Fall 10-10-2018

Dynamic Hierarchical Cache Management for Cloud RAN and Multi- Access Edge Computing in 5G Networks

Deepika Pathinga Rajendiran
San Jose State University

Dynamic Hierarchical Cache Management for Cloud RAN and Multi-Access Edge Computing in 5G Networks

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By

Deepika Pathinga Rajendiran

October 2018

The Designated Writing Project Committee Approves the Writing Project Titled

Dynamic Hierarchical Cache Management for Cloud RAN and Multi-Access Edge Computing in 5G Networks

by

Deepika Pathinga Rajendiran

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

October 2018

Dr. Melody Moh	Department of Computer Science
Dr. Robert Chun	Department of Computer Science
Dr. Teng Moh	Department of Computer Science

ABSTRACT

Cloud Radio Access Networks (CRAN) and Multi-Access Edge Computing (MEC) are two of the many emerging technologies that are proposed for 5G mobile networks. CRAN provides scalability, flexibility, and better resource utilization to support the dramatic increase of Internet of Things (IoT) and mobile devices. MEC aims to provide low latency, high bandwidth and real-time access to radio networks. Cloud architecture is built on top of traditional Radio Access Networks (RAN) to bring the idea of CRAN and in MEC, cloud computing services are brought near users to improve the user's experiences. A cache is added in both CRAN and MEC architectures to speed up the mobile network services. This research focuses on cache management of CRAN and MEC because there is a necessity to manage and utilize this limited cache resource efficiently. First, a new cache management algorithm, H-EXD-AHP (Hierarchical Exponential Decay and Analytical Hierarchy Process), is proposed to improve the existing EXD-AHP algorithm. Next, this paper designs three dynamic cache management algorithms and they are implemented on the proposed algorithm: H-EXD-AHP and an existing algorithm: H-PBPS (Hierarchical Probability Based Popularity Scoring). In these proposed designs, cache sizes of the different Service Level Agreement (SLA) users are adjusted dynamically to meet the guaranteed cache hit rate set for their corresponding SLA users. The minimum guarantee of cache hit rate is for our setting. Net neutrality, prioritized treatment will be in common practice. Finally, performance evaluation results show that these designs achieve the guaranteed cache hit rate for differentiated users according to their SLA.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my project advisor, Dr. Melody Moh, for her excellent guidance, and encouragement throughout the course of this project.

My sincere thanks to my committee members, Dr. Robert Chun, and Dr. Teng Moh for their useful comments, and their valuable time for reviewing my work.

Special thanks to my husband, for his endless support, and understanding, and for never giving up on me. Special love to my son for his patience, and being flexible to my schedule, and his love for me helped me to keep going. I would like to say my heartily thank you to my mom for her motivation and blessings.

Last but not the least, a big thank you for my wonderful family and friends for their moral support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AHP – Analytical Hierarchy Process

BBU – Base Band Unit

CD – Cache Distribution

CHR – Cache Hit Rate

CRAN – Cloud Radio Access Networks

CWR – Cloud Write Rate

DH – Dynamic Hierarchical

EXD – Exponential Decay

GGDH – Good Guess Dynamic Hierarchical

IDH – Improved Dynamic Hierarchical

LFU – Least Frequently Used

MEC – Multi-access Edge Computing

MG – Minimum Guarantee

PBPS – Probability Based Popularity Scoring

RRH – Remote Radio Head

RRM – Reverse Random Marking

TD – Traffic Distribution

1. INTRODUCTION

For Fifth Generation (5G) mobile networks many fast-growing technologies such as Cloud Radio Access Networks (CRAN), Multi-Access Edge Computing (MEC), Millimeter Wave, Massive Multiple-Input Multiple-Output (MIMO) has been proposed. Among these technologies, we are going to use CRAN and MEC for our research.

These technologies are introduced to handle the traffic data volume caused by the rapidly growing Internet of Things (IoT) and mobile devices [7, 24]. There is a cache involved in both CRAN and MEC technologies to increase the speed of cellular network services [7, 11, 24, 26]. The memory size of these cache resources is limited and therefore there is a necessity to manage these cache resources efficiently. With these kinds of challenges, the network service providers are demanded to meet the mobile user's satisfaction. This research focuses on User Equipment Context (UEC) cache management for CRAN and files content cache management for MEC [7, 24]. We will discuss UEC and files content in the following section. So, the idea is to increase the Cache Hit Rate (CHR) of CRAN and MEC by managing the available resource efficiently to improve the overall user's experience.

This paper first proposes a new algorithm H-EXD-AHP (Hierarchical Exponential Decay and Analytical Hierarchy Process), to improve the existing algorithm EXD-AHP [22, 24]. Then, three dynamic cache management algorithms are designed, and they are implemented on the new algorithm: H-EXD-AHP and an existing algorithm: H-PBPS (Probability Based Popularity Scoring) [11, 24]. The aim of these dynamic cache algorithms is to achieve the guaranteed cache hit rate corresponding to the user's Service Level Agreement (SLA). The minimum guarantee of cache hit rate is for our setting to improve SLA 3 and SLA 4 users. Net neutrality, prioritized treatment will be in common practice. Cache sizes are partitioned for differentiated SLA users and they are dynamically adjusted to achieve the guaranteed cache hit rate. This paper is organized as follows: The following section discusses the background of CRAN and MEC architectures. Section 2 also presents the related works of this research. Section 3 describes the existing cache management algorithms. Section 4 discusses the preliminary results of the existing algorithms. Section 5 presents the proposed dynamic cache algorithms. Section 6 gives us the performance

evaluation of the proposed algorithms, which is followed by the conclusion section. This work is a continuation of our research on cloud computing [8, 16], CRAN [10, 11, 20, 22, 23, 24], edge and fog computing [3, 5, 14], and IOT, mobile and 5G networks [14, 18, 19, 21].

2. BACKGROUND AND RELATED WORKS

2.1 CRAN and MEC Architectures

Traditional Radio Access Networks (RAN) is a distributed architecture which consists of Long Term Evolution (LTE) Macro Base Station (MBS) (evolved Node B or eNodeB) and User Equipment (UE) [2]. Each Remote Radio Head (RRH) of eNodeB has its own Base Band Unit (BBU). They are connected to the core network or Evolved Packet Core (EPC) and the internet. RRH is responsible for transmitting and receiving wireless signals. Whereas BBU helps in converting Internet Protocol (IP) packets to signals, manages Quality of Services (QoS) and user mobility [24]. CRAN is a centralized cloud architecture where BBUs are separated from their RRHs, virtualized and pooled together. The architecture of CRAN is shown in Figure 1 [24, 25]. Each BBU can be represented as a Virtual Machine (VM) and each VM has its own cache memory. The size of this cache memory involved in the BBU pool is limited, so it is important to manage this resource efficiently [24, 25].

For each user, a UEC record information is stored in the secondary cloud memory. This UEC information contains user's ID, state information of the current event or session, subscription details etc. Basically, it is like a metadata about the users. So, instead of retrieving this information from the secondary cloud, it will be easier to access if it is stored in the BBU pool cache. Our aim is to increase the CHR of UEC in the BBU pool. By this, we are reducing the traffic of secondary cloud storage [24]. CRAN aims for better scalability, flexibility and better resource utilization.

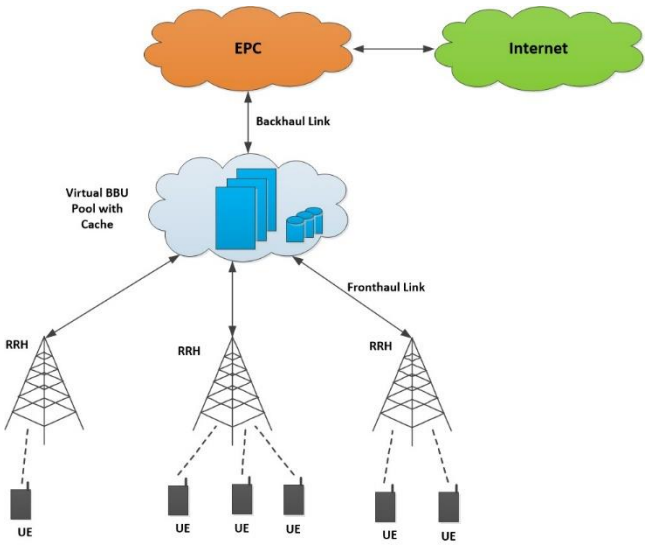

Figure 1. CRAN architecture.

In MEC, cloud computing services are brought near the users to improve the user's experience [26]. These services include but not limited to content caching, task offloading, storage, computation. For our research, we are going to focus on content file caching of MEC. Like UEC caching from CRAN, we are trying to increase the CHR of file contents in MEC servers. A MEC server with cache is deployed near each MBS [7]. The architecture of MEC is shown in Figure 2 [7, 26]. Of course, the cache size here also is limited, so there is a necessity to manage this resource efficiently. These servers can share its cached items with each other using the communications between MBS. To achieve communication between MBS, X2 interface can be used [7]. MEC aims to reduce latency and backhaul traffic flow of the networks.

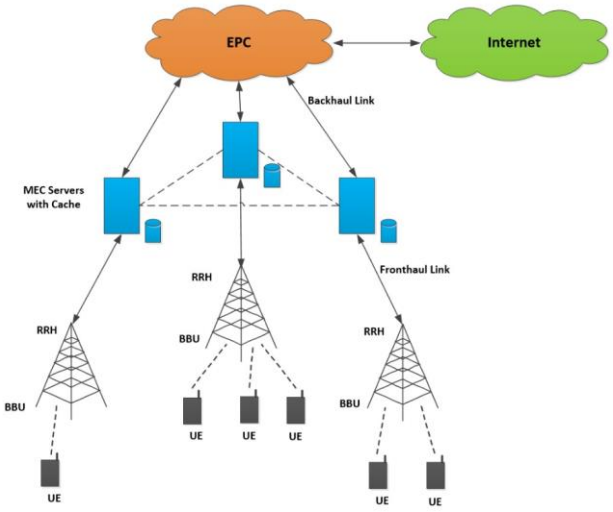

Figure 2. MEC architecture.

Caching on the edge (near the users) has been proven effective. Instead of retrieving data from the internet, it will be easier and quicker if there is content readily available as near as possible to the users [26]. CRAN and MEC technologies can be combined and form a new hybrid architecture as shown in Figure 3 [12, 13]. CRAN uses centralized BBU pool whereas MEC servers usually work with distributed MBS [12, 13]. In this hybrid architecture, MEC's request can be sent and received via MBS (RRH and BBU pool) [12]. For 5G networks, it can be either CRAN architecture or MEC architecture or combination of both architectures.

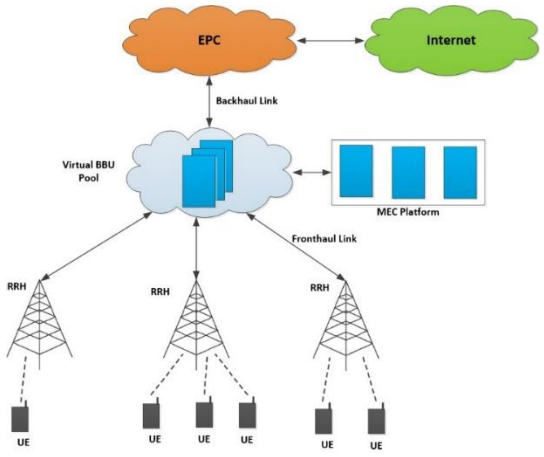

Figure 3. Hybrid CRAN and MEC architecture.

2.2 Related Studies

Cache management problem is widely studied in wireless mobile networks. Following are the some of the related works discussed.

Floratou et al proposed adaptive Selective Least Recently Used – K (SLRU-K) and adaptive Exponential Decay (EXD) caching algorithms for Big SQL, in which they mentioned that their parameter K and parameter a value for adaptive SLRU-K and adaptive EXD respectively has significant impact on changing workload, so they are changed dynamically according to the workload resulted in better performance than existing algorithms [4]. For our research, we are using EXD for scoring the elements in the cache. If parameter a value is higher then, in the cache more recent elements will be placed and if parameter a value is smaller then more frequent elements will be given importance in the cache [4].

Gomes, Braun, Monteiro used the Analytical Hierarchy Process (AHP) to calculate the weight of mobility, non-intersecting content, free storage, relative size of mobility group, and cost of migration [6, 17]. A matrix is formed using these and importance is calculated by AHP weights [17]. In this, they are trying to keep the popular contents in the edge cache. They show us the design and strategies of mobile edge migrations. Using their simulation results, the authors show that they can reduce the latency and increase the cache hit at the edge caches [6].

Tsai and Moh adopted the above two papers and came up with an algorithm called EXD-AHP scoring algorithm in which lowest score UEC is evicted to make space for highest score UEC [22, 24]. They have used four different levels of SLA users according to their mobility, basic and premium services. Using their algorithm, network traffic and cloud writes were reduced which increased their cache hit when compared to other existing algorithms [24]. We have enhanced this scoring algorithm.

Tsai and Moh experimented with several Load Balancing (LB) algorithms to manage cache efficiently for CRAN [23]. Among their LB algorithms, Shortest Queue (S_{queue}) algorithm is considered as best performing algorithm. We are going to use the same algorithm for our LB. Their

results show that they can decrease the queue size and service time of cache which reduced the network latency for 5G networks [22, 23].

Kaur and Moh proposed new algorithms for cache management in 5G [11]. In this, for scoring the UEC records, they have used Probability Based Popularity Scoring (PBPS). One such algorithm is Reverse Random Marking (RRM) with PBPS (RRM+PBPS) in which a certain percentage of UECs are marked after it reaches a threshold. If we want to evict the records, we can only evict from unmarked records [11]. If all the records are marked, then increase the threshold and continue the same process [11]. Another algorithm is that their PBPS scoring technique is combined with Hierarchy is called PBPS+Hierarchy in which the cache is partitioned and allocated to users according to their service levels [11]. These algorithms which are simple in design were compared in terms of CHR, latency, and network traffic. This PBPS+Hierarchy algorithm is also extended for our work along with Tsai and Moh's work [11, 22, 24].

Huang, Zhao, and Zhang used cooperative multicast caching mechanism in MEC between base stations to utilize the resources efficiently [9]. Cooperative means if the content is not in the small base station (SBS) instead of accessing it from MBS we can try to access from another SBS. Multicast means instead of serving multiple requests separately and storing it in each SBS, we can multicast the popular videos to all. Thereby saving storage space and decreasing energy consumption. They demonstrated that by caching in the edge, network latency can be reduced, and CHR can be increased [9].

Tran, Hajisami, and Pompili cached in a hierarchical manner [25]. Their research experiment showed us that caching in the edge (RRH) is better than caching in the BBU pool (cloud cache) since RRH is nearer to the users. Their performance metrics were CHR, latency, and backhaul traffic load [25].

3. EXISTING CACHE MANAGEMENT ALGORITHMS

In this section, we are going to discuss some of the existing cache management algorithms for 5G mobile networks. Before that, the following flowchart gives a general idea of the structure

of the algorithms. This flowchart shown in Figure 4 is adapted from Tsai and Moh's work [24] and this flowchart is also used in Kaur and Moh's work [11]. A brief description of the flowchart follows.

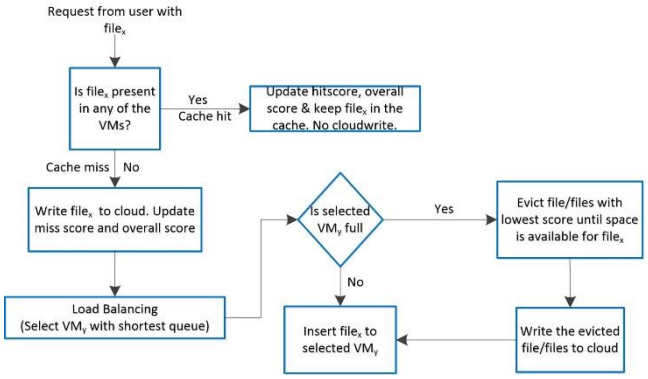

Figure 4. Cache management flowchart.

Please note that if we are using the algorithm or flowchart for BBU caching from CRAN then the "file" is UEC records and if we are using MEC server caching then it is file content. Each user with file or UEC request is incoming and if that file or UEC is already present in any of the Virtual Machine (VM) cache, then it is a cache hit and keeps that file or UEC in the cache. We must update the hit score and calculate the overall score according to the algorithm and there is no need to write in the cloud. On the other hand, if the requested file is not present in any of the VMs then it is a cache miss. We must write that filename in the cloud. After that, we must update the miss score and calculate the overall score according to the algorithm. Now we are trying to add this file to any one of the VMs so that if it is requested in future again it will be present in the cache. To select any one of the VMs, we are using S_{queue} LB algorithm [22, 23]. If the selected VM is full, evict the file with the lowest score until space is available for this file. Write the evicted file or files to the cloud. After eviction when space is available, insert this file into VM [11, 24].

3.1 Least Frequently Used (LFU) Algorithm

LFU algorithm is used as a baseline comparison algorithm in the preliminary results section. LFU is a classic algorithm which keeps track of the number of times files are being

accessed. When it's time of eviction, the least number of accessed or least frequently used files are evicted [15].

3.2 EXD-AHP Algorithm

Tsai and Moh proposed EXD-AHP algorithm which uses a scoring method to decide which files need to be present in the cache and which files need to be evicted from the cache [22, 24]. To calculate the scoring both EXD and AHP weights calculations are considered [4, 17]. For EXD, there is a parameter a in which we can tune it to keep the recently or frequently used files in the cache. Higher the parameter a value, the system keeps recently used files in the cache and smaller the parameter a value it leans towards the frequency of elements [4]. AHP weights are calculated by creating a matrix based on the SLA users [17]. If the file is accessed at time ui1+Δu for the first time after ui1, then scoring is calculated as follows [24]:

$$S_i(ui1+\Delta u) = S_i(ui1)*e^{-a\Delta u}+W_{AHP} \qquad (1)$$

If the file didn't get requested at a time interval [ui1, (ui1+Δu)], then the score is calculated as follows [23]:

$$S_i(ui1+\Delta u) = S_i(ui1)*e^{-a\Delta u} \qquad (2)$$

Where $S_i(ui1)$ is the score (weight), $e^{-a\Delta u}$ is EXD calculation and W_{AHP} is AHP weight. From equation (1) and (2), we can tell that the score of the file depends on both EXD and AHP weight calculations [24].

3.3 PBPS Cache Management Algorithms

Kaur and Moh proposed PBPS algorithms which also use a scoring technique to determine whether a file should be present in VM or not [11]. This scoring is calculated using the equation (3).

$$\frac{hit\ score\ Li}{\#\ requests}\left(1-\left(\frac{hit\ score\ Li}{\#\ requests}\right)\right)^{\frac{miss\ score}{\#\ requests}} \qquad (3)$$

As we can see from equation (3), the score is calculated based on both cache hits and cache misses. At a given point an overall score can be calculated for a file which tells us how popular the file is based on hits and misses. The higher the score, the higher the popularity and less chance of eviction. This file will probably stay inside the cache because of its high PBPS score. The requests are constantly changing so the scores are calculated dynamically. The algorithm uses a rewarding system, in which the quantity of reward is varied according to the SLA users [11].

3.3.1 RRM with PBPS (RRM+PBPS) Algorithm

This algorithm proposed by Kaur and Moh uses PBPS scoring method and in addition to that, it uses RRM [11]. When files are being requested and it is in the cache then it is a cache hit. If the file score exceeds a certain threshold it is marked. Then if we must evict some files to make room for new files, eviction process happens from unmarked files. If all the files are marked, then the threshold value M_t is marginally increased to unmark a certain number of files [11]. Below is the algorithm for RRM +PBPS.

RRM with PBPS (RRM+PBPS) Algorithm

1. For each $file_x$ request;
2. If $file_x$ is present in any one of the VMs; /* cache hit
3. Update PBPS hit score and calculate the overall score
   Using equation (3); (no cloudwrite)
4. Return;
5. If $file_x$'s score exceeds M_t, then mark $file_x$;
6. Else leave it as unmarked;
7. If all the files are marked, then increase M_t, then
   recalculate marked and unmarked files.
8. Else /*cache miss
9. Update PBPS miss score and calculate the overall score
   Using equation (3);
10. Write $file_x$ to the cloud;
11. LB: S_{queue} to select VM

12. If the selected VM has free space for $file_i$;
13. Insert $file_x$ to selected VM;
14. Return;
15. Else if the VM is full;
16. Find $file_y$ which is an unmarked file from the cache;
17. Evict $file_y$;
18. Write $file_y$ to the cloud;
19. Insert $file_x$ in the cache;
20. Return;
21. Update all other files in the cache using equation (3)
With total number request parameter updated;

3.3.2 PBPS+Hierarchy (H-PBPS) Algorithm

This algorithm also uses the PBPS scoring method to calculate the addition and eviction of files in the cache [11]. On top of this, the entire cache is divided, and each cache partition is dedicated to the users according to their SLA. This forms a hierarchy of users. We are giving preferential treatment to the users, so the higher preferred users will get the larger size cache partition. The file addition or eviction happens in their allocated cache partition only [11]. The algorithm logic is similar to the flowchart and PBPS+Hierarchy algorithm is as follows:

PBPS+Hierarchy (H-PBPS) Algorithm

1. For each $file_x$ request with SLA L_i;
2. If $file_x$ is present in any one of the VMs; /* cache hit
3. Update PBPS hit score and calculate the overall score
Using equation (3); (no cloudwrite)
4. Update content of the $file_x$ with L_i in the cache;
5. Return;
6. Else; /* cache miss
7. Update PBPS miss score and calculate the overall score
using equation (3);
8. Write $file_x$ L_i to the cloud;

9. Select one of the VM using S_{queue} LB;
10. If the selected VM has free space for $file_x$;
11. Insert $file_x$ with L_i;
12. Return;
13. Else if the VM is full;
14. Find $file_y$ which has the lowest score;
15. Evict $file_y$ from L_i;
16. Write $file_y$ to the cloud;
17. Insert $file_x$ L_i in the cache;
18. Return;
19. Update all other files in the cache using equation (3)
With total number request parameter updated;

4. PRELIMINARY RESULTS

4.1 Experiment Parameter

For our experiment setup, we have used the CloudSim simulator [1]. This simulator is very popular and effective to cloud based applications. Without worrying about underlying cloud infrastructure, we can build our cloud architecture on top of CloudSim. The parameter values for Tsai and Moh's research are provided by Nokia Lab researchers [24]. For our work, we mostly use their simulation values [24]. The following Table 1 shows the experiment parameter for our simulations [11, 24]. The simulation values used here is not fixed meaning we can change all the parameter values and experiment the simulation results. For example, number of users and number of user requests can be changed. Also, number of virtual machines can be generalized.

Table 1. Simulation Parameters and Values

Parameters	Values
Host and VM	1 and 4
VM Cache Sizes	0.75 GB & 2 GB
Arrival rate of files into the network	1400 files/sec

No of Users and requests	25000 and 420,000
File Sizes	200 KB fixed and distributed, and 2000 KB fixed and distributed using Normal (Gaussian) Distribution
Network Bandwidth	1 Gbps
QoS Level Users	SLA 1: High Mobility; Premium, SLA 2: Low Mobility; Premium, SLA 3: High Mobility; Basic, SLA 4: Low Mobility; Basic
EXD parameter a and LB	10^{-3} and S_{queue}
Analytical Hierarchical Process Weights	SLA1: 0.58; SLA2: 0.28; SLA3: 0.10; SLA4: 0.04
PBPS Hit Rewards	SLA1: 1; SLA2: 0.75; SLA3: 0.5; SLA4: 0.25

4.2 Preliminary Performance Evaluation

In this preliminary performance evaluation section, we made some small changes like changing the parameter of the existing algorithms and analyzed the results. This section results gave us the inspiration to propose new algorithms and in the following section, those proposed algorithms are also tested and analyzed. The performance metrics used in this section are Cache Hit Rate (CHR), Cloud Write Rate (CWR), and Network traffic.

4.2.1 Cache Hit Rate (CHR)

CHR is an important performance evaluation metric for cache management problems. The following results use CHR for its performance measure. For different service levels, CHR can be calculated using the following equation (4) [24]:

$$\text{CHR of } L_i = \frac{\text{Total number of Li cache hits}}{Total\ number\ of\ file\ requests\ from\ Li} \qquad (4)$$

Different values of EXD parameter a

The first experiment we did was to try different values for EXD parameter a. Cache size for this experiment is 0.75 GB and file size is 200 KB distributed. Tsai and Moh's work have 3

different algorithms which use this EXD parameter a [24]. They are EXD, EXD-AHP+1, EXD-AHP. This parameter a value decides whether the recent files need to be placed in the cache or the frequently used files need to be present in the cache. If parameter a value is high the recently accessed files will be kept in cache and vice versa [4, 24]. The experimented different values of parameter a using CHR performance metric is shown in Figure 5. From the result, a = 10^{-3} gives better CHR. So, we used this value for our further experiment. If the system leans towards the recency of elements, then the CHR is very less (a = 10^{-1} and 10^{-2}) and if the algorithm keeps the frequently used files in the cache then the chances of CHR is higher (a = 10^{-3} to 10^{-12}). Among the three EXD algorithms, EXD-AHP algorithms give the highest CHR. So, we chose this algorithm for our enhancement.

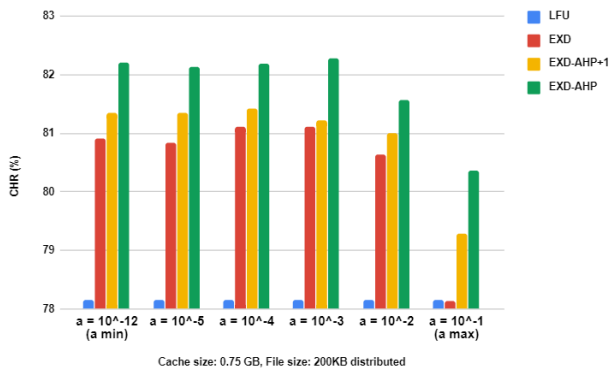

Figure 5: Cache Hit Rate (CHR) (%) for different values of EXD parameter a

Different file sizes

In this, we used different file sizes for EXD-AHP and H-PBPS algorithms [11, 24]. File size fixed means all the requested files will be of the same size, for example, 200 KB fixed means all the file sizes requested will be of 200 KB and distributed means the requested files sizes are varied. For distribution of different file sizes, Gaussian (Normal) distribution is used. In Table 2, different file sizes used for the simulations are displayed. Cache size used for this experiment is 0.75 GB. Figure 6 Comparing 200 KB and 2000 KB file sizes, the small file size gives the highest CHR and comparing fixed and distributed file sizes, fixed file size gives the highest CHR. We can use 200 KB fixed files for UEC of CRAN since it is a metadata of users. Other distributed file sizes can be used for MEC file content caching. Comparing EXD-AHP and H-PBPS algorithms, EXD-AHP performs better because it has higher CHR.

Table 2. Different File Sizes

#	File Size
1.	200 KB fixed
2.	200 KB distributed
3.	2000 KB fixed
4.	2000 KB distributed

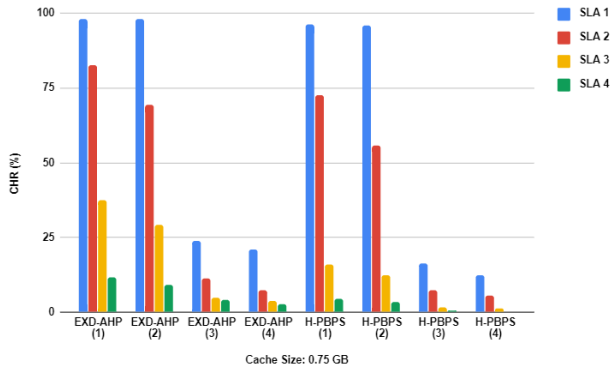

Figure 6: Cache Hit Rate (CHR) (%) for different file sizes

Different cache sizes

Next, we experimented with different cache sizes (0.75 GB and 2 GB). File size used for this experiment is 200 KB fixed and distributed. Table 3 displays the list of different cache sizes used for EXD-AHP and H-PBPS algorithms [11, 24]. CHR for different cache sizes is displayed in Figure 7. Of course, cache size 2 GB gives better CHR than cache size 0.75 GB. As usual, fixed file sizes give better CHR than distributed file sizes. In 0.75 GB cache size, EXD-AHP algorithm gives better CHR than H-PBPS algorithm. In 2 GB cache size, H-PBPS algorithm gives 100% cache hit for SLA 1 and 2 whereas EXD-AHP algorithm gives better CHR for all the SLAs overall. After this experiment, we think that EXD-AHP performs better than H-PBPS so far.

Table 3. Different Cache Sizes

#	File Size	Cache Size
1.	200 KB fixed	0.75 GB
2.	200 KB distributed	0.75 GB
3.	200 KB fixed	2 GB
4.	200 KB distributed	2 GB

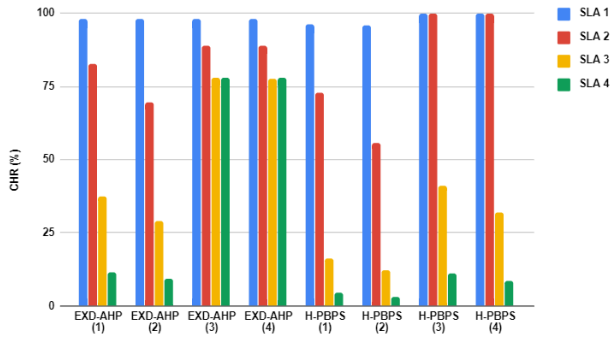

Figure 7: Cache Hit Rate (CHR) (%) for different cache sizes

Different cache management algorithms

In this, CHR for different existing cache management algorithms is evaluated in Figure 8. Cache size used here is 0.75 GB and file size is 200 KB distributed. LFU and RRM+PBPS are used as baseline algorithms. For RRM+PBPS the threshold increase is 10%. For H-PBPS, the default hierarchy partition used is 70% , 20%, 8 %, 2% for SLA 1, 2, 3, 4 users respectively [11]. We have tried different cache size partitions for SLA levels and it is mentioned in Table 4. We can see that if the cache size partition is changed, CHR is also changed accordingly. This result gave us the motivation to do two things. 1. We wanted to apply this hierarchy cache partition to EXD-AHP algorithm and enhance it to perform even better. 2. We wanted to dynamically change the cache partition according to the Minimum Guarantee of CHR set for the different SLA users.

Table 4. Different Cache Distribution

#	CD for SLA1	CD for SLA2	CD for SLA3	CD for SLA4
1.	70 %	20 %	8 %	2 %
2.	55 %	28 %	11 %	6 %
3.	47 %	25%	20 %	8 %

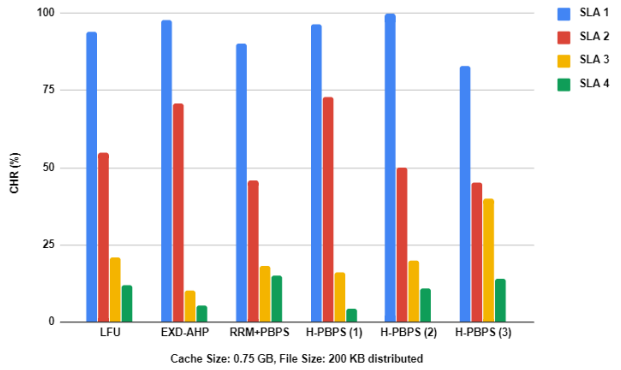

Figure 8: Cache Hit Rate (CHR) (%) for different cache management algorithms

4.2.2 Cloud Write Rate (CWR)

CWR is another performance metric for these cache algorithms. CWR is defined as the number of cloud writes by the number of requests [24]:

$$\text{CWR of } L_i = \frac{\text{Total number cloud writes}}{Total\ number\ of\ file\ arrivals\ in\ Li} \qquad (5)$$

If there is a cache hit for a request, then there is no cloud write. If there is a cache miss, then that file must be written in the cloud and after the addition of that file, any evicted file from cache must be written in the cloud. So, there will be 2 cloud writes [24]. Cloud Write Rate and Cache Hit Rate are inversely proportional to each other. H-PBPS uses different cache distribution from Table 4. Here also we can notice that CWR varies according to the different cache distributions. Figure 9 displays the CWR for different SLA users and CWR is calculated using

equation (5). SLA 1 has zero cloud writes for H-PBPS (2) and SLA 4 has the highest cloud writes for H-PBPS (1). Cache size also has a huge impact on Cloud Write Rate. If the cache size is too small, then almost all the SLAs will have 2 cloud writes. As the cache size grows the Cloud Write Rate will also be decreased [24].

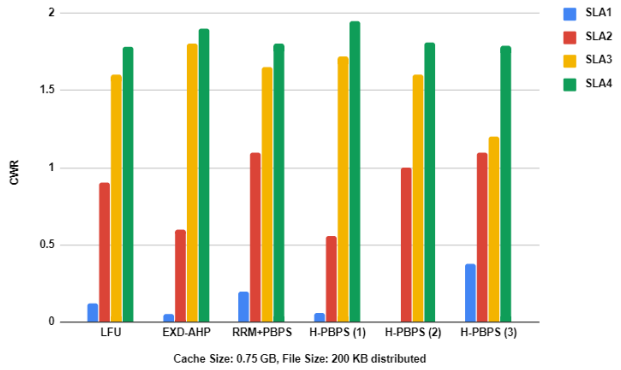

Figure 9: Cloud Write Rate (CWR) for different cache management algorithms

4.2.3 Network traffic

Network traffic is a performance metric which is calculated based on cloud writes. Because whenever there is a cache miss, there is a need for files to travel from cache to cloud which creates network traffic. Assuming each file size is 200 KB on average, it is calculated as follows [24]:

$$\text{Network Traffic of } L_i = \frac{\text{Number of Li cloudwrites} * 200 * 8 * 1000}{Simulation\ Time} \quad (6)$$

Network traffic of Figure 10 is calculated using equation (6). This traffic result is not based on the different SLA users. This is the overall traffic result for different cache management algorithms. CHR and network traffic are inversely proportional and CWR and network traffic are directly proportional because the higher the cloud writes the higher the network traffic going to be. EXD-AHP algorithm has less network traffic overall. Different cache distribution of H-PBPS is used from Table 4. This shows that by changing the cache distribution the traffic can also be decreased. So, this also gave us the idea of applying the hierarchy cache partition to EXD-AHP and try to decrease the traffic even more.

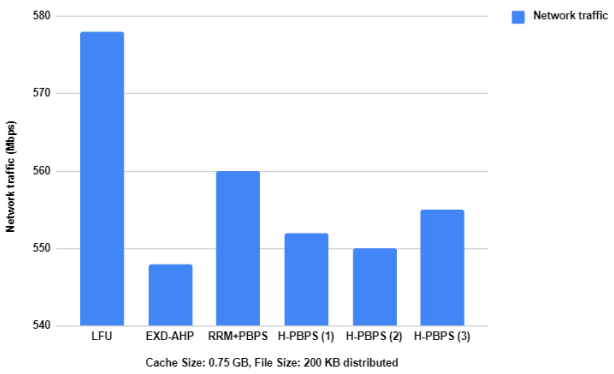

Figure 10: Network Traffic (Mbps) for different cache management algorithms

5. PROPOSED CACHE MANAGEMENT ALGORITHMS

The previous section results gave us the motivation to design the following proposed algorithms. For the first part, inspired by the change in cache distribution of H-PBPS, we applied the hierarchical part to EXD-AHP and designed a new algorithm called H-EXD-AHP algorithm. For the second part, we designed 3 algorithms that dynamically changed the cache distribution to meet the Minimum Guarantee of CHR for differentiated users according to their SLA. We applied these 3 algorithms to H- EXD-AHP and H-PBPS. We will see in detail about those Dynamic Hierarchy (DH) algorithms in this section.

5.1 New: H-EXD-AHP Algorithm

In this algorithm, EXD-AHP scoring is used from the previous section [23]. Inspired by H-PBPS algorithm from the previous section, a hierarchical layer of cache is allocated for differentiated SLA users according to their preferred treatment [6]. The total cache is divided according to the type of SLA users. Of course, premium SLA users have large cache size and basic users are allocated with small cache size. The H-EXD-AHP algorithm is as follows:

H-EXD-AHP Algorithm

1. For each $file_x$ request with SLA L_i;
2. Calculate new EXD-AHP score using equation (4);
3. Update score of remaining files using equation (5);
4. If $file_x$ is present in any one of the VMs; /* cache hit

5. Update score of $file_x$;
6. Return;
7. Else; /* cache miss
8. Write $file_x$ L_i to the cloud;
9. Select one of the VM using S_{queue} LB;
10. If the selected VM has free space for $file_i$;
11. Insert $file_x$ with L_i;
12. Return;
13. Else if the VM is full;
14. Find $file_y$ which has the lowest score;
15. Evict $file_y$ from L_i;
16. Write evicted file to the cloud;
17. Else if $file_x$ score is smaller than the lowest score;
18. Evict lowest scored file from the cache;
19. Write $file_y$ to the cloud;
20. Insert $file_x$ L_i in the cache;
21. Return;

5.2 New: Dynamic Hierarchy (DH) Cache Management Algorithms

5.2.1 DH Algorithm

This algorithm is an enhancement for H-EXD-AHP and H-PBPS algorithms. In those algorithms, the cache size partitioned for SLA users is fixed and cannot be changed dynamically according to the runtime requirement. In the DH algorithm, for a given traffic distribution (TD), the algorithm dynamically adjusts the Cache Distribution (CD) to improve CHR and tries to meet the Minimum Guarantee (MG) of CHR [11]. The minimum guarantee of cache hit rate is for our setting to improve SLA 3 and SLA 4 users. Net neutrality, prioritized treatment will be in common practice.

The first step is identical to either H-EXD-AHP scoring or H-PBPS scoring. For a given TD, and with an initial CD, CHR is measured for each SLA. Minimum Guarantee of Cache Hit Rate is set for different SLA users. While if any one of the SLA's CHR didn't meet the MG (CHR

< MG) then, Surplus is measured for each SLA whose CHR > MG. Next, the surpluses are arranged in increasing order. Then choose the highest surplus L_i so that it can give some of its cache sizes to deficit SLA. Choose the highest preferred SLA, whose CHR didn't meet the MG. Give X % (in our case 20 %) of cache size from surplus CD to deficit CD. Remove X % (20 %) of the surplus CD and update the new cache sizes. Do this until all SLA's MG is met. Return the number of iterations took to achieve the MG to see how fast we can get to the final cache distribution.

DH Algorithm

1. Use H-EXD-AHP or H-PBPS algorithm;
2. Measure CHR_i of each SLA L_i for i = 1,2...n;
3. While at least any one of the $CHR_i < MG_i$ for L_i;
4. Measure $Surplus_i = CHR_i - MG_i$ for each L_i
   where $CHR_i > MG_i$;
5. Arrange $Surplus_i$ in increasing order;
6. Choose L_i, which has the highest $Surplus_i$;
7. Choose L_j, which has the highest SLA preference
   AND $CHR_j < MG_j$ for j = 1,2...n and $i \neq j$;
8. Set $CD_j = CD_j + [CD_i * (X/100)]$;
9. Set $CD_i = CD_i - [CD_i * (X/100)]$;
10. GoTo Step 1;
11. END While; * when $CHR_i >= MG_i$ for each L_i
12. Return;

5.2.2 Improved (I) DH Algorithm

This IDH algorithm is an improved version of DH algorithm, in this instead of borrowing from only one surplus SLA, we are going to borrow from top two highest surplus SLA and give it to deficit SLA users. From the first highest surplus we borrowed X %(in our case 20 %) of its cache size and second highest surplus we borrowed Y % (15 %). Below is the IDH algorithm.

IDH Algorithm

1. Use H-EXD-AHP or H-PBPS algorithm;
2. Measure CHR_i of each SLA L_i for i = 1,2...n;
3. While at least any one of the $CHR_i < MG_i$ for L_i;
4. Measure $Surplus_i = CHR_i - MG_i$ for each L_i
   where $CHR_i > MG_i$;
5. Arrange $Surplus_i$ in increasing order;
6. Choose L_i, which has the highest $Surplus_i$;
7. *Choose L_j, which has the second highest $Surplus_j$ for $j = 1,2...n$ and $i \neq j$;*
8. *Choose L_k, which has the highest SLA preference AND $CHR_k < MG_k$ for $k = 1,2,...,n$, $k \neq i$ and $k \neq j$;*
9. *Set $CD_k = CD_k + [CD_i * (X/100)] + [CD_j * (Y/100)]$ where $X > Y$;*
10. *Set $CD_i = CD_i - [CD_i * (X/100)]$;*
11. *Set $CD_j = CD_j - [CD_j * (Y/100)]$;*
12. GoTo Step 1;
13. END While; * when $CHR_i >= MG_i$ for each L_i
14. Return;

In this, we are Choosing the first and second highest surplus. Choose the highest preferred SLA L_j, whose CHR didn't meet the MG. Give X % (20 %) of CD_i AND Y % (15 %) of CD_j to CD_k where X > Y. Remove X % (20 %) of CD_i from CD_i. Remove Y % (15 %) of CD_j from CD_j and update all cache sizes. End while all SLA's MG is met. Return the number of iterations took to achieve the MG to see how long it takes to execute our algorithms.

5.2.3 Good Guess (GG) DH Algorithm

In this GGDH algorithm, instead of borrowing cache sizes from surplus SLAs, we try to meet the MG of CHR using GG formula. From previous algorithms, it is observed that for a given

TD_i, CD_i is directly proportional to CHR_i for each L_i, $CD_i \alpha CHR_i$. Taking out the proportionality, $CD_i = k_i * CHR_i$.

$$k_i = CD_i / CHR_i \qquad (7)$$

Similarly, to find Good Cache Distribution GCD_i, it is directly proportional to MG_i. $GCD_i \alpha MG_i$. Taking out the proportionality,

$$GCD_i = k_i * MG_i \qquad (8)$$

To get the GCD_i value for each L_i, we must find the value of constant k_i for each L_i from equation (7) and substitute that k_i in equation (8). GGDH algorithm is as follows:

GGDH Algorithm

1. Use H-EXD-AHP or H-PBPS algorithm
2. Measure CHR_i of each SLA L_i for i = 1,2...n;
3. While at least any one of the $CHR_i < MG_i$ for L_i;
4. Calculate k_i using equation (7);
5. Substitute k_i from equation (7) and calculate GCD_i using equation (8);
6. If GCD_i of each L_i sum is not equal to 100 then normalize;
7. GoTo Step 1;
8. END While; * when $CHR_i >= MG_i$ for each L_i
9. Return;

5.3 New: DH-EXD-AHP Cache Management Algorithms

5.3.1 DH-EXD-AHP Algorithm

In the DH-EXD-AHP algorithm, we are adding DH algorithm to H-EXD-AHP and CD is dynamically changed to achieve the MG. Highest surplus gives X% of its CD to the deficit CD in our case 20%.

DH-EXD-AHP Algorithm

1. Use H-EXD-AHP algorithm;
2. Use DH algorithm;

5.3.2 IDH-EXD-AHP Algorithm

This algorithm is like IDH algorithm. The only difference is H-EXD-AHP scoring is added to IDH algorithm.

IDH-EXD-AHP Algorithm

1. Use H-EXD-AHP algorithm;
2. Use IDH algorithm;

5.3.3 GGDH-EXD-AHP Algorithm

GGDH algorithm for EXD-AHP is as follows:

GGDH-EXD-AHP Algorithm

1. Use H-EXD-AHP algorithm;
2. Use GGDH algorithm;

5.4 New: DH-PBPS Cache Management Algorithms

The following DH-PBPS cache algorithms are like DH-EXD-AHP algorithms except in these we are using PBPS scoring techniques. The algorithms are as follows:

5.4.1 DH-PBPS Algorithm

DH-PBPS Algorithm

1. Use H-PBPS algorithm;

2. Use DH algorithm;

5.4.2 IDH-PBPS Algorithm

IDH-PBPS Algorithm

1. Use H-PBPS algorithm;
2. Use IDH algorithm;

5.4.3 GGDH-PBPS Algorithm

GGDH-PBPS Algorithm

1. Use H-PBPS algorithm;
2. Use GGDH algorithm;

6. PERFORMANCE EVALUATION OF PROPOSED CACHE MANAGEMENT ALGORITHMS

In this section, the performance evaluation of proposed algorithms is discussed. First, we applied hierarchical structure to EXD-AHP algorithm and designed H-EXD-AHP. We can now compare this new algorithm with H-PBPS and see which performs better. Next, we are going to analyze the 3 Dynamic Hierarchy (DH) algorithms for H-EXD-AHP and H-PBPS.

6.1 Cache Hit Rate for H-EXD-AHP algorithm

CHR for H-EXD-AHP is displayed and compared with H-PBPS algorithm in Figure 11 and Figure 12. Table 4 gives the different cache partition used for the algorithms. Figure 11 uses a small cache size and small distributed file size (that is 0.75 GB and 200 kB distributed) and Figure 12 uses 2 GB as cache size and 2000 KB distributed file size. The problem we notice with DH-PBPS algorithms is it gives good CHR for SLA 1 and SLA 2 but for SLA 3 and SLA 4 its CHR is less. Whereas H-EXD-AHP algorithm performs better because CHR is higher for all the SLA users when compared with H-PBPS algorithms.

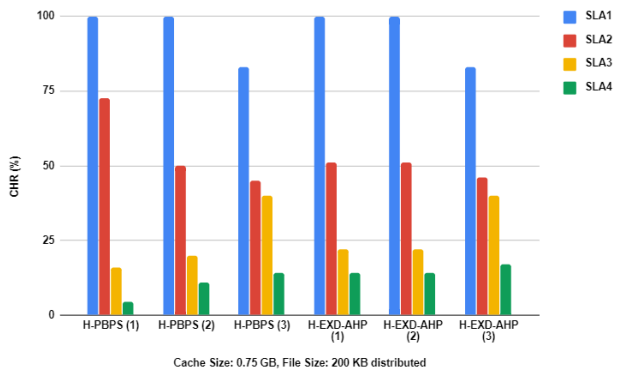

Figure 11: Cache Hit Rate (CHR) (%) for H-EXD-AHP (Cache Size 0.75 GB, File Size 200 KB distributed)

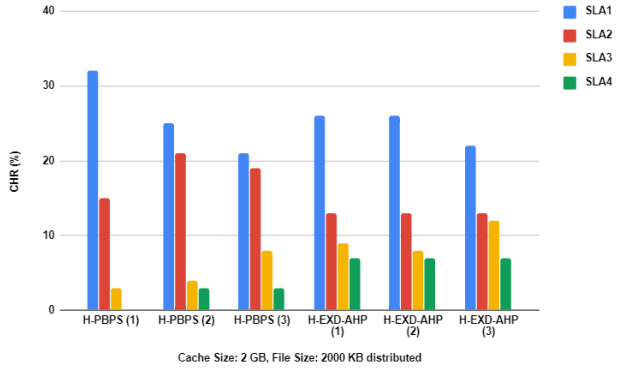

Figure 12. Cache Hit Rate (CHR) (%) for H-EXD-AHP (Cache Size 2 GB, File Size 2000 KB distributed)

6.2 Number of iterations for DH algorithms

This subsection returns the number of iterations it took to achieve the Minimum Guarantee. The minimum guarantee of cache hit rate is for our setting to improve SLA 3 and SLA 4 users. Because in existing algorithms more importance is given to SLA 1 and SLA 2 users. Net neutrality, prioritized treatment will be in common practice. Because we need to know how fast the algorithm can change its cache distribution dynamically. Also, to recall, Dynamic Hierarchy borrows from the highest surplus (X % = 20 % in our case) and gives it to deficit cache partition. Improved Dynamic Hierarchy borrows from first two highest surpluses (X % = 20 %, Y % = 15 % where X > Y) and gives it to deficit cache size.

6.2.1 DH-EXD-AHP Cache Results

Initial Cache Distribution**:** 25% for each SLA. For Table 5, traffic distribution is 25% for each SLA and MG is varied. It displays the number of iterations needed to reach the MG for different DH algorithms. For Table 6, MG is 60% for SLA1, 50% for SLA2, 35% for SLA3 and 30% for SLA4 and traffic distribution is varied. Like Table 5, this also displays the number of iterations needed to reach MG for different dynamic cache algorithms.

Table 5. DH-EXD-AHP: No. of Iterations (Different Minimum Guarantee)

Minimum Guarantee of Cache Hit Rate (%)	95, 50, 20, 10	80, 45, 35, 15	70, 40, 30, 20	60, 50, 35, 30
DH-EXD-AHP	8	5	4	4
IDH-EXD-AHP	6	4	4	3
GGDH-EXD-AHP	2	2	2	2

Table 6. DH-EXD-AHP: No. of Iterations (Different Traffic Distribution)

Traffic Distribution (%)	95, 50, 20, 10	80, 45, 35, 15	70, 40, 30, 20	60, 50, 35, 30
DH-EXD-AHP	5	4	5	4
IDH-EXD-AHP	4	4	4	4
GGDH-EXD-AHP	2	2	2	2

6.2.2 DH-PBPS Cache Results

Following tables gives us the number of iterations required to meet MG for the proposed DH algorithms. For this experiment, initial cache size distribution is considered as 25% for each SLA. In Table 7, TD is 25% for each SLA and MG is varied. In Table 8, MG is 60% for SLA1 users, 50% for SLA2 users, 35% for SLA3 users, 30% for SLA4 users and TD is varied. Results indicate that GGDH algorithm performs better than both DH and IDH algorithms. Comparing

Dynamic Hierarchy of EXD-AHP and PBPS algorithms, EXD-AHP Dynamic Hierarchy algorithms have a smaller number of iterations.

Table 7. DH-PBPS: No. of Iterations for (Different Minimum Guarantee)

Minimum Guarantee of Cache Hit Rate (%)	95, 50, 20, 10	80, 45, 35, 15	70, 40, 30, 20	60, 50, 35, 30
DH-PBPS	9	5	4	5
IDH-PBPS	7	4	3	3
GGDH-PBPS	2	2	2	2

Table 8. DH-PBPS: No. of Iterations for (Different Traffic Distribution)

Traffic Distribution (%)	30, 25, 15, 30	30, 20, 15, 35	25, 15, 10, 50	30, 20, 10, 40
DH-PBPS	6	4	6	4
IDH-PBPS	4	5	5	4
GGDH-PBPS	2	2	2	2

7. CONCLUSION

This paper adopted two scoring algorithms as follows: 1. H-PBPS, 2. EXD-AHP [11, 23]. We changed the cache partition for H-PBPS algorithm [11]. Inspired by this algorithm, we proposed H-EXD-AHP algorithm which leads to DH algorithms. In these DH algorithms, using the hierarchy of SLA users, we dynamically changed the cache size distribution. The algorithms can adapt to the changing MG needs. The minimum guarantee of cache hit rate is for our setting to improve SLA 3 and SLA 4 users. Because the existing algorithms has more cache hit rates for SLA 1 and SLA2. In reality, Net neutrality, prioritized treatment will be in common practice. Comparably DH-EXD-AHP algorithms performed a little bit better. Among three DH algorithms,

GGDH algorithm gave us the smaller number of iterations. Currently, we are only changing the cache distribution of different SLA users dynamically. Throughout the simulation, the traffic distribution remains the same. We have changed the traffic distribution manually and run the simulation. So, for future work, to change the traffic distribution dynamically for different SLA users can be considered. Cache management problem in 5G is a very interesting area. For future work, other 5G technologies such as millimeter wave, MIMO and other areas such energy efficiency, security can also be explored.

REFERENCES

[1] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. 2011. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience* 41, 1 (2011), 23–50.

[2] Checko, Aleksandra, et al. "Cloud RAN for mobile networks—A technology overview." IEEE Communications surveys & tutorials 17.1 (2015): 405-426.

[3] Tejaswiini Choudharik, Melody Moh, and T.-S. Moh, "Prioritized Task Scheduling in Fog Computing," in *Proc. of the ACM Annual Southeast Conference (ACMSE),* Richmond, KY, Mar 2018.

[4] Floratou, A., et al. "Adaptive Caching in Big SQL using the HDFS Cache," Proceedings of the Seventh ACM Symposium on Cloud Computing (SoCC'16), New York, NY, USA, 321-333, 2016.

[5] Gao, L., and M. Moh., Joint Computation Offloading and Prioritized Scheduling in Mobile Edge Computing. In *Proceedings of IEEE International Conference on High Performance Computing and Simulation*, Orleans, France, July 2018.

[6] Gomes, Andre, Torsten Braun, and Edmundo Monteiro. "Enhanced caching strategies at the edge of lte mobile networks." IFIP Networking Conference (IFIP Networking) and Workshops, 2016.

[7] T. Hou, G. Feng, S. Qin, and W. Jiang, "Proactive Content Caching by Exploiting Transfer Learning for Mobile Edge Computing," *GLOBECOM 2017 - 2017 IEEE Global*

Communications Conference, Singapore, 2017, pp. 1-6. doi: 10.1109/GLOCOM.2017.8254636

[8] J. Huang, K. Wu, and M. Moh. 2014. Dynamic Virtual Machine migration algorithms using enhanced energy consumption model for green cloud datacenters. *2014 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 902–910.

[9] X. Huang, Z. Zhao, and H. Zhang, "Cooperate Caching with Multicast for Mobile Edge Computing in 5G Networks," 2017 IEEE 85th Vehicular Technology *Conference (VTC Spring)*, Sydney, NSW, 2017, pp. 1-6.

[10] Karneyenka, U., Mohta, K., and Moh, M. "Location and Mobility Aware Resource Management for 5G Cloud Radio Access Networks," 2017 Inf. Conference on High Performance Computing and Simulation (HPCS), Genoa, Italy, July 2017.

[11] Gurpreet Kaur, and Melody Moh. Cloud Computing Meets 5G Networks: Efficient Cache Management in Cloud Radio Access Networks. Proceedings of The 2018 Annual ACM Southeast Conference, Richmond, Kentucky, March 2018.

[12] T. Li, C. S. Magurawalage, K. Wang, K. Xu, K. Yang, and H. Wang, "On Efficient Offloading Control in Cloud Radio Access Network with Mobile Edge Computing," *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, Atlanta, GA, 2017, pp. 2258-2263.

[13] D. Lin, Y. Hsu, and H. Wei, "A Novel Forwarding Policy under Cloud Radio Access Network with Mobile Edge Computing Architecture," *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*, Washington, DC, 2018, pp. 1-9.

[14] Moh, M. and R. Raju. (Invited Paper) Machine learning techniques for security of Internet of Things (IoT) and fog computing systems. *Proceedings of IEEE International Conference on High Performance Computing and Simulation*, Orleans, France, July 2018.

[15] Podlipnig, Stefan, and Laszlo Böszörmenyi. "A survey of web cache replacement strategies." ACM Computing Surveys 35.4 (2003): 374-398.

[16] Reguri, V. R., Kogatam, S., and Moh, M. "Energy Efficient Traffic-Aware Virtual Machine Migration in Green Cloud Data Centers," Proceedings of the Second IEEE International Conference on High Performance and Smart Computing, New York, April 2016.

[17] R. Saaty, “The analytic hierarchy process—what it is and how it is used,” Mathematical Modelling, vol. 9, no. 3–5, pp. 161 – 176, 1987.

[18] S. Sathyanarayana, and M. Moh, Joint Route-Server Load Balancing in Software Defined Networks using Ant Colony Optimization, Proceedings of the International Conference on High Performance Computing and Simulation (HPCS), Innsbruck, Austria, July 2016.

[19] B. Shahriari, and M. Moh. “Intelligent Mobile Messaging for Urban Networks.” Proceedings of the 12th IEEE Int. Conf. onWireless and Mobile Computing, Networking and Communications (WiMob), New York, October 17-19, 2016.

[20] B. Shahriari, M. Moh, and T.-S. Moh, "Generic Online Learning for Partial Visible Dynamic Environment with Delayed Feedback," in *Proc. of the International Conference on High Performance Computing and Simulation (HPCS)*, Genoa, Italy, July 2017.

[21] Gary Su, and Melody Moh. 2018. Improving Energy Efficiency and Scalability for IoT Communications in 5G Networks. *Proc. of 12th ACM Int. Conf. on Ubiquitous Information Management and Communication (IMCOM),* Langkawi, Malaysia, Jan 2018.

[22] Tsai, C., and Moh, M. “Abstract: Cache Management and Load Balancing for 5G Cloud Radio Access Networks,” ACM Symposium on Cloud Computing, Santa Clara, USA, Sept 2017.

[23] C. Tsai, and M. Moh, "Load balancing in 5G cloud radio access networks supporting IoT communications for smart communities," *2017 IEEE Int. Symposium on Signal Processing and Information Technology (ISSPIT)*, Bilbao, 2017, pp. 259-264.

[24] Tsai, C., and Moh, M. "Cache Management for 5G Cloud Radio Access Networks," Proceedings of ACM International Conference on Ubiquitous Information Management and Communication, Langkawi, Malaysia, January 2018.

[25] X. Tuyen Tran, A. Hajisami, D. Pompili, "Cooperative hierarchical caching in 5G cloud radio access networks", *IEEE Netw.*, vol. 31, no. 4, pp. 35-41, Jul. 2017.

[26] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A Survey on Mobile Edge Networks: Convergence of Computing, Caching, and Communications," in *IEEE Access*, vol. 5, pp. 6757-6779, 2017.