An ICN-based Approach for Service Caching in Edge/Fog Environments

Hamza Ben-Ammar and Yacine Ghamri-Doudane
La Rochelle University, L3i Lab., France.
Emails: {hamza.ben_ammar, yacine.ghamri}@univ-lr.fr

***Abstract*—Edge and Fog computing represent today a realistic alternative to traditional cloud data centers in order to support data-intensive and time-sensitive applications, such as those laying under the Internet of Things (IoT) umbrella. One of the main problems associated to this is the service placement problem, or how to efficiently manage the available computing and storage resources while deploying the plethora of application services, to be made available to clients, at the network edge/fog? Due to the similarities shared by this problem and the traditional data caching problem, multiple researches started looking at the adaptation of the Information-Centric Networking (ICN) paradigm to answer the aforementioned question, giving birth to what we depict as an ICN-Edge/Fog architecture. Leveraging such an architecture, we propose in this paper a novel service caching strategy, called 3Q, that is the first to argue on the caching of both, the service instances, consuming computing resources, as well as their associated source codes, consuming storage resources. Being characterized by its low-complexity and its low-overhead, 3Q proves to achieve near-optimal results in terms of cached services hits, latency and cloud usage.**

***Index Terms*—Service Caching, Edge Computing, Fog Computing, Information-Centric Networking.**

I. Introduction

Edge Computing (EC) and Fog Computing (FC) [1] are promising paradigms aiming at overwhelming some of the limitations faced by traditional cloud data centers, especially those related to the support of time-sensitive applications and eco-responsibility (as cloud data centers have an important energy footprint). However, EC and FC solutions face stringent challenges in terms of resources allocation for the support of heterogeneous applications at the network edge/fog. Indeed, they are composed of multiple distributed nodes that are characterized by their very limited computing and storage resources, in comparison to those available in centralized (and geographically localized) cloud data centers. A relevant issue that needs to be addressed is the service placement problem (SPP), which relates to the process of determining where to place different services, namely their instances (i.e. computing resource consumption) as well as their associated source codes (i.e. storage resource consumption). Such a service placement needs to answer the question of how to distribute workloads of a placed application on multiple locations. In this paper, one of our arguments is that such a workload is not only to be considered in terms of computing resources, as mostly considered in the literature, but also in terms of storage resources as well (i.e. for source codes storage).

Several existing works have addressed the service placement problem in edge and fog environments and more generally in network architectures having distributed and limited computational and storage resources [2]. They can be classified following multiple criteria: single or multi-objective optimization, centralized or distributed solution, etc. More precisely, SPP has been studied from different angles, that led to various proposals, such as integer programming [3], meta-heuristics [4] or constrained optimization [5]. The problem of how to optimize the placement of services in the edge/fog computing environment shares few similarities with the well-investigated problem of cache management [6], [7], which started with computer architecture and continued later with web caching to then regain new interests with the arrival of the ICN paradigm [8]. These similarities led researchers to wisely think about combining both approaches, including the design of specific and efficient network architectures allowing to combine both, such as the one in [9], where the main idea is to handle the "low-level" network communication operations on behalf of applications. This includes some basic tasks such as enabling the users requests to invoke applications hosted at edge and fog nodes, delivering the applications from selected nodes in a timely manner and allowing the reuse of computing results by other similar tasks.

In this paper, our objective is to revisit the service placement problem while an ICN-based Edge/Fog architecture is deployed. More precisely, we argue that a caching strategy in such a context must rely on two main features: (i) handling the caching of both, the service instances as well as their source codes (or software images), and (ii) keeping the overhead as low as possible by adopting a strategy that does not require any explicit cooperation among nodes. So, we propose in this paper 3Q, a low-complexity caching strategy that uses three queues at each edge/fog node, in order to manage which services to admit in the service instances cache and which one to admit to the source code cache. Extensive evaluations show that such a simple caching strategy proves to have cached services hits, latency and cloud usage performances that are close to the ones obtained by an optimal (oracle) strategy.

The remainder of the paper is organized as follows. Section II reviews the main related work. This is followed in Section III by our proposal for Service Placement in an ICN-based Edge/Fog architecture. In Section IV, we analyze the performance of our proposal. Finally, Section V concludes this paper and presents some future research directions.

II. Related Work

Several papers have presented different solutions to the SPP problem. In [4], the authors propose a genetic algorithm to maximize the instances of deployed services in the fog nodes as an alternative to the cloud. The authors in [5] handled the SPP problem in Fog Computing by providing a generic and evolving constraint programming model. In [3], a framework based on Network Functions Virtualization (NFV) is proposed to deploy in the fog nodes, services provided by the cloud. In their work, the deployment problem is formulated as an Integer-Linear Programming (ILP) problem. These proposals are examples of interesting solutions to the SPP problem. However, they all significantly increase the network overhead, as they require centralized information to be collected as well as a collaboration among network nodes.

To the best of our knowledge, there are only few studies that dealt with the Service Placement Problem in edge/fog environments that are built following the ICN principles and proposing online and distributed caching and cahe replacement strategies that are characterized by their low overhead. Hou et al. [10] tackled in their paper the service provisioning problem in edge-clouds from a theoretical point of view. They propose an online algorithm for the dynamic reconfiguration of edge-clouds where their limited capacity, operational costs and the unknown arrival patterns of requests are considered. Their solution, named retrospective download with least recently used, is an LRU-based replacement strategy with low complexity where the history of previously requested services is used to decide which service to replace in the edge-cloud when it reaches its maximum capacity. However, the edge nodes in their system model are not geographically distributed as they are considered as forming a single entity. In [11], the authors argue that the SPP is similar to the resource storage allocation problem and studied it as such. Using the principles of cache management and by considering delay-sensitive services, they propose an uncoordinated resource allocation where the edge-cloud nodes are used as on-path opportunistic resources. Their solution is based on the combination of well-known scheduling strategies and cache replacement strategies. Nevertheless, they suppose in their work that the source code necessary to instantiate the different proposed services/applications are available in advance in every edge/fog node of the network.

III. Towards an ICN-based Service Instances and Source Codes Caching Strategy

A. System model

We consider in this work an ICN-based Edge/Fog Computing environment, similar to the one presented in [9], and we use one of the most popular concrete implementations of ICN, which is the Named-Data Networking [12], also known as Content-Centric Networking or CCN.

Edge and Fog computing are natively service-centric architectures where clients, asking for specific services from the edge of the network, can get it from any edge/fog node offering the service in the network. The ICN paradigm matches perfectly the objective and the mechanisms of EC and FC through the use at the network layer of application-defined naming. Let's recall that in CCN, the content's name (services in our case) is the only identifier of a specific data (respectively, a service in our case). To request and retrieve data, two types of packets are commonly used [12]: Interest Packets and Data Packets. Users ask for specific data by sending Interest packets, which are forwarded towards the data sources using the Forwarding Information Base (FIB). A record of the forwarded Interests is kept in the Pending Interest Table (PIT) in order to keep track of the Interests waiting for a data packet. When a node receives multiple requests for the same data, the Interest packets are aggregated in one entry in the PIT and only the first one is routed. Once the request is found, it is automatically routed back to the clients on the reverse path. All the nodes along this path can store a copy of the requested data to answer future demands. In our case, clients requests regards the available running services in the network.

Let $G = (V, E)$ be the graph representing an ICN-Edge/Fog network, where $V = \{v_1, \ldots, v_M\}$ depicts the nodes of the network and $E \subset V \times V$ is the set of links connecting the nodes. Each node in the network has computation capabilities (CPU and RAM) to run the services and storage capacity to host their source codes allowing to instantiate pre-installed services as an answer to client requests. Unlike ICN, where each node has only one cache to save in the network copies of the flowing data packets, every node in our work will have two separate caches: one to host the running services available to the users and the other is used to store the source codes. Let $S = \{s_1, \ldots, s_R\}$ be the set of services available to the users. We assume that all the accessible applications in the system along with their source codes have an identical size. The node capacity is then expressed in terms of the number of maximum running services and the number of application source codes (or software images, in case where containers are used) that can be stored. The pattern of the interests sent by the users is depicted by the Independent Reference Model (IRM) [13] where these are generated in a independent and identical distributed sequence of requests. More precisely, the probability p_r to request an item s_r from the catalog S of R services follows a popularity law in which the items composing the catalog of services are ranked decreasingly according to their popularity from 1 to R. All the available services are hosted permanently at one or more servers within the network. In the rest of the paper and for the sake of readability, we will use the term service/application interchangeably as well as the terms rank/popularity and source code/image.

B. Service caching and management

In this work, our aim is to propose an online strategy with low-complexity using ICN's on-path caching feature to decide how the services and their images are cached and managed within the network. The intended objectives are to reduce the cloud load, minimize the average distance and delay for the users to get their requested applications and improve the efficiency of the edge/fog nodes resources. We assume that all the services available for the users and their source codes are initially only hosted in the cloud. In order for any node wanting to serve a specific application upon the reception of an interest for it, it has first to download from the cloud the corresponding source code in order to be able to instantiate the service afterword.

Whenever a client sends a request asking for a specific service, the first node that receives it will check whether it has the application available in its cache. If there is a hit, then this node will provide the requested service. In case of a miss, we suppose that even if a node has an image of the demanded application in its cache, launching the service takes more time than getting it from an another node or the cloud. The user's demand will then be forwarded to the next nodes toward the cloud checking for the presence of the application. Once it is found, it is sent in the reverse path towards the end user and following the adopted caching policy, the service's image is cached or not at each node that receives it in order to create an instance of the service. Once one of the node's caches is full and a new caching decision is to be enforced, then one item has to be selected for eviction to make room for the new element to be cached.

The caching scheme has to manage basically two aspects of the services cache and the source code store: the cache replacement policy and the caching decision. The first one is the algorithm responsible for managing the cache. When the cache is full, an item is chosen by the algorithm to be discarded. To achieve this, one can cite the following existing algorithms: First In First Out (FIFO), Least Frequently Used (LFU), Least Recently Used (LRU), to name a few. The second one determines which object should be cached and at which node. In other words, whenever a service requested by an end user is travelling towards its destination node, an intermediate node has to choose whether to ask for its source code in order to launch the application locally. This depends on the implemented caching strategy. Among the examples of such strategies that are commonly used in ICN, we can cite the Leave Copy Everywhere (LCE) [14], the Leave Copy Down (LCD) [14] and the Two Queue (2Q) [15], for instance. Schemes like these three examples have the specificity of being of low-complexity and of low-induced network overhead as there is no information exchange or cooperation between the different nodes, which is our aim in this work.

C. LCE and LCD

In multi-cache systems in general and more specifically in ICN, the default caching strategy that can be used is the Leave Copy Everywhere (LCE) where every data packet received by each node along the downloading path is systematically cached. It is a very basic mechanism that does not imply any overhead or communication cost. Such schemes can be used in performance evaluation campaigns to have the lower bound values in terms of performance. Another interesting technique having a low overhead cost but with a better efficiency is the Leave Copy Down (LCD). Instead of saving a copy of the data packet in all intermediates node caches as in LCE, LCD consists on performing the caching operation only in the immediate downstream node on the path from the hit location. This mechanism creates a kind of filtering system where contents gradually move towards the users with the popular ones to be the first to be cached as close as possible to clients.

D. 2Q algorithm

On the basis of the algorithm named LRU-K presented in [16], Johnson et al. proposed the Two Queue scheme (2Q). LRU-K is basically an enhancement of the classical LRU algorithm while keeping track of the timing of the K last occurrences of each element entering the cache, thus allowing to have an estimation of their reference inter-arrival times. The item whose K^{th} most recent access is furthest in the past will then be evicted when the cache is full (LRU-1 is equivalent to the classical LRU). It is shown that the most important gain of the LRU-K method is achieved when $K = 2$ (LRU-2) [16], but with the cost of having a relatively high complexity, as each item access requires $\log(N)$ operations to manipulate a priority queue (N being the cache size). 2Q is similar and performs as well as the LRU-2 algorithm but unlike the latter, it has a constant complexity time overhead. Instead of cleaning cold elements from the main buffer like LRU-2, 2Q admits to the cache only the ones that are more likely to be requested in the near future. When a request is received by a cache using 2Q, the requested object's hash is first placed in a virtual cache (called $A1$), which is managed as a FIFO queue. If an item is requested during its $A1$ residency, it is then promoted to the main cache (called Am). The authors, then, propose another version of 2Q in which the $A1$ queue is partitioned into $A1in$ and $A1out$. The $A1in$ queue along with Am form the physical cache and $A1out$ is a virtual cache, which will contain only items hashes. The most recent first accesses are stored in $A1in$, which is managed as a FIFO queue. When objects are evicted from $A1in$, they are remembered in $A1out$. Upon arrival of a request and if it is present in the $A1out$ queue, then it is cached in Am. The item to be discarded in 2Q is chosen either from $A1in$ or Am. One should also note that the sizes of the different queues ($A1in$, $A1out$ and Am) are sensitive to the requests patterns and should be tuned carefully.

E. 3Q: a 2Q-inspired algorithm for service caching

In this paper we propose a caching scheme that we call 3Q, which is similar to the first version of 2Q but where the virtual buffer and the main store are both managed using LRU. The virtual cache's role is to allow only popular services to be cached by detecting the services that are requested more than once within a short period of time. The 2Q algorithm was initially proposed in the context of page replacement algorithms for computer operating systems and it was then shown to be efficient in ICN [17]. We believe that adopting a 2Q like algorithm in the context of services placement in an ICN-Edge/Fog architecture can also lead to very good performance. As it was explained earlier, each node in the network has to have the service's source code in order to

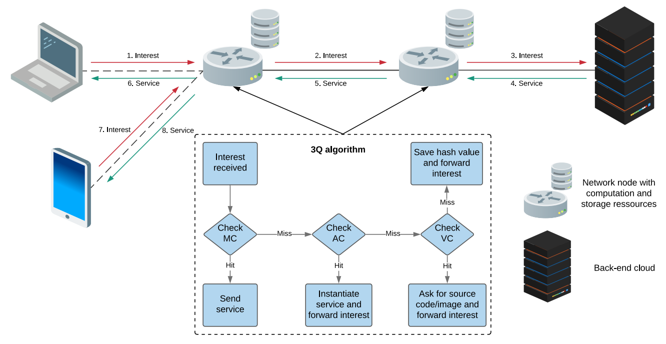

Figure 1. 3Q functioning in an ICN-based Edge/Fog architecture.

instantiate it and be able to provide it to users. So, in addition to the virtual cache (VC) and the main cache (MC) that hosts the services, there is an auxiliary cache (AC) for the source codes, thus having three queue in total for each node (see Figure 1). Once a node receives the request issued from a client looking for a specific service, it checks if it has a copy running in the MC. If there is a hit, then it provides the application to the user. In case of a miss, it looks for the presence of its source code in the AC. If a copy is found, it instantiates the corresponding service, but the interest is also sent upward to other nodes since we assume that loading an application from its source code or from its image takes more time than getting it from another node or even from the cloud. If not, it is the turn of the VC to be checked. If there is a hit and while the node forwards the interest to the other nodes toward the cloud, it also asks for the service's source code (which can be received from one of the upward nodes or the cloud) in order to save it in the AC and then instantiate a copy of it in the MC. In case of a miss, a hash value of the requested application is saved in the VC waiting to be requested again while being present in this buffer and thus to trigger its caching. Here again, the interest is forwarded upward.

F. Optimal caching strategy (OPT)

When designing a caching scheme, the aim is generally to maximize the overall hit ratio in order to reduce the cloud load and maximize the cache hits at the nodes close to users, which reduces the average latency to get a requested object. This is achieved by creating a mechanism that tries to detect and cache the most popular and requested data packets as near as possible to end users and to have the lowest possible redundancy between neighboring nodes. In this case, an "optimal" strategy (that we call OPT) works as follow: if we suppose that the services popularities are known in advance and that all the caches of services of the network's nodes have the same size C, then the most C popular services ($s_1, \ldots, s_C$) are exclusively cached at the nodes located at a one-hop distance from the users, then the second C popular items ($s_{C+1}, \ldots, s_{2C}$) are cached at a 2-hop distance, etc. Such strategy gives the upper bound in terms of performance.

IV. Results

A. Evaluation settings

We evaluate in this section the performance of 3Q and compare it to the following caching strategies: LCE, LCD and OPT. As for the cache replacement policy, we will focus in this work on LRU to manage each node's caches. We could have considered other cache management algorithms like FIFO, Random or LFU but LRU is known to perform much better than FIFO and Random. As for LFU, it requires a more complex replacement process and do not adapt very well in time with the popularity variations of the elements to be cached.

The different evaluations are conducted using ccnSim [18], a discrete-event and chunk-level simulator of CCN (ccnSim was modified in order to handle the management of services instead of contents). We use during the simulations a complete binary tree network topology containing 31 nodes (with a total of five levels). The root node is attached to the back-end cloud system, which hosts all the services available to the users and their source codes. The clients, attached to the leaves nodes of the network, generate their requests for the different services according to a Poisson process with a rate of 1 request/sec (this rate corresponds to each aggregation of users attached to each leaf node). We consider in the topology a uniform latency of 25 ms between each two nodes except for the inter-connection between the root and its child nodes where the latency is set to 100 ms (to represent a higher latency to access the back-end cloud system compared to the hop-to-hop node latency). For our simulations, we consider a catalog of services containing 10000 services whose popularity distribution follows the Zipf's law, where each service has a probability to be requested that is equal to: $p_r = r^{-\alpha} / \sum_{i=1}^{R} i^{-\alpha}$, α being the skew of the distribution. The Zipf's law has been used in many fields, including the modeling of the services popularity distribution in Cloud and Edge/Fog Computing [19]. We suppose that the network's nodes have service cache sizes following the uniform distribution, with values varying from 50 to 250, and source code caches having its double in size, as storage capacity is generally greater than computation capacity according to price trends. The size of the virtual cache in 3Q is also a parameter to be tuned, which was set to 10% of the services cache size (we obtained the best results using this value). We tested different values of the Zipf's law parameter α (0.8, 1.0 and 1.2). The numerical results depicted in Figure 2 and Figure 3 represent the mean values taken over 30 simulation runs (using a confidence interval of 95%) and in which 2×10^6 requests from users are sent in the network after the system reaches stability (i.e. after all service caches become full).

B. Results and analysis

We display in Figure 2 the average service cache hit for the entire network (2(a), 2(b) and 2(c)) as well as for the edge nodes only (2(d), 2(e), and 2(f)) to highlight the efficiency of cache resource usage. We can see from the results how

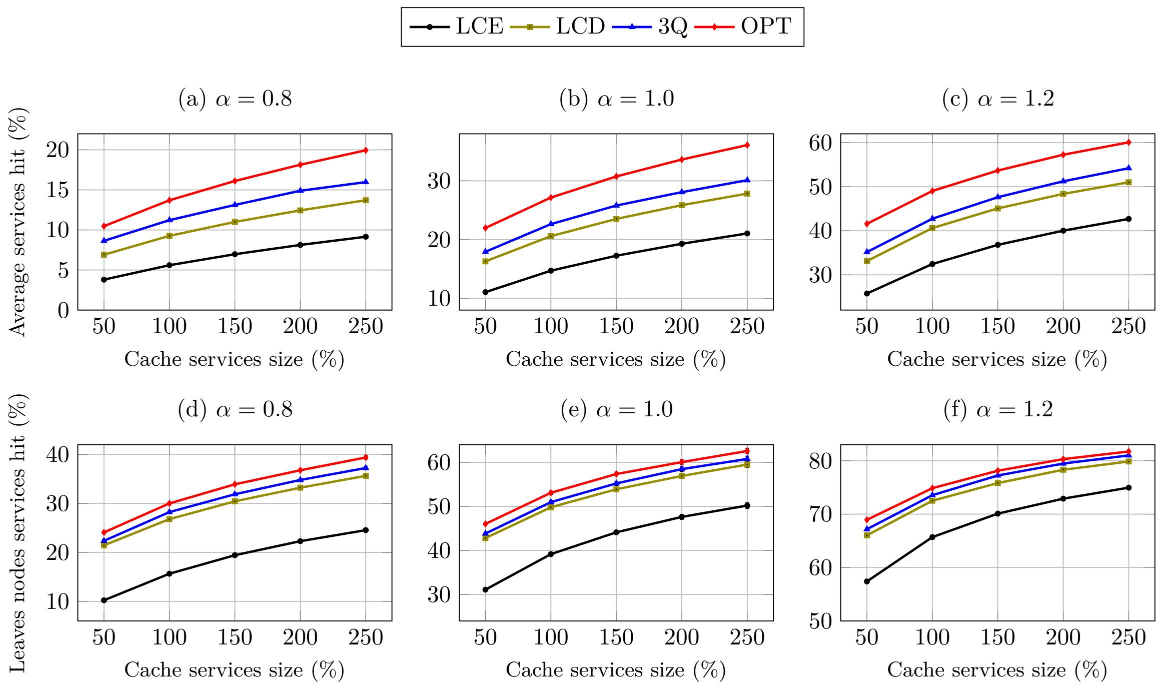

Figure 2. Average cache services hit in all the network and at the leaves nodes.

using low cost strategies (LCD and 3Q), i.e. strategies that do not require any sort of cooperation and information exchange between nodes, can be effective as the cache hit performance is greatly improved compared to LCE, which is the default caching technique. We denote, however, a superiority of 3Q over LCD. This is even more clear when we focus on the cache hit performance of the edge nodes only (2(d), 2(e), and 2(f)) as the performance of 3Q, and at slightly lower extent LCD, are very close to OPT. This also illustrates the difficulty of taking an efficient advantage of the nodes resources located at higher levels of the network topology, resulting in lower performance compared to edge nodes, which have the chance to serve more popular services as they are the first to receive the clients requests. Furthermore, one should note that the performances are heavily impacted by the service popularity distribution and the cache size. Lower value of α means less difference in the service popularity, and vice-versa, which makes it more difficult to have multiple hits for the same item in the cache (and vice-versa). As for the cache size, a lower value means a higher probability of discarding from the cache running services that could be requested in the near future.

Figure 3 depicts the network performance in terms of latency reduction ratio (3(a) and 3(b)), cloud services load reduction ratio (3(c) and 3(d)) and cloud source code load reduction ratio (3(e) and 3(f)). These are obtained while considering various values of α and the cache size. The cloud load reduction metric represents how much (in percentage) the solicitation of the back-end cloud is decreased due to the caching process at the edge/fog nodes. The results shows how 3Q outperforms the other caching strategies achieving performances close to those of OPT. For example, 3Q achieves 96% of the OPT performance in terms of latency reduction ratio when α and the cache size are set respectively to 1.2 and 250. These results indicate clearly that 3Q is more successful compared to LCE and LCD in serving most popular services, especially in the lower levels of the network topology.

Overall, we can conclude from these results that an uncoordinated caching strategy, like 3Q, can achieve near optimal performance as defined by OPT.

V. Conclusion and Future Work

We studied in this paper the service placement problem in the context of Edge/Fog computing environments. We started first by highlighting how the ICN paradigm can be used to tackle this problem from a new perspective. By leveraging the in-network caching feature of ICN, the services along with their source codes (or software images) can be managed at the Edge and Fog nodes using a specifically designed caching strategy. To keep the network overhead as low as possible, we focused on caching schemes that does not require explicit cooperation between nodes. To do so, we proposed in this paper a service caching strategy called 3Q, an extension of a caching algorithm called 2Q that was shown to be efficient for data caching in ICN. 3Q works in a distributed manner where each node within the network is autonomous, inducing only a marginal control overhead and cost. The conducted simulations showed the closeness of our proposal to an optimal solution, i.e. a solution knowing the popularity distribution of the services in advance, in terms of cached services hits, latency and cloud usage. As per our future works, we aim to explicitly incorporate the QoS requirements (e.g. deadline satisfaction)

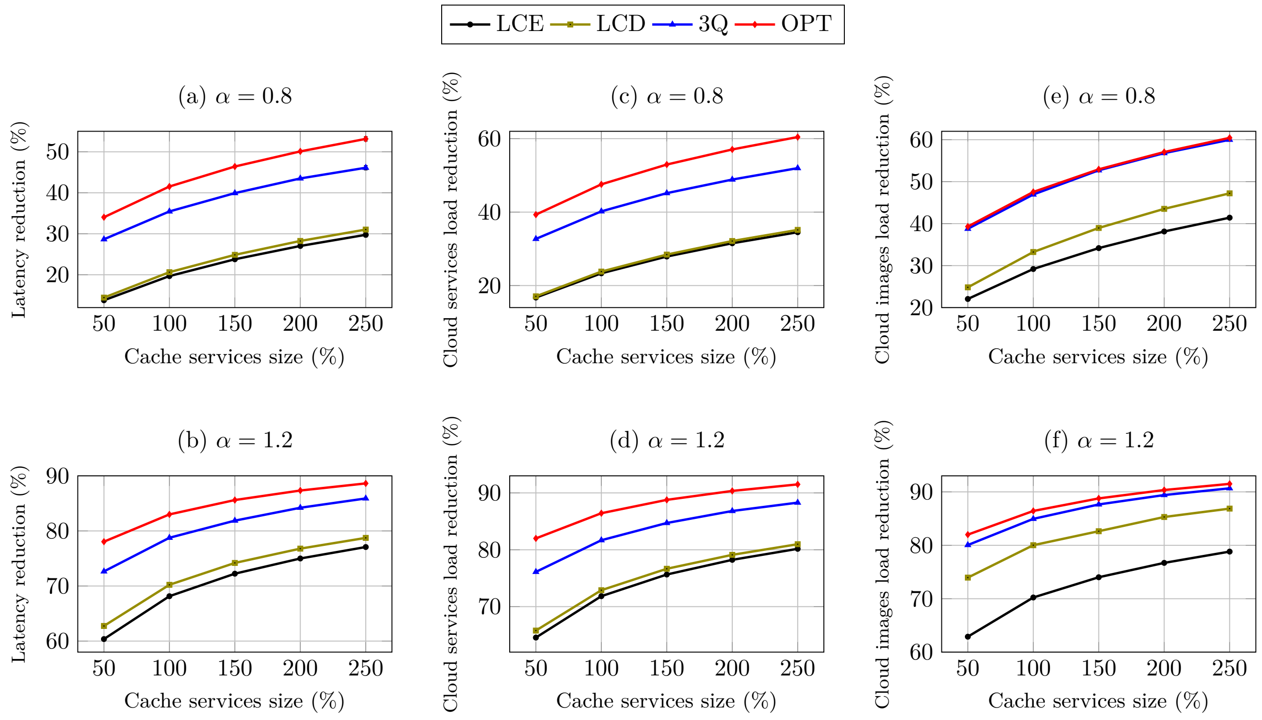

Figure 3. Network performance comparison in terms of latency and cloud load.

in the service caching process. We also aim at developing analytical tools in order to quantify the cost/efficiency ratio when studying and using caching strategies and their network overhead impact.

References

[1] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, “All one needs to know about fog computing and related edge computing paradigms: A complete survey,” *Journal of Systems Architecture*, vol. 98, pp. 289 – 330, Sep. 2019.

[2] F. A. Salaht, F. Desprez, and A. Lebre, “An overview of service placement problem in fog and edge computing,” in *Research Report RR-9295, Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP,LYON, France*, Nov. 2019, pp. 1–43.

[3] R. I. Tinini, L. C. M. Reis, D. M. Batista, G. B. Figueiredo, M. Tornatore, and B. Mukherjee, “Optimal placement of virtualized bbu processing in hybrid cloud-fog ran over twdm-pon,” in *2017 IEEE Global Communications Conference*, Dec. 2017, pp. 1–6.

[4] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, “Optimized iot service placement in the fog,” *Service Oriented Computing and Applications*, vol. 11, pp. 427–443, Oct. 2017.

[5] F. Ait Salaht, F. Desprez, A. Lebre, C. Prud’homme, and M. Abderrahim, “Service placement in fog computing using constraint programming,” in *2019 IEEE International Conference on Services Computing (SCC)*, Jul. 2019, pp. 19–27.

[6] H. Ben-Ammar, Y. Hadjadj-Aoul, G. Rubino, and S. Ait-Chellouche, “A versatile markov chain model for the performance analysis of CCN caching systems,” in *IEEE Global Communications Conference*, Dec. 2018, pp. 1–6.

[7] H. Ben-Ammar and Y. Hadjadj-Aoul, “A GRASP-based approach for dynamic cache resources placement in future networks,” *Journal of Network and Systems Management*, pp. 1–21, Mar. 2020.

[8] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. Katsaros, and G. C. Polyzos, “A survey of information-centric networking research,” *IEEE Communications Surveys and Tutorials*, vol. 16, pp. 1024–1049, Jul. 2014.

[9] S. Mastorakis, A. Mtibaa, J. Lee, and S. Misra, “ICedge: When edge computing meets information-centric networking,” *IEEE Internet of Things Journal*, jan 2020.

[10] I.-H. Hou, T. Zhao, S. Wang, and K. Chan, “Asymptotically optimal algorithm for online reconfiguration of edge-clouds,” in *Proceedings of the 17th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Jul. 2016, p. 291–300.

[11] O. Ascigil, T. K. Phan, A. G. Tasiopoulos, V. Sourlas, I. Psaras, and G. Pavlou, “On uncoordinated service placement in edge-clouds,” in *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Dec. 2017, pp. 41–48.

[12] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard, “Networking named content,” in *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, Dec. 2009, pp. 1–12.

[13] P. Flajolet, D. Gardy, and L. Thimonier, “Birthday paradox, coupon collectors, caching algorithms and self-organizing search,” *Discrete Applied Mathematics*, vol. 39, pp. 207–229, Nov. 1992.

[14] N. Laoutaris, H. Che, and I. Stavrakakis, “The LCD interconnection of LRU caches and its analysis,” *Performance Evaluation*, vol. 63, pp. 609–634, Jul. 2006.

[15] T. Johnson and D. Shasha, “2Q: A low overhead high performance buffer management replacement algorithm,” in *Proceedings of the 20th International Conference on Very Large Data Bases*, Apr. 1994, pp. 439–450.

[16] E. J. O’Neil, P. E. O’Neil, and G. Weikum, “The LRU-K page replacement algorithm for database disk buffering,” *ACM SIGMOD Record*, vol. 22, pp. 297–306, Jun. 1993.

[17] H. Ben-Ammar, Y. Hadjadj-Aoul, G. Rubino, and S. Ait-Chellouche, “On the performance analysis of distributed caching systems using a customizable markov chain model,” *Journal of Network and Computer Applications*, vol. 130, pp. 39–51, Mar. 2019.

[18] R. Chiocchetti, D. Rossi, and G. Rossini, “ccnSim: An highly scalable ccn simulator,” in *IEEE International Conference on Communications*, Jun. 2013, pp. 2309–2314.

[19] G. Li, J. Wu, J. Li, K. Wang, and T. Ye, “Service popularity-based smart resources partitioning for fog computing-enabled industrial internet of things,” *IEEE Transactions on Industrial Informatics*, vol. 14, pp. 4702–4711, Oct. 2018.