Saving Energy on Edge: In-Memory Caching for Multi-Tier Heterogeneous Network

Saving Energy on Edge: In-Memory Caching for Multi-Tier Heterogeneous Network

Jianwen Xu, Kaoru Ota, and Mianxiong Dong

Abstract

Recent years have witnessed billions of new manufactured sensors, equipments and machines being connected to our almost omnipotent Internet. While enjoying the comfort and convenience brought by Internet of Things (IoT), we also have to face tremendous energy consumption and carbon emissions which even cause climate deterioration. Extended from cloud computing, edge/fog computing and caching provide new thoughts on processing big data generated from distributed IoT devices. With the purpose to help deal with the data explosion problem by edge caching, in this paper we apply in-memory storage & processing to reduce energy consumption. We design two kinds of Time to Live (TTL) in four cache replacement policies to cache data at edge. We carry out simulation experiment in a 3-tier heterogeneous network structure using Random Waypoint (RWP) model and test the performance of in-memory caching and traditional method. The analysis results manifest that our in-memory method is able to obtain better energy efficiency in edge caching and has stable & low backhaul rate.

Introduction

From wearable devices to house furnishings, from vehicles to industrial facilities, nearly all that we ever regarded as *things* have been sharing the same comfort and convenience brought by the Internet. As a result, we are already living in a new Internet of Things (IoT) era. Since user/client volume is generally taken as a metric to show the network size, this time with the addition of devices and machines, our Internet may have to face the network connections and data quantity in the near future several times of the present. Moreover, as big data generated by countless IoT devices is in high demand on real-time processing, which means we have to spend less time on much larger amount of data with small single computations. Such condition calls for additional requirement in data transmission & processing and brings huge energy consumption and carbon emissions leading to climate deterioration such as greenhouse effect.

Together with the development of big data, cloud computing provides new thought of sharing computer processing resources. Nowadays we are able to pay rental and use computing resources with not need to purchase high-priced servers or data centers, just like borrowing books from library. However, traditional cloud computing may not be suitable for handling distributed computational tasks in IoT scenarios. Considered as extension of cloud computing, edge/fog computing attaches much importance to making real-time responses to massive users in which most requests require no complex calculation and can be processed by small devices at the edge of network. While in conventional method all computing tasks are allocated to a limited number of dedicated servers, with edge computing we are able to rely on a mass of mobile devices, routers and other dedicated edge devices to send immediate feedback to nearby requests. Therefore, as wanted contents have already been cached locally, edge computing can perform tasks without visiting remote data centers and reduce time cost and energy consumption in procedures of transmission and propagation.

In computing, cache is a temporary place for information or data storage, and caching refers to the process of storing and reading in a cache. Just like modern CPU cache structure and web cache technology, the cache can serve as an inherent trade-off to help balance the gap between different processing speeds and storage sizes. As a result, storage capacity provided by caching can improve computational performance by reducing extra latency and energy consumption.

Although devices in edge computing scenario may not own multi-level CPUs or web cache softwares, caching can still occur in divided storage space with suitable strategies and methods. In conventional ways we can cache data or contents in disk, which means there is no need to forward it upwards when a request asks for anything already saved in current edge device.

In-memory, which is broadly regarded as the RAM (Random Access Memory) inside a computer, stands for high-speed operation capability compared to NVM (Non-Volatile Memory) like hard disk drive (HDD). Processing through in-memory can save a lot of time spent on I/O operations as well as energy consumption. Since the common storage volume of in-memory is much less than disk, we need better space utilization and scheduling while designing caching method with in-memory. Additionally in IoT scenarios we do not need single edge device to cache too many contents since IoT data is easy to be out of date. That is to say, in-memory processing would be very appropriate for edge caching and improve energy efficiency of edge computing.

In this paper, we will focus on the solutions of edge caching based on in-memory processing from mathematical modeling, caching method designing to simulation and analysis. The main contributions are as follows.

- Design a 3-tier heterogeneous network structure using Random Waypoint (RWP) model;
- Design two in-memory edge caching methods based on two kinds of Time to Live (TTL);

- Consider four cache replacement policies including First In First Out (FIFO), Least Frequently Used (LFU), Least Recently Used (LRU), Random Replacement (RR);
- Choose total energy consumption and backhaul rate as metrics to compare and analyze experimental results of in-memory edge caching and conventional disk method under the same simulation setups.

This paper is divided into six sections to cover all aspects of the research. We introduce the background and sort out the whole work flow. We present related works on edge caching and in-memory processing. We set up the mathematical model and elaborate the details of raised problem. We propose the designed in-memory edge caching methods using four replacement policies. We give results of simulation and comparative analysis between in-memory edge caching method and conventional method. We summarize the previous work and draw conclusions.

Related Work

In this section, we present some related works about edge computing, edge caching, and then introduce some researches on in-memory storage and processing.

Edge Computing and Edge Caching

Cloud services have long remained a part of people's lives, ever since cloud computing became known in 2005 and was quickly utilized in a wide variety of fields. Together with the current IoT boom, in the come-at-able 2020, total amount of data created by any device will reach 600 zettabytes (ZB) per year while annual global data center IP traffic will only be 15.3 ZB at the end of this decade [1]. As a result, in the near future, we are no longer able to put all computing tasks on the cloud and pin our hopes on continuously updating hardware levels, increasing the number of end equipments. We need edge devices to share the workload and solve the bottleneck in data transmission and processing [2].

Edge computing, before attracting wide attention and extensively applied among research institutions, is already studied by a number of technology companies, such as the key players including *Cisco Systems Inc.* and *HP*, etc. Early in 2012, Bonomi *et al.* from *Cisco* start from making clear the position of edge computing in IoT era and prove that fog owns the characteristics of serving as platforms for IoT services from connected vehicle, smart grid to smart city. They define the fundamental characteristics as low latency & location awareness, widespread geographical distribution, mobility, large numbers of devices, predominant role of wireless connection, streaming and real-time applications and heterogeneity [3]. Vaquero*et al.* from *HP* offer a comprehensive definition *the fog* to include cloud, sensor network, peer-to-peer network, etc. They also combine Network Function Virtualization (NFV) and Software-Defined Networking (SDN) together to achieve a new *Softwareisation* network management [4].

Many works on edge computing in recent years focus on interdisciplinary researches and try to find the relation with other fields to help promote common development. Liu *et al.* design a device-to-device video recovery system based on heterogeneous network for picocell edge users. In the paper they discuss the possibilities of achieving the video on demand (VoD) application to improve the current performance [5]. As a branch discipline, mobile edge computing pays more attention on wireless communication among smartphones, tablets and other hand-held devices. Sardellitti *et al.* consider a multiple-input and multiple-output (MIMO) multicell system and design a whole set of joint optimization algorithms for mobile edge computing [6]. Research group from the European Telecommunications Standards Institute (ETSI) regards mobile edge computing to an independent field of study and combine with the fifth generation (5G) mobile networks. They also analyze the market drivers and business value of mobile edge computing services [7].

In order to further utilize edge devices to balance the workload in the expanding network, caching on edge can make a contribution on reducing bandwidth usage, server load and so on. Researches on edge caching also have many different directions. Early in 2005 before cloud computing entering the public consciousness and widely applied in production and living, Ramaswamy *et al.* propose the idea of building cache clouds to deal with documents in edge networks. In the paper they design a dynamic hashing scheme to improve document placement in cache clouds [8]. Gabry *et al.* put forward a maximum-distance separable (MDS) encoded caching scheme to achieve energy-efficient edge computing in heterogeneous network [9]. In recent years, with fast development of wireless communication, caching on mobile/wireless edge becomes a research hotspot. Liu *et al.* summarize the design aspects and challenges of wireless edge caching. They focus on two key features of content delivery traffic and compare the performance of caching at base stations and users [10].

In-Memory Storage and Processing

Compared by disk storage like HDD, flash memory and faster Solid-State Drive (SSD), in-memory or main memory mainly refers to volatile RAM could spend the same amount of time while reading/writing data regardless of physical location. Even though still limited by fault-tolerance, consistent power supply and high manufacturing cost, from all kinds of electronic equipments, personal computers, to large professional servers, we still can not rely on main memory to store our data for long time. However, the last decade has witnessed rapidly decreasing cost of main memory and growing demand of high-speed computing which makes it possible for turning in-memory into the new disk.

Related works on in-memory storage and processing involve different levels from application domain analysis, technical breakthrough to business development prospect. Zhang *et al.* introduce the recent years' development in in-memory big data management and processing. In the paper they classify and summarize all existing commercial and academic management systems for in-memory operations [11]. Beneventi *et al.* apply in-memory processing tools to help do machine learning in High-Performance Computing (HPC) infrastructure models [12].

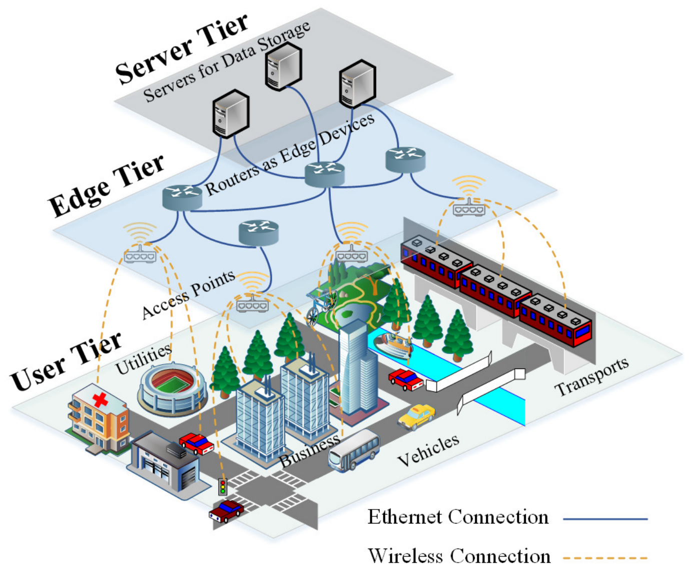


Fig. 1. A 3-tier heterogeneous network structure

Problem Formulation

In this section, we design a 3-tier heterogeneous network structure as the experimental scenario for simulate in-memory edge caching for big data. As shown in Fig. 1, from top to bottom a 3-tier network model can be described as follows.

- *Server Tier:* in this tier multiple servers play the tole of cloud data centers, each file is only stored in one of the servers;
- *Edge Tier:* in this tier we use routers both as forwarders and edge devices which can cache files in packets passed by;
- *User Tier:* a tier made of user nodes that keep moving randomly and requesting files originally stored in servers or cached in routers.

System Outline

In our designed network structure, user nodes are moving in the *RWP* model which is one of the most popular mobility models applied in mobile ad hoc network (MANET) [13]. In Fig. 1, user nodes (u_1, u_2, ..., u_n) in *User Tier* send packets to request files at random time intervals and move to next positions under normal distribution before next sending. Each file, in an unfixed size, is randomly saved in one of the servers (s_1, s_2, ..., s_n) in *Server Tier*. Then in *Edge Tier*, routers (r_1, r_2, ..., r_n) as edge devices will check if the needed files are already cached in storage before forwarding to neighbor r or upward to s. Since information or contents are easy to be outdated, once the original files in *Server Tier* being modified or deleted, all cached copies become useless. That is to say, edge caching need to consider Time to Live (TTL) to make sure that most requests be satisfied with unexpired files [14]. Transmissions between *User Tier* & *Edge Tier* are wireless broadcasting, those inside *Edge Tier* are wired broadcasting and those between *Edge Tier* & *Server Tier* are wired point-to-point.

Normally RAM is associated with volatile types of memory whose data storage would be lost when power is off. That is to say, in-memory may not support long time storage which can be suitable for scenarios of edge caching. As a result, we consider both TTL for cached data and consistency of in-memory storage in designing caching methods.

Performance Metrics

In face of mass date generated by billions of IoT devices, we always prefer less energy consumption on data transmission and processing. For this reason, edge caching aims at reducing repeated data transmission from original servers which means unnecessary energy consumption on packet delivery between *Edge Tier* and *Server Tier* can be saved. Moreover, caching itself also consumes extra energy while keeping RAM or disk memory running. To determine and sum up the overall cost of the entire simulation on the 3-tier network architecture, we may consider 3 parts. The part to maintain all devices in 3 tiers is not expressed in the equation since it is a fixed cost and can only be reduced by shutting down some devices [9]. In summary, we use two equations to present the calculation of total energy consumption.

$$E_{total} = E_{cache}(t) + E_{trans}$$

$$E_{cache}(t) = \omega \sum_{i=1}^{n} (s_i^{cache} t_i)$$

in which $E_{cache}(t)$ and E_{trans} respectively stands for caching energy cost and transmission energy cost. $E_{cache}(t)$ is running time correlated and can be calculated by energy consumption per byte ω. n represents the times when the current caching size of all r in *Edge Tier* is changed. Thus we sum up the n products of variational caching sizes s_i^{cache} and their duration t_i.

$$\begin{aligned} E_{trans} =& E_{send} + E_{recv} \\ &= \sum_{j=1}^{x} (m_{send} s_i^{trans} + b_{send} + m_{recv} s_i^{trans} + b_{recv}) \end{aligned}$$

Comparatively, E_{trans} has no relation to time and only depends on size of data being transmitted s_i^{trans}. Here we separately calculate the energy consumption while devices in three tiers sending and receiving packets. Moreover, as a heterogeneous network model, communications between *User Tier* & *Edge Tier* and among r inside *Edge Tier* are regarded as wireless while those from *Edge Tier* to *Server Tier* and backward are Ethernet transmissions. m and b are linear coefficients obtained from experimental results [15].

To figure out the total energy cost as close as possible to the practical situation, we take some more details into consideration. First, different bit rates of wireless and Ethernet transmissions. Second, the wave propagation speed, we respectively choose speed of light and thick coax as the communication media for wireless and wired.

Besides energy consumption, we also pay attention to how edge caching help reduce workload on end servers. We add a backhaul rate as another metric to test what is percentage may in-memory edge caching takes in completing the task of fetching files from servers across tiers.

In-memory Edge Caching Method

In this section, we propose two caching methods based on different TTL designs, TTL of requested times (TRT) and caching time (TCT).

TRT is counting how many times a cached file being requested by u in *User Tier*. When the needed file being found at any s in *Server Tier* and sent back, each r in the full path may check if it already has the copy of the file. If not, r will cache the file in its in-memory or disk. Later once a r receives request and finds the needed file in cached data, it may send back the copy and count the requested times of the cached file. If number of requested times reaches the maximum value being set, the cached file will be dropped. Accordingly in the calculation equation of $E_{cache}(t)$ of last section, s_i^{cache} is changed and a s_{i+1}^{cache} is needed.

Rather than counting requested times, the other caching method TCT keeps a timer for each cached file. Caching and routing follow the same rules of the first method, similarly when any timer reaches the maximum value, the cached file will be dropped.

In addition, both methods consider the volume of in-memory/disk storage, that is, if the next file to be cached exceeds the memory volume of r, before caching the newcome data we have to free up some space by popping out cached data. As result, to decide which one to drop when the volume of disk or in-memory is full, we apply four common cache replacement policies based on different ideas on how to improve the performance of edge caching. Another point to note is, the drop behaviors in cache replacement policies have no relation to TTL designs since both are needed to guarantee the usability of cached data, that is, still alive and within the capacity of the current r.

First, First In First Out (FIFO) policy regards volume disk/in-memory as a FIFO queue and drops the head of queue that gets pushed in the earliest when queue is full. Second, Least Frequently Used (LFU) policy does not consider the order of cached files and chooses the cached file with the least requested times currently. Third, similar to LFU, Least Recently Used (LRU) also focus on the cached files which are not so popular but chooses the least recently used one to drop. Last, the Random Replacement (RR) policy serves as a contrast to compare the performance of TTL designs with the other policies.

Simulation and Analysis

In the section, we carry out experimental simulations to compare the performance of edge caching in in-memory and conventional disk memory under two TTL designs.

The simulation scenario is a 10 km^2 square open area, and we set 2,000 user nodes in *User Tier* with random initial positions. Each user node randomly moves to next position under normal distribution after sending request packet to fetch one of the 200 files originally stored in one of the five servers in *Server Tier*. We use 100 routers in *Edge Tier* as edge devices to cache the files locally. Four cache replacement policies are applied in designing edge caching methods.

As shown in Table I, the setups of experiment include bit rate & wave propagation speed of packet transmission, *Edge Tier*'s device settings and energy consumption coefficients of both transmission and caching. As a result, to calculate the total energy consumption E_{total}, we have to count the number of packets and sum up their sizes, then compute the time cost from transmission and reading/writing from disk/in-memory. We carry out 10 rounds of simulations in two designed edge caching methods & four cache replacement policies. We repeat the part of each method & policy 10 times and get the average results. The simulation environment is *MATLAB R2017b*.

TABLE I
Experiment Setups

Bit Rate of Transmission	
Wireless (802.11ad)	6.8 *Gbps*
Ethernet	10 *Gbps*
Wave Propagation Speed of Transmission	
Wireless (air)	c (speed of light)
Ethernet (thick coax)	0.77 c
Device Settings of Edge Tier	
MTR of Disk (SSD)	2500 *MB/s*
MTR of In-Memory (DDR3)	6400 *MB/s*
Disk Volume	256 *MB*
In-Memory Volume	25 *MB*
Energy Consumption Coefficients	
	10^{-8} *J/MB* 10^{-6} *J*
Broadcast Send	2.1 × size + 272
Point-to-Point Send	0.48 × size + 431
Broadcast Receive	0.26 × size + 50
Point-to-Point Receive	0.12 × size + 316
Power Consumption of Caching	8 ×10^{-3} *W/MB*

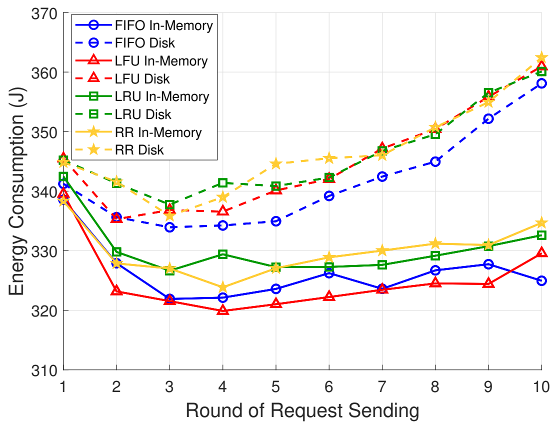


Fig. 2. Total Energy Consumption of Edge Caching in TTL of Requested Times (TRT)

Fig. 2 shows the simulation result of total energy consumption of TRT method in four cache replacement policies. Blue, red, green and yellow represent the policies of FIFO, LFU, LRU and RR. The solid lines and dotted lines respectively stand for caching in in-memory storage and disk storage. From the patterns of eight broken lines, we can know that the overall energy consumption of traditional disk caching is larger than in-memory caching. In our 10 rounds of simulation, the trends of disk and in-memory methods are also different. Energy consumption of disk in TRT varies like a checkmark symbol which firstly falls down a little and then after a smooth transition period increases rapidly to a high value. Our in-memory method in TRT shows a similar trend at the first half from round 1 to 5 but become steady after then which means energy consumption on edge caching may finally turns into a stable state. The differences of the trends are in line with our expectation since disk which own larger storage volumes and lower MTR call for more unit energy consumption and longer operation time. The positive correlation of the two factors lead to the continuing growth of total consumption.

Different cache replacement policies also have respective functions on the results of total energy cost. From the four lines of disk caching in the top half of the figure we can see three of them, LFU, LRU and RR fluctuate until coinciding in the end. Only blue line of FIFO policy shows a slight advantage in total energy consumption which means a simpler rule maybe more suitable for disk caching by TRT. Then from the other four colorful lines of in-memory caching, although still some changes in order of energy cost values, we may get the overall order of four policies as RR, LRU, FIFO and LFU. RR policy does suffer from random choice of dropping cached files which costs more energy in total. The special case of round 4 when LRU exceeds RR may be explained by some occasional error when RR just drops the right files which are not requested much later. LFU policy seems to own the optimal performance in in-memory caching by TRT because of the most suitable dropping choice which focuses on reserving popular files probably from the server directly connected to the current router with only one hop and dropping a most unpopular one which may come from a remote server after several hops of forwarding. Green line applying LRU also pays attention to the popularity of cached files but directly considers the caching time and drops the one not being requested of the longest interval. However, compared to LFU, the practice of LRU may face exceptions like some popular files being dropped due to the timing that the other files are just being cached or requested.

The experimental result of the other TTL design is shown in Fig. 3. In comparison with TRT, the trend of disk's four colorful lines of different cache replacement policies is relatively smoother, gaps between each two of the policies are visible. However, the order of lines in different policies us some kind of abnormal. Green line applying LRU shows the highest total energy consumption and RR policy even achieve high performance in value. In our point of view, since disk have far larger storage volume than in-memory, although more files are cached live for enough time, not many of them finally waited for a request or even get a remote request which may save no time than fetching the original file from servers. As a result, replacement policies trying to delay the time for dropping popular files may in turn cause more energy cost.

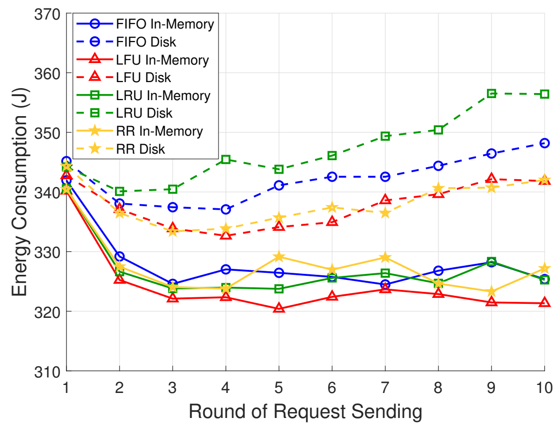


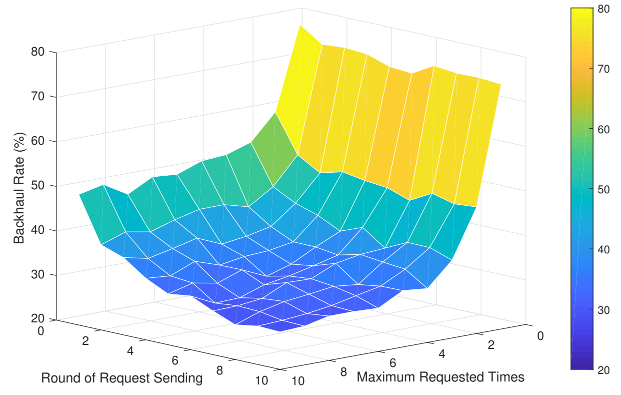
Fig. 3. Total Energy Consumption of Edge Caching in TTL of Caching Time (TCT)

The patterns of in-memory by TCT are similar with TRT. RR policy fluctuates more violent. LRU seems to improve its performance compared to TRT and even shows a certain degree of convergence with FIFO. In our point of view, replacement policy here share the same method with TCT by dropping cached files by keeping check the their timestamps. The collective effect may lead to some promotion in efficiency.

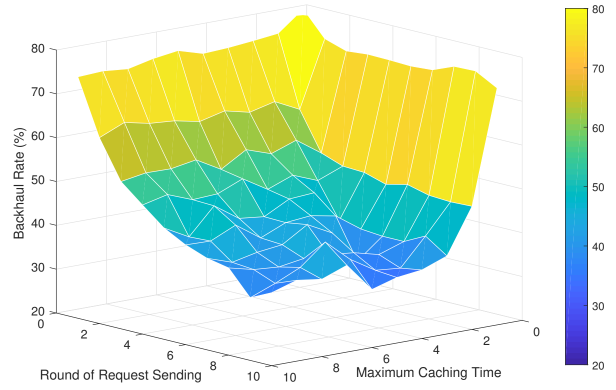
To make clear how TRT and TCT methods may help reallocate the whole workload and achieve energy efficient caching, we add the experimental results of backhaul rates. As the values of results in different cache replacement policies are similar to each other, here we choose FIFO as an example. We change the set maximum requested times & caching time of each single file being cached in routers as edge devices and get two 3-D surface graphs (values of maximum caching time are corresponding values in simulation).

Under the same coordinate axis, Fig. 4 shows the variation of backhaul rates by rounds of packet sending and maximum requested times/caching time. First in Fig. 4a, when number of requested times is small, regardless of which round of packet sending, about more 70 percent of requests still have to reach *Server Tier* to get needed files. The other side, numerical range of the same maximum times in 10 rounds changes not so great which means although stay high in value with small requested times number, backhaul rate of TRT method can rapidly reach steady state as number increases. Finally more than 70 percent of requests can be satisfied by in-memory edge caching.

Then in Fig. 4a, the main difference of the pattern is the variation when we continue to send packets. That is, no matter how long the caching time is set, *Server Tier* still has to take charge of most workload at the first rounds. However, with more and more files being fetched originally from end servers and then cached in in-memory of passed routers, TCT method also can reach a comparatively ideal low backhaul rate, though higher in value than TRT as well as some fluctuations.



(a) TTL of Requested Times (TRT)



(b) TTL of Caching Time (TCT)

Fig. 4. Backhaul Rate of In-Memory Edge Caching in 2 TTL Designs

Conclusion

In this paper, we focus on how to improve energy efficiency of edge caching using in-memory storage and processing. Here we build a 3-tier heterogeneous network structure and propose two edge caching methods using different TTL designs & cache replacement policies. We use total energy consumption and backhaul rate as the two metrics to test the performance of in-memory caching method and compare with conventional method based on disk storage. The simulation results show that in-memory storage and processing can help save more energy in edge caching and share considerable workload in percentage.

Acknowledgment

This work is partially supported by JSPS KAKENHI Grant Number JP16K00117, JP15K15976, and KDDI Foundation. Mianxiong Dong is the corresponding author.

References

[1] Cisco Public White Paper, "Cisco Global Cloud Index: Forecast and Methodology, 2015-2020," [Online]. Available: www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf, accessed Sep. 2017.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, Oct. 2016, pp. 637-646.

[3] F. Bonomi, R. Milito, J. Zhu and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, no. 4, 2012, pp. 13-16.

[4] L. M. Vaquero and L.Rodero-Merino, "Finding Your Way in the Fog: Towards a Comprehensive Definition of Fog Computing," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, Oct. 2014, pp. 27-32.

[5] Z. Liu, M. Dong, H. Zhou, X. Wang, Y. Ji and Y. Tanaka, "Device-to-device Assisted Video Frame Recovery for Picocell Edge Users in Heterogeneous Networks," *2016 IEEE International Conference on Communications (ICC)*, May. 2016, pp. 1-6.

[6] S. Sardellitti, G. Scutari and S. Barbarossa, "Joint Optimization of Radio and Computational Resources for Multicell Mobile-Edge Computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, Jun. 2015, pp. 89-103.

[7] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher and V. Young, "Mobile Edge Computing: A Key Technology towards 5G," *ETSI White Paper*, vol. 11, no. 11, 2015, pp. 1-16.

[8] L. Ramaswamy, L. Liu and A. Iyengar, "Cache Clouds: Cooperative Caching of Dynamic Documents in Edge Networks," *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, Jun. 2005, pp. 229-238.

[9] F. Gabry, V. Bioglio and I. Land, "On Energy-Efficient Edge Caching in Heterogeneous Networks," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, Dec. 2016, pp. 3288-3298.

[10] D. Liu, B. Chen, C. Yang and A. F. Molisch, "Caching at the Wireless Edge: Design Aspects, Challenges, and Future Directions," *IEEE Communications Magazine*, vol. 54, no. 9, Sep. 2016, pp. 22-28.

[11] H. Zhang, G. Chen, B. C. Ooi, K. L. Tan and M. Zhang, "In-Memory Big Data Management and Processing: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 7, Jul. 2015, pp. 1920-1948.

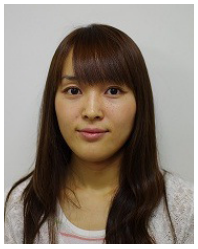
[12] F. Beneventi, A. Bartolini, C. Cavazzoni and L. Benini, "Continuous Learning of HPC Infrastructure Models using Big Data Analytics and In-Memory processing Tools," *Design, Automation Test in Europe Conference Exhibition (DATE 2017)*, Mar. 2017, pp. 1038-1043.

[13] C. Bettstetter, G. Resta and P. Santi, "The Node Distribution of the Random Waypoint Mobility Model for Wireless Ad Hoc Networks," *IEEE Transactions on Mobile Computing*, vol. 2, no. 3, Jun. 2003, pp. 257-269.

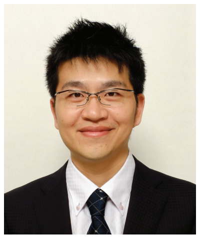
[14] Z. Zhou, K. Ota, M. Dong and C. Xu, "Energy-Efficient Matching for Resource Allocation in D2D Enabled Cellular Networks," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 6, Jun. 2017, pp. 5256-5268.

[15] L. M. Feeney and M. Nilsson, "Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment," *Proceedings of IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, vol. 3, 2001, pp. 1348-1557.

Jianwen Xu received the B.Eng degree in Electronic and Information Engineering from Dalian University of Technology (DLUT), China, in 2014, and M.Eng degree in Information and Communication Engineering from Shanghai Jiaotong University (SJTU), China, in 2017. He is currently pursuing the Ph.D. degree in Electrical Engineering at Muroran Institute of Technology, Japan. His main fields of research interest include distributed system, Internet of things.



Kaoru Ota received her M.S. degree in computer science from Oklahoma State University in 2008, and her B.S. and Ph.D. degrees in computer science and engineering from the University of Aizu in 2006 and 2012, respectively. She is currently an assistant professor with the Department of Information and Electronic Engineering, Muroran Institute of Technology. She serves as an Editor for IEEE Communications Letters.



Mianxiong Dong received his B.S., M.S., and Ph.D. in computer science and engineering from the University of Aizu. He is currently an associate professor in the Department of Information and Electronic Engineering at Muroran Institute of Technology. He serves as an Editor for IEEE Communications Surveys & Tutorials, IEEE Network, IEEE Wireless Communications Letters, IEEE Cloud Computing, and IEEE Access.