maeder, D. Sabella, et al. Cloud technologies for flexible 5G radio access networks. *IEEE Communications Magazine*, 52(5):68–76, 2014.

[58] X. Zhang, W. Cheng, and H. Zhang. Heterogeneous statistical QoS provisioning over 5G mobile wireless networks. *IEEE Network*, 28(6):46–53, 2014.

[59] M. Qiu, Z. Chen, L. Yang, X. Qin, and B. Wang. Towards power-efficient smartphones by energy-aware dynamic task scheduling. In *IEEE 14th International Conference on High Performance Computing and Communication & IEEE 9th International Conference on Embedded Software and Systems*, pages 1466–1472, Liverpool, UK, 2012.

[60] J. Li, M. Qiu, J. Niu, Y. Zhu, M. Liu, and T. Chen. Three-phase algorithms for task scheduling in distributed mobile DSP system with lifetime constraints. *Journal of Signal Processing Systems*, 67(3):239–253, 2012.

[61] C. Wang, Y. Zhu, S. Zhou, X. Gu, J. Jiang, and M. Qiu. A scalable embedded system for massive medical signal processing. In *IEEE 12th International New Circuits and Systems Conference*, pages 432–435, Trois-Rivieres, QC, Canada, 2014.

[62] X. Zhang, Z. Yang, L. Shangguan, Y. Liu, and L. Chen. Boosting mobile apps under imbalanced sensing data. *IEEE Transactions on Mobile Computing*, PP(99):1–12, 2014.

[63] D. Wubben, P. Rost, J. Bartelt, M. Lalam, V. Savin, M. Gorgoglione, A. Dekorsy, and G. Fettweis. Benefits and impact of cloud computing on 5G signal processing: Flexible centralization through cloud-RAN. *IEEE Signal Processing Magazine*, 31(6):35–44, 2014.

[64] M. Chen and A. Ksentini. Cache in the air: exploiting content caching and delivery techniques for 5G systems. *IEEE Communications Magazine*, 52(2):131–139, 2014.

[65] J. Li, M. Qiu, J. Niu, Y. Chen, and Z. Ming. Adaptive resource allocation for preemptable jobs in cloud systems. In *10th International Conference on Intelligent Systems Design and Applications*, pages 31–36, Cairo, Egypt, 2010.

[66] W. Shi, M. Wu, S. Wang, M. Guo, B. Peng, B. Ouyang, and T. Chen. Local resource accessing mechanism on multiple mobile platform. In *IEEE 14th International Conference on High Performance Computing and Communication & IEEE 9th International Conference on Embedded Software and Systems*, pages 1716–1721, Liverpool, UK, 2012.

[67] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya. Heterogeneity in mobile cloud computing: taxonomy and open challenges. *IEEE Communications Surveys & Tutorials*, 16(1):369–392, 2014.

[68] J. Li, C. Liu, and J. Yu. Context-based diversification for keyword queries over XML data. *IEEE Transactions on Knowledge and Data Engineering*, 27(99):1–14, 2014.

[69] A. Charland and B. Leroux. Mobile application development: web vs. native. *Communications of the ACM*, 54(5):49–53, 2011.

[70] K. Gai, M. Qiu, S. Jayaraman, and L. Tao. Ontology-based knowledge representation for secure self-diagnosis in patient-centered telehealth with cloud systems. In *The IEEE 2nd International Conference on Cyber Security and Cloud Computing*, pages 98–103, New York, NY, USA, 2015. IEEE.

[71] K. Gai, M. Qiu, L. Chen, and M. Liu. Electronic health record error prevention approach using ontology in big data. In *17th IEEE International Conference on High Performance Computing and Communications*, pages 752–757, New York, USA, 2015.

[72] S. Elnagdy, M. Qiu, and K. Gai. Cyber incident classifications using ontology-based knowledge representation for cybersecurity insurance in financial industry. In *The 2nd IEEE International Conference of Scalable and Smart Cloud*, pages 301–306. IEEE, 2016.

[73] G. Alipui, L. Tao, K. Gai, and N. Jiang. Reducing complexity of diagnostic message pattern specification and recognition on in-bound data using semantic techniques. In *The 2nd IEEE International Conference of Scalable and Smart Cloud*, pages 267–272. IEEE, 2016.

[74] S. Jayaraman, L. Tao, K. Gai, and N. Jiang. Drug side effects data representation and full spectrum inferencing using knowledge graphs in intelligent telehealth. In *The 2nd IEEE International Conference of Scalable and Smart Cloud*, pages 289–294. IEEE, 2016.

[75] P. Samadi, H. Mohsenian-Rad, V. Wong, and R. Schober. Real-time pricing for demand response based on stochastic approximation. *IEEE Transactions on Smart Grid*, 5(2):789–798, 2014.

[76] R. DeStefano, L. Tao, and K. Gai. Improving data governance in large organizations through ontology and linked data. In *The 2nd IEEE International Conference of Scalable and Smart Cloud*, pages 279–284. IEEE, 2016.

[77] S. Elnagdy, M. Qiu, and K. Gai. Understanding taxonomy of cyber risks for cybersecurity insurance of financial industry in cloud computing. In *The 2nd IEEE International Conference of Scalable and Smart Cloud*, pages 295–300. IEEE, 2016.

[78] C. Asamoah, L. Tao, K. Gai, and N. Jiang. Powering filtration process of cyber security ecosystem using knowledge graph. In *The 2nd IEEE International Conference of Scalable and Smart Cloud*, pages 240–246. IEEE, 2016.

[79] M. Qiu, K. Gai, H. Zhao, and M. Liu. Privacy-preserving smart data storage for financial industry in cloud computing. *Concurrency and Computation: Practice and Experience*, 30(5):e4278, 2018.

[80] K. Gai, M. Qiu, M. Liu, and H. Zhao. Smart resource allocation using reinforcement learning in content-centric cyber-physical systems. In *International Conference on Smart Computing and Communication*, pages 39–52. Springer, 2017.

[81] F. Dabiri, T. Massey, H. Noshadi, H. Hagopian, C. Lin, R. Tan, and et al. A telehealth architecture for networked embedded systems: a case study in in vivo health monitoring. *IEEE Transactions on Information Technology in Biomedicine*, 13(3):351–359, 2009.

[82] M. Qiu, E. Khisamutdinov, Z. Zhao, C. Pan, J. Choi, N. Leontis, and P. Guo. RNA nanotechnology for computer design and in vivo computation. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 371(2000):20120310, 2013.

[83] M. Qiu, E. Sha, M. Liu, M. Lin, S. Hua, and L. Yang. Energy minimization with loop fusion and multi-functional-unit scheduling for multidimensional DSP. *Journal of Parallel and Distributed Computing*, 68(4):443–455, 2008.

[84] M. Qiu, L. Yang, Z. Shao, and E. Sha. Dynamic and leakage energy minimization with soft real-time loop scheduling and voltage assignment. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(3):501–504, 2010.

[85] F. Hu, Q. Hao, M. Qiu, Y. Wu, J. Frye, D. Pontillo, J. Finamore, A. Bhutani, Z. McGarvey, and D. Phillips. Low-power electroencephalography sensing data RF transmission: hardware architecture and test. In *Proceedings of the 1st ACM international workshop on Medical-grade wireless networks*, pages 57–62. ACM, 2009.

[86] M. Qiu, Z. Ming, J. Li, and et. al. Three-phase time-aware energy minimization with DVFS and unrolling for chip multiprocessors. *J. of Syst. Architecture*, 58(10):439–445, 2012.

[87] M. Qiu, J. Niu, L. Yang, X. Qin, S. Zhang, and B. Wang. Energy-aware loop parallelism maximization for multi-core DSP architectures. In *Proceedings of the 2010 IEEE/ACM International Conference on Green Computing and Communications & International Conference on Cyber, Physical and Social Computing*, pages 205–212. IEEE Computer Society, 2010.

[88] J. Niu, C. Liu, Y. Gao, and M. Qiu. Energy efficient task assignment with guaranteed probability satisfying timing constraints for embedded systems. *IEEE Trans. on Parallel and Distributed Syst.*, 25(8):2043–2052, 2014.

[89] K. Gai, M. Qiu, H. Zhao, and W. Dai. Privacy-preserving adaptive multi-channel communications under timing constraints. In *The IEEE International Conference on Smart Cloud 2016*, page 1, New York, USA, 2016. IEEE.

[90] H. Zhao, M. Chen, M. Qiu, K. Gai, and M. Liu. A novel pre-cache schema for high performance android system. *Future Generation Computer Systems*, 2015.

[91] Y. Guo, Q. Zhuge, J. Hu, J. Yi, M. Qiu, and E. Sha. Data placement and duplication for embedded multicore systems with scratch pad memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(6):809–817, 2013.

[92] M. Qiu, D. Cao, H. Su, and K. Gai. Data transfer minimization for financial derivative pricing using Monte Carlo simulation with GPU in 5G. *Int'l J. of Communication Systems*, 2015.

[93] M. Maheu, M. Pulier, J. McMenamin, and L. Posen. Future of telepsychology, telehealth, and various technologies in psychological research and practice. *Professional Psychology: Research and Practice*, 43(6):613, 2012.

[94] H. Zhao, M. Qiu, K. Gai, and X. He. Optimal solution to intelligent multi-channel wireless communications using dynamic programming. *The Journal of Supercomputing*, PP:1–15, 2018.

[95] K. Gai, K.K.R. Choo, M. Qiu, and L. Zhu. Privacy-preserving content-oriented wireless communication in internet-of-things. *IEEE Internet of Things Journal*, PP(99):1, 2018.

[96] L. Xu, W. He, and S. Li. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, 2014.

[97] L. Fernandes, J. Souza, G. Pessin, P. Shinzato, D. Sales, C. Mendes, M. Prado, R. Klaser, A. Magalhães, and A. Hata. CaRINA intelligent robotic car: architectural design and applications. *Journal of Systems Architecture*, 60(4):372–392, 2014.

[98] K. Gai, M. Qiu, Z. Xiong, and M. Liu. Privacy-preserving multi-channel communication in edge-of-things. *Future Generation Computer Systems*, PP:1, 2018.

[99] L. Chen, Y. Duan, M. Qiu, J. Xiong, and K. Gai. Adaptive resource allocation optimization in heterogeneous mobile cloud systems. In *The 2nd IEEE International Conference on Cyber Security and Cloud Computing*, pages 19–24, New York, USA, 2015. IEEE.

[100] K. Gai, M. Qiu, L. Tao, and Y. Zhu. Intrusion detection techniques for mobile cloud computing in heterogeneous 5G. *Security and Communication Networks*, 9(16):3049–3058, 2016.

[101] K. Gai, M. Qiu, Z. Ming, H. Zhao, and L. Qiu. Spoofing-jamming attack strategy using optimal power distributions in wireless smart grid networks. *IEEE Transactions on Smart Grid*, 8(5):2431 – 2439, 2017.

[102] L. Tao, S. Golikov, K. Gai, and M. Qiu. A reusable software component for integrated syntax and semantic validation for services computing. In *9th International IEEE Symposium on Service-Oriented System Engineering*, pages 127–132, San Francisco Bay, USA, 2015. IEEE.

[103] K. Gai, M. Qiu, H. Zhao, L. Tao, and Z. Zong. Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing. *Journal of Network and Computer Applications*, 59:46–54, 2015.

[104] K. Gai, M. Qiu, B. Thuraisingham, and L. Tao. Proactive attribute-based secure data schema for mobile cloud in financial industry. In *The IEEE International Symposium on Big Data Security on Cloud*, pages 1332–1337, New York, USA, 2015. IEEE.

[105] C. Dobre and F. hafa. Intelligent services for big data science. *Future Generation Computer Systems*, 37:267–281, 2014.

[106] K. Gai and M. Qiu. An optimal fully homomorphic encryption scheme. In *IEEE 3rd International Conference on Big Data Security on Cloud*, pages 101–106, Beijing, China, 2017. IEEE.

[107] K. Gai, M. Qiu, Y. Li, and X. Liu. Advanced fully homomorphic encryption scheme over real numbers. In *The 4th International Conference on Cyber Security and Cloud Computing*, pages 64–69, New York, USA, 2017. IEEE.

[108] H. Lu, J. Li, and M. Guizani. Secure and efficient data transmission for cluster-based wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):750–761, 2014.

[109] M. Qiu, K. Gai, and Z. Xiong. Privacy-preserving wireless communications using bipartite matching in social big data. *Future Generation Computer Systems*, PP:1, 2017.

[110] K. Gai, M. Qiu, M. Chen, and H. Zhao. SA-EAST: security-aware efficient data transmission for ITS in mobile heterogeneous cloud computing. *ACM Transactions on Embedded Computing Systems*, 16(2):60, 2016.

[111] A. Lajunen. Energy consumption and cost-benefit analysis of hybrid and electric city buses. *Transportation Research Part C: Emerging Technologies*, 38:1–15, 2014.

[112] L. Yu, T. Jiang, and Y. Cao. Energy cost minimization for distributed internet data centers in smart microgrids considering power outages. *IEEE Transactions on Parallel and Distributed Systems*, 26(1):120–130, 2015.

[113] K. Gai, M. Qiu, and H. Hassan. Secure cyber incident analytics framework using monte carlo simulations for financial cybersecurity insurance in cloud computing. *Concurrency and Computation: Practice and Experience*, 29(7), 2017.

[114] K. Gai, M. Qiu, H. Zhao, and J. Xiong. Privacy-aware adaptive data encryption strategy of big data in cloud computing. In *The 2nd IEEE International Conference of Scalable and Smart Cloud (SSC 2016)*, pages 273–278, Beijing, China, 2016. IEEE.

[115] Y. Li, K. Gai, L. Qiu, M. Qiu, and H. Zhao. Intelligent cryptography approach for secure distributed big data storage in cloud computing. *Information Sciences*, 387:103–115, 2017.

[116] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376, 2015.

[117] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate. A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud Computing*, 3(1):11–17, 2015.

[118] A. Betzler, C. Gomez, I. Demirkol, and J. Paradells. CoAP congestion control for the Internet of Things. *IEEE Communications Magazine*, 54(7):154–160, 2016.

[119] L. Costa, J. Rabaey, A. Wolisz, M. Rosan, and M. Zuffo. Swarm OS control plane: an architecture proposal for heterogeneous and organic networks. *IEEE Transactions on Consumer Electronics*, 61(4):454–462, 2015.

[120] Y. Lee, W. Hsiao, C. Huang, and T. Seng-cho. An integrated cloud-based smart home management system with community hierarchy. *IEEE Transactions on Consumer Electronics*, 62(1):1–9, 2016.

[121] M. Cintuglu, O. Mohammed, K. Akkaya, and A. Uluagac. A survey on smart grid cyber-physical system testbeds. *IEEE Communications Surveys & Tutorials*, 19(1):446–464, 2017.

[122] S. Fang, D. Xu, Y. Zhu, J. Ahati, H. Pei, J. Yan, and Z. Liu. An integrated system for regional environmental monitoring and management based on internet of things. *IEEE Transactions on Industrial Informatics*, 10(2):1596–1605, 2014.

[123] K. Gai, M. Qiu, X. Sun, and H. Zhao. Security and privacy issues: A survey on FinTech. In *Int'l Conf. on Smart Comput. and Communi.*, pages 236–247, Shenzhen, China, 2016. Springer.

[124] J. Wood, R. Beecham, and J. Dykes. Moving beyond sequential design: Reflections on a rich multi-channel approach to data visualization. *IEEE transactions on visualization and computer graphics*, 20(12):2171–2180, 2014.

[125] H. ElSawy and E. Hossain. On stochastic geometry modeling of cellular uplink transmission with truncated channel inversion power control. *IEEE Transactions on Wireless Communications*, 13(8):4454–4469, 2014.

[126] Z. Mo, W. Su, S. Batalama, and J. Matyjas. Cooperative communication protocol designs based on optimum power and time allocation. *IEEE Transactions on Wireless Communications*, 13(8):4283–4296, 2014.

[127] M. Yigit, V. Gungor, E. Fadel, L. Nassef, N. Akkari, and I. Akyildiz. Channel-aware routing and priority-aware multi-channel scheduling for WSN-based smart grid applications. *Journal of Network and Computer Applications*, 71:50–58, 2016.

[128] K. Almotairi and X. Shen. A distributed multi-channel MAC protocol for ad hoc wireless networks. *IEEE Transactions on Mobile Computing*, 14(1):1–13, 2015.

[129] K. Gai, M. Qiu, H. Zhao, and X. Sun. Resource management in sustainable cyber-physical systems using heterogeneous cloud computing. *IEEE Transactions on Sustainable Computing*, PP(99):1, 2017.

[130] H. Zhao, M. Qiu, and K. Gai. Empirical study of data allocation in heterogeneous memory. In *International Conference on Smart Computing and Communication*, pages 385–395. Springer, 2017.

[131] M. Qiu and J. Li. *Real-time Embedded Systems: Optimization, Synthesis, and Networking*. CRC Press, 2011.

[132] M. Qiu, J. Niu, F. Pan, Y. Chen, and Y. Zhu. Peak temperature minimization for embedded systems with DVS transition overhead consideration. In *2012 IEEE 14th Int'l Conf. on High Performance Computing and Communication & 2012 IEEE 9th Int'l Conf. on Embedded Software and Systems*, pages 477–484, 2012.

[133] S. Yin, Y. Tian, J. Xie, X. Qin, M. Alghamdi, X. Ruan, and M. Qiu. Reliability analysis of an energy-aware raid system. In *2011 IEEE 30th Int'l Performance Computing and Communications Conference (IPCCC)*, pages 1–8. IEEE, 2011.

[134] A. Essén. The emergence of technology-based service systems: A case study of a telehealth project in sweden. *J. of Service Management*, 20(1):98–121, 2009.

[135] K. Kanoun, H. Mamaghanian, N. Khaled, and D. Atienza. A real-time compressed sensing-based personal electrocardiogram monitoring system. In *IEEE DATE*, pages 1–6, France, 2011.

[136] M. Qiu, H. Li, and E. Sha. Heterogeneous real-time embedded software optimization considering hardware platform. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1637–1641, 2009.

[137] M. Cartwright, S. Hirani, L. Rixon, M. Beynon, H. Doll, et al. Effect of telehealth on quality of life and psychological outcomes over 12 months (whole systems demonstrator telehealth questionnaire study): nested study of patient reported outcomes in a pragmatic, cluster randomised controlled trial. *BMJ: British Medical Journal*, 346, 2013.

[138] J. Wang, M. Qiu, and B. Guo. High reliable real-time bandwidth scheduling for virtual machines with hidden markov predicting in telehealth platform. *Future Generation Computer Syst.*, 49:68–76, 2014.

[139] X. Tang, K. Li, M. Qiu, and E. Sha. A hierarchical reliability-driven scheduling algorithm in grid systems. *J. of Parallel and Distributed Computing*, 72(4):525–535, 2012.

[140] Z. Chen, M. Qiu, Y. Zhu, X. Qin, and G. Quan. Improving phasor data concentrators reliability for smart grid. *Transactions on Emerging Telecommunications Technologies*, 2013.

[141] J. Li, M. Qiu, J. Niu, L. Yang, Y. Zhu, and Z. Ming. Thermal-aware task scheduling in 3D chip multiprocessor with real-time constrained workloads. *ACM Trans. on Embedded Computing Systems (TECS)*, 12(2):24, 2013.

[142] M. Qiu and E. Sha. Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems. *ACM Transactions on Design Automation of Electronic System*, 14(2):25, 2009.

[143] M. Qiu, J. Liu, J. Li, Z. Fei, Z. Ming, and E. Sha. A novel energy-aware fault tolerance mechanism for wireless sensor networks. In *2011 IEEE/ACM International Conference on Green Computing and Communications*, pages 56–61. IEEE, 2011.

[144] M. Raad and L. Yang. A ubiquitous smart home for elderly. *Information Systems Frontiers*, 11(5):529–536, 2009.

[145] S. Fauvel, A. Agarwal, and R. Ward. Compressed sensing and energy-aware independent component analysis for compression of eeg signals. In *IEEE Int'l Conf. on Acoustics, Speech and Signal Processing*, pages 973–977, Vancouver, BC, Canada, 2013.

[146] N. Botezatu, R. Lupu, and A. Stan. Energy-aware routing for e-health wireless sensor networks. In *E-Health and Bioengineering Conference (EHB), 2013*, pages 1–4. IEEE, 2013.

[147] K. Gai, M. Qiu, L. Tao, and Y. Zhu. Intrusion detection techniques for mobile cloud computing in heterogeneous 5G. *Security and Communication Networks*, PP:1–10, 2015.

[148] M. Qiu, H. Su, M. Chen, Z. Ming, and L. Yang. Balance of security strength and energy for a PMU monitoring system in smart grid. *IEEE Communications Magazine*, 50(5):142–149, 2012.

[149] M. Qiu, L. Zhang, and etc. Security-aware optimization for ubiquitous computing systems with SEAT graph approach. *J. of Computer and Syst. Sci.*, 79(5):518–529, 2013.

[150] M. Qiu, W. Gao, M. Chen, J. Niu, and L. Zhang. Energy efficient security algorithm for power grid wide area monitoring system. *IEEE Transactions on Smart Grid*, 2(4):715–723, 2011.

[151] Y. Guo, Q. Zhuge, J. Hu, M. Qiu, and E. Sha. Optimal data allocation for scratch-pad memory on embedded multi-core systems. In *IEEE International Conference on Parallel Processing*, pages 464–471, 2011.

[152] M. Qiu and K. Gai. *Mobile Cloud Computing: Models, Implementation, and Security*. CRC Press, 2017.

[153] L. Qiu, K. Gai, and M. Qiu. Optimal big data sharing approach for tele-health in cloud computing. In *IEEE International Conference on Smart Cloud*, pages 184–189, New York, USA, 2016. IEEE.

[154] N. Botts, B. Thoms, A. Noamani, and T. Horan. Cloud computing architectures for the underserved: Public health cyberinfrastructures through a network of healthatms. In *IEEE 43rd Hawaii Int'l Conf. on Syst. Sci.*, pages 1–10, Honolulu, HI, USA, 2010.

[155] Y. Li, K. Gai, Z. Ming, H. Zhao, and M. Qiu. Intercrossed access control for secure financial services on multimedia big data in cloud systems. *ACM Transactions on Multimedia Computing Communications and Applications*, 12(4s):67, 2016.

[156] K. Gai, M. Qiu, and H. Zhao. Privacy-preserving data encryption strategy for big data in mobile cloud computing. *IEEE Transactions on Big Data*, PP(99):1–11, 2017.