PERFORMANCE ANALYSIS OF TIME AND COST EFFICIENT DATA PRE-FETCHING IN MOBILE CLOUD COMPUTING

by

Kaveh Khorramnejad

BSc., Science & Research University of Fars, Iran, 2012

A thesis

presented to Ryerson University

in partial fulfilment of the requirements for the degree of

Master of Applied Science in the

program of Electrical and Computer Engineering

Toronto, Ontario, Canada, 2017

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

Abstract

Performance Analysis of Time and Cost Efficient Data Pre-fetching in Mobile Cloud Computing

© Kaveh Khorramnejad, 2017

Master of Applied Science

Electrical and Computer Engineering

Ryerson University

Recently, many methods and algorithms have been proposed in pre-fetching area. However, pre-fetching integrated with workload scheduling approaches have not been investigated as much. Initially, this thesis reviews the principles of the existing prefetching strategies considering latency and cost factor as primary objectives. Later, it focuses on an integrated workload scheduling and pre-fetching model to enhance the performance of response time and minimize the cost. Furthermore, response time and cost problems are formulated and to overcome the total response time and cost problems a heuristic approach is proposed. Integrated model is tested for wide range of variables and, the effects of various parameters such as processing speed and pre-fetcher's utilization are analysed and compared. Finally, based on the results integrated pre-fetching and workload scheduling model outperforms either of them, individually. Thus, this thesis can contribute to the the new solutions in this area.

Index Terms — Mobile Cloud Computing, Workload-Scheduling, Pre-Fetching.

Acknowledgments

I would like to thank all those who contributed in some way to the work described in this thesis. First and foremost, I would like to express my special appreciation and thanks to my supervisors, Alagan Anpalagan and Ling Guan, for their supervision and support. Every meeting has contributed to the progress of my research and thesis. Their patience and understanding are also what makes this journey more enlightening and rewarding. The members of RRM+RAN Research Group and WINCORE lab members contributed immensely to my personal and professional time at Ryerson. The group has been a source of friendships as well as good advice and collaboration. I also want to thank Dr. X. Nan for his initial help and fellow researcher, Lilatul Ferdouse, for her helpful recommendations along the path. They were a great help, and their assistance is greatly appreciated.

Additionally, I would like to extend my gratitude to the Department of Electrical and Computer Engineering and the School of Graduate Studies of Ryerson University for their financial supports and all the incredible opportunities they have provided for me. I am grateful for the funding sources that allowed me to pursue my graduate school studies.

Last but certainly not least, I would like to acknowledge my family and friends, without whose help and encouragement I would not be able to accomplish my goal and finish my degree.

Kaveh Khorramnejad

Ryerson University

Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABREVIATIONS

CC	Cloud Computing
MCC	Mobile Cloud Computing
HR	Hit Rate
BHR	Byte Hit Ratio
VM	Virtual Machine
ML	Machine Learning
PF	Pre-fetcher
QoS	Quality of Service
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
SMD	Smart Mobile Devices
DLS	Dynamic Loop Scheduling
CASA-DB	Deadline Based Context Aware Simulated Annealing
SA	Simulated Annealing
CSB	Cloud Service Broker
ARC	Adoptive Ratio Controller
FE	Future Execution
DCE	Double Core Execution
AMP	Aggressive Meta-data Pre-fetching
OBC	Object Boundary Circle

ANN	Artificial Neural Network
CBR	Cased Based Reasoning
NCST	Non Compact Suffix Trees
CART	Classified and Regressive Trees
LRU	Last Recently Used
LFU	Last Frequently Used
FIFO	First In First Out

Chapter 1

Introduction

Communication over the Internet with minimum latency is more critical at present than at any time in history. With the evolution of the Internet, data transmission methods have changed; no longer point to is point data transmission with traditional cables and physical connections required. The topic of cloud computing is relishing a growing popularity among practitioners, scientists as well as researchers. Knowing that based on the latest industry reports 69% of companies are forecasted to invest in the technology, and 80% of companies report some degree of influence of cloud computing on their business based on Oxford Economics studies in 2015. However, simultaneous speed of data transmission comes to concern. To help enhance the speed over the internet, cloud computing was introduced. Cloud technology helped increase the ability to provide computing services to users faster and more securely. The vast computing resources lead to the cloud being selected as the integrating stage for the mobile environment. The cloud has services like software, platform and infrastructure.

Since devices face some limitations such as power, memory and computing power, cloud providers offer services such as scalable and reconfigurable computing, storage and network services. Cloud Computing is a technology with growing popularity for all kinds of organizations including for profit and non-profit organizations. The main advantages that lead to this popularity are unlimited storage, backup and recovery, cost efficiency, easy maintenance and quick deployment. Even though cloud computing has made our life much easier, there are yet many challenges to face. Latency is the most important factor to satisfy when it comes to delay-sensitive transactions like banking, stock market and multi-million dollar trading businesses.

To classify the in-demand areas in Fig. 1.1, we divided some of the hot research topics into two broad categories based on their delay sensitivity: Social Areas (Education, Social media, Entertainments, etc.) and Technical Areas (Engineering, Business, Management, etc.). As it is presented, these two vast areas have an intersection in some of the subjects and this is where delay

delicacy comes to concern. In this thesis study, we proposed a pre-fetching approach to improve the efficiency in these areas.

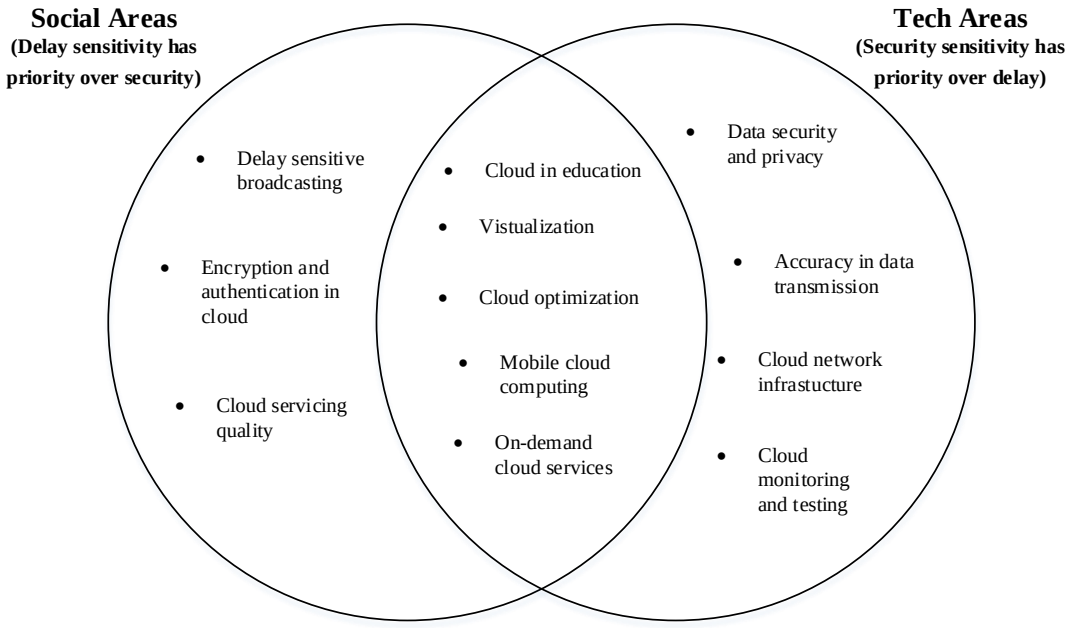


Fig. 1.1. Cloud computing in-demand research areas diagram.

Hence, pre-fetcher was introduced. Pre-fetcher gives us the flexibility to store the critical and vital information closer to the source of the request. In this case, the access will be much faster. Pre-fetchers can be defined in various storage sizes with different processing abilities.

Scheduling the workload and breaking it down into smaller pieces while allocating it to different servers could manage and minimize the budget assigned to particular task or project. Compared to other pre-fetching strategies, the hardware required to implement software initiated prefetching is modest. Most of the methods' complexity lies in the judicious placement of fetch instructions within the application. Choosing where to place a fetch instruction relative to the corresponding load or store instruction is known as pre-fetch scheduling. Software prefetching can often take advantage of compile-time information to schedule pre-fetches more accurately than hardware techniques. In practice, it is not possible to predict exactly when to schedule a pre-fetch so that data arrives in the cache exactly when the processor will request it. The execution time between the pre-fetch and the matching load or store may vary, as will memory latencies. These uncertainties must be considered when deciding where in the program to place fetch instructions. It is often challenging to schedule the pre-fetching process accurately. If the compiler schedules

fetch too late, the data will not be in the cache when the processor pre-requisites it. If the compiler schedules the fetches too early, the cache may discard the data before it can be used to make room for data that the controller fetches later. Early pre-fetches also might displace data in the cache that the processor is using at the time. This situation, in which there is a miss that would not have occurred without prefetching, is called cache pollution. The most efficient pre-fetching scheduling method so far is a combination of both hardware and software (hybrid) which will be discussed in detail in chapter 2 of this thesis.

1.1. Motivation

Cloud based multi-media services face three main challenges: Latency, Cost and Scalability. In traditional networks before the existence of Internet and cloud, providers used to have physical cable connections in order to transmit data. Cable connectivity is known as the most secure way to send and receive data. However, point-to-point data transmission is not practical when it comes to a larger scale; for instance, connecting a building from the east side of the city to the west side of the city. As soon as large scale data transmission came to concern, wireless connectivity and access point were invented to satisfy this need. Wireless networking has expanded rapidly, and the users are increasing every day. Unlike the cable connections, wireless networking is shared and accordingly, bandwidth is not necessarily dedicated to a single user. Within the radius range of the wireless transmitter, data can be received, but once the mobile user is not in the range, the connection is lost. This was the beginning of the need for cellular communication. Cellular data transmission was so successful until battery lives could not keep up with the users need. Another lack in this connectivity was a memory, cellular users mostly cell phone users did not have unlimited storage to store all the information they require. To overcome this challenge, cloud computing was introduced. In cloud computing, cloud service providers do not face those shortcomings, no longer was storage capacity nor battery life considered as a problem, accessibility and saleability were addressed and solved to some extent. However, delay became a new challenge to researchers in this area. Pre-fetching was one of the techniques that was proposed to reduce the delay in multi-media delay sensitive mobile communication. This technique can reduce the delay significantly if it is well designed and implemented. In this thesis, we focus on a pre-fetching technique to solve latency problems. In this study, cost and transmission speed are directly relevant

factors, meaning that, once the rate of data transmission is increasing, the cost is also increasing and vice versa. Hence, cost efficient, data pre-fetching was proposed to satisfy both delay and cost.

1.2. Contributions

This thesis initially targets multi-media cloud and end users communication in existence of pre-fetching method. Later, it explains and analyses different works in this area and, then we propose a system model to solve the time and cost problem associated with data access and transmission. In the end, as it is presented in figures and the analysis in Chapter 4 of this thesis, the response time and cost (access delay) problems which are known as convex mixed-integer problem (NP-hard) are solved using the heuristic algorithm in Matlab because of the complexity of the problem. In Chapter 3, system model is designed and proposed to show the dynamics of the interaction between multi-media cloud, end users and, pre-fetcher. Characteristics and attributes of each component and entity in the model are discussed and introduced. A number for each object is assumed and assigned prior to simulation to analyse different objectives. In Chapter 4, problem is defined and formulated using the assumed values in the model. Constraints and the objective function are formulated. The problem type is indicated as NP-hard and the solution is proposed to be heuristic algorithm. Time problem is proposed and solved, and cost penalty formulation is proposed to find the optimal solution. Simulation on the proposed methods was done in Matlab. The data used for simulation was extracted from previous work done in this area by Ferdouse in [1]. Time minimization objective is satisfied as well as cost problem which is optimized and generated in graphs. Evaluation of proposed approach was achieved and analysed using figures generated from Matlab, and the comparison is made in Chapter 4 of the thesis. The comparison was made between the old method and the proposed solution in existence of pre-fetcher.

Main contributions in this thesis are as follows:

- A new model for total time optimization in cloud networks is presented.
- Cost optimization for this model is described and solved.
- The model is tested and the results are shown for a wide variety of variables.
- With the proper values, a significant decrease in time and cost can be achieved.

1.3. Organization of Thesis

The rest of this thesis is organized as follows:

Chapter 2 first gives an introduction to cloud computing and its benefits and challenges, an overview of recent advances in mobile cloud computing (MCC) then a review of pre-fetching method and techniques, hardware controlled, software controlled and, hybrid.

Chapter 3 presents the system model and architecture. Investigates multi-media cloud and pre-fetcher communication problems, problem formulation then discusses feasible approaches to solve the time and cost problems.

Chapter 4 studies simulation results and performance evaluation using Matlab then discusses each figure, the parameters and the performance analysis.

Chapter 5 is the conclusion and future work

Chapter 2

Review on Data Pre-fetching & Workload Scheduling Schemes on Mobile Cloud Computing

Background and Literature Review

In today's world, almost everything is done over the Internet and with the help of local networks (e.g. WLAN, Ethernet, and Home Networks). Demand for a fast and reliable network is rapidly growing, specifically in the last decade. Hence, the invention of Internet and its combination with traditional local networks made it possible to access information from anywhere and at any time. Accordingly, various methods have been proposed to increase the speed of data transmission while avoiding the possible collisions to lessen the delay over the Internet connection, which led to the invention of cloud computing. Cloud computing can be described as a model for enabling ubiquitous, convenient and on-demand networks access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort from the user side and minimal service provider interaction.

2.1. Cloud Computing

Nowadays, we are surrounded by information, and the primary concern we are facing is how to receive and store these data in the fastest and most efficient was possible. On the other hand, accessing stored information would be another major topic.

To overcome limitations such as bandwidth, scalability, availability, storage and cost, cloud computing (CC) services have been invented. CC allows users to access applications and services without the need of installing them and have their files available through any device and at any time over an Internet connection.

What is Cloud Computing? In the simplest terms, cloud computing means storing and accessing data and programs over the Internet instead of one's computer hard drive. The cloud is

just a metaphor for the Internet, as the cloud symbol was originally used to represent the Internet in flowcharts and diagrams. The fundamental differences between cloud computing and traditional computing can be explained as follows:

I. Cloud computing is the ability to access data anywhere and any time. While in traditional computing, user can access the data only on the system in which he has stored it. If the user has to access the data in another system, he needs to save it in an external storage medium.
II. Cloud computing allows a company to pay for only as much capacity as needed, and to bring more online as soon as required. This pay-for-what-you-use model resembles the way electricity is consumed. Unlike traditional computing which demands considerable software and hardware establishment, once you reach over your space capacity you need to install more. Cloud computing offers a number of environmental benefits that traditional hosting approaches find difficult to match such as fast access and availability.

That is to say, cloud computing will take the world by storm, because of its merits which give this technology an advantage over traditional computing.

Major CC deployment models fall into the following categories as it is demonstrated in Fig. 2.1: Private, Community, Public and Hybrid. Private clouds are hosted from within the cloud, from a remote server or organization that owns it as their property. The main purpose of private cloud is to be dedicated and operated by an organization exclusively. On the contrary, community cloud is meant to be used for several organizations having common policies, services or concerns. Similar to private cloud, a community cloud is also hosted from within or from a remote server externally. When it comes to public cloud, it can be understood from its name that it is a part of the major cloud infrastructure which is exposed to the public to be accessed, revised and stored remotely. Hybrid clouds can be a combination of two or more clouds while each has their individual features as they are bundled together. One of the remarkable benefits of using hybrid cloud is cloud bursting, meaning that under the high volume of demands for a private cloud, it can access public cloud to provide the service and will be released as the demand volume decreases.

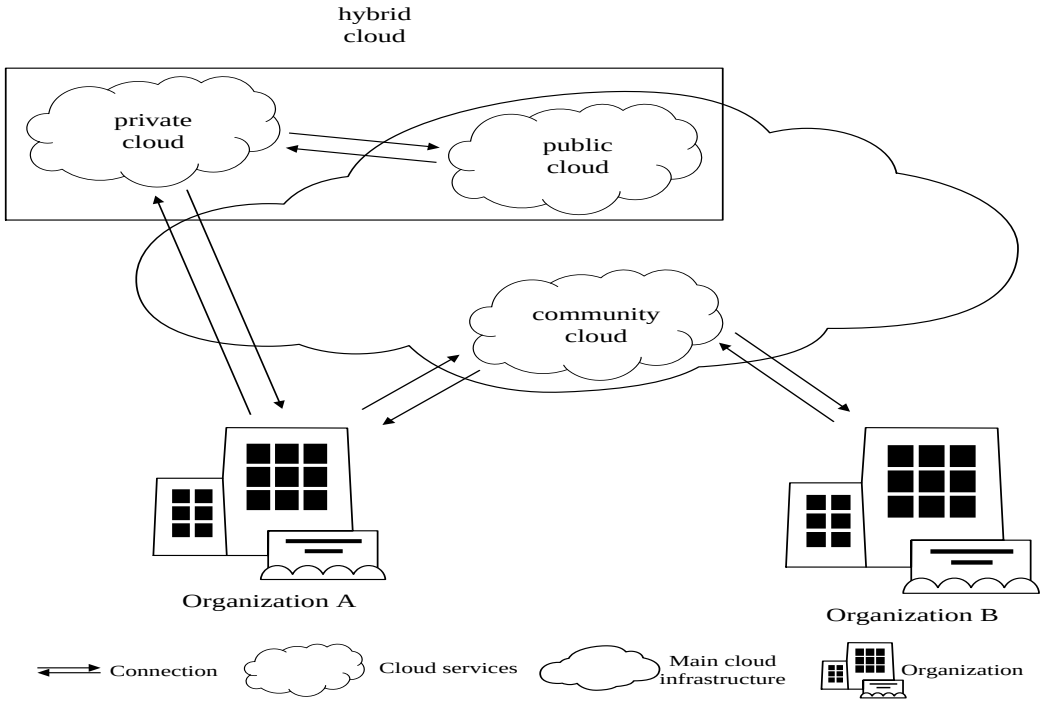


Fig. 2.1. Cloud Categorization Modeling.

Meanwhile, cloud computing enhances data accessibility, flexibility and significantly reduces the cost of implementation. However, the high demand of the services specifically by mobile users leads to major network congestions issues.

Cloud services can be categorized into these major service models such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). The most flexible cloud computing model is known as IaaS, which is designed for automatic deployment of servers, processing power, storage, and networking. In IaaS users have more control over their infrastructure than PaaS or SaaS service users. The prominent use of IaaS includes the actual development and deployment of PaaS, SaaS, and web-scale applications.

PaaS operates on a level below SaaS, technically providing a platform which software can be developed on. As with most cloud services, PaaS is built on virtualization technology. Businesses can demand resources as they need them, scaling as demand grows, rather than investing in hardware with redundant resources.

Use of SaaS applications tends to reduce the cost of software ownership by removing the need for technical staff to manage install, maintain, and upgrade software. SaaS applications are usually provided on a subscription model.

Table 2.1 below shows the differences between IaaS, PaaS and SaaS.

Table 2.1: Categorized based on U, V for different applications (new service model vs. traditional IT).

	Application	Runtimes	Databases	Servers	Virtualization	Storage	Networking
Old IT	U	U	U	U	U	U	U
IaaS	U	U	U	U	V	V	V
PaaS	U	V	V	V	V	V	V
SaaS	V	V	V	V	V	V	V

U: user manages V: managed by vendor

2.2. Mobile Cloud Computing (MCC)

In this section, mobile devices are reviewed and the benefits of using them, along with their interactions with cloud services are mentioned.

Cloud computing is the delivery of computing services over the Internet on the pay-per-use basis. The cloud computing model allows access to information and resources on an anytime and anywhere basis. Cloud services are very useful as they include online file storage, social networking, webmail, and online business applications, etc. By using these services, business persons can use software and hardware that are managed by third parties at remote locations. Cloud computing also provides a shared pool of resources, including data storage space, networks, specialized corporate and user applications. Moreover, there are challenges in this regard that need to be addressed and discussed.

As discussed previously, to resolve the issues considering the limitations we face in mobile cloud computing (MCC) e.g. mobility, communication, and portability, a comprehensive understanding of MCC is required. The main benefits of MCC are, extending battery lifetime, improving data storage capacity, processing power and improving reliability. Mobile cloud

computing is the advanced version, or it is a combination of the two most important practical computing paradigm: cloud computing and mobile computing. As MCC is based on the cloud concept, the centralized applications, services and resources are accessed over the wireless network technologies based on the web browsers of smartphones. Many of business persons are attracted by MCC as a profitable business option since it reduces the development, execution cost of mobile applications, and mobile users are enabled to acquire new technology on-demand basis. Like every other technology, MCC is faced with many challenges. Below some of the main challenges are discussed.

- Scalability: Overcoming this challenge plays a vital role in any technology. In MCC, maintaining scalability is vital and constantly needs to be assured. Hence, in end user side we are make sure that first the user is an authorized user of our system and has already passed through the procedure provided by the local server. Meantime, on the cloud side, scalability has to be consequently maintained in public and private level or both. Any minor neglect might cause the service to be unreachable, and this means more cost.
- Offloading: Considering this factor will lead to have a significant saving in power consumption, overall load on mobile devices and result in overall battery life. Offloading can be regarded as a solution to augment mobile systems features by migrating computation to more resourceful computers such as servers; this makes it different from the traditional client-server architecture besides it varies from migration model used in multiprocessor systems and grid computing where a process is migrated for load balancing. The major difference is where it migrated programs to, a server outside of the users' immediate network.
- Availability: It has been mentioned that wired networks are more reliable in the sense of connectivity compared to wireless one. That is another major challenge that needs to be considered in order to have a better performance in mobile networks. Bandwidth limitation causes us to use different algorithms to minimize the chance of congestion during peak hours, and that is where availability comes to action and becomes vital.

As an instance, the offloading challenge is taken into consideration and sees what the possible options and limitations we have are, and how to overcome them. Several factors are involved in enhancing efficiency in offloading the consumptional tasks in the cloud, which are

naturally dynamic. Based on the nature of these entities, cloud setup varies from time to time. Hence, sometimes according to the band width and the amount of data load needed to be transferred, due to high power (battery) consumption, it is not practically efficient. Formula (2.1) is used to calculate the saved energy (P_s) while studying the discussion above.

$$P_s = P_c \times \frac{C}{M} - P_i \times \frac{C}{S} - P_{tr} \times \frac{D}{B} \tag{2.1}$$

Where,

P_c is the power consumption by the mobile when executing instructions,

P_i is the idle power consumption of the mobile,

P_{tr} is the power consumption while transmitting data,

C is the total number of instructions in the given computation,

S is the time taken by the cloud server to process instructions,

M is the time taken by the mobile to process the instructions,

D is the total amount of data to be sent in bytes,

B is the total available network bandwidth.

The initial component calculates the overall power (battery) consumption if the mobile device has finished the calculation. The rest calculates the power consumption if the tasks are sent to the cloud server side. It can be understood from the above formula that, if the result is greater than zero, the offloading computation task is done effectively, and if the result is less than zero, then it can be interpreted that the task was not done beneficially. Accordingly, it is obvious that not every task is going to be offloaded practically beneficial based on the given time frame.

$$P_s = \frac{M}{C} \times \left(\frac{P_c}{1} - \frac{P_i}{F} \right) - P_{tr} \times \frac{D}{B}. \tag{2.2}$$

With respect to, $S = F \times M$

Meaning that the cloud server is faster than the mobile processor core user F times. Considering that, while replacing the above condition in formula (2.1), leads to formula (2.2) [2]. To maximize the outcome, the value of F should be significantly large to minimize the whole value for P_i / F. This means cloud server has to function multiple times faster than the mobile processor.

As we can observe in the example given above, in MCC problems, several steps and trade-offs need to be taken to get an efficient result. In the next section, we will discuss how to prevent some of these issues by scheduling the workload and pre-fetching information with different strategies.

2.2.1. Workload (Resource) Scheduling

Even though relying on offloading mechanism as mentioned earlier, MCC can considerably improve the computational ability of smart mobile devices (SMDs), yet developing a reliable platform in MCC can be challenging. The major issue is how to achieve a power efficient computation offloading.

Applications on the mobile devices need to be split into the smaller sequence of tasks which can be run on either a mobile device named local computing or on the cloud service called cloud computation. That is where scheduling the workload comes to action, in order to host SMDs running complicated apps. Clearly, it is not practical to execute them just on SMDs due to due to lack of sufficient resources [3].

There are several major questions to be answered in detail, with the previous work done in this area and various approaches used to explain those issues. The main questions are as follow:

- Could the cost of power efficiency be minimized while considering the task sequence in cloud source by allocating the data transmission energy in every offloading calculation?
- Could we manage and control the CPU clock frequencies [4] approach locally, to optimize the power efficiency cost?
- If two major constraints, task dependency and apps complication, occur simultaneously, what is going to happen with the calculation of offloading selection?

The enforcement is a need according to apps complexity time which is not an easy task to maintain in delay sensitive apps.

Moreover, pre-fetching could be considered as an individual alternative or as a combined technique with load scheduling to work around different constraints to optimize the outcome.

2.2.2. Pre-fetching

Pre-fetching is technically predicting the information that will be needed in the near future and have it ready (generally in a temporary storage called cache which refreshes its inventory every once in a while) for when the actual request will be received. Based on that, it is important to use techniques to predict the information as accurately as possible; otherwise, not only it will not lessen the processing time and cost, but will it increase the overhead and act counter-productively. There are numerous techniques proposed in recent studies which will be discussed later.

Pre-fetching overcomes the boundary of passive caching by predicting and retrieving the cache resources for future requests proactively. Pre-fetching does not decrease the data exchange rate, but unfortunately, it sometimes increases due to inaccuracy in prediction algorithms. For instance, the algorithm might predict the majority of the requests appropriately based on future demand. Yet, there will be some unnecessary fetched resources.

Every approach has its own weaknesses mainly in pre-fetching overhead, but we will discuss how to resolve the problems related to delay, performance, large data, and data availability with the pre-fetching mechanism. Hence, the pre-fetching method should have conditions and specifications such as:

- Acknowledging the successful previously fetched data for future use, e.g. pre-fetch the data that is more likely going to be requested next.
- Data has to be requested not too early or not too late for a specific request, in other words, it needs to be timely.
- It also needs to be stored in memory properly, for instance, it will not be efficient if it is stacked behind the data that was pre-fetched but not used.

Considering the above conditions, we will categorize the previous work done in this area into tables and have a comparison between them [5].

2.3. Workload Scheduling

Scheduling tasks in every system gives us an order. Using that order, we can predict and pre-fetch the information that is more likely going to be requested in future. Hence, we need an organized method to distribute the given tasks to improve the overall system performance, in meantime resource cost and response time will be minimized.

To review some proposed scheduling approaches, we classified them into three general categories: Level Based, Policy Based and Algorithm Based.

2.3.1. Level based Scheduling

In order to have an optimized system to schedule the load in the cloud, we will categorize them into two levels, *user level* and *task level.* In user level, requests for a particular service are going to be distributed to several clusters based on the load density in each of the clusters (clusters consist of a group of computers offering the same service or task). This level helps to avoid local congestion in the first place. In other words, prevention should be considered in user level. On the other hand, in task level, a multimedia service on the cloud is going to be broken into tasks, and those tasks will be assigned to virtual machines either in a sequential order or simultaneously.

Considering the heterogeneous nature of resource capacities, to optimize the load scheduling we need to have an adaptive system for varying loads. Moreover, it is difficult to satisfy all precedence constraints due to precedence conditions in multimedia services while assigning a task to virtual machines (VMs). Operation structure is not the same in multimedia services, which can be classified into three general types: parallel, sequential and mixed. In the parallel case, tasks can be executed simultaneously. In the sequential case, each task has to wait until the previous task is completed before it can be executed. In a mixed structure, we can do a combination of both.

D. Sengupta et al. [6] propose string scheduler approach. String scheduler clusters Graphic Processor Unit (GPU) scheduling tasks into a mixture of load balancing and per-device scheduling, device- level scheduling benefits from using all parts of GPU, load balancing checks and validates the priority order of GPU tasks.

2.3.2. Policy based Scheduling

A. Ilyushkin et al. [7], in another study, present four scheduling policies, such as strict reservation policy, scaled Level of Parallelism (LoP) policy, future eligible sets policy and backfilling policy to deal with the oscillation workload of workflows (WFs). Their most important conclusion was that any form of processor reservation for WFs with no runtime predictions solely reduces the total system performance, causing significantly low maximal utilizations.

Based on the mentioned policy proposed previously in [8], P. Gupta proposed a phase assisted approach that facilitates workload scheduling task while minimizing the contentions. To achieve that, and to evaluate the CPU resource contention for a mixed bundle of loads which share the same resources, they developed an analytical approach. Based on that, nodes with a higher risk of contention will be filtered, and they would be easy to select from fewer nodes to allocate them for a new schedule. They recognized the threshold bound between two workloads which later can be used to estimate how likely they are going to occur at the same time. The approach was applied to all the nodes in the cluster using an OpenStack framework, and 98.77% accuracy was achieved in their estimation if the workload is mixed and using the shared CPU resources.

2.3.3. Algorithm based Scheduling

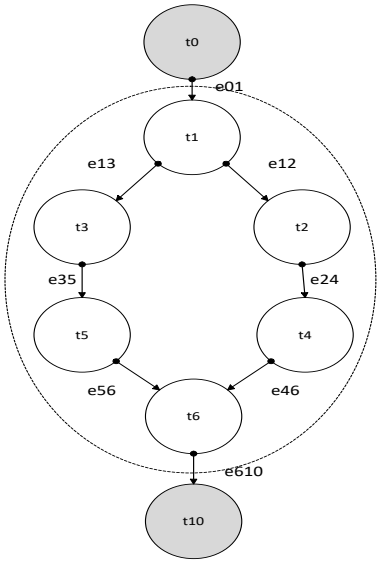
Balasubramaniam et al. [9] compared two different algorithms to study the behaviour of the dynamic loop scheduling (DLS) and the divisible load theory (DTL) methods on two network topologies, called single level tree network and linear array network. DLS and DLT are two well-known algorithms used in the scheduling of divisible workloads. According to the simulation results, DLS overtakes the DLT in the existence of runtime load disparity. Meanwhile, DLT overtakes DLS in the lack of it. Higher communication costs favour DLT while higher computing costs favour the DLS.

What we can interpret from the above-proposed approaches, is that the loads and tasks come in different types; hence we need to apply a proper method to handle them. In the meantime, overcoming the scheduling objection would have a cost, which can be either time, maintenance or space as a trade-off.

H. Ghoumain et al. [10] looked at the problem from a different angle. The authors define resource allocation as processing and placing the most suitable set of virtual machines from a pool

of resources for a certain number of tasks. Later it was proposed to do resource allocation based on simulated annealing algorithm (SA). Here we are going to discuss some of the proposed approaches.

There are sensors in mobile devices which the context manager will collect data from, and transfer them as a whole bundle to Cloud Service Broker (CSB) as well as manage the updates and data on brokerage to see if any new updates or changes happen in the context. In Fig. 2.2, authors considered location, device and the data rate in the context.



Application Workflow $W = (T, E)$,

Bundle of tasks $T = \{t_1, t_2, t_3, ..., t_n\}$,

Dependency indicator between tasks t_i and t_j & $e_{ij} \in E$

Fig. 2.2 Task directed diagram for sub-workflow [10].

The authors explain that this algorithm is based on a melting system at a very high temperature and cooling it off to a freezing point gradually afterwards. The algorithm of the SA is based on the algorithm proposed by M.Wang [11].

To interpret the flowchart in Fig. 2.3 which refers to annealing-deadline based algorithms as an example, we need to take execution time into consideration which will be achieved when all the assigned tasks are completed. Maximum time of all tasks for a user defines the execution time that users' workflows need.

Where, Max (completed time for the whole task) = execution time of all tasks

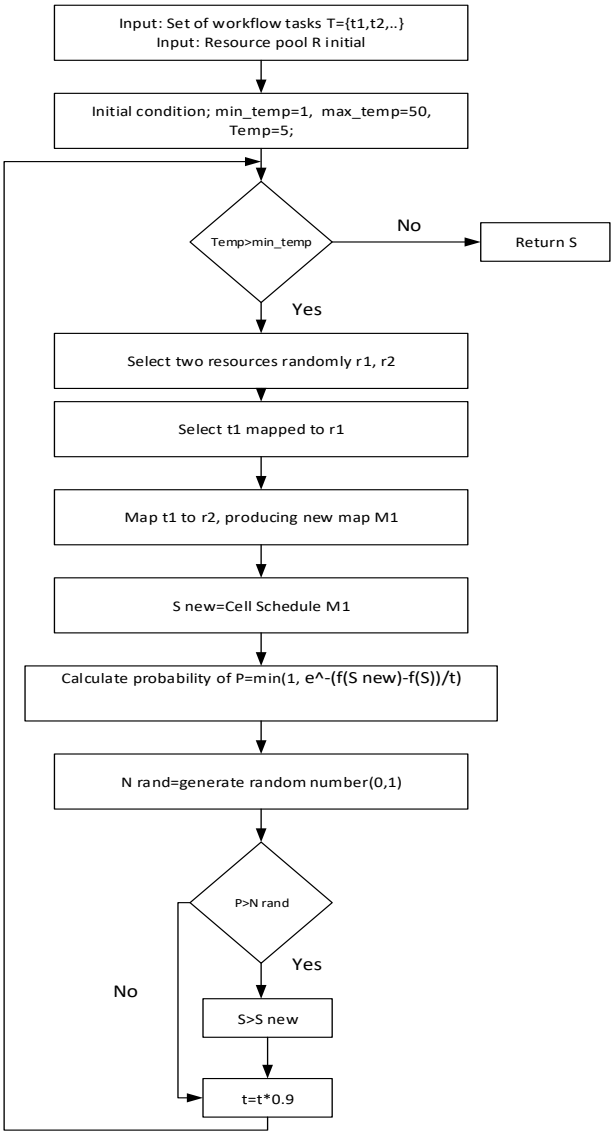


Fig. 2.3 Flow chart of Annealing-Deadline based algorithm [10].

Mostly for sub systems we use the deadline based simulation annealing algorithm (SA-DB), CASA-DB, context aware deadline based simulated annealing algorithm is used implementing context aware scheduling for allocating workload. To sum up, to compare cloud service enactment in both context free and context aware we use both algorithms.

2.4. Pre-Fetching

As human knowledge is increasing every day, demand for technologies, cloud services and mobile devices are also increasing accordingly. Hence, to catch up with the human need cloud computing and mobile devices came to an integration called mobile cloud computing (MCC),

which offers numerous benefits to clients. However, the significant growth in the amount of MCC user's delay and finally congestion in the network was also noticed. Considering the major limitations MCC faces such a battery, storage and bandwidth, the pre-fetching approaches were proposed. MCC architecture can be summarized in three layers: presentation phase, application phase and database phase [5].

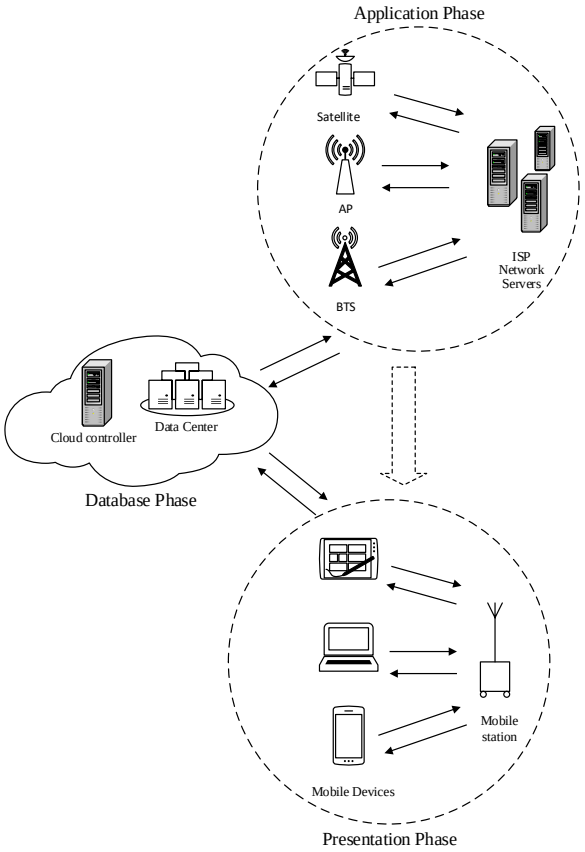


Fig. 2.4 MCC three layer architecture [5].

In the model presented in Fig. 2.4, reliability and scalability need to be concerned in addition to latency which was mentioned before.

Based on recent research done in this area, pre-fetching is known as the best way to handle these problems. Accordingly, approaches are classified in three broad categories: Static (re-active), Dynamic (pro-active) and Mixed (hybrid).

2.4.1. Hardware Controlled (Re-active)

In hardware controlled data prefetching, prefetching is implemented in hardware [12]. Several methods support hardware controlled prefetching. History-based prediction is the most efficient one used among hardware controlled data prefetching schemes. According to these studies, a pre-fetch engine (PE) is used to predict future data references and to create prefetching schemes. With no need of using user interface, all the entities of prefetching are implemented on a processor. PE observes the old record of data request and accesses or the history of cache misses to forecast which data will be requested. Runahead execution uses idle sets or cores to run instructions when a CPU is delayed or idle. Dual-Core Execution (DCE) approach uses an idle core of a dual-core processor to construct large, dispersed instruction Ganusov et al.'s future execution (FE) [13] uses an idle core using value estimate to pre-execute future loop repetitions. Another hardware controlled prefetching scheme worth mentioning is off-line analysis. Kim et al. [14] proposed a method where data access patterns are analyzed for hotspots of code that are often executed. For applications that refer to similar data access pattern, using this scheme is beneficial.

2.4.2. Software Controlled (Pro-active)

A real-time prefetching algorithm was proposed by Li and Wu [15], relying on sequential pattern meaning that it can hide data access latency which is a major factor impacting the quality of service (QoS). This can resolve the problem by accurately predicting and pre-fetching the data which will be requested by the user to local storage hosted by nodes. They concluded that using pre-fetching decreases the overhead in sending data and prevents wrong pre-fetching, trading off for extra load on the network as a result.

In real-time services especially multimedia and online streaming services, any small jitter or delay will have a significant impact on the performance. We can mention that in web cluster systems, the biggest problem is memory and memory management due to the gap between disk and processor. Heung Ki in [16] proposes a double-prediction-by-partial-match (DPS), which can be used within the web frameworks and create an adaptive ratio controller (ARC) to identify the ratio dynamically.

In a similar work to [15], authors in [17] studied a pre-fetching approach on hybrid flash-disk memory (combination of hard disk and flash memory). In their study, they proposed a two-

level method considering file and block level in a sequential order access in pre-fetching. This technique improves the performance on pre-fetching while it does not act efficiently in the cloud. The reason is that the overhead on the network caused by pre-fetch is over fitted and contains lots of data objects. Y. Chen in [18] proposes an algorithm-level feedback-controlled adaptive (AFA) method to overcome the delay in accessing data in high-performance computing. The author analyses data access to the cache, and it can dynamically run the pre-fetching algorithm.

X. Wang in [19] proposes a method of pre-fetching to resolve weak QoS of video streaming services over the MCC; for instance, intermittent interferences and latency in buffering. In his study, the author discusses two sections: cloud-assisted adaptive video streaming and social-aware video prefetching. The benefit of using this approach is that it enables the videos to be stored in the cloud in an efficient way, and it trades off power, cost and scalability to achieve it. The previously mentioned approach was taken from the ideas of J. Kang in [20], while J. Kang studied on battery life time prediction in mobile devices based on their usage. For instance, the battery consumption ratio while making voice calls, using data or waiting for phone calls was considered yet regarding scalability and data validation needs further study.

In [21], G. Yunwen proposes pre-fetching intelligence analysis and computing technologies technique. It records user usage pattern, history of service data and view records to improve the response time in terms of speed and quality. The miss ratio indicates performance outcomes and the main factors loss indicates view alignment in this technique. The other disadvantage of using this approach is that it cannot reflect the urgent need for the technique while learning the algorithm offline.

In addition to that, Kung in [22], presents speculative solutions for computer cloud programming. To overcome latency, workload's heterogeneous node affection on the node with slower or overload hardware is considered to decrease the load. However, complexity and memory issues still remain. In [23], Lin proposed an approach to improve the performance of large-scale distributed memory systems. They provided an aggressive metadata on pre-fetching an affinity based metadata pre-fetching (AMP) method, then they proposed the method in large scale distributed storage systems. By historical access, it applies a proper mining information to identify the metadata by AMP.

Table 2.2, provides a summary of pros and cons in the previously studied approaches in this report, and it is similar to the work the author did in [5].

Table 2.2: Comparison table of approaches in pre-fetching.

Author(s)	**Proposed Solution**	**Weakness**
Lin et al. (2008)[23]	Proposed an *Affinity based Metadata Pre-fetching (AMP)* for metadata servers in *large-scale* distributed storage systems streaming and socially aware video pre-fetching.	Aggressive prefetching
Li and Wu (2012)[15]	Proposed *a real-time data pre-fetching* algorithm based on sequential pattern mining to hide the data access delay.	Network overhead
Kung et al. (2010)[22]	Propose a *pipeline-based scheduling strategy* to decrease implementation *delays*	Storage overheads
Yoon et al. (2010)[17]	Proposed the 'two-level' pre-fetching level and considered *both file-level and block-level* with sequential access patterns in pre-fetching.	Heavy burden on the network
Yunwen et al. (2011)[21]	The approach is based on user model in CC using *intelligent analysis and computing technologies.*	Unused resources in network
Kang et al. (2011)[20]	Proposed *prediction model based on usage patterns*, such as the battery consumption ratio while making voice calls, using data communication, or waiting for calls	Privacy and security on validating data through data log
Wang et al. (2013)[19]	Proposed a framework to enhance the quality of *video services* for mobile users, which includes two parts: cloud assisted adaptive video	energy and cost regarding the usage patterns of mobile users

Pre-fetching is considered as the best way to handle latency problems, however over using this method over time may cause overhead storage, and may lead to network congestion. Using artificial intelligence (AI) would help to overcome this problem or at least to make it more reliable.

By applying AI, using methods that consider human intelligence in decision making and learning pattern which in the last stage will become machine intelligence machine learning (ML). ML will minimize the overhead and even the chance of congestion in the network by predicting data which is more likely going to be requested in future more accurately using overaggressive progressing.

2.4.3. Hybrid Controlled

Machine learning helps us obtain a higher performance in the long run, based on what the machine is going to gain by learning from the pattern being used and predict more relevant and useful information. Here, we will see what are works done in applying ML in pre-fetching are, what the gain and loss(trade-offs), pros and cons are. The common ML types used in the report are as follows: Artificial Neural Network (ANN), Cases Based Reasoning (CBR), Classification and Regression Trees (CART), K-Nearest Neighbor (K-NN).

For web mapping service, Garcia in [24] uses ANN in pre-fetching to forecast requested data to estimate future request based on the previous history. Complexity to decrease over-fitting as well as large scale sample remains a challenge.

To bring up an example of ML, and specifically NN in real life, we can discuss its sensitive role in medical and operation purposes. Rajkumar in [25], using neural network studied the cancer diagnosis since artificial neural network (ANN) is prominent in decision support systems. However, it has a problem when it comes to converging to function and network mean square error. Basically, the web prefetching requires two steps: anticipating future pages of users and preloading them into a cache. This means the web prefetching also involves the caching. However, the web caching and prefetching were addressed separately by many researchers in the past.

On another study, Chandrasekaran in [26], researched on different ways to optimal the multi-path machine based cloud computing, online. This technique can be considered as a hybrid since both hardware (machine) and software prefetching are involved. In the results, the author classified approaches into three categories: cutting the force prediction based on empirical relationships, a surface roughness prediction module based on neural network and the tool life prediction model based on empirical relationships. They were working on achieving an optimum point for different procedures and facilitating interactions to access the way various manufacturers

share their data information. On the down side, problems occur while being implemented in a private cloud.

In [27], Sawar et al. studied the comparison of cases retrieval between case based reasoning and neural networks. To achieve more accurate predictions, the authors examined the benefit hybrid of case based reasoning and neural networks. But the most important part to be concerned with this hybrid approach was retrieval which yields related solutions to the current problem. The authors compromise the working of case based reasoning (CBR) and NN to execute the case retrieval. To populate the case, non-compact suffix trees (NCST) was used. The retrieval was applied employing CBR and NN using a bundle of cases randomly chosen. NN acts better than CBR when implementing the case retrieval. Later in [28], the author used hybrid intelligence using ANN and CBR. CBR did not have a static nature in this study while ANN is used for alteration phase of CBR. Compared with the previous study, in this case, CBR performed better than ANN due to the smoother procedure of former history.

Web caching and prefetching are two effective solutions to lessen Web service bottleneck, reduce traffic over the Internet and improve the scalability of the Web system. The Web caching and prefetching can complement each other since the web caching exploits the temporal locality for predicting revisiting requested objects, while the web prefetching utilizes the spatial locality for predicting next related web objects of the requested Web objects. Thus, a combination of the web caching and the web prefetching doubles the performance compared to single caching. This paper reviews principle and some the existing web caching and prefetching approaches. Firstly, we have reviewed principles and existing works of web caching. This includes the conventional and web caching. Secondly, types and categories of prefetching have been presented and discussed briefly. Moreover, the history-based prefetching approaches have been concentrated and discussed with a review of the related works for each approach in this survey. Finally, this study has presented some studies that discussed the integration of web caching and web prefetching together.

Maurer et al. [29] applied approaches to configure cloud infrastructure management, adaptively. They examined the hierarchical assembly of probable reallocation actions implemented and evaluated two information management methods: CBR and a rule-based approach to achieve the mentioned aim for one reallocation level. Regardless, the disadvantage is scalability and learning performance.

Classification and Regression Trees (CART) is a prominent competent in ML studies. Compared with other ML approaches it does not need a large amount of input from analysts [5], [19]. CART is a strong decision tree for data mining, pre-processing and predictive modelling Singh et al. [30].

Based on [30], the CART is a non-parametric statistical procedure to analyse categorization problems. CART applies a decision tree to show the possible ways to classify data. Accordingly, even though we do not have an autonomous test model, for newly received data, we know how any tree will perform. CART is a robust method and simplifies the understanding even for normal people and allows us to deal with misplaced variables. Regardless, it is yet problematic if one tries to capture the linear and additive structure. Moreover, K-Nearest Neighbour (K-NN) is one of the complementary approaches to order items relying on closest training cases in the feature.

K-NN is one of the best approaches when there is no information in hand, on big data bundles. K-NN is classified with the internal election among the neighbours, meaning that it will be chosen if the majority of its neighbours vote for it, then the object will be assigned to the class most common among its nearest neighbor-NN used for user pattern categorization, mostly joint filtering.

According to the scheme, Park et al. [31] proposes a prefetching data structure and OBC (Object Boundary Circle) in order to lessen the user's time being tuned to broadcast channel. Based on the presented approaches, if the needed objects have already been fetched into the cache, the user can execute the K-NN query processing with no need for tuning the channel. Considering the broadcast model, the author is proposing; the channel lets concurrent access concoctions to an arbitrary number of clients. Hence, that benefits from the efficient usage of scarce bandwidth. Yet, studies in this area do not have any solution for moving object database as wells as cache replacement.

The accuracy and coverage of the standard technique for standard joint filtering were used in [32] by Mobasher et al. In foretelling web usage mining tasks, the value of K was chosen according to the analytical sensitivity of the best enactment in terms of coverage and exactness.

Table 2.3 shows a comparison of the ML approaches on prefetching methods, their pros and cons [5]. Consequently, in this review, ML schemes will be used to optimize the outcome that

is utilized for the pre-fetching. Considering ML limitations, it has a great deal of complexity. It is difficult to find case data samples, as an inexact prediction for the chosen case where it needs more training, and it is more time consuming to organize new information.

Table 2.3: Performance analysis using ML methodologies [5].

Approach	Pros	Cons
Artificial Neural Networks (ANN) (Rajkumar et al., 2013; Garcia et al., 2013; Sarwar etal., 2012 Cahndrasekaran et al.)	For teaching it has the ability to engage how to do jobs relying on data given even with major network damage, various network abilities may be remembered.	To diminish over fitting, it needs a great deal of computational determination Large data sample size is needed
Case-based Reasoning (CBR). Sarwar et al., (2012) Maurer et al.	No need for broad maintenance No need for proficiency. Failed cases can be reviewed.	Difficult to collect case data. Restricted forecasts for the items that have been perceived. Needs a large training set (CBR integrated with ANN)
Classification and Regression Trees (CART). (Yohannes and Hoddinott 1999; Yohannes and Webb 1999)	Easy to understand. Can handle misplaced variables.	Comparatively new and slightly unknown. Since CART is a new technique, it is hard to find statisticians with importance expertise in this technique.
K-Nearest Neighbour (K-NN) (Park et al., 2005; Mobasher et al., 2002)	Can handle large data bundles Simple to implement and use Easy to explain estimate	A lot of memory required to store entire information. Time consuming

Furthermore, there are several areas that need to be applied to these classes of ML. The prior work has used ML in diverse domains, like the medical field, on web-based and virtual machines. Therefore, the proposed idea advocates using ML for pre-fetching on MCC to maximize the performance and resolve the existing restrictions of former studies.

Chapter 3

System Model and Problem Formulation

Unlike the traditional cloud-based servicing system which had a lot of latency and inconveniences, nowadays there is a foggy structure introduced which can enhance the response time significantly. In our work, pre-fetching is considered to speed up the access of data in the foggy-type network where frequently accessed data is stored closer to servers. Fig. 3.1 shows the interaction between multimedia cloud and end users in the presence of data pre-fetcher. In traditional cloud based servicing, all the calculation and decisions are made in the cloud itself.

3.1. System Model Architecture and Dynamics

The multi-media cloud consists of *N* clusters, and the master server has a load balancer and computing server. There are *M* number of users sending a service request to either multi-media cloud or pre-fetcher. If all of the requests are directed to multi-media cloud, load balancer splits them over *N* clusters. Computing server computes the likelihood of each request and accordingly some will be transferred to pre-fetcher. Pre-fetcher's functionality is similar to traditional cache memories in terms of maintenance and replacement policies, which in this study adaptive replacement algorithm is being used in particular.

In this technique, as it is shown in Fig. 3.1 when users are sending their requests $(\lambda \text{ requests/sec})$ to the cloud, the master server is the first entity that receives the requests and distributes them among different servers (based on processing cost, processing time, etc.) to process them. Afterward, computing server is responsible for checking if a copy of processed results needs to be stored in pre-fetcher for future similar requests. Otherwise, it sends it back directly to the user. In every stage, whenever the results are achieved either by the multi-media cloud or pre-fetcher, it will reply to the sender. If the answers are not found, pre-fetcher sends an update request to update its database, if necessary. In this model, there are exceptions where the answer is not found in pre-fetcher and after checking with multi-media cloud, yet the likelihood is not enough to pre-fetch and store the information. In this case, a direct response to the sender is more advantageous toward the response time optimization goal.

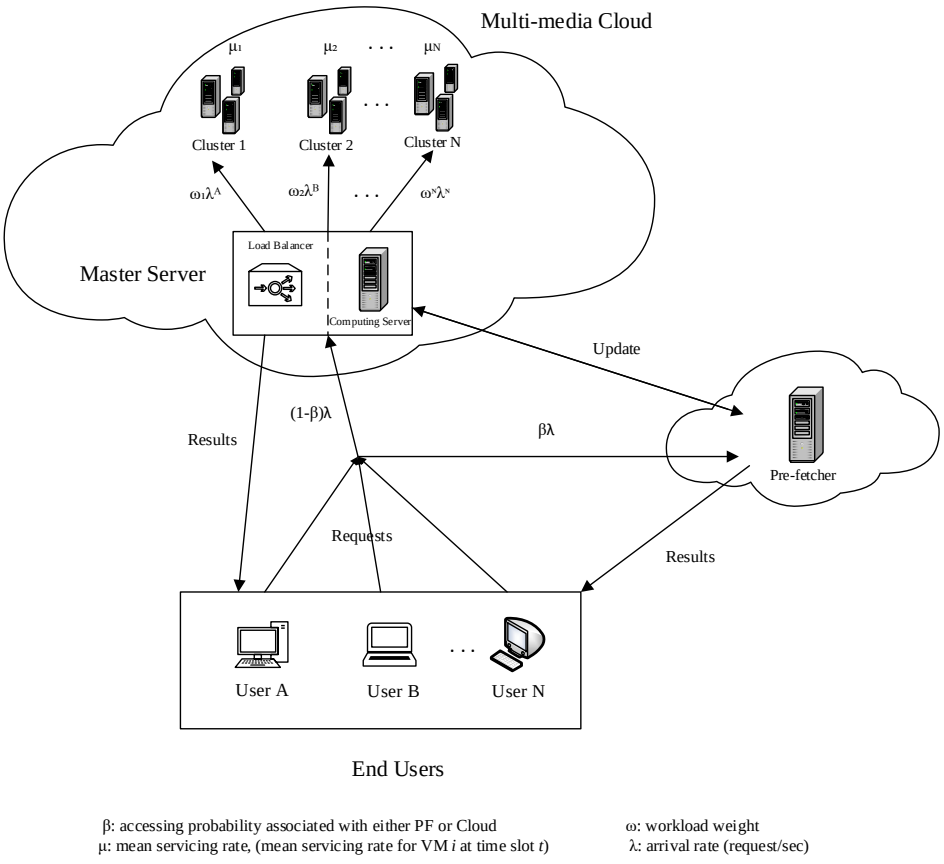


Fig. 3.1 Data transmission between multimedia cloud and end users in the presence of pre-fetcher.

To speed up the response time, we propose a pre-fetching mechanism. In this technique, once the request is processed and is ready to be sent back to the user, it will update pre-fetcher's memory before transmitting data to the user, if needed. Based on that, we expect an extra delay in data transmission at the beginning (e.g. in the learning phase), since cache memory is being filled with the data that in the near future is more likely to be requested. Once the pre-fetcher's memory is filled with necessary data, we expect an increased speed in data transmission since most of the future requests are already stored and exist in the cache memory which results in faster response time, the fair use of resources and lowers the hit rate.

Now if we look closely at the interaction between pre-fetcher, end users and multimedia cloud, there are a few things that need to be taken into consideration: a) How often does the pre-fetcher needs to be updated and what algorithm does it follow? b) How should the pre-fetcher react if a request does not exist in the cache memory? c) Until when and under what conditions has pre-fetching resulted in performance increase? To answer the first question, we begin by introducing the pre-fetcher's caching algorithms.

3.2. Review on Caching

While a processor executes instructions, it requires data from the memory and the system where both data and instructions are stored. However, the main memory is usually very slow. In the time it takes to fetch one piece of data, a modern processor could have executed hundreds of instructions. Obviously, it makes the memory system a significant performance bottleneck. Hence, there are numerous methods introduced to cope with this bottleneck; the most common and effective are caching. As shown in Fig. 3.2, a fast subsystem, the cache, is located between the processor and the main memory. The major task of caching to pursue is to keep the most commonly accessed information in a closer and more accessible location, which is called cache the main component of fog computing networks.

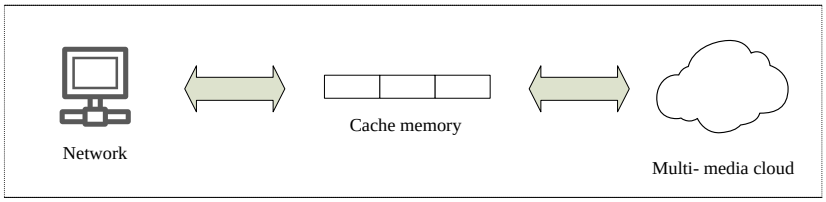


Fig.3.2 A simple caching structure.

Cache generally keeps a copy of the most recently referenced information found in the main memory system. The time it takes to execute an instruction is usually close to the time required to fetch a piece of information from the cache. This boosts up the average memory accessing time while reducing the "memory bottleneck" mentioned above [33].

There are two prominent metrics for measuring the cache performance: hit rate and access time. Hit rate (HR) is simply the percentage of accesses which are cache hits [34], [35]. The access time is the amount of time it takes to service a hit (T_{hit}) or miss (T_{miss}). From these metrics, the average access time of a memory access can be computed. The advantage of direct-mapped caches

is a faster access time (a lower T_{hit}) while they tend to have a lower hit rate (HR). Fully-associative caches usually have longer access times but a higher hit rate. T_{miss} is the time it takes for access to be fulfilled if the cache does not have a copy of the data, so it has very little to do with the cache itself. Rather, it is dependent upon the memory system behind the cache. In fact, T_{miss} can be an important parameter when designing a cache [33]. If T_{miss} is high, increasing the hit rate at the cost of T_{hit} will be more beneficial than if T_{miss} were lower.

In general, a cache consists of two parts, the data part which the information stored in the cache, and the tag part which indicates which address the data came from. Each tag has enough information to recognize from which address the data came. When a memory access occurs, one or more tags are searched and compared to the memory address of the memory access. If a tag matches the address of the memory access, the data associated with that tag is handed to the processor by the cache rather than requiring access to the main memory to get the same data.

The one-to-one relationship between tags and data is illustrated in Fig. 3.3. Ideally, the cache will keep the data that is the most likely to be requested. In practice, however, this is usually not feasible.

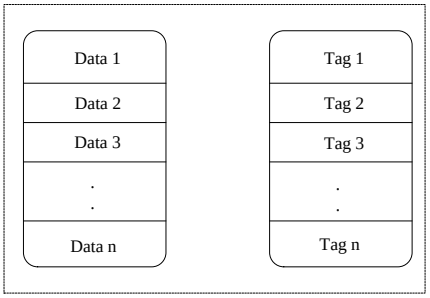


Fig. 3.3 A cache with data and tags.

The cache would need to search through each of the tags to find where the data is in the cache (if it is there at all). Since the whole point of the cache is to speed up access time, the time required to search all of the tags is usually not acceptable. A standard solution to this problem is to assign each memory location a single place in the cache where it can reside. Since the main memory is much larger than the cache, many memory locations share the same assigned place in the cache. This means that only one of those memory locations could be in the cache at any given time. The net result is that flexibility is traded away for speed. This type of cache is called a direct-

mapped cache. A cache where the data may be kept anywhere is referred to as a fully-associative cache.

A compromise between the direct-mapped cache and the fully-associative cache is the set-associative cache. A set-associative cache is broken into sets of locations, all of equal size. Rather than mapping each main memory location to a particular line in the cache, the memory location is mapped to a set of lines. A given main memory location is now restricted to a very small number of locations in the cache. This increases the flexibility of placing data into the cache while still greatly reducing the number of locations which need to be searched. Caches, where the set size is of size n, are referred to as "*n*-way associative caches."

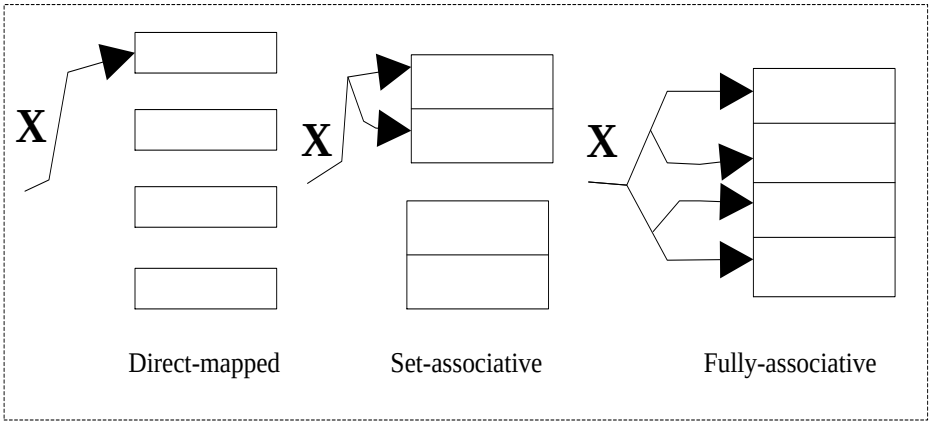


Fig. 3.4 If X is some address, this figure shows the different locations its cache block cloud be placed.

Figure 3.4 illustrates the organization of these three caches. One can think of both a direct-mapped cache and a fully-associative cache as extremes of a set-associative cache.

Numerous types of caching have been introduced, and each of them has its own pros and cons. In this case of study, we consider fully-associative mapping.

Prefetching technology is a widely-used computer system technology, which hides system expands at every delay of the computer system. The main purpose of using prefetching technology is to predict future needs by history records, which can effectively increase response speed of user requests. The indicators of performance inspection include prefetching accuracy rate and prefetching loss rate.

Prefetching accuracy rate shows the degree of prefetching algorithm occupied system resources, and the resources include computing resource and memory resource. Prefetching loss

rate indicates the results of pre-fetching, and the major factor that affects prefetching loss rate is visit file alignment.

There are three ways to get data sets, pre-fetch by user behaviours, pre-fetch by needs and collaborative pre-fetch [36]. User behaviours prefetching mines user visits alignment patterns by analysing user history records, the pattern was fed back to the user model, and the candidate data sets were decided by visit alignment pattern. Prefetching based on needs can determine candidate data sets of a new user, by user frequent access information of service model. At last, collaborative prefetching filters the candidate prefetching data sets and gets prefetching data sets. Based on user behaviour, obtaining candidate prefetching data sets is decided by current data set of users needed and data set of 2 kinds of sequential patterns predicted to visit, which was obtained by the offline learning algorithm. Prefetching algorithm wastes some network resources, and it cannot reflect the urgency of prefetching needs. To reduce network resources, we use online learning algorithm separating data sets into urgent alignment and not urgent alignment by the same time interval of the pattern. If a data set is in the urgent alignment, it will be visited and pre-fetched immediately. When the urgent alignment is empty, or the network load is low, system pre-fetches the data sets in the not urgent alignment.

In this research, we are using collaborative pre-fetching since, splitting large data to smaller chunks and tagging them based on urgency will be easier to work with and process as well as having less overhead. The splitting and decision making is done in computing server located in multi-media cloud.

3.3. Replacement Policies

If the cache is full and a new block needs to be put into the cache, some other block must be evicted. Clearly, the block that is to be evicted must be from the same set as the new block. In a direct-mapped cache, there is no decision which needs to be made—there is only one block to choose from. However, in a set-associative or fully-associative cache, some decisions must be made [36], [37].

3.3.1. A-Least Recently Used (LRU) Algorithm

One of the favourite cache replacement algorithms for researchers is LRU algorithm. It uses a principle called temporal locality. Based on timestamp it replaces the page which has not

been used for an extended period of time. This algorithm is useful only for the conventional system rather than for Internet services because while taking replacement decisions it takes into consideration the time since last access and not access frequency. In Internet services, different user requests need to be handled which are not inter dependent and have different access characteristics hence optimal performance is not obtained in the case of Internet services. LRU is straightforward and easy to use in the conventional system has constant time and space overhead, and captures regency or clustered locality of reference. In the case of cloud, it does not provide optimal performance. There is a Lock contention problem, to move a page to a most recently used position on every hit it incurs, an overhead which is unacceptable, and access frequency is not taken into consideration. By scanning, LRU can be easily polluted.

3.3.2. B-Least Frequently Used (LFU) Algorithm

In LFU, history of accesses is used for predicting the probability of upcoming references. The frequency of access counts of all the lines is maintained by the cache, and whenever there is a new access, cache replaces the line which has been used least frequently. In Internet environment, if LFU is used there is a higher chance that one or the other document will always be replaced by a new document with high frequency without the considerations of the access probabilities of the replaced and the new documents. When videos and other such large documents are obtained they fill out the cache completely which might destroy the proper functioning of the cache. LFU is simple and easy to use in the conventional system. LFU does not offer optimal performance for Internet systems.

3.3.3. C-First in First Out (FIFO) Algorithm

The system maintains a list of all the pages in the memory where the first page at the head is the oldest one whereas page at the end is the recent page being added. Whenever there is a request for a page, and a miss occurs, the system removes a page from the head of the list and adds the new page at the end of the list. This process is known as First in First out (FIFO). But this method fails if it removes the data which is important but not recently used. Hence, FIFO is rarely used in its pure form. Low-overhead paging algorithm. It is not very effective. The system needs to keep track of each frame. Sometimes, it behaves abnormally. This behaviour is called Beladys

anomaly and might throw out important pages. Same problems are faced in case of Internet systems.

3.3.4. D-Adaptive Replacement Algorithm

The adaptive replacement cache algorithm, keeps track of both frequently used pages and recently used pages and maintains a balance between the features of workloads. Workloads contain changing frequency, temporal locality, sequential I/Os, etc. ARC is easy to implement, and its running time per request is substantially independent of the cache size. As compared to LRU it has a performance gain with respect to hit ratio for variable cache sizes. It is a low-overhead algorithm that responds online to changing access patterns. Features like regency and frequency of workload are exploited in a self-tuning fashion, time and space complexity is low, but for a broad range of workload and cache size, it outperforms LRU. It has a cache hit serialization problem.

Table 3.1 Comparison of Cache Replacement Policies [37].

Order	Replacement Policy	Structure	Small Cache Size			Large Cache Size		
			Hit Ratio	Byte Hit Ratio	Cost Reduction	Hit Ratio	Byte Hit Ratio	Cost Reduction
A	Last Recently Used(LRU)	It works on the timestamp. Last recently used page is replaced on a page fault.	✔	✘	✘	✘	✘	✘
B	Last Frequently Used(LFU)	Uses the access history to predict the probability of subsequent reference.	✔	✘	✘	✘	✘	✘
C	First in First Out(FIFO)	Used FIFO method as in queue.	✔	✘	✘	✘	✘	✘
D	Adaptive Replacement Algorithm	Keep track of both frequently used pages and maintain a balance between the features of workload.	✔	✔	✘	✔	✔	✘

In section b, we have analysed different cache replacement policies and in this section comparative study will be presented. Various policies and their performance metric parameters are listed in Table 3.1.

From the table, it can be understood that, all the mentioned cache replacement algorithms achieve significant improvement in hit ratio when the cache size is small, whereas, only ARC achieves improvement in hit ratio when the cache size is large. ARC gives significant improvement for both small as well as large cache size in the byte hit ratio (BHR).

3.4. Problem Formulation & Analysis

Multi-media processing enforces new challenges to cloud computing, according to attributes like being delay sensitive, high computation intensity and significant bandwidth demands. Accordingly, we designed a model Fig. 3.1 to enhance the multimedia-end user communication process. Introduced model consists of three main components: *Multi-media Cloud, Pre-fetcher and End Users*. Multi-media cloud has a core which is called *Master Server*. Master server receives the portion of the workload sent to multi-media cloud. *Load Balancer* is responsible to distribute the workload over *N* server clusters for processing. *Computing Server* is the intelligent core of master server which calculates the probability of the next call on the same request and sends a copy to pre-fetcher.

Each server cluster receives a portion of the total workload and processes it with different processing speed until the entire workload is processed. Then computing server gradually fills the *storage clusters* in pre-fetcher by sending a copy of the critical data for next call.

Pre-fetcher contains storage clusters to store data which is more likely to be requested next. Once the memory is full, it will be overridden based on Adaptive Replacement Algorithm. There are *M* number end users (providing tests) considered in the model, and the probability of end users reaching pre-fetcher is $\beta\lambda$ versus the probability of reaching multi-media cloud which is $(1-\beta)\lambda$. The more load sent to pre-fetcher, the more overhead and delay expected. The expected delay will be discussed and optimized later in this chapter. However, it cannot be reduced to zero if the use of the pre-fetcher is incorporated with processing.

3.4.1 Problem Definition

In this section, we introduce and formulate two main problems (Response time and Cost) in a combined pre-fetcher/cloud system and will discuss the simulation results in next chapter. In order to understand the problem better, parameters and variables are defined in the Table 3.2. Furthermore, we explain each parameter and talk about their role in the formula.

Variable	Description
M	number of given tasks
N	number of VMs
K	maximum number of paths
i	VM indexing indicator
j	task indexing indicator
k	path indexing indicator
β	accessing probability associated with pre-fetcher
λ	arrival rate (requests/sec)
μ	mean servicing rate, μ_i (mean servicing rate for VM i)
γ	processor's speed multiplier
α	allocation index, α_{ij} (allocation index between VM i and task j) for pre-fetcher
θ	allocation index, θ_{ij} (allocation index between VM i and task j) for cloud
ω	workload weight to VM i
$g(\gamma)$	cost function(delay) associated with pre-fetcher
C	constant delay value in cost function
m	pre-fetcher processor's speed increment indicator in the cost function
m'	pre-fetcher usability increment indicator in the cost function

Table 3.2. Variable definition.

Here, we are introducing each variable and the use of it in our proposed model. The intraction of these variables will be discussed in future chapters.

M : It is associated with the number of the tasks scheduled in each path, while *j* is used for indexing.

N : It is the number of virtual machines (VMs), where *i* is used for indexing the VM.

K : Indicates the maximum number of paths where *k* donates path index.

β : In the proposed system model shown in Fig. 3.1, we introduce a probability (or presence of time) of using the pre-fetcher, while accordingly $(1-\beta)$ is the probability that the workload is sent to multi-media cloud.

λ : Donates the requests arrival rate to the system.

γ : It is the pre-fetcher processor's speed multiplier, where $\gamma > 1$ and $\mu_{i,PF} = \gamma\ \mu_{i,Cloud}$. Here, we assume that the pre-fetcher works faster than the cloud processor.

μ : It is the mean servicing rate, where μ_i donates the mean servicing rate for $\mathrm{VM_i}$.

θ : It is the VM allocation indicator, where θ_{ij} donates the allocation index between $\mathrm{VM_i}$ and task *j*. That is,

$$\theta_{ij} = \begin{cases} 1, \text{ if } \mathrm{VM}_i \text{ is allocated to task } j, \\ 0, \text{ otherwise.} \end{cases} \tag{3.1}$$

ω : Donates the workload weight with a constraint that ω_i is workload weight associated with $\mathrm{VM_i}$ and the summation of workload weight has to be equal to 1, and it means that all the workload is completed by the processors.

$g(\gamma)$*:* Introduced cost function with interpreted as processing delay in our pre-fetcher in the system model.

C: Fixed value or a constant multiplier that is directly associated with the delay.

m : Processor speed increment indicator, as it is increasing, cost function increases exponentially.

m'*:* Pre-fetchers usability probability indicator, as it is increasing, cost function increases exponentially.

3.4.2. Response Time Minimization Problem and Analysis

The mean service rate from all the paths will be $\sum_{i=1}^{N}\sum_{j=1}^{M}\theta_{ij}\mu_{ij}$. The service process of each path follows an M/M/1 queuing system [38]. The average response time of path k can be calculated as:

$$T_k = \sum_{i=1}^{N} \frac{\omega_k}{\sum_{i=1}^{N}\sum_{j=1}^{M}\theta_{ij}\mu_i - \omega_k\lambda_i}. \tag{3.2}$$

The optimization problem for workload scheduling of each server can be formulated as [38], [39]:

$$\underset{\omega_k,\theta_{ij}}{\text{Minimize}}: T_{Cloud} = \sum_{k=1}^{K} T_k = \sum_{k=1}^{K}\sum_{i=1}^{N} \frac{\omega_k}{\sum_{i=1}^{N}\sum_{j=1}^{M}\theta_{ij}\mu_i - \omega_k\lambda_i}, \tag{3.3}$$

$$\text{Subject to: } \theta_{ij} = \{1,0\}, \forall i \in \{1,2,...,N\}, \forall j \in \{1,2,...,M\},$$

$$\text{C1: } \sum_{j=1}^{M}\theta_{ij} = 1 \quad \forall i \in N,$$

$$\text{C2: } \sum_{i=1}^{N}\theta_{ij} \geq 1 \quad \forall j \in M,$$

$$\text{C3: } \sum_{k=1}^{K}\omega_k = 1 \quad \forall i \in N,$$

$$C4: \omega_k\lambda_i \leq \sum_{i=1}^{N}\sum_{j=1}^{M}\theta_{ij}\mu_i.$$

The objective of the workload scheduling problem is to minimize the total mean response time of the system. Constraint, C1 and C3 illustrate that at time instance *t*, each VM serves only one task whereas constraint, C2 and C4 indicate that one task needs more than one VM. Constraint C5 is the workload conservation constraint. The constraint C6 is the queuing stability constraint.

Total mean response time can be formulated as:

$$T_{Total} = (1-\beta)T_{Cloud} + (\beta)T_{PF} \tag{3.4}$$

If β is the probability (or percentage of time) of using pre-fetcher in time slot *t*,

$$\beta = \begin{cases} 1, \text{ if only pre-fetcher is used,} \\ 0, \text{ if only cloud is used,} \\ > 0 \text{ and } < 1 \text{ otherwise.} \end{cases} \tag{3.5}$$

Pre-fetcher´s mean servicing time T_{PF} can be formulated as:

$$T_{PF} = \sum_{k=1}^{K} \frac{\omega_{k,PF}}{\sum_{i=1}^{N}\sum_{j=1}^{M} \alpha_{ij}\mu_{i,PF} - \omega_{k,PF}\lambda_{i,PF}}. \tag{3.6}$$

The optimization problem for workload scheduling of each server in the presence of pre-fetcher can be formulated as:

$$\underset{\omega_k,\theta_{ij},C_{PF}}{\text{Minimize}}: T_{Total} = (1-\beta)T_{Cloud} + \beta T_{PF} \tag{3.7}$$

$$\underset{\omega_k,\theta_{ij},\alpha_{ij}}{\text{Minimize}}: T_{Total} = (1-\beta)\sum_{k=1}^{K}\sum_{i=1}^{N} \frac{\omega_{k,Cloud}}{\sum_{i=1}^{N}\sum_{j=1}^{M}\theta_{ij}\mu_{i,Cloud} - \omega_{k,Cloud}\lambda_{i,Cloud}} + \beta\sum_{k=1}^{K} \frac{\omega_{k,PF}}{\sum_{i=1}^{N}\sum_{j=1}^{M}\alpha_{ij}\mu_{i,PF} - \omega_{k,PF}\lambda_{i,PF}},$$

Subject to: $\theta_{ij} = \{1,0\}, \alpha_{ij} = \{1,0\}, \forall i \in \{1,2,...,N\}, \forall j \in \{1,2,...,M\}, \ \beta = [0,1]$

$\forall i \in \{1,2,...,N\}, \ \text{k} = \{1,2,3,...,K\}, \ \gamma > 1,$

C1: $\sum_{j=1}^{M}\theta_{ij} = 1 \quad \forall i \in N,$

C2: $\sum_{i=1}^{N}\theta_{ij} \geq 1 \quad \forall j \in M,$

C3: $\sum_{j=1}^{M}\alpha_{ij} = 1 \quad \forall i \in N,$

C4: $\sum_{i=1}^{N}\alpha_{ij} \geq 1 \quad \forall j \in M,$

C5: $\sum_{k=1}^{K}\omega_{k,Cloud} = 1 \quad \forall i \in N,$

C6: $\omega_{k,Cloud}\lambda_{i,Cloud} \leq \sum_{i=1}^{N}\sum_{j=1}^{M}\theta_{ij}\mu_{i,Cloud},$

C7: $\sum_{k=1}^{K}\omega_{k,PF} = 1 \quad \forall i \in N,$

C8: $\sum_{i=1}^{N}\mu_{i,PF} > \sum_{i=1}^{N}\lambda_{i,PF},$

C9: $\mu_{i,PF} = \gamma\ \mu_{i,Cloud}.$

Similar to Eq. (3.4), the objective of workload scheduling with the pre-fetcher is to minimize the total mean service response time [40], [41]. Constraint C5 is the cloud workload conservation constraint, C6 is the queuing stability constraint associated with cloud. C7 is the workload conservation constraint related to pre-fetcher, Constraint C8 is pre-fetcher´s queuing stability constraint, and C9 shows the processing capacity the relation between the processor in multimedia-cloud and pre-fetcher. Since pre-fetcher needs to process the requests faster, it is γ times faster than the processor in multimedia-cloud. Pre-fetcher requests arrives with probability of $(1-\beta)$. This problem is a complex problem and not easy to solve with regular optimization methods such as branch and bound, linear programming relaxations, etc. Hence, we choose the heuristic algorithm to solve the problem. We developed Algorithm 3.1 for pre-fetcher and Algorithm 3.2 for multi-media cloud.

3.4.3. Heuristic Approach

A heuristic approach, is any approach to problem solving, learning, or discovery that employs a practical method not guaranteed to be optimal, but satisfactory for the instantaneous goals. Where finding an optimal solution is impossible or impractical, heuristic methods can be used to speed up the process of finding an acceptable solution. Heuristics can be mental shortcuts that ease the cognitive load of making a decision. Examples of this method include using a rule of thumb, an educated guess, an intuitive judgment, guesstimate, stereotyping, profiling, or common sense.

Many optimization problems in computer science have been proven to be NP-hard, and it is unlikely that polynomial-time algorithms solve these problems. Alternatively, they are solved using heuristics algorithms, which provide a sub-optimal solution that, hopefully, is arbitrarily close to the optimal one. Such problems are found in a wide range of applications, including artificial intelligence, game theory, graph partitioning, database query optimization, etc. Since our response time problem is known as a convex mixed-integer problem, it is not easy to solve with mathematical and theoretical solutions. Hence, we propose a heuristic (exhaustive search) approach to optimize the response time and cost problems as follows. We also proposed an optimal scheme later to compare and analyse the performance between two approaches.

Algorithm 3.1: Pre-fetcher

Input: $\mu_{PF}, \mu_{Cloud}, \alpha_i$, Number of VMs

Output: $\beta, \alpha_{i,Cloud}, T_{PF}$

1: Compute $\gamma = \frac{\mu_{PF}}{\mu_{Cloud}}$ which is the cost of using one class-*i* VM to process one unit request, respectively. Let set $\gamma = \{1, 2, 3, ..., Z\}$.

2: Sort set γ in ascending order.

3: **repeat**

4: Select the smallest γ from the γ set.

5: **if** $\sum_{i=1}^{N} \mu_{i,PF} > \sum_{i=1}^{N} \lambda_{i,PF}$ is satisfied.

6: Calculate Bayesian weight of each path ($\omega_{k,PF}$) from [1] **then**

7: Check the constraint $\sum_{k=1}^{K} \omega_{k,PF} = 1$.

8: Compute $\beta = \left(\frac{A - A'}{(m'+1) C \gamma^{m}} \right)^{1/m'}$.

9: Reserve the $i - t_n$ class VM and schedule the workload among VM.

10: **end if.**

11: **until** all the VMs are processed.

12: Compute the remaining arrival rate from Cloud $\alpha_i - (1-\beta)\alpha_{i,PF} = \alpha_{i,Cloud}$.

13: Compute time/cost = T_{PF} .

14: **Return** $\beta, \alpha_{i,Cloud}, T_{PF}$.

Algorithm 3.2: Multi-media Cloud

Input: $\alpha_{i,Cloud}, \beta, \mu_{Cloud}, T_{PF}$

Output: Time/Cost: T_{Total}

1: Sort $i - t_n$ Class VM of accessing order of cost.

2: **repeat**

3: **if** $\sum_{i=1}^{N} \mu_{i,Cloud} > \sum_{i=1}^{N} \lambda_{i,Cloud}$ is satisfied,

4: Compute Bayesian/Optimum weight $\omega_{k,Cloud}$ from [1] **then**

5: **until** all requests are processed.

6: Check the constraint $\sum_{k=1}^{K} \omega_{k,Cloud} = 1$.

7: Reserve the $i - t_n$ class VM & schedule the workload among VM.

8: **End if**

9: **until** all the remaining request ($\alpha_{i,Cloud}$) are processed.

10: Compare total time $T_{Total} = T_{Cloud} + T_{PF}$.

11: **Return** T_{PF}

As it was discussed previously, heuristic approach was proposed and algorithms are provided for cloud (Algorithm 3.1) and pre-fetcher (Algorithm 3.2). In every algorithm, there are several steps to be followed in order to achieve the optimized solution. In Chapter 4, we study the performance analysis on the produced results. Additionally, after providing the simulation results, we analyse how different factors interact and impact each other.

3.4.4. Cost Problem and Analysis

According to our system model in Fig. 3.1, there is delay associated with pre-fetcher's functionality and in 3.8 it is introduced as $T_{\$}(\text{sec})$. This delay is a time frame that pre-fetcher requires to react towards the received request. This delay is considered as a cost in our problem formulation since the units are in milliseconds [42]. The cost problem for workload scheduling presence of pre-fetcher can be formulated as follows in (3.8).

$$T_{Total} = T_{QoS} + T_{\$} \tag{3.8}$$

$$T_{\$} = \beta g(\gamma) = \beta C \gamma^{m} \beta^{m'},\ C > 0$$

$$\underset{\omega_k \theta_{ij} \alpha_{ij}}{\text{Minimize}}: T_{Total} = (1-\beta)\sum_{k=1}^{K}\sum_{i=1}^{N}\frac{\omega_{k,Cloud}}{\sum_{i=1}^{N}\sum_{j=1}^{M}\theta_{ij}\mu_{i,Cloud} - \omega_{k,Cloud}\lambda_{i,Cloud}} + \beta\left(\sum_{k=1}^{K}\sum_{i=1}^{N}\frac{\omega_{k,PF}}{\sum_{i=1}^{N}\sum_{j=1}^{M}\alpha_{ij}\mu_{i,PF} - \omega_{k,PF}\lambda_{i,PF}} + C\gamma^{m}\beta^{m'}\right),$$

$$\text{Subject to: } \theta_{ij} = \{1,0\}, \alpha_{ij} = \{1,0\}, \forall i \in \{1,2,...,N\}, \forall j \in \{1,2,...,M\},\ \beta = [0,1]$$
$$\forall i \in \{1,2,...,N\},\ \text{k}= \{1,2,3,...,K\},\ \gamma > 1,\ \text{C} > 0.$$

$$\text{C1: } \sum_{j=1}^{M}\theta_{ij} = 1 \quad \forall i \in N, \qquad C6: \ \omega_{k,Cloud}\lambda_{i,Cloud} \le \sum_{i=1}^{N}\sum_{j=1}^{M}\theta_{ij}\mu_{i,Cloud},$$

$$\text{C2: } \sum_{i=1}^{N}\theta_{ij} \ge 1 \quad \forall j \in M, \qquad \text{C7: } \sum_{k=1}^{K}\omega_{k,PF} = 1 \quad \forall i \in N,$$

$$\text{C3: } \sum_{j=1}^{M}\alpha_{ij} = 1 \quad \forall i \in N, \qquad \text{C8: } \sum_{i=1}^{N}\mu_{i,PF} > \sum_{i=1}^{N}\lambda_{i,PF},$$

$$\text{C4: } \sum_{i=1}^{N}\alpha_{ij} \ge 1 \quad \forall j \in M, \qquad \text{C9: } \mu_{i,PF} = \gamma\ \mu_{i,Cloud},$$

$$\text{C5: } \sum_{k=1}^{K}\omega_{k,Cloud} = 1 \quad \forall i \in N, \qquad \text{C10: } 0 < \text{T}_{Cloud-}\text{T}_{PF} < (m'+1)C\gamma^{m}.$$

Similarly to previous time problem formulation and inspired from what Nan did in [38], [39], we have constraint C1 and C3 referring to time instance *t*, each VM serves only one task whereas constraint C2 and C4 demonstrates that one task needs more than one VM. Constraint C5 is the cloud workload conservation constraint, C6 is the queuing stability constraint associated with cloud. C7 is the workload conservation constraint related to pre-fetcher, constraint C8 is pre-fetcher´s queuing stability constraint, and C9 shows the processing capacity relation between the

processor in multimedia-cloud and pre-fetcher. Since pre-fetcher needs to process the requests faster, it is γ times faster than the processor in multimedia-cloud. C10 is the cloud and pre-fetcher's response time difference threshold constraint that ensures that the difference is always a positive number and less than a maximum. Accordingly, workload probability for pre-fetcher (β) in our cost optimization problem can be analytically found as follows:

$$
\begin{aligned}
&T_{Total} = T_{QoS} + T_{\$} \\
&T_{QoS} = (1-\beta)T_{Cloud} + \beta T_{PF} \\
&T_{\$} = \beta\left(C\gamma^{m} f(\beta)\right)
\end{aligned}
\qquad (3.9)
$$

$$
\overrightarrow{Assume}\ f(\beta) = \beta^{m'}
$$

$$
A = \sum_{k=1}^{K}\sum_{i=1}^{N} \frac{\omega_{k,Cloud}}{\sum_{i=1}^{N}\sum_{j=1}^{M}\theta_{ij}\mu_{i,Cloud} - \omega_{k,Cloud}\lambda_{i,Cloud}} \quad C > 0
$$

$$
A' = \sum_{k=1}^{K}\sum_{i=1}^{N} \frac{\omega_{k,PF}}{\sum_{i=1}^{N}\sum_{j=1}^{M}\alpha_{ij}\mu_{i,PF} - \omega_{k,PF}\lambda_{i,PF}} \quad , \quad 0 \le \beta \le 1
$$

$$
\begin{aligned}
\overrightarrow{Then}\ T_{Total} &= (1-\beta)A + \beta\left(A' + C\gamma^{m}\beta^{m'}\right), \\
&= (1-\beta)A + \beta A' + C\gamma^{m}\beta^{m'+1},
\end{aligned}
$$

, when $A > A'$

$$
\partial T \big/ \partial \beta = -A + A'(m'+1)C\gamma^{m}\beta^{m'} = 0,
$$

$$
\Rightarrow \quad \beta = \left(\frac{A - A'}{(m'+1)C\gamma^{m}}\right)^{1/m'}.
$$

In our assumption, initially, we consider a function of β to relate the cost problem to the probability of received workload in pre-fetcher, as well as replace the cloud's and pre-fetcher's objective functions with A and A' to simplify (3.9). The optimum β , (that is the percentage of traffic sent through pre-fetcher) will have different values when the other parameters are changing.

Chapter 4

Simulation Results & Performance Evaluation

In this chapter, we perform a number of simulations on the previously proposed optimization problem to evaluate the performance of the results compared to the previous model [1]. All simulations are written and done in Matlab scripts. In order to discuss and analyse, the results are categorized based whether the values of γ and β are fixed or changing. γ is the speed indicator in pre-fetcher's processor and β is the distribution probability of received workload in pre-fetcher and accordingly $(1-\beta)$ is the probability of workload reaching to multi-media cloud in our system model. In Table 4.1, variables and their value/range used in our Matlab simulation are shown as follow.

Table. 4.1 Simulation setup and variable value/range.

Variable	Symbol	Value/Range
PF processing speed multiplier	γ	1-50
PF utilization	β	[0,1]
Arrival rate (requests/sec)	λ	1000-9000
Processing delay (sec)	C	1×10^{-4}
PF processing cost factor	m	[0.2,0.25]
PF utilization cost factor	m'	1
Processing speed (sec)	μ	[10,20,30]
Number of clusters	N	3

4.1. Response Time

In this part of our study, the primary objective is the total response time minimization. Hence, we performed simulations while changing different variables to see how pre-fetching scheme is advantageous towards the goal. The main variables in our model are β, γ and λ. Accordingly, we discuss each scenario individually and analyse the results.

4.1.1. Response time study with fixed γ and changing β

In this section, after getting the results through performing simulations, we analyse them. We compare the generated result after implementation of pre-fetcher $(\beta = 1)$ and the former result where pre-fetcher was not considered $(\beta = 0)$, where processing speed in pre-fetcher is assumed to be fixed $(\gamma = 4)$ which is four times faster than processing time in multi-media cloud. This comparison is made while we have response time (sec) on Y axis and arrival rate (requests/sec) on the X axis.

As expected, the response time in the results is ascending, while using pre-fetcher benefits us by saving a significant amount of time in the long run. This means, instead of financially investing on data transmission, a portion of the investment can be done to improve and maintain the processor in pre-fetcher.

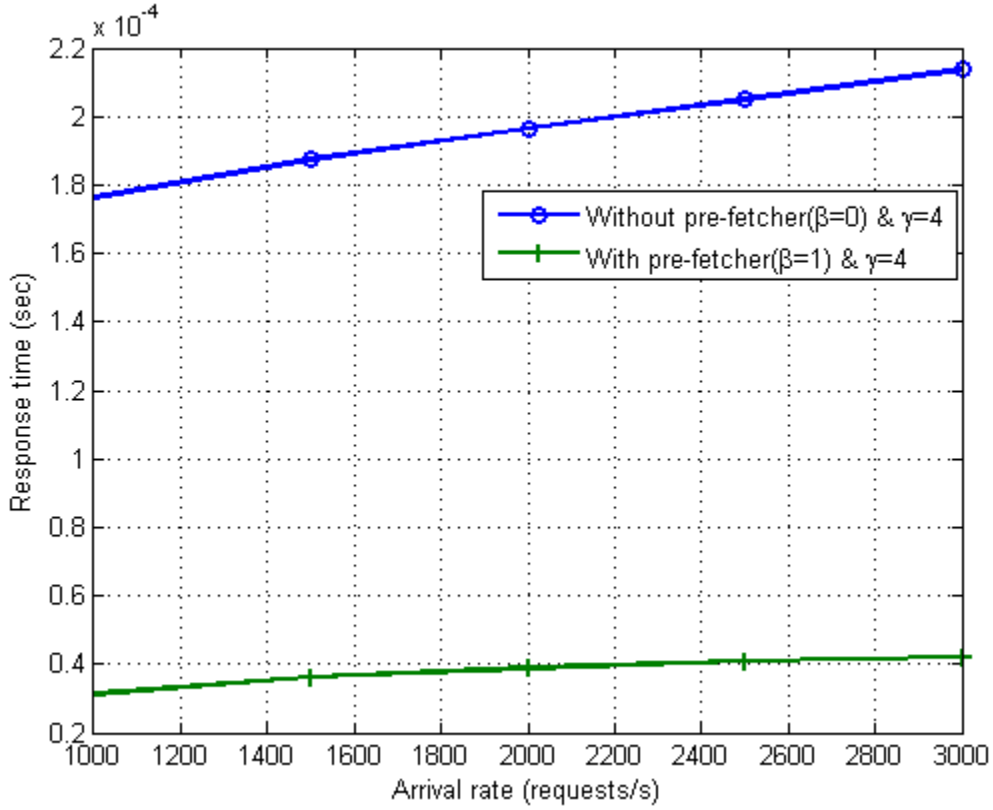


Fig. 4.1 Response time vs. arrival rate in solo use of the pre-fetcher or multi-media cloud.

As presented in Fig. 4.1, the top curve represents the response time of multi-media cloud based on the arrival rate without considering pre-fetcher. According to Fig. 4.1, as the number of the requests/sec increases, we have a slight elevation in the curve which means requests will need a bit of extra time to be processed and sent back to the source of the request. The curve with lower response time represents the solo use of pre-fetcher without considering any load being shared or sent to multi-media cloud. The results show that there is a remarkable performance enhancement in response time when only pre-fetcher is responding to requests. The reason behind the performance enhancement is because the requests already exist in the memory of the pre-fetcher and they just need to be located and sent back, versus in multi-media cloud, there is an extra step required which is computing the request. Therefore, their response process in the pre-fetcher is significantly faster. In Fig. 4.2, metrics and increment units are considered to stay the same similar to the previous figure. The value of γ is also is fixed and deemed to be four while β is changing.

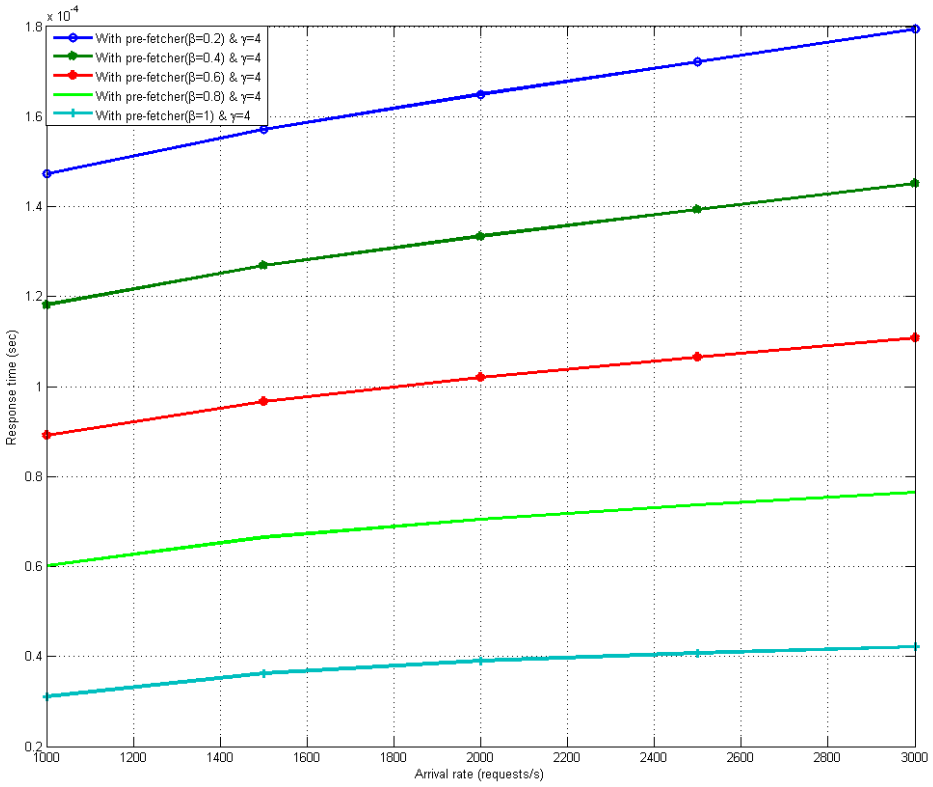


Fig. 4.2 Response time vs. arrival rate while the workload is shared between multi-media cloud and pre-fetcher at probability distribution of β.

In Fig. 4.2, we increase the workload distribution probability from 0 to 1 for 0.2 units in every increase to show how it would affect the response time as we move toward pushing more workload to pre-fetcher and less to multi-media cloud. As expected, in every increase share of pre-fetcher, the response time drops, and it is minimum in the purple curve when $\beta = 1$. All the results at the results shown there are with the assumption of $\gamma = 4$, which means processor in pre-fetcher is running four times faster than the in multi-media cloud.

4.1.2. Response time study on fixed β and changing γ

In this part of the simulation, we study the effect of varying distributed assigned workload when both processors in multi-media cloud and pre-fetcher are running at the same speed. As it is presented and expected, there is no preference in terms of choosing cloud over pre-fetcher or vice versa. As it can be interpreted from our problem formulation, our simulated pre-fetcher is advantageous where it is running at faster processing speed compared to multi-media cloud. However, in Fig. 4.3 we simulated a situation where both cloud and pre-fetcher are running at the same processing speed, to show that there is no difference in total response time. In other words, having a pre-fetcher does not help if its processing speed is at the same level as cloud. It is also shown, that regardless of whether we increase the pre-fetcher utilization factor, the increase is still linear and at all various utilization levels there is no difference between using cloud and pre-fetcher.

Even though, increasing processing speed in pre-fetcher decreases response time, it also increases the cost (processing delay) associated with processing time. The delay, is also considered and optimized using proposed heuristic and optimal. Results are shown and compared later in this chapter. Initially, we perform simulations on pre-fetcher with considering the impact of delay and later, after considering that impact of cost (processing delay) we compare the results and show how an integrated model using pre-fetcher and cloud computing is beneficial towards minimization of total response time. Having said that, in our model we only study the dynamic communication of multi-media cloud, pre-fetcher and end users. We will also find the optimal β for different situations, since it changes based on circumstances from time to time.

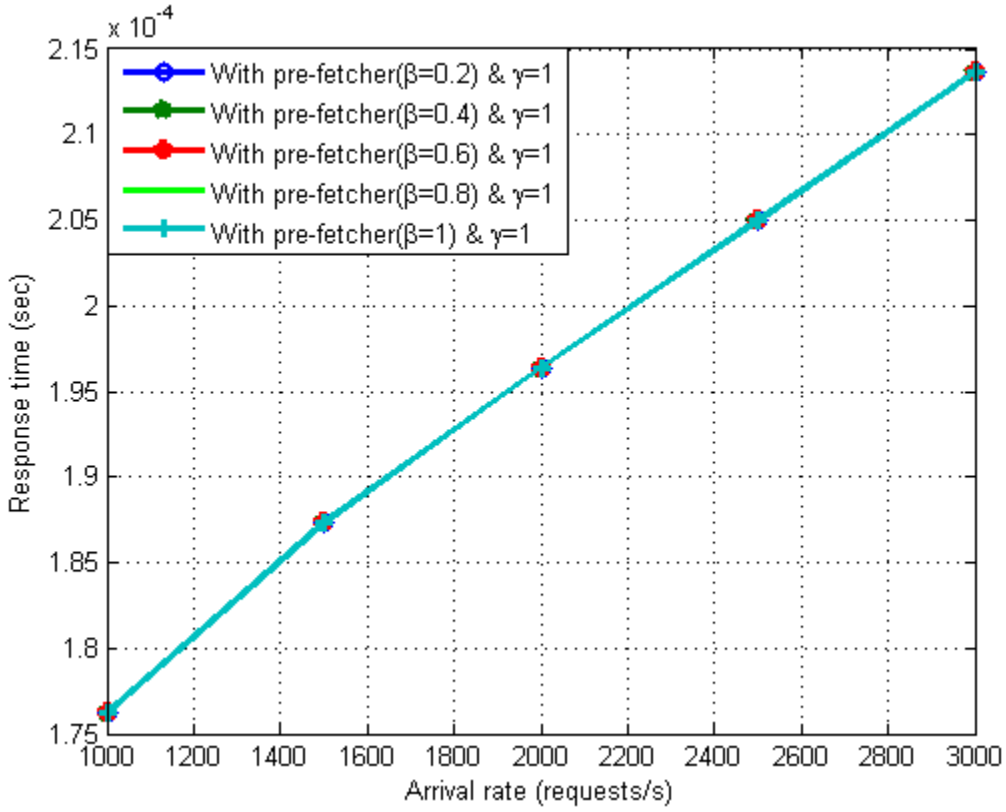


Fig. 4.3 Response time vs. arrival rate while the workload is shared between multi-media cloud and pre-fetcher at probability distribution of β .

In Fig. 4.3, we can see that all curves are aligned with the same slope on the top of each other and ascending as more requests arrive per second. It is understandable that among all simulation results, it is least beneficial while pre-fetcher is considered to be running and is processed at the speed level as a cloud. In this case, not only is it not beneficial time-wise but it is a waste of resources and energy and also needs maintenance. In practice, this scenario is mostly used for load balancing purposes and it is only shown to simplify the comparison.

In Fig. 4.4, all different situations are shown next to each other from (a) to (d). We can see that in all the curves, trend increment is almost linear. As arrival rate increases, all the curves are slightly shifting down which means lower response time and efficiency towards our goal. As presented in the comparison, we investigate four different scenarios in which we study the pre-fetcher processing speed impact on response time mainly while the arrival rate is increasing.

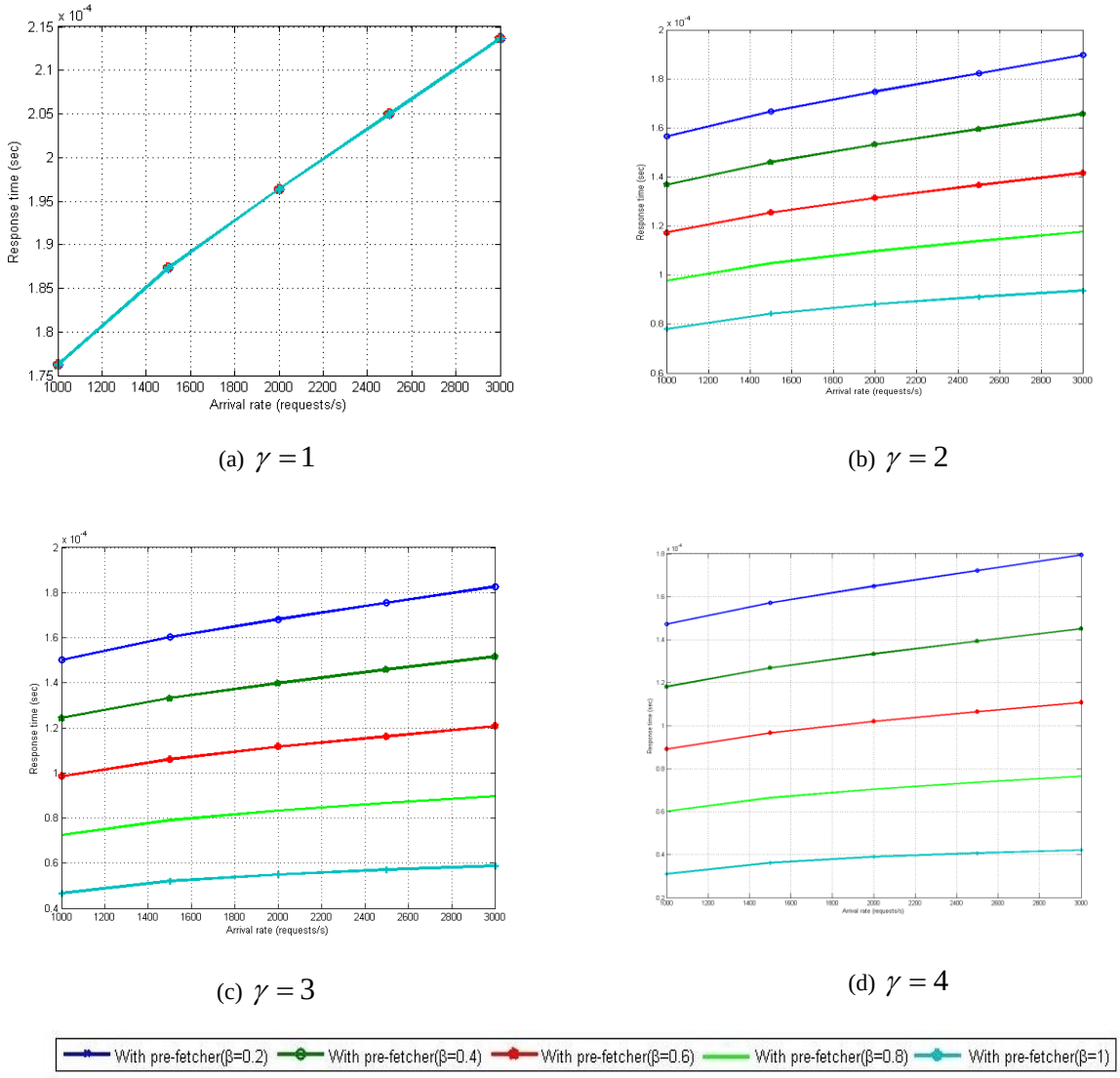


Fig. 4.4 Response time comparison when γ is fixed at different speeds while β and λ are changing.

As shown in Fig. 4.4, we can compare the decrease of each curve's total response time as we move from (a) towards (d). As we increase the arrival rate, all curves have a slight increase, but as a result of increasing pre-fetcher's processor power, they have faster response time. In (a) for instance, the lowest curve $(\beta = 1)$ response time, starts from almost 1.77ms while in (b), it decreases to approximately 0.98ms, in (c) it is 0.72ms and in (d) is it 0.6. Unlike (a) where there is no preference in using cloud over pre-fetcher, the trend of all (b), (c) and (d) is ascending and identical.

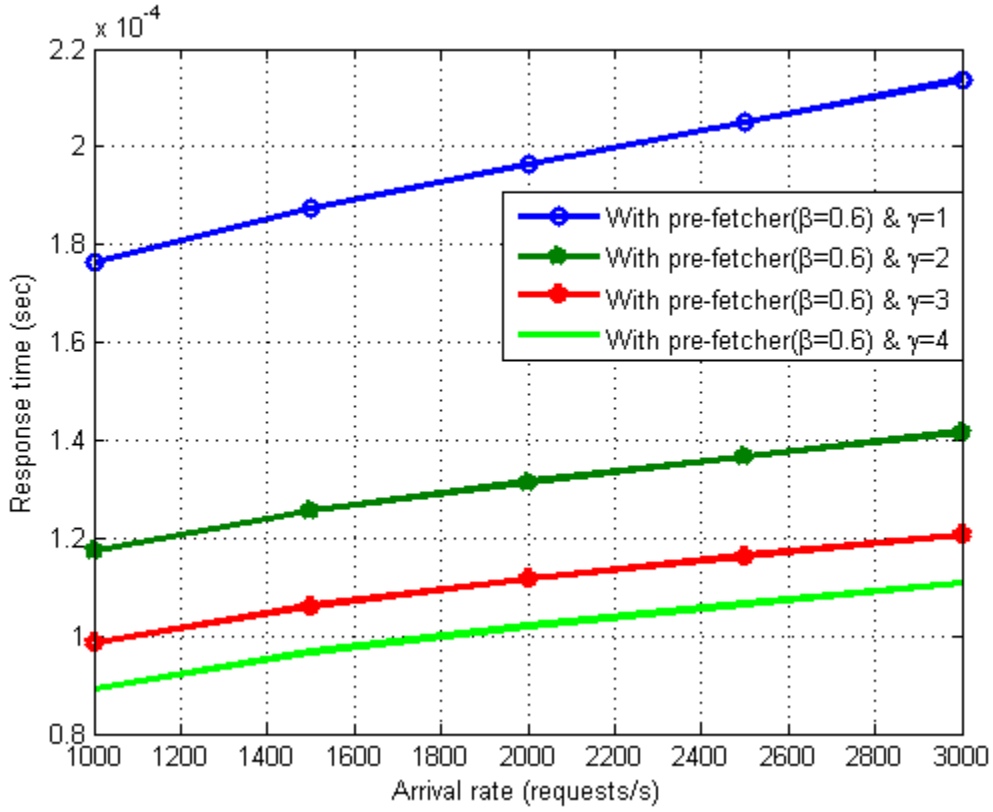


Fig. 4.5 Workload response time vs. arrival rate while the workload is shared between multi-media cloud and pre-fetcher at probability distribution of $\beta = 0.6$.

In this part, we study the response time behaviour of the arrival rate $\lambda = [1000, 3000, 500]$ and, $\gamma = [1, 4, 1]$ at the probability distribution of $\beta = 0.6$. As presented in Fig. 4.5, we observe the speed enhancement while speed multiplier γ is increasing as expected, the shortest response time belongs to the bottom curve with $\gamma = 4$. Another noticeable result that we can see in Fig. 4.5 is that the response time gap between each curve is significantly decreasing as a result of speed enhancement. Having said that, all the results and the performance analysis are on using pre-fetcher without considering the delay associated with its processor.

4.1.3. Response time study while β is changing

This figure could be one of the most interesting comparisons. In this figure we have total response time (sec) on Y axis versus workload distribution probability (β) changing on X axis from 0.2 to 1 with 0.1 increment unit. As can be interpreted, when the response time increases and the load is pushed toward the pre-fetcher, only the top line is constant value at $\gamma = 1$, since is it running at the same speed as the cloud, the other curves are all descending. Descending curves show that the response time decreases as the processor's processing time increases and it reaches its minimum on the bottom curve.

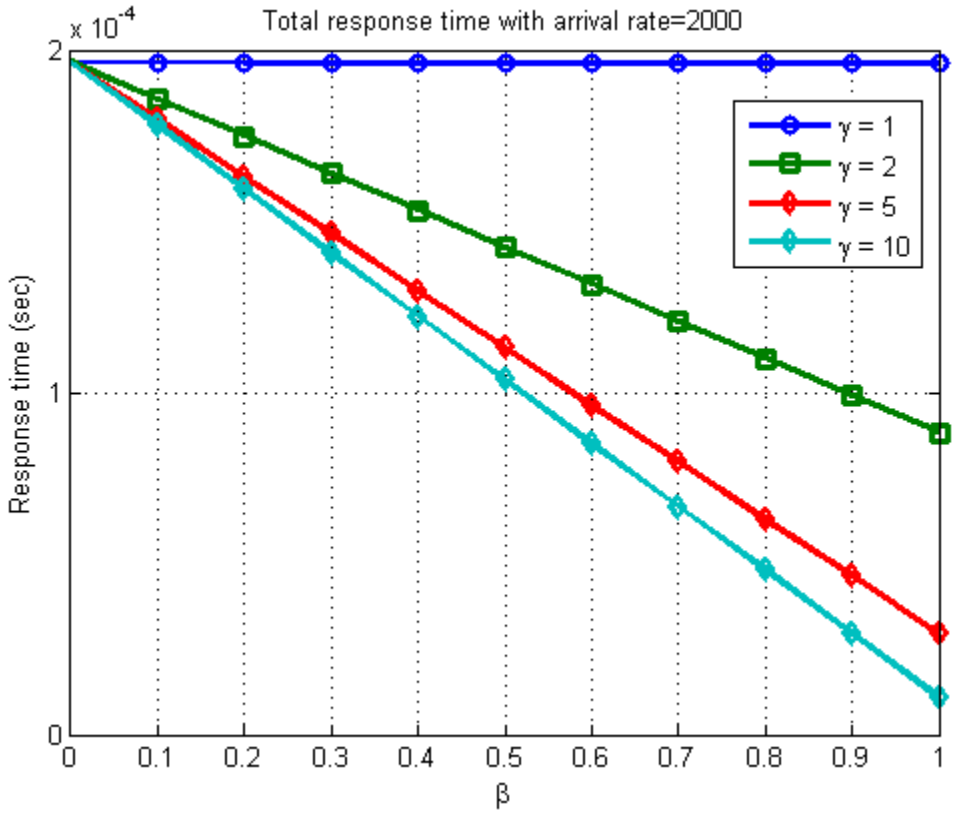


Fig. 4.6 Total response time vs. probability distribution while pre-fetcher's processor speed multiplier is changing.

As the workload is leaning more towards pre-fetcher, the processing speed comes to attention more than before. As we can observe in Fig. 4.6, the bottom curve has the minimum total response time, and the total response time is minimized when entire workload is sent to pre-fetcher. Accordingly, a faster processor is needed which means more cost, and we will study that in the next part.

4.2. Cost Consideration

In this section, we investigate the cost improvements based on the simulation results. We will change the values of β, γ and, λ to observe the changes in total cost and total response time. Pre-fetchers processing is considered to be changing from 1 to 50.

4.2.1 Total cost improvement while PF utilization is $\beta = 0.6$ and λ is increasing

In Fig. 4.7, we observe the simulation result when only 0.6 ($\beta = 60\%$) of the total workload is sent to pre-fetcher and arrival rate is increasing. γ is also considered to be increasing from 0 to 50. As expected, the total cost (sec) is increasing while a number of requests/sec increases. This cost is associated with the delay that occurs when pre-fetcher is processing the requests.

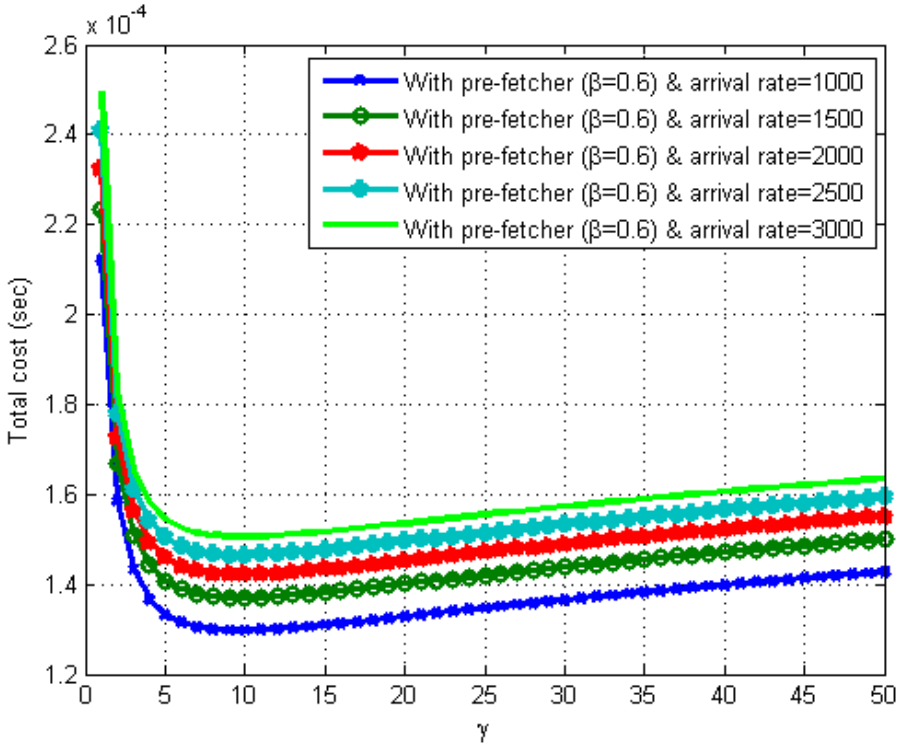


Fig. 4.7 Impact of increasing arrival rate and γ on Total cost when $\beta = 0.6$.

As presented, the initial γ is set to be 1 in all curves, which means processor of the multi-media cloud and pre-fetcher are both running at the same speed.

4.2.2. Total cost improvement while $\lambda = 3000$ and PF utilization is increasing

As presented in Fig.4.8, in this simulation arrival rate is set to be fixed at 3000 while the use of pre-fetcher is increasing. Total cost is linear and constant when there is no pre-fetcher. While pre-fetcher is operating, initially $\beta = 1$ has the minimum total cost (sec) and, as γ is increasing $\beta = 0.6$ takes its places and becomes the overall optimum β for this particular instance.

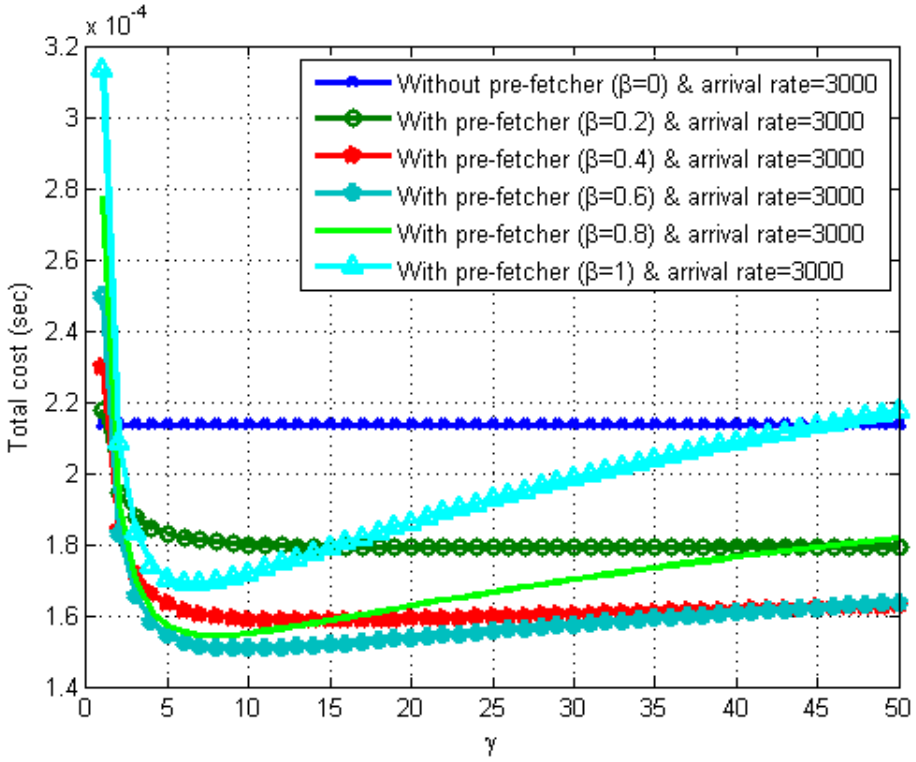


Fig. 4.8 Impact of increasing β and γ when $\lambda = 3000$.

The value of optimal β varies as we change the simulation criteria and change the arrival rate or pre-fetcher's processing speed. For example, if we take $\gamma = 50$, the optimum value for β is 0.4 and similarly optimal β changes its place based on the condition applied in the simulation. In other words, the percetage of using pre-fetcher varies from time to time, and it is not always best to allocate the entire workload to pre-fetcher even though it operates faster than cloud.

4.2.3. Response time Vs. β while PF is processing at fixed speed $\gamma = 15$

One of our goals is to find the optimal response time for the amount of workload sent to pre-fetcher. In order to study that, we performed a simulation where arrival rate is set to be 2000 requests/sec and $\gamma = 15$. Values of *m* are considered to be 0.25 and m' is considered as 1.

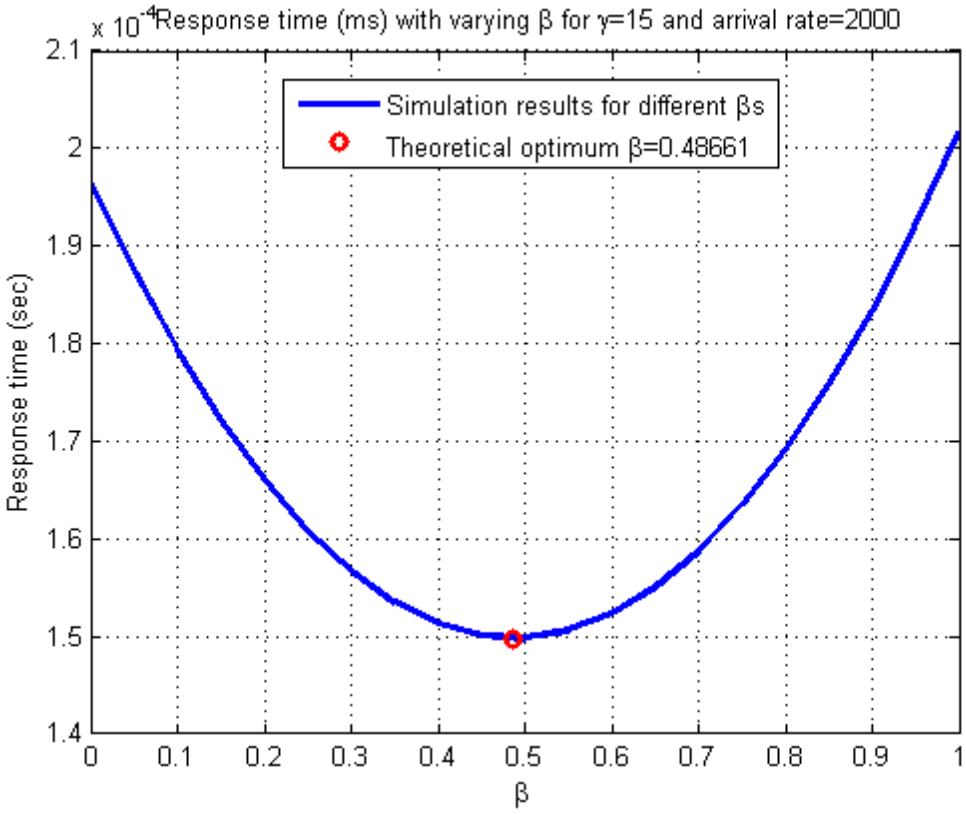


Fig. 4.9 β vs. Response time (sec) with for $\gamma = 15$, and arrival rate=2000.

Based on the results, response time is optimum when $(\beta = 0.48)$ almost half of the workload is sent to pre-fetcher when $\gamma = 15$. The theoretically optimum β is also calculated and marked on Fig. 4.9 to show that both simulation and theoretical results are aligned at the same point. Staring from $\beta = 0$, response time (sec) is considerably high, and it decreases slowly while the use of pre-fetcher increase up to 0.5, and after that, it starts rising again until it hits $\beta = 1$. It is understandable that if only pre-fetcher is used, it is not the optimum condition, since response time and accordingly costs are increasing.

In Fig. 4.10 and Fig. 4.11, we investigate the impact of changing γ and λ, on optimum β. As presented, in both figures theoretically calculated optimum β is marked as well. As we can observe from the trends as increase in the arrival rate of the response time slightly increases, the β optimum point moves to the right. In other words, if the amount of the requests/sec is increasing, it is more beneficial to push the load toward pre-fetcher.

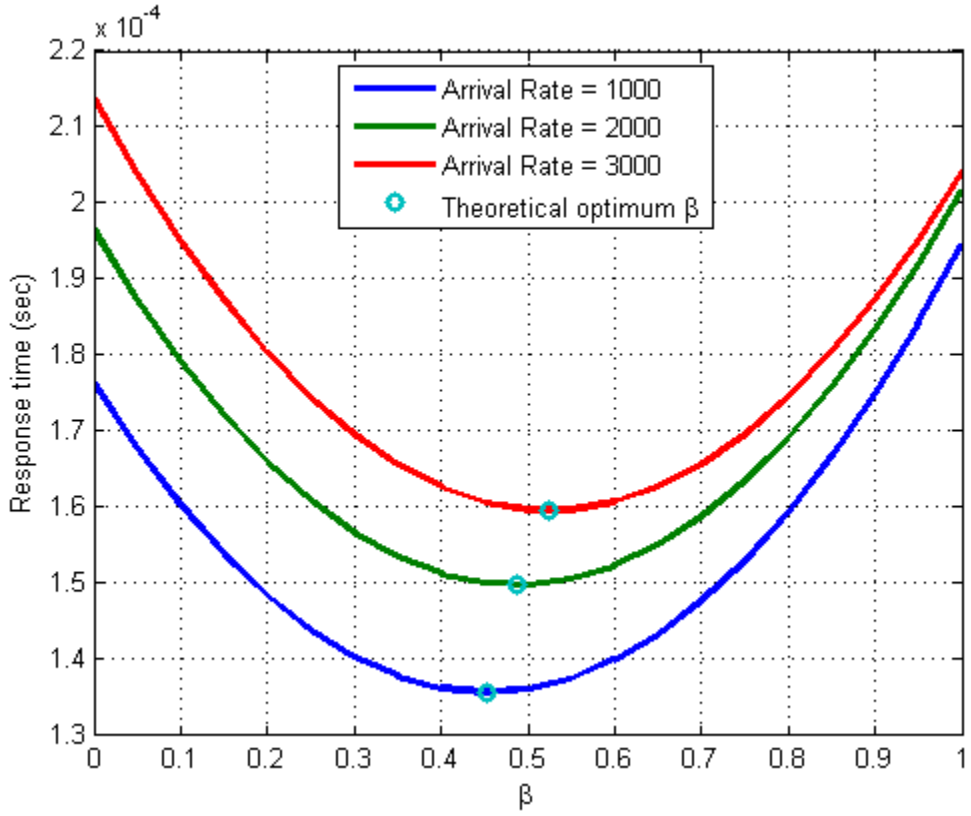


Fig. 4.10 Comparison of optimum β values while $\gamma = 15$ and λ is changing.

As shown in Fig. 4.10, all the curves for response time are slowly decreasing to the optimal point, and then it increases. The gap between all curves is also shrinking as we move away from the optimal point using (3.9) and we allocate more load toward pre-fetcher to handle. The optimum point of β aligns with the theoretical value β in all the curves. The response time gap between curves is also maximized at the optimal point and minimized at $\beta = 1$. According to Fig. 4.10, we see how different arrival rates would affect the optimal value of β .

In Fig. 4.11, we study the changing trend of the curves according to the processing speed, while arrival rate is fixed to be $\lambda = 2000$. Unlike the previous figure, in this figure, the gap between curves is slowly increasing as we push more load toward pre-fetcher.

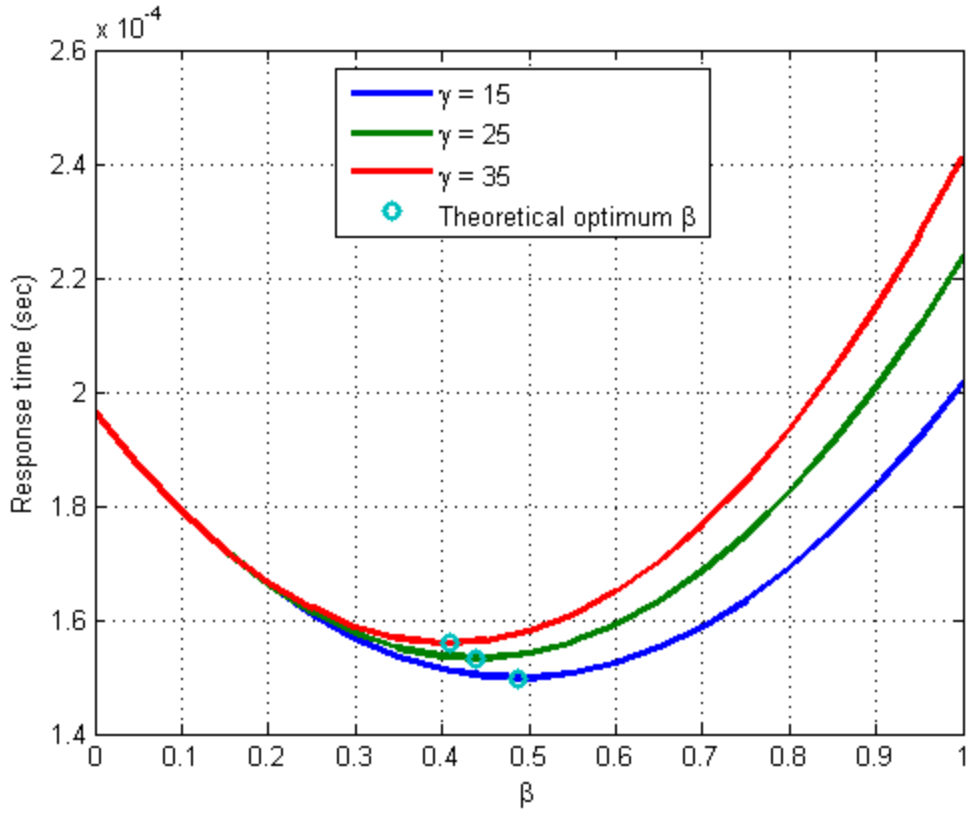


Fig. 4.11 Comparison of optimum β values while γ is changing and $\lambda = 2000$.

As we push more load to pre-fetcher, initially response time decreases to the optimal point and after that, increases to maximum where all the workload is allocated to pre-fetcher. All the curves start basically from the same starting point and initially, their response time decrease is linear to $\beta = 0.5$.Using pre-fetcher is most advantageous at pre-fetcher's utilization optimum point. Similar to the previous figure, the optimum point of β aligns with theoretical value β in all the curves. In both Figures 4.10 and 4.11, we can observe the impact of varying γ and λ when $\beta = [0,1]$.

4.2.4 Cost decrease (%) Vs. arrival rate (requests/sec) while γ is changing

Cost is one of the most important factors in every problem. Earlier, based on the preformed simulation, we showed that using pre-fetcher is an excellent way to reduce the total response time. Pre-fetcher has a delay which was formulated in the previous chapter and here in Fig. 4.10 we present the overall percentage improvement while pre-fetcher is being used at different speeds. The arrival rate us also considered to be changing from 1000 to 3000 requests/sec.

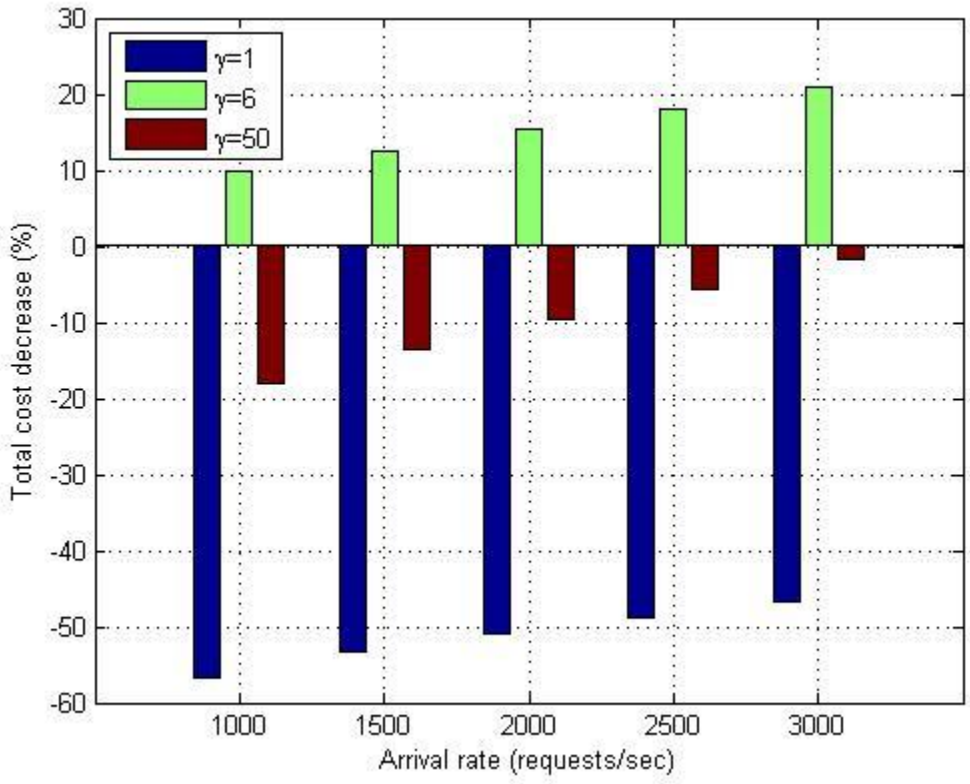
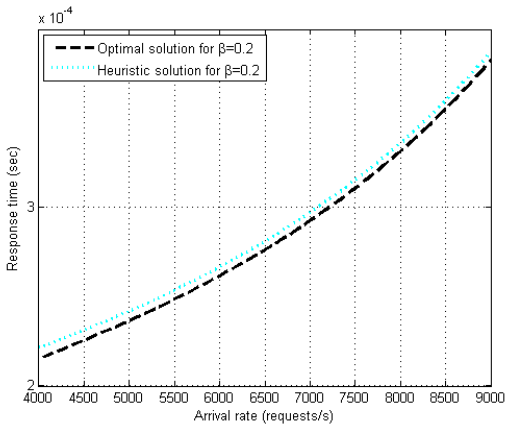


Fig. 4.12 Cost decrease improvement vs. Arrival rate.

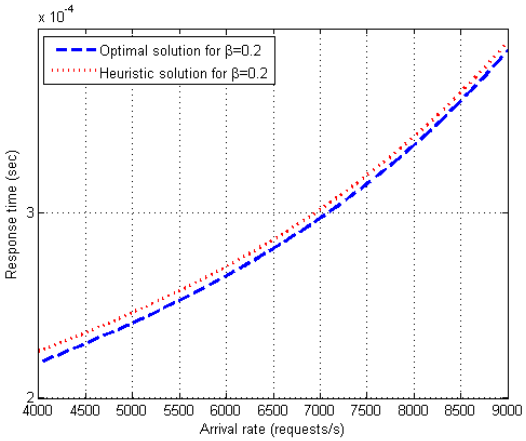
The bar assigned to $\gamma = 1$, shows a negative decrease where all the bars are below 0. It means that there is no improvement when both pre-fetcher and cloud are processing the requests at the same speed. The bar assigned to $\gamma = 50$ is better than $\gamma = 0$, but is also below 0 and shows a negative value which means if pre-fetcher processes data remarkably faster than multi-media cloud it is also not constructive since cost function is not satisfied within our criteria.

The bar allotted to $\gamma = 6$ has a positive value and its amount is increasing while requests are increasing per second. We can argue that every bar which has a positive percentage value is advantageous toward our goal with total cost decrease.

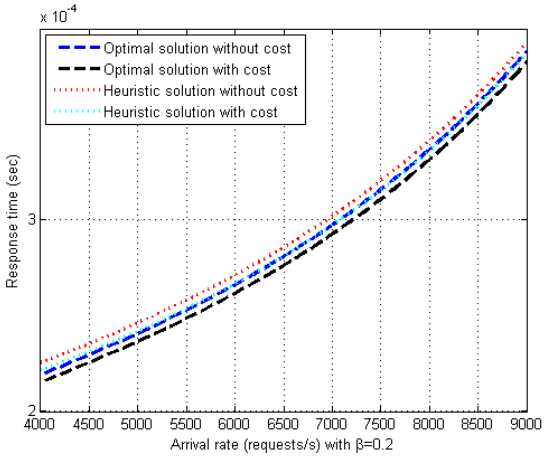
4.3. Optimal and Heuristic Optimization Performance analysis



(a) Cost optimization



(b) Time optimization



(c) Cost vs. time optimization

Fig. 4.13 Comparison between optimal and heuristic optimization approaches with and without cost factor.

We preformed both heuristic and optimal optimizations on the total response time. In this simulation, pre-fetcher processing speed, γ is set to be 3, pre-fetcher utilization factor, β is set to be 0.2 while arrival rate changes from 4000 requests/sec to 9000 requests/sec. The heuristic scheme has longer response time than the optimal scheme with a small difference. However, the minimum optimal scheme has more complexity in computation. The optimal scheme weights vary with the change of λ, while weights in the heuristic scheme are constant since it is normalized by

the service rate. In (a) after the implementation of the cost function, we can see a small elevation compared to (b) which does not have the cost. In, (c) both (a) and (b) are shown for comparison. Based on the assumption in 4.1 and the optimization schemes, in Table 4.2 we have weight ω as follows [1]:

$$\begin{aligned} \overrightarrow{Assume}\ \xi_{i,Cloud} &= \sum_{i=1}^{N}\sum_{j=1}^{M}\theta_{ij}\mu_{i,Cloud} \\ \xi_{i,PF} &= \sum_{i=1}^{N}\sum_{j=1}^{M}\alpha_{ij}\mu_{i,PF}, \text{ where } \mu_{i,PF} = \gamma\mu_{i,Cloud}, \\ &= \sum_{i=1}^{N}\sum_{j=1}^{M}\alpha_{ij}\gamma\mu_{i,PF} \end{aligned} \tag{4.1}$$

Scheme	Cloud
Optimal	$\omega_{i,Cloud} = \dfrac{\lambda\sqrt{\xi_{i,Cloud}} + \xi_{i,Cloud}\sum_{j=1}^{N}\sqrt{\xi_{j,Cloud}} - \sqrt{\xi_{i,Cloud}}\sum_{j=1}^{N}\xi_{j,Cloud}}{\lambda\sum_{j=1}^{N}\sqrt{\xi_{j,Cloud}}}$
Heuristic	$\omega_{i,\text{ Cloud}} = \dfrac{\xi_{i,\ Cloud}}{\sum_{i=1}^{N}\xi_{i,\ Cloud}}$
Scheme	**Pre-fetcher**
Optimal	$\omega_{i,PF} = \dfrac{\lambda\sqrt{\xi_{i,PF}} + \xi_{i,PF}\sum_{j=1}^{N}\sqrt{\xi_{j,PF}} - \sqrt{\xi_{i,PF}}\sum_{j=1}^{N}\xi_{j,PF}}{\lambda\sum_{j=1}^{N}\sqrt{\xi_{j,PF}}}$
Heuristic	$\omega_{i,PF} = \dfrac{\xi_{i,\text{PF}}}{\sum_{i=1}^{N}\xi_{i,\text{PF}}}$

Table 4.2 Workload formulation based on optimization and heuristic schemes.

Chapter 5

Conclusion and Future Work

Previously work was done with the primary objective in convex workload scheduling problem for cloud based applications. The authors propose path selection algorithm to reduce response time by adjusting weights among the paths of tasks. In this thesis, we propose a pre-fetching system model to enhance data transmission speed (delay) between multi-media cloud and end users and finally penalizing it with a cost function. The proposed problem was more complex compared to previous similar work (NP-hard), and the heuristic algorithm was considered to be the solution. In this study, we used a pre-fetcher that adopts and learns new frequent requests to cache and store them prior to the actual request.

5.1. Conclusion

Proposed response time minimization and cost optimization problems in this work study were simulated and solved by a heuristic algorithm. Accordingly, the solution to these problems can be found somewhere between max and min, which contains the optimal solution with respect to time and cost.

As it is presented in Chapter 4, based on the simulation results, we can observe:

- Significant response time improvement compared to previous study results.
- Proposed solution is cost optimized while it has fairly less processing time
- Complexity is also reduced since no computation is done in designed pre-fetcher model and the only processing time belongs to searching for data when it is requested and caching from the multi-media cloud to update itself. Decision making and computation core are assumed to be in multi-media cloud to reduce the overhead.

Data accessing speed is also improved since the end user requests most of the time existed and cached from the pre-fetcher and not the cloud. In other words, data transmission skips nodes and routers by reaching directly to pre-fetcher which most of the time hast the request answers stored.

Based on the mentioned contribution and improvements above, we can conclude that proposed solution is an enhancement to previous models and solutions.

5.2. Future Work

Even though our goal was to design and develop a pre-fetcher, there are times that request does not exist within the storage of the pre-fetcher (miss-ratio). The multi-media cloud communication method is becoming a major focus of researchers nowadays, yet intelligence of pre-fetching and caching learning schemes can be improved and would be an interesting topic to work on with regard to reducing miss-ratio of intelligent caching schemes in future studies.

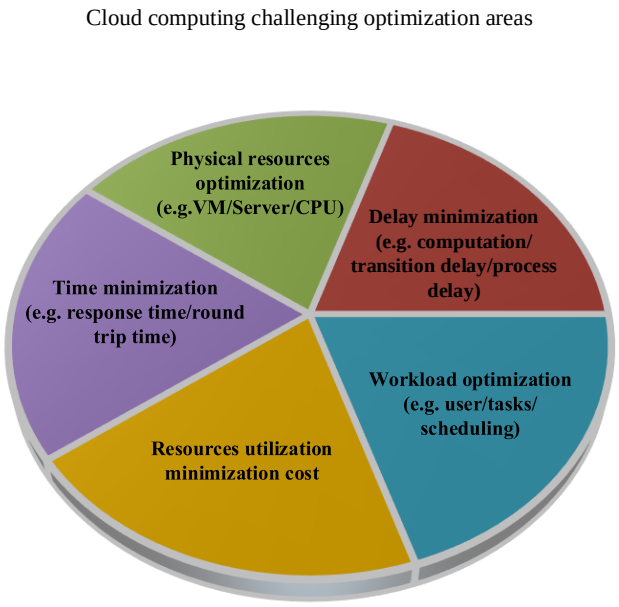


Fig. 5.1 Future work hot research areas

Another side that was not discussed in this study is energy efficacy of the proposed method. An energy efficient scheme in either multi-media cloud, end user side or pre-fetcher could be implemented, and their way of data transmission could be considered as another field of research in future studies. For creating a better vision, some of the interesting research areas are categorized and brought to attention in Fig. 5.1.

References

[1] L. Ferdouse, M. Li, L. Guan, and A. Anpalagan, "Bayesian workload scheduling in multimedia cloud networks," *2016 IEEE 21st International Workshop on Computer Aided Modelling and Design of Communication Links and Networks (CAMAD)*. pp. 83–88, 2016.

[2] K. Kumar and Y. H. Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?," *Computer*, vol. 43, no. 4. pp. 51–56, 2010.

[3] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. pp. 1–9, 2016.

[4] X. Lin, Y. Wang, Q. Xie, and M. Pedram, "Task Scheduling with Dynamic Voltage and Frequency Scaling for Energy Minimization in the Mobile Cloud Computing Environment," *IEEE Transactions on Services Computing*, vol. 8, no. 2. pp. 175–186, 2015.

[5] N. S. Hussien, S. Sulaiman, and S. M. Shamsuddin, "Review on pre-fetching for Mobile Cloud Computing," *2013 IEEE Conference on e-Learning, e-Management and e-Services*. pp. 130–135, 2013.

[6] D. Sengupta, A. Goswami, K. Schwan, and K. Pallavi, "Scheduling Multi-tenant Cloud Workloads on Accelerator-Based Systems," *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*. pp. 513–524, 2014.

[7] A. Ilyushkin, B. Ghit, and D. Epema, "Scheduling Workloads of Workflows with Unknown Task Runtimes," *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. pp. 606–616, 2015.

[8] P. Gupta, S. G. Koolagudi, R. Khanna, M. Ganguli, and A. N. Sankaranarayanan, "Analytic technique for optimal workload scheduling in data-center using phase detection," *5th International Conference on Energy Aware Computing Systems & Applications*. pp. 1–4, 2015.

[9] M. Balasubramaniam, I. Banicescu, and F. M. Ciorba, “Scheduling Data Parallel Workloads - A Comparative Study of Two Common Algorithmic Approaches,” *2013 42nd International Conference on Parallel Processing*. pp. 798–807, 2013.

[10] H. Ghoumain, “CONTEXT-AWARE RESOURCE ALLOCATION AND SCHEDULING FOR HYBRID,” Ryerson University, 2015.

[11] M. Wang and W. Zeng, “A Comparison of Four Popular Heuristics for Task Scheduling Problem in Computational Grid,” *2010 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM)*. pp. 1–4, 2010.

[12] S. Byna, Y. Chen, and X. H. Sun, “A Taxonomy of Data Prefetching Mechanisms,” *2008 International Symposium on Parallel Architectures, Algorithms, and Networks (i-span 2008)*. pp. 19–24, 2008.

[13] I. Ganusov and M. Burtscher, “Future execution: a hardware prefetching technique for chip multiprocessors,” *14th International Conference on Parallel Architectures and Compilation Techniques (PACT’05)*. pp. 350–360, 2005.

[14] J. Kim, K. V Palem, and W.-F. Wong, “A framework for data prefetching using off-line training of Markovian predictors,” *Proceedings. IEEE International Conference on Computer Design: VLSI in Computers and Processors*. pp. 340–347, 2002.

[15] J. Li and S. Wu, “Real-time Data Prefetching Algorithm Based on Sequential Patternmining in Cloud Environment,” *2012 International Conference on Industrial Control and Electronics Engineering*. pp. 1044–1048, 2012.

[16] H. K. Lee, B. S. An, and E. J. Kim, “Adaptive Prefetching Scheme Using Web Log Mining in Cluster-Based Web Systems,” *2009 IEEE International Conference on Web Services*. pp. 903–910, 2009.

[17] U. K. Yoon, H. j. Kim, and J. y. Chang, “Intelligent Data Prefetching for Hybrid Flash-Disk Storage Using Sequential Pattern Mining Technique,” *2010 IEEE/ACIS 9th International Conference on Computer and Information Science*. pp. 280–285, 2010.

[18] Y. Chen, H. Zhu, and X. H. Sun, “An Adaptive Data Prefetcher for High-Performance Processors,” *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. pp. 155–164, 2010.

[19] X. Wang, T. Kwon, Y. Choi, H. Wang, and J. Liu, “Cloud-assisted adaptive video streaming and social-aware video prefetching for mobile users,” *IEEE Wireless Communications*, vol. 20, no. 3. pp. 72–79, 2013.

[20] J.-M. Kang, S. Seo, and J. W.-K. Hong, “Personalized battery lifetime prediction for mobile devices based on usage patterns,” *J. Comput. Sci. Eng.*, vol. 5, no. 4, pp. 338–345, 2011.

[21] G. Yunwen, W. Shaochun, Y. Bowen, and L. Jiazheng, “The Methods of Data Prefetching Based on User Model in Cloud Computing,” *2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*. pp. 463–466, 2011.

[22] H. T. Kung, C. K. Lin, D. Vlah, and G. B. Scorza, “Speculative pipelining for compute cloud programming,” *2010 - MILCOM 2010 MILITARY COMMUNICATIONS CONFERENCE*. pp. 2026–2034, 2010.

[23] L. Lin, X. Li, H. Jiang, Y. Zhu, and L. Tian, “AMP: An Affinity-Based Metadata Prefetching Scheme in Large-Scale Distributed Storage Systems,” *2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*. pp. 459–466, 2008.

[24] R. García, E. Verdú, L. M. Regueras, J. P. de Castro, and M. J. Verdú, “A neural network based intelligent system for tile prefetching in web map services,” *Expert Syst. Appl.*, vol. 40, no. 10, pp. 4096–4105, 2013.

[25] B. Rajkumar, T. Gopikiran, and S. Satyanarayana, “Neural network design in cloud computing,” *Int. J. Comput. Trends Technol.*, vol. 4, no. 2, pp. 6–7, 2013.

[26] M. Chandrasekaran, M. Muralidhar, and U. S. Dixit, “Online optimization of multipass machining based on cloud computing,” *Int. J. Adv. Manuf. Technol.*, pp. 1–12, 2013.

[27] S. Sarwar, Zia-ul-Qayyum, O. A. Malik, B. Rizvi, H. F. Ahmed, and H. Takahashi, "Performance comparison of case retrieval between Case Based Reasoning and Neural Networks in Predictive Prefetching," *2009 6th International Symposium on High Capacity Optical Networks and Enabling Technologies (HONET)*. pp. 57–61, 2009.

[28] S. Sarwar, Z. Ul-Qayyum, and O. A. Malik, "A hybrid intelligent system to improve predictive accuracy for cache prefetching," *Expert Syst. Appl.*, vol. 39, no. 2, pp. 1626–1636, 2012.

[29] M. Maurer, I. Brandic, and R. Sakellariou, "Adaptive resource configuration for Cloud infrastructure management," *Futur. Gener. Comput. Syst.*, vol. 29, no. 2, pp. 472–487, 2013.

[30] Y. Singh, P. K. Bhatia, and O. Sangwan, "A review of studies on machine learning techniques," *Int. J. Comput. Sci. Secur.*, vol. 1, no. 1, pp. 70–84, 2007.

[31] K. Park, M. Song, and C.-S. Hwang, "Broadcasting and Prefetching Schemes for Location Dependent Information Services.," in *W2GIS*, 2004, pp. 26–37.

[32] B. Mobasher, H. Dai, T. Luo, and M. Nakagawa, "Using sequential and non-sequential patterns in predictive Web usage mining tasks," *2002 IEEE International Conference on Data Mining, 2002. Proceedings.* pp. 669–672, 2002.

[33] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies," *ACM Comput. Surv.*, vol. 35, no. 4, pp. 374–398, 2003.

[34] T. Koskela, J. Heikkonen, and K. Kaski, "Web cache optimization with nonlinear model using object features," *Comput. Networks*, vol. 43, no. 6, pp. 805–817, 2003.

[35] K.-Y. Wong, "Web cache replacement policies: a pragmatic approach," *IEEE Network*, vol. 20, no. 1. pp. 28–34, 2006.

[36] M. Farhan, "Intelligent web caching architecture," Universiti Teknologi Malaysia, 2006.

[37] W. Ali, S. M. Shamsuddin, and A. S. Ismail, "A survey of web caching and prefetching," *Int. J. Adv. Soft Comput. Appl*, vol. 3, no. 1, pp. 18–44, 2011.

[38] X. Nan, Y. He, and L. Guan, “Optimal task-level scheduling for cloud based multimedia applications,” *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. pp. 3771–3775, 2013.

[39] X. Nan, Y. He, and L. Guan, “Optimal allocation of virtual machines for cloud-based multimedia applications,” *2012 IEEE 14th International Workshop on Multimedia Signal Processing (MMSP)*. pp. 175–180, 2012.

[40] X. Nan, Y. He, and L. Guan, “Towards optimal resource allocation for differentiated multimedia services in cloud computing environment,” *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. pp. 684–688, 2014.

[41] R. Kapur, “A workload balanced approach for resource scheduling in cloud computing,” *2015 Eighth International Conference on Contemporary Computing (IC3)*. pp. 36–41, 2015.

[42] R. Kapur, “A cost effective approach for resource scheduling in cloud computing,” *2015 International Conference on Computer, Communication and Control (IC4)*. pp. 1–6, 2015.