Asymptotically Optimal Algorithm for Online Reconfiguration of Edge-Clouds

I-Hong Hou*, Tao Zhao*, Shiqiang Wang†, and Kevin Chan‡
*Department of ECE, Texas A&M University, College Station, TX, USA
†Department of EEE, Imperial College London, London, UK
‡US Army Research Laboratory, Adelphi, MD, USA
{ihou, alick}@tamu.edu, shiqiang.wang11@imperial.ac.uk, kevin.s.chan.civ@mail.mil

ABSTRACT

"Edge-clouds," which are small servers located close to mobile users, have the potential to greatly reduce delay and backhaul traffic of mobile applications by moving cloud services closer to users at the edge. Due to their limited storage capacity, proper configurations of edge-clouds have a significant impact on their performance. This paper proposes a tractable online algorithm that configures edge-clouds dynamically solely based on past system history without any assumptions on the arrival patterns of mobile applications. We evaluate the competitive ratio, which quantifies the worst-case performance in comparison to an optimal offline policy, of our policy. We prove that the competitive ratio of our policy is linear with the capacity of the edge-cloud. Moreover, we also prove that no deterministic online policy can achieve a competitive ratio that is asymptotically better than ours. The utility of our online policy is further evaluated by traces from real-world data centers. These trace-based simulations demonstrate that our policy has better, or similar, performance compared to many intelligent offline policies that have complete knowledge of all future arrivals.

CCS Concepts

•**Networks** → **Cloud computing; Data center networks;** •**Theory of computation** → *Caching and paging algorithms;*

1. INTRODUCTION

Many emerging mobile applications rely on cloud computing technology to greatly expand the capability of resource-constrained mobile devices. In a typical scenario, a mobile device sends a request, such as a picture containing text in a foreign language, to a data center, and the data center generates a response, such as translations of the text, using its massive computational power and storage. However, the long distance between mobile devices and data centers can result in significant delay and burden on the backhaul connection, which can limit the development of real-time and data-intensive applications. The concept of "edge-clouds," also called cloudlets, edge computing, fog computing, etc., has been proposed to address this issue [4, 7, 8, 12, 15, 19]. Edge-clouds are small servers deployed close to mobile users, such as at the locations of cellular base stations or WiFi access points. They can host a small number of popular services, and provide timely response to requests of these services directly without communicating with remote data centers.

Edge-clouds have very limited storage, and can only host a small number of services. Therefore, the configurations of edge-clouds can have significant impact on their performance. While there have been many studies on the optimal configurations of edge-clouds, most of them assume either that the edge-clouds have reasonably good predictions about future requests [9, 18], or that the arrivals of requests follow some stochastic process [2, 5, 11, 20, 21]. These studies then use predictions or observations of past arrivals to determine the services that the edge-clouds should host, and to achieve the optimal average performance. However, in many scenarios, requests for services are generated by events in the physical world, which is difficult to predict or model as a stochastic process. We consider FirstNet [10], which is a network architecture designed for first responders during emergencies, as an example. When a catastrophic event occurs, first responders may generate a swarm of requests based on the rapidly changing environments. Algorithms that rely on predictions or stochastic models can result in poor performance for these systems. Further, many mission-critical systems require "worst-case" performance guarantees, instead of "average" performance guarantees.

In this paper, we study online algorithms that reconfigure edge-clouds dynamically without making any assumptions on the arrival patterns of requests. We propose a model that captures the limited storage of edge-clouds, the cost of forwarding requests to remote data centers, and the cost of reconfiguring the edge-clouds by downloading the whole service application and database. We then aim to minimize the total cost, including costs of forwarding requests and downloading services, for any sequence of request arrivals. As online algorithms have no knowledge about future arrivals, we evaluate the performance of an online policy by its *competitive ratio*, defined as the largest possible ratio between the cost of the online policy and the minimum cost, under any sequence of arrivals.

This material is based upon work supported in part by, the U. S. Army Research Laboratory and the U. S. Army Research Office under contract/grant number W911NF-15-1-0279, the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

MobiHoc'16, July 04-08, 2016, Paderborn, Germany

DOI: http://dx.doi.org/10.1145/2942358.2942363

We first focus on a homogeneous system where all services require the same amount of storage, and have the same forwarding cost as well as the same downloading cost. Using an observation of the optimal policy, we propose a simple online policy, *retrospective download with least recently used* (RL), for edge-cloud reconfiguration. We prove that the competitive ratio of our RL policy only grows linearly with the capacity of edge-clouds. We further prove that no deterministic online algorithms can achieve a competitive ratio that is sublinear with the capacity of edge-clouds. Therefore, our RL policy indeed achieves the optimal asymptotic performance.

We also address several practical issues of RL. We demonstrate that it can be implemented as a linear-time algorithm. We also propose a simple extension of RL for heterogeneous systems where different services may be associated with different costs, and require different amounts of storage.

We evaluate the performance of our RL policy using real-world traces of data centers. We compare our policy against another randomized online policy, and three offline policies that have the complete knowledge of all future arrivals. These three policies correspond to solutions based on stochastic optimization, cache management, and dynamic programming, respectively. Each of them achieves the optimal performance under some specific scenarios. Simulation results show that our policy achieves better, or at least similar, performance compared to these offline policies in both homogeneous systems and heterogeneous systems.

The rest of the paper is organized as follows. Section 2 reviews some related studies. Section 3 formally describes the problem of edge-cloud reconfiguration. Section 4 establishes a property of the offline optimal policy that is vital to the design of our online policy. Section 5 introduces our online policy. Section 6 derives the competitive ratio of our policy. Section 7 establishes a lower-bound on competitive ratio for all deterministic online policies and proves that no deterministic online policies can be asymptotically better than ours. Section 8 addresses some practical issues, including how to extend our policy for heterogeneous systems and how to implement our policy with low complexity. Section 9 compares our policy against several others, including many existing offline policies, through trace-based simulations. Finally, Section 10 concludes the paper.

2. RELATED WORK

The dramatic increase in network traffic, particularly due to the proliferation of mobile devices and Internet of Things (IoT), has made it critical to offload some computing and storage jobs from data centers to the edge. Taleb and Ksentini [19] have proposed an architecture for edge-clouds. The utility of edge-clouds has been demonstrated by various prototypes [14, 16].

Managing the very limited resource is a fundamental challenge of edge-clouds. Tadrous et al. [18] have considered systems where one can predict the popularity of services, so that edge-clouds can proactively download and cache popular services during off-peak hours. Llorca et al. [9] have studied the content placement problem and proposed an optimal offline policy. These studies rely on an accurate prediction of future requests.

There are many studies that employ stochastic optimization for edge-cloud reconfiguration. Amble et al. [2] have considered a system where request arrivals follow an independent and identically distributed (i.i.d.) stochastic process with unknown distribution, and proposed a stochastic control policy that maximizes the capacity region of the system. Qiu et al. [11] employ Lyapunov optimization to maximize the performance of edge-clouds. Borst et al. [5] have proposed an algorithm for the caching problem in IPTV networks. Their studies assume a static system where the popularity of services does not change over time. On the other hand, Wang et al. [21] and Urgaonkar et al. [20] have considered dynamic systems where request arrivals are modeled as a Markov process, and proposed solutions that achieve the optimal long-term average performance. However, these solutions based on stochastic optimization cannot be applied when the request arrival process is non-ergodic.

The problem of edge-cloud reconfiguration bears some similarities with the classic cache management problem in computer architecture. In the cache management problem, requests for data arrive sequentially, and a "miss" occurs when the data is not stored in the cache. One then needs to minimize the total number of misses. When one has the complete knowledge of all future requests, the famous Belady's Algorithm [3] achieves the optimal performance. There are many studies on the competitive ratios of online policies without any knowledge of future requests. Sleator and Tarjan [17] have established a least recently used (LRU) policy that has the optimal competitive ratio among all deterministic policies. Achlioptas et al. [1] have studied the competitive ratios of randomized policies. Reineke and Grund [13] have developed a tool that computes the relative competitive ratio between two policies automatically. In the cache management problem, one cache miss corresponds to one download. No request forwarding to the back-end cloud is allowed. However, in the edge-cloud reconfiguration scenario, forwarding is permitted and incurs a smaller cost compared to downloading.

3. SYSTEM MODEL

We consider an edge-cloud system with an edge-cloud and a back-end cloud that are connected through a backhaul connection. The edge-cloud and the back-end cloud jointly host a set $\mathbb{S}$ of services, numbered as $S_1, S_2, \ldots$. The back-end cloud has massive capacity, and can host all services. On the other hand, the edge-cloud only has limited capacity and can only host K services. Without loss of generality, we assume that, when the system starts, the edge-cloud hosts services $S_1, S_2, \ldots, S_K$.

Requests for services arrive at the edge-cloud sequentially. We use $r_n \in \{S_1, S_2, \ldots\}$ to denote the service requested by the n-th request. If r_n is hosted by the edge-cloud when the request arrives, then the edge-cloud can serve the request immediately without causing any significant delay and burden on the backhaul connection. On the other hand, if r_n is not hosted by the edge-cloud, then the edge-cloud has two choices: First, it can forward this request to the back-end cloud for processing. This will cause some delay as well as some traffic on the backhaul connection. We hence say that forwarding a request to the back-end cloud incurs a cost of one unit. Second, the edge-cloud can instead download and replicate the whole service r_n at the edge-cloud. Downloading a service causes much higher delay and traffic. Therefore, we say that each service download incurs a cost of M units, with $M \geq 1$. On the other hand, since the edge-cloud hosts the service r_n after the download, it can then serve subsequent requests for r_n without incurring any costs. The edge-cloud can only host K services. Therefore, when it downloads a service, it also needs to delete a service from its storage.

We aim to minimize the total cost of the system by intelligently reconfigure the set of services hosted by the edge-cloud. Intuitively, if we know that a service will have a lot of requests in the near future, we should download this service so that all these requests only incur M units of cost. Otherwise, we should simply forward all these requests to the back-end cloud without downloading the service, and pay one unit of cost for each request. In practice, however, we may not have accurate prediction for future requests. In such cases, we need to rely on online policies that assume no infor-

mation about future requests. We evaluate the performance of an online policy by its *competitive ratio*, which is formally defined as follows:

Let OPT be the optimal offline policy that minimizes the total cost of the system. In order to minimize the total cost, OPT needs to have full information about all future request arrivals. Let η be an online policy that makes its decision solely based on past events. For a given sequence of request arrivals, $r_1, r_2, \ldots$, let C_{OPT} be the total cost of OPT, and C_η be the total cost of η. Note that the total costs, C_{OPT} and C_η, are functions of the sequence $r_1, r_2, \ldots$, but we omit the sequence to simplify notation. There may be multiple policies that achieve the minimum cost. In this case, we choose OPT to be one that makes the most downloads among all policies with minimum cost. The competitive ratio of η is then the largest possible value of $\frac{C_\eta}{C_{OPT}}$, over all sequences of request arrivals.

DEFINITION 1. *An online policy η is said to be β-competitive if $\frac{C_\eta}{C_{OPT}} \leq \beta$, for every sequence of request arrivals.*

An online policy with low competitive ratio has similar performance with the optimal offline policy. Therefore, we aim to develop an online policy with a low competitive ratio, as well as a lower-bound of competitive ratio for all online policies.

To facilitate the analysis of online policies, we use $OPT(n)$ to be the subset of services hosted by OPT after the n-th arrival, for a given sequence of request of arrivals, $r_1, r_2, \ldots$. Similarly, $\eta(n)$ is defined as the subset of services hosted by η after the n-th arrival. Again, note that $OPT(n)$ and $\eta(n)$ are functions of the sequence of arrivals.

For a service S_i, we use $x_i(n) := \mathbb{1}\{r_n = S_i\}$ to denote the indicator function that the n-th request is one for service S_i. Therefore, $\sum_{l=n}^{m} x_i(l)$ is the number of requests for S_i between the n-th arrival and the m-th arrival.

Before proceeding to the next section, we note that the system model in this section is a homogeneous one: All services have the same cost of forwarding requests, and the same cost of downloading. Moreover, all services require the same amount of storage. While our analytical results mainly focus on the homogeneous system, we will show that our policy can be easily extended to heterogeneous systems in Section 8, and we will evaluate our policy for heterogeneous systems in Section 9.

4. A BASIC PROPERTY OF OPT

Given a sequence of request arrivals, calculating the optimal policy OPT may still incur high complexity. In this section, we establish a basic property of OPT that can be later used to develop our online policy.

THEOREM 1. *Suppose we are given a sequence $r_1, r_2, \ldots$, and $OPT(n)$, which is the subset of services hosted by OPT after the n-th arrival. If there exist $m > n$, a service $S_i \notin OPT(n)$, and another service $S_j \in OPT(n)$ such that:*

$$\sum_{l=n+1}^{m} x_i(l) \geq \sum_{l=n+1}^{m} x_j(l) + 2M, \tag{1}$$

then OPT downloads at least one service between the $(n+1)$-th arrival and the m-th arrival.

We say that a policy hosts a service if, under the said policy, the edge-cloud hosts the service.

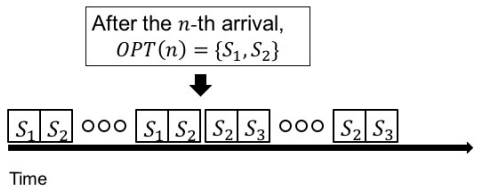


Figure 1: An example illustrating Theorem 1.

Before proving Theorem 1, we first use Fig. 1 to illustrate an example. Suppose the edge-cloud can host two services, and the first n arrivals are $S_1, S_2, S_1, S_2, \ldots$, followed by $S_2, S_3, S_2, S_3, \ldots$. After the n-th arrival, $OPT(n) = \{S_1, S_2\}$. Between the $(n+1)$-th arrival and the $(n+4M)$-th arrivals, we have $2M$ requests for S_3 and no requests for S_1, that is, $\sum_{l=n+1}^{n+4M} x_1(l) = 0$, and $\sum_{l=n+1}^{n+4M} x_3(l) = 2M$. Theorem 1 then states that OPT downloads at least one service between the $(n+1)$-th arrival and the $(n+4M)$-th arrival. Note that Theorem 1 does not specify which service to download, and which service to delete from the edge-cloud.

We now prove Theorem 1.

PROOF OF THEOREM 1. We prove Theorem 1 by contradiction. Suppose OPT does not download any service, and therefore does not delete any service, between the $(n+1)$-th arrival and the m-th arrival, that is, $OPT(l) = OPT(n)$, for all $n+1 \leq l \leq m$.

We construct a different policy η as follows: η hosts the same subset of services as OPT before the n-th arrival and after the $(m+1)$-th arrival, that is, $\eta(l) = OPT(l)$, for all $l \leq n$, and all $l \geq m+1$. After the n-th arrival, η downloads S_i and deletes S_j so that $\eta(n+1) = OPT(n) \backslash \{S_j\} \cup \{S_i\}$. After the m-th arrival, η downloads S_j and deletes S_i, and then follows the same decisions that OPT makes so that $\eta(m+1) = OPT(m+1)$.

We now compare the costs of η and OPT. Since η and OPT are the same before the n-th arrival and after the m-th arrival, they incur the same amount of costs in these two durations. Between the n-th arrival and the m-th arrival, η downloads two services and OPT downloads none. Therefore, η needs to pay $2M$ units of cost. Meanwhile, η needs to pay one unit cost for each request for S_j, and there are $\sum_{l=n+1}^{m} x_j(l)$ of them, while OPT needs to pay one unit cost for each of the $\sum_{l=n+1}^{m} x_i(l)$ requests for S_i. The two policies incur the same amount of costs for all other requests. In summary, we have:

$$C_\eta = C_{OPT} + 2M + \sum_{l=n+1}^{m} x_j(l) - \sum_{l=n+1}^{m} x_i(l) \leq C_{OPT},$$

where the last inequality follows by (1). Therefore, η also minimizes the total cost. Moreover, η makes two more downloads than OPT. Recall that OPT is defined as the policy that makes the most downloads among all policies with minimum cost. The existence of η therefore contradicts with assumptions of OPT. □

5. AN ONLINE POLICY FOR EDGE-CLOUD RECONFIGURATION

We now introduce an online policy. The policy needs to consist of two parts: deciding whether to download a service, and, when a download occurs, deciding which service to delete from the edge-cloud. We propose *retrospective download with least recently used* (RL) policy that uses a *retrospective download* (RD) policy for the first part, and a *least recently used* (LRU) policy for the second part.

RD is used to determine whether to download a service, and is formally defined as follows:

DEFINITION 2. *When a request $r_n = S_i$ whose service is not currently hosted by the edge-cloud arrives, RD downloads and replicates S_i at the edge-cloud if there exists a number τ and a service S_j such that $S_j \in RL(l)$ and $S_i \notin RL(l)$, for all $n - \tau \leq l \leq n - 1$, and*

$$\sum_{l=n-\tau}^{n} x_i(l) \geq \sum_{l=n-\tau}^{n} x_j(l) + 2M. \quad (2)$$

The intuition of RD comes from Theorem 1. Suppose $S_j \in OPT(n-\tau)$, $S_i \notin OPT(n-\tau)$, and (2) holds, then Theorem 1 states that OPT downloads at least one service between the $(n-\tau)$-th arrival and the n-th arrival. RD then downloads S_i when, in retrospect, it finds OPT would have already downloaded a service.

When RD decides to download a service, we need to choose a service to delete from the edge-cloud. We propose a *least recently used* (LRU) policy as follows:

DEFINITION 3. *Suppose the edge-cloud decides to download a service at the n-th arrival. For each service S_i currently hosted by the edge-cloud, let τ_i be the smallest number such that there are $2M$ requests for S_i in the past τ_i arrivals, that is, $\sum_{l=n-\tau_i}^{n} x_i(l) = 2M$. LRU deletes the service with the largest τ_i.*

A service with large τ_i does not have many recent requests, since it needs to go further back in time to find $2M$ requests. LRU therefore deletes the service that is least recently used.

RL uses RD to decide whether to download a service, and LRU to decide which service to delete. We use Fig. 1 to illustrate the operation of RL. Suppose $RL(n) = \{S_1, S_2\}$, that is, RL hosts S_1 and S_2 after the n-th arrival. At the $(n+4M)$-th arrival, we find that $\sum_{l=n+1}^{n+4M} x_3(l) = 2M \geq \sum_{l=n+1}^{n+4M} x_1(l) + 2M$, and RD decides to download S_3 at the $(n+4M)$-th arrival. Meanwhile, between the n-th arrival and the $(n+4M)$-th arrival, there are $2M$ requests for S_2 and none for S_1. We then have $\tau_1 > \tau_2 = 4M$, and LRU decides to delete S_1 to accommodate S_3.

6. THE COMPETITIVE RATIO OF RL

This section establishes the competitive ratio of RL by proving the following theorem:

THEOREM 2. *RL is $10K$-competitive, where K is the number of services that can be hosted by the edge cloud.*

6.1 Overview of the Proof

We will compare the performance of RL and OPT. Given OPT and the arrival sequence, we can divide the arrival sequence into frames, $\{1, 2, \ldots, t_1\}$, $\{t_1+1, t_1+2, \ldots, t_2\}$, $\ldots$, so that OPT downloads services only at the end of frames, i.e., after the t_1-th, t_2-th, $\ldots$, arrivals.

We will calculate the costs of OPT and RL in each frame. We define the cost of OPT in a frame as the sum of the download cost at the beginning of the frame and all the costs of forwarding requests to the back-end cloud during the frame. On the other hand, it can be challenging to define the cost of RL in each frame properly. Consider the example in Fig. 2. RL downloads S_3, and incurs M units of download cost, shortly after the second frame begins. The decision to download S_3 actually involves many requests in the first frame. It is then unreasonable to count all the M units of download cost against the second frame. Instead, we separate the

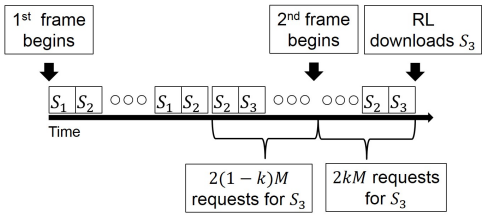


Figure 2: An example for dividing the costs into frames.

M units of download cost as follows: Suppose there are only $2kM$, $k < 1$, requests for S_3 in the second frame when RL downloads it, we say that the download is a *partial download* of fraction k, and only incurs kM units of cost. On the other hand, another partial download of fraction $1-k$ occurs at the end of the first frame, and incurs $(1-k)M$ units of cost. This separation does not change the total cost of RL. To further simplify notation in the next section, we say that, at the end of a frame, all services not in RL have a partial download, possibly with fraction 0. The cost of RL in a frame will then consist of all the download costs, including those from partial downloads, and all the costs of forwarding requests to the back-end cloud, in this frame.

Below, we will calculate the costs of OPT and RL in each frame, and prove Theorem 2 by showing that RL incurs at most $10K$ times as much cost as OPT in each frame.

6.2 Costs in a Frame

Without loss of generality, we calculate the costs in a frame $[t_g+1, t_{g+1}]$. We use $C_{OPT}(g)$ and $C_{RL}(g)$ to be the costs of OPT and RL in this frame, respectively.

We first consider a service $S_i \in OPT(t_g+1)$. Suppose that RL downloads S_i a total number of E_i times, and deletes it from the edge-cloud D_i times. Note that E_i includes partial downloads. Also, $D_i \geq E_i - 1$, as a service needs to be deleted first in order to be downloaded again.

Let $f_{i,z}$ be the number of requests for S_i that RL forwards to the back-end cloud between the $(z-1)$-th download and the z-th download of S_i. Fig. 3 illustrates an example of the downloads and deletions of S_i in a frame, as well as the definition of $f_{i,z}$. S_i then incurs at most

$$E_i M + \sum_z f_{i,z} \quad (3)$$

units of cost under RL.

On the other hand, we have the following lemma for OPT:

LEMMA 1. *Suppose there exists $n < m$, and S_i such that $OPT(l) = OPT(n)$, $S_i \in OPT(l)$, and $S_i \notin RL(l)$, for all $n \leq l \leq m$, then the number of requests that OPT forwards to the back-end cloud between the n-th arrival and the m-th arrival is at least*

$$\sum_{l=n}^{m} x_i(l) - 4M. \quad (4)$$

PROOF. See Appendix A. □

By Lemma 1, if $f_{i,z} > 4M$, for some i, z, then, during the time of these $f_{i,z}$ requests for S_i, OPT forwards at least $f_{i,z} - 4M$ requests to the back-end cloud, and incurs at least $f_{i,z} - 4M$ units of cost. Apply this argument for all z, and we have $C_{OPT}(g) \geq$

We say that the beginning of the frame is also the 0-th download of S_i.

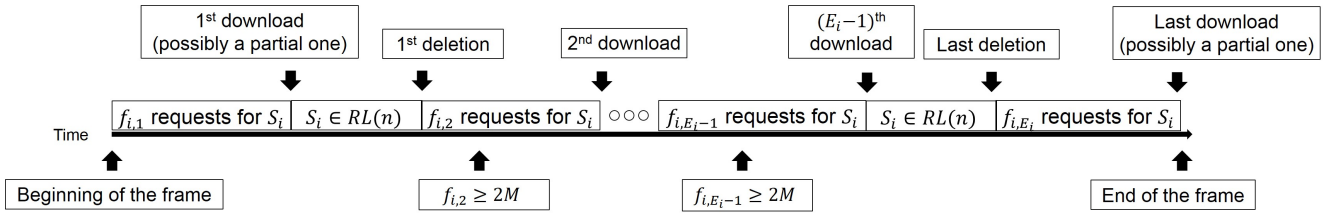


Figure 3: An example of the downloads and deletions of S_i in a frame.

$\sum_z \max\{0, f_{i,z} - 4M\} + M$, where the last term is the download cost at the beginning of the frame. We now have the first bound on $C_{OPT}(g)$:

$$C_{OPT}(g) \geq \max_{i:S_i \in OPT(t_\tau+1)} \sum_{z=1}^{E_i} \max\{0, f_{i,z} - 4M\} + M$$
$$\geq \max_{i:S_i \in OPT(t_\tau+1)} \sum_{z=1}^{E_i} f_{i,z} - 4ME_i + M =: B_1. \quad (5)$$

Next, we consider the deletion of S_i. We have the following lemma for OPT.

LEMMA 2. *Suppose RL deletes a service S_i at the n-th arrival and at the m-th arrival, $n < m$, and $OPT(l) = OPT(m)$, for all $n \leq l \leq m$, then OPT forwards at least $2M$ requests to the back-end cloud between the n-th arrival and the m-th arrival.*

PROOF. See Appendix B. □

By Lemma 2, for every $z > 1$, OPT forwards at least $2M$ requests between the $(z-1)$-th deletion and the z-th deletion of S_i. This gives us the second bound on $C_{OPT}(g)$:

$$C_{OPT}(g) \geq \max_{i:S_i \in OPT(t_\tau+1)} 2M(D_i - 1) + M$$
$$\geq \max_{i:S_i \in OPT(t_\tau+1)} 2ME_i - 3M =: B_2. \quad (6)$$

Finally, we consider a service $S_j \notin OPT(t_g + 1)$. Suppose there are A_j requests for S_j in this frame. OPT needs to forward all these requests to the back-end cloud, for all $S_j \notin OPT(t_g+1)$, which gives us the third bound on $C_{OPT}(g)$:

$$C_{OPT}(g) \geq \sum_{j:S_j \notin OPT(t_\tau+1)} A_j + M =: B_3. \quad (7)$$

On the other hand, with the concept of partial download, RL downloads S_j at most $\frac{A_j}{2M}$ times. S_j then at most incur an amount of

$$A_j + M \times \frac{A_j}{2M} = 1.5A_j \quad (8)$$

units of cost.

Combining (3) and (8) gives us a bound on $C_{RL}(g)$:

$$C_{RL}(g) \leq \sum_{i:S_i \in OPT(t_\tau+1)} (E_iM + \sum_z f_{i,z}) + \sum_{j:S_j \notin OPT(t_\tau+1)} 1.5A_j \quad (9)$$

We are now ready to prove Theorem 2.

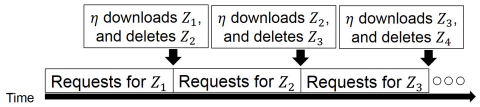


Figure 4: An example for the proof of Theorem 3.

PROOF OF THEOREM 2.

$$C_{RL}(g)$$
$$\leq \sum_{i:S_i \in OPT(t_\tau+1)} (E_iM + \sum_z f_{i,z}) + \sum_{j:S_j \notin OPT(t_\tau+1)} 1.5A_j$$
$$\leq K \left[\max_{i:S_i \in OPT(t_\tau+1)} (E_iM + \sum_z f_{i,z}) + \sum_{j:S_j \notin OPT(t_\tau+1)} 1.5A_j \right]$$
$$\leq K(B_1 + 2.5B_2 + 6.5B_3) \leq 10KC_{OPT}(g),$$

for every frame. □

7. A THEORETICAL LOWER BOUND OF THE COMPETITIVE RATIO

In this section, we prove that the competitive ratio of any deterministic online policy is at least K. Since the competitive ratio of RL is $10K = \theta(K)$, this implies that RL is asymptotically optimal among all deterministic online policies.

THEOREM 3. *The competitive ratio of any deterministic online policy is at least K.*

PROOF. Given a deterministic online policy η, we construct a sequence of arrivals as follows: When the system starts, the first δ_1 arrivals are all requests for a service $Z_1 \notin \{S_1, S_2, \ldots, S_K\}$. Recall that the edge-cloud hosts services $\{S_1, S_2, \ldots, S_K\}$ when the system starts. Therefore, the service Z_1 is not initially hosted by the edge-cloud. If η never downloads Z_1, then we choose δ_1 to be arbitrarily large, and the system ends after δ_1 arrivals of Z_1. In this case, OPT can download Z_1 at the first arrival, and only incurs a cost of M, while η incurs a cost of δ_1. The competitive ratio of η is then $\frac{\delta_1}{M}$, which can be made arbitrarily large. From now on, we can assume that η downloads Z_1 after a finite number of requests for Z_1, and choose the value of δ_1 so that η downloads Z_1, and deletes a service, after δ_1 arrivals of Z_1. Let $Z_2 \in \{S_1, S_2, \ldots, S_K\}$ be the service that η deletes.

We construct the remaining of the sequence of arrivals iteratively: For all $1 < u \leq K$, there are δ_u requests for service Z_u after the first $\sum_{v=1}^{u-1} \delta_v$ arrivals. If η never downloads Z_u, we can make the competitive ratio arbitrarily large by choosing δ_u to be arbitrarily large. Therefore, we can assume that η downloads Z_u after a finite number of requests for Z_u, and choose δ_u to be that number. Let $Z_{u+1} \in \{S_1, S_2, \ldots, S_K, Z_1\}$ be the service that η

deletes when it downloads Z_u. The system ends after the last δ_K requests for Z_K. Fig. 4 illustrates such a sequence.

By the construction of our sequence, η makes K downloads and therefore incurs a cost of at least KM. On the other hand, there are at most K different services among $\{S_1, S_2, \ldots, S_K, Z_1\}$ that have any requests in this sequence. Therefore, at least one service in $\{S_1, S_2, \ldots, S_K\}$ does not have any requests. Let Z^* be that service. An offline policy can then download Z_1 and delete Z^* when the system starts. This only incurs a cost of M, as all subsequent requests are hosted by the edge-cloud after the first download. Therefore, the competitive ratio is at least $\frac{KM}{M} = K$. □

8. PRACTICAL ISSUES

8.1 Extensions to Heterogeneous Systems

Our analysis so far has assumed that all services are homogeneous. In practice, some services are very sensitive to delays, while others are not. Different services also require different storage and have different download cost. We now discuss how to address these heterogeneous features.

We model the heterogeneous features as follows: Forwarding a request for S_i to the back-end cloud incurs a cost of F_i, and downloading the service S_i to the edge-cloud incurs a cost of M_i, with $M_i \geq F_i \geq 1$. Each service S_i requires a storage space of W_i, and the edge-cloud only has a capacity of K. Therefore, if the edge-cloud hosts a subset $\mathbb{T}$ of services, we then have $\sum_{i:S_i \in \mathbb{T}} W_i \leq K$. Our previous analysis corresponds to the special case where $F_i \equiv 1$, $M_i \equiv M$, and $W_i \equiv 1$.

We have the following theorem for OPT in heterogeneous systems:

THEOREM 4. *Suppose we are given a sequence $r_1, r_2, \ldots$, and $OPT(n)$, which is the subset of services hosted by OPT after the n-th arrival. If there exists $m > n$, a service $S_i \notin OPT(n)$, and another service $S_j \in OPT(n)$ with $W_i \leq W_j$ such that:*

$$\sum_{l=n+1}^{m} F_i x_i(l) \geq \sum_{l=n+1}^{m} F_j x_j(l) + M_i + M_j, \qquad (10)$$

then OPT downloads at least one service between the $(n+1)$-th arrival and the m-th arrival.

PROOF. The proof is virtually the same as the proof of Theorem 1. □

With the intuition provided by Theorem 4, we can modify RD and LRU as follows:

DEFINITION 4. *When a request $r_n = S_i$ whose service is not currently hosted by the edge-cloud arrives, RD downloads and replicates S_i at the edge-cloud if there exists a number τ and a service S_j such that $S_j \in RL(l)$ and $S_i \notin RL(l)$, for all $n - \tau \leq l \leq n-1$, and*

$$\sum_{l=n-\tau}^{n} F_i x_i(l) \geq \sum_{l=n-\tau}^{n} F_j x_j(l) + M_i + M_j. \qquad (11)$$

DEFINITION 5. *Suppose the edge-cloud decides to download a service at the n-th arrival. For each service S_i currently hosted by the edge-cloud, let τ_i be the smallest number such that $2M_i \leq \sum_{l=n-\tau_i}^{n} F_i x_i(l)$. LRU sorts all services in descending order of τ_i, and deletes services in this order until there is enough space to download the new service.*

8.2 Implementation and Complexity

This section discusses the implementation and complexity of RL. We focus on the homogeneous system to simplify notation. However, it is straightforward to extend the implementation for heterogeneous systems.

We first discuss the implementation of the retrospective download (RD) policy. Define

$$b_{ij}(n) := \left[\max_{\tau: i \in RL(l), j \notin RL(l), \forall n-\tau \leq l \leq n-1} \sum_{l=n-\tau}^{n} x_j(l) - \sum_{l=n-\tau}^{n} x_i(l) \right]^+,$$

for all $i \in RL(n-1)$ and $j \notin RL(n-1)$, where $x^+ := \max\{0, x\}$. By the definition of RD, a service S_j will be downloaded at the n-th arrival if $b_{ij}(n) \geq 2M$, for some $i \in RL(n-1)$. Finding the value of $b_{ij}(n)$ can be transformed into the well-known maximum subarray problem as follows: Construct a sequence of integers $\{a_n\}$ such that $a_n = 1$ if $x_j(n) = 1$, $a_n = -1$ if $x_i(n) = 1$, and $a_n = 0$, otherwise. We then have

$$b_{ij}(n) = \left[\max_{\tau: i \in RL(l), j \notin RL(l), \forall n-\tau \leq l \leq n-1} \sum_{l=n-\tau}^{n} a_l \right]^+,$$

which can be computed easily as follows: If service S_i is downloaded at the n-th arrival, set $b_{ij}(n) = 0$, for all j. Otherwise, $b_{ij}(n) = [b_{ij}(n-1) + x_j(n) - x_i(n)]^+$.

Next, we discuss the implementation of LRU. When RL decides to download a service, LRU needs to compute τ_i for all services S_i hosted by RL, where τ_i is chosen so that there are $2M$ requests for S_i in the last τ_i requests. In order to obtain τ_i, each service can maintain the arrival times of its last $2M$ requests, which can be easily done by using a queue of size $2M$ to store the arrival times of past requests.

It is straightforward to extend the above discussions for heterogeneous systems. The complete pseudocode of RL for heterogeneous systems is described in Alg. 1. It is easy to check that the time complexity of RL is $O(|\mathbb{S}|)$ for homogeneous systems and $O(K|\mathbb{S}|)$ for heterogeneous systems, where $|\mathbb{S}|$ is the number of all services. The space complexity is $O(|\mathbb{S}|^2)$. Even when the number of unique services is as large as 10^4, the memory footprint of RL is only about 400 MB, which is easily manageable for most modern data center servers.

9. TRACE-BASED SIMULATIONS

In this section, we compare the performance of RL against three offline policies and one online policy using real-world data traces.

9.1 Overview of the Trace

We use a data set from Google cluster [6] to obtain the sequences of service requests. This data set registers the request arrivals from a Google cluster over seven hours, and contains more than three million requests. Each request has a "ParentID" that identifies its service. In this data set, there are 9,218 unique services. The most popular service has 208,306 requests, while 5,150 services only have one request. In all simulations in this section, we break the data set into ten non-overlapping parts, and run policies on these ten parts individually. We then report the average performance of these ten parts.

9.2 Evaluated Policies

In addition to our RL policy, we also implement three different existing offline policies and one online policy. The three offline policies are derived from the optimal solutions based on cache

Algorithm 1 Retrospective download with least recently used (RL)

$b_{ij} \leftarrow 0, \forall i, j$
$n \leftarrow 0$
Initialize a queue, q_i, with $\lceil \frac{2M_i}{F_i} \rceil$ elements of $0, \forall i$
while a new request arrives **do**
  Suppose the request is for service i^*
  $n \leftarrow n + 1$
  Push n into q_{i^*}, and pop out an element
  if $i^* \in RL(n-1)$ **then**
    Serve this request at the edge-cloud
    for $j \notin RL(n-1)$ **do**
      $b_{i^*j} \leftarrow (b_{i^*j} - F_{i^*})^+$
  else if $i^* \notin RL(n-1)$ **then**
    $m \leftarrow$ False
    for $j \in RL(n-1)$ **do**
      $b_{ji^*} \leftarrow b_{ji^*} + F_{i^*}$
      if $b_{ji^*} \geq M_j + M_{i^*}$ and $W_{i^*} \leq K$ **then**
        $m \leftarrow$ True
    if m **then**
      $\tau_j \leftarrow n - (\text{head of } q_j), \forall j \in RL(n-1)$
      Delete j in descending order of τ_j until there is enough space to download i^*
      Download i^*
      $b_{i^*j} \leftarrow 0, \forall j$
      for j that have just been deleted **do**
        $b_{ij} \leftarrow 0, \forall i$
    else
      Forward this request to the back-end cloud

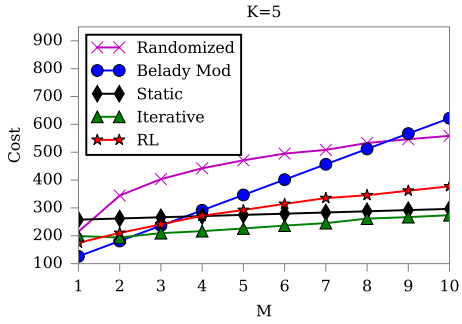
management, stochastic optimization, and dynamic programming, respectively. We describe the implementations of these policies for both homogeneous systems and heterogeneous systems.

Belady Modified. Belady's Algorithm is known to be the offline optimal solution to the cache management problem in computer architecture [3]. It minimizes cache misses by swapping in a new item and deleting the item which will be used in the farthest future. To adopt it in the edge cloud reconfiguration scenario, we make a small modification and allow the requested service to be forwarded instead of always downloaded when the next occurrence of the new service is farther than those of the existing edge services. Belady Modified then achieves the optimal performance when $M = 1$ for homogeneous systems. For heterogeneous systems, Belady Modified keeps deleting services whose next request is farthest in the future until it has enough space to download the new service.

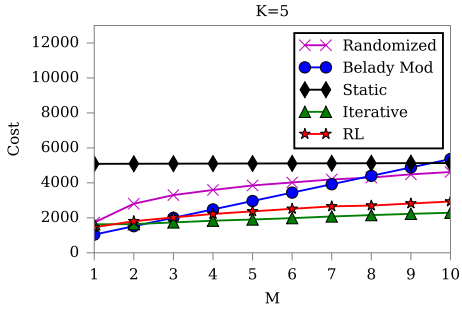
Offline Static. In homogeneous systems, Offline Static computes the frequency of all service requests and simply chooses the top K popular services to host at the edge-cloud. In heterogeneous systems, Offline Static hosts a subset of services so that their total space is no more than K, and the sum of their frequencies is maximized. When the arrivals of requests follow an i.i.d. stochastic process, most online policies that employ stochastic optimization will converge to Offline Static.

Offline Iterative. Given the complete trace, it is possible to compute the optimal solution, OPT, using dynamic programming. However, even for the homogeneous system, the complexity of dynamic programming is at least $O(\binom{|\mathbb{S}|}{K})$ per request. Even when K is as small as 5, our implementation finds that dynamic programming cannot be completed within a reasonable amount of time. Therefore, we instead implement the following iterative policy for homogeneous systems:

Since the edge-cloud can host K services, we say that the edge-cloud has K spaces, numbered as $L_1, L_2, \ldots, L_K$, and each of them can host one service. Offline Iterative algorithm finds the edge service at each of the K spaces iteratively. First, it uses dynamic programming to find services hosted in L_1 so as to minimize the total cost when $K = 1$. Given the solutions for $L_1, L_2 \ldots, L_k$, the policy then uses dynamic programming to find the services hosted in L_{k+1} so that the total cost is minimized when $K = k + 1$, and $L_1, \ldots, L_k$ are given. This policy achieves the optimal performance when $K = 1$. We only test this policy for homogeneous systems since it cannot be easily extended for heterogeneous systems.



(a) Total cost over 10^3 requests



(b) Total cost over 10^4 requests

Figure 5: Cost comparison with different download costs M.

Online Randomized. We consider an online baseline policy in addition to the above offline policies. Intuitively, a reasonable policy should download more often when the downloading cost is small. When a request whose service is not hosted at the edge-cloud arrives, Online Randomized downloads a new service with probability $\frac{1}{M}$, or with probability $\frac{F_i}{M_i}$ in heterogeneous systems. To make room for the new downloaded service, Online Randomized deletes services randomly until there is enough space to download the new service. Since Online Randomized is a randomized policy, we report its average performance over 10 i.i.d. simulation runs on each part of the data set.

9.3 Performance Evaluations for Homogeneous Systems

We implement all the above policies and run the algorithms with different parameters of K and M. For each pair of parameters, we evaluate the total costs over 10^3 requests and over 10^4 requests.

In each run, we set the initial edge services to be the K services with lowest service IDs. The average costs of the aforementioned algorithms are compared in Fig. 5 and Fig. 6.

Fig. 5 compares the costs of the above algorithms while fixing $K = 5$. RL performs very well when compared with other policies. In most settings, Offline Iterative is slightly better than RL, but the difference is very limited. We note that Offline Iterative is a very intelligent policy that requires the knowledge of all future arrivals and has very high complexity. The result that our policy, being an online policy with low complexity, is only slightly worse than Offline Iterative suggests that it works well in practice. Belady Modified achieves better performance than RL when $M = 1$, as it is indeed the optimal policy when $M = 1$. However, as M becomes larger, it quickly becomes much worse than RL. Also note that RL has much better "real-world" performance than that the theoretical result guarantees since the competitive ratio is based on a worst case analysis. Offline Static can be better than RL when we only evaluate it over 10^3 requests, but has worse performance than RL when we evaluate it over 10^4 requests. With more requests, the system witnesses more variations, and therefore Offline Static becomes worse. Finally, Online Randomized performs poorly in all settings.

Fig. 6 shows the costs with different K with $M = 5$. Similar to Fig. 5, Offline Iterative is only slightly better than RL in all settings. Recall that Offline Iterative minimizes the total cost when $K = 1$. This result therefore shows that our RL is close to optimal. Offline Static performs worse than our policy when we evaluate it over 10^4 requests. Both Belady Modified and Online Randomized are worse than RL under all settings.

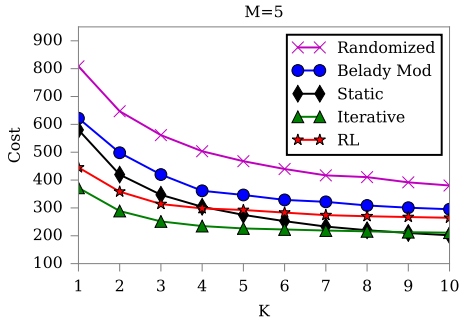
9.4 Performance Evaluations for Heterogeneous Systems

We further evaluate the performance for heterogeneous systems. To create heterogeneity, we assign different forwarding costs $F_i = 1, 2, 3$, downloading costs $M_i = 5, 10, 15$, and space requirements $W_i = 1, 2, 3$ for different services by their IDs. We set the initial edge services with the services with lowest IDs which can fit in the storage while skipping those exceeding the remaining capacity.

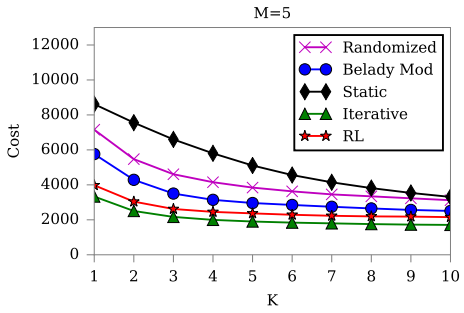
The costs of different policies with varying edge-cloud capacity K are shown in Fig. 7. We can see that RL achieves the minimum cost among all policies under almost all settings. Although we only establish that RL is asymptotically optimal for homogeneous systems, this result suggests that RL remains a desirable solution even in heterogeneous systems.

10. CONCLUSIONS

This paper studies dynamic reconfiguration of edge-clouds. We study a model that captures the limited capacity of edge-clouds, the unknown arrival patterns of requests, and the operational costs of edge-clouds, including the cost of forwarding requests to the back-end cloud and the cost of downloading a new service. We propose an online policy, RL, that aims to reduce the total cost under any arbitrary sequence of arrivals. We evaluate the competitive ratio of RL, and prove that its competitive ratio is at most $10K$, where K is the capacity of the edge-cloud. Moreover, we prove that the competitive ratio of any deterministic online policy is at least $\theta(K)$, and therefore RL is asymptotically optimal. The performance of RL is further evaluated through simulations using data traces from real-world data centers. We compare RL against three offline policies and one online policy. Simulation results demonstrate that our RL achieves better, or similar, performance compared to the other policies in all scenarios. We will consider the theoretical analysis of randomized polices and heterogeneous systems for future work.



(a) Total cost over 10^3 requests



(b) Total cost over 10^4 requests

Figure 6: Cost comparison with different K.

11. REFERENCES

[1] ACHLIOPTAS, D., CHROBAK, M., AND NOGA, J. Competitive analysis of randomized paging algorithms. In *Algorithms-ESA'96*. Springer, 1996, pp. 419–430.

[2] AMBLE, M. M., PARAG, P., SHAKKOTTAI, S., AND YING, L. Content-aware caching and traffic management in content distribution networks. In *2011 Proceedings IEEE INFOCOM*.

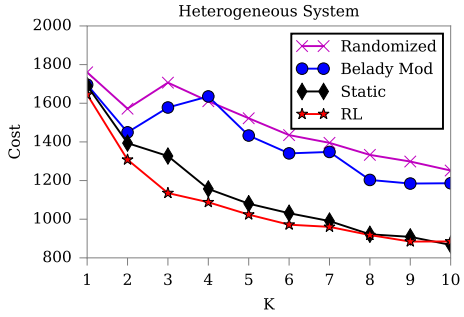
[3] BELADY, L. A. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal 5*, 2 (1966), 78–101.

[4] BONOMI, F., MILITO, R., ZHU, J., AND ADDEPALLI, S. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing* (2012), ACM, pp. 13–16.

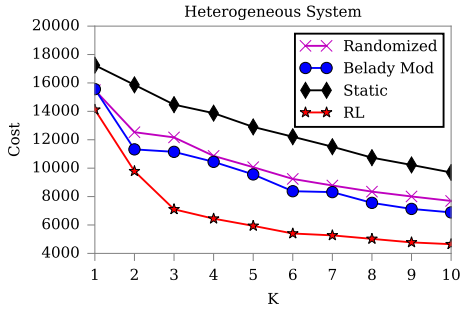
[5] BORST, S., GUPT, V., AND WALID, A. Distributed caching algorithms for content distribution networks. In *2010 Proceedings IEEE INFOCOM* (2010), IEEE, pp. 1–9.

[6] HELLERSTEIN, J. L. Google cluster data. Google research blog, Jan. 2010. Posted at http://googleresearch.blogspot.com/2010/01/google-cluster-data.html.

[7] LEWIS, G., ECHEVERRÍA, S., SIMANTA, S., BRADSHAW, B., AND ROOT, J. Tactical cloudlets: Moving cloud computing to the edge. In *2014 IEEE Military Communications Conference (MILCOM)* (2014), IEEE, pp. 1440–1446.



(a) Total cost over 10^3 requests



(b) Total cost over 10^4 requests

Figure 7: Cost comparison in heterogeneous system.

[8] LIU, J., ZHAO, T., ZHOU, S., CHENG, Y., AND NIU, Z. CONCERT: A cloud-based architecture for next-generation cellular systems. *IEEE Wireless Communications Magazine 21*, 6 (Dec. 2014), 14–22.

[9] LLORCA, J., TULINO, A. M., GUAN, K., ESTEBAN, J., VARVELLO, M., CHOI, N., AND KILPER, D. Dynamic in-network caching for energy efficient content delivery. In *2013 Proceedings IEEE INFOCOM* (2013), IEEE, pp. 245–249.

[10] MOORE, L. K. The first responder network and next-generation communications for public safety: Issues for congress. Congressional Research Service, Library of Congress.

[11] QIU, X., LI, H., WU, C., LI, Z., AND LAU, F. Cost-minimizing dynamic migration of content distribution services into hybrid clouds. In *2012 Proceedings IEEE INFOCOM* (2012), IEEE, pp. 2571–2575.

[12] RAVINDRAN, R., LIU, X., CHAKRABORTI, A., ZHANG, X., AND WANG, G. Towards software defined icn based edge-cloud services. In *2013 IEEE 2nd International Conference on Cloud Networking (CloudNet)* (2013), IEEE, pp. 227–235.

[13] REINEKE, J., AND GRUND, D. Relative competitive analysis of cache replacement policies. In *ACM Sigplan Notices* (2008), vol. 43, ACM, pp. 51–60.

[14] SATYANARAYANAN, M., CHEN, Z., HA, K., HU, W., RICHTER, W., AND PILLAI, P. Cloudlets: at the leading edge of mobile-cloud convergence. In *2014 6th International Conference on Mobile Computing, Applications and Services (MobiCASE)* (2014), IEEE, pp. 1–9.

[15] SATYANARAYANAN, M., LEWIS, G., MORRIS, E., SIMANTA, S., BOLENG, J., AND HA, K. The role of cloudlets in hostile environments. *IEEE Pervasive Computing 12*, 4 (2013), 40–49.

[16] SATYANARAYANAN, M., SIMOENS, P., XIAO, Y., PILLAI, P., CHEN, Z., HA, K., HU, W., AND AMOS, B. Edge analytics in the internet of things. *IEEE Pervasive Computing*, 2 (2015), 24–31.

[17] SLEATOR, D. D., AND TARJAN, R. E. Amortized efficiency of list update and paging rules. *Communications of the ACM 28*, 2 (1985), 202–208.

[18] TADROUS, J., ERYILMAZ, A., AND EL GAMAL, H. Proactive content distribution for dynamic content. In *2013 IEEE International Symposium on Information Theory Proceedings (ISIT)* (2013), IEEE, pp. 1232–1236.

[19] TALEB, T., AND KSENTINI, A. Follow me cloud: interworking federated clouds and distributed mobile networks. *IEEE Network 27*, 5 (2013), 12–19.

[20] URGAONKAR, R., WANG, S., HE, T., ZAFER, M., CHAN, K., AND LEUNG, K. K. Dynamic service migration and workload scheduling in edge-clouds. *Performance Evaluation 91* (2015), 205–228.

[21] WANG, S., URGAONKAR, R., ZAFER, M., HE, T., CHAN, K., AND LEUNG, K. K. Dynamic service migration in mobile edge-clouds. In *Proceedings of IFIP Networking* (2015).

APPENDIX

A. PROOF OF LEMMA 1

This section provides a proof for Lemma 1. By assumptions of the lemma, $OPT(l) \neq RL(l)$, for all $n \leq l \leq m$. Therefore, for every $l \in [n, m]$, there exists some service that is in $RL(l)$ but not in $OPT(l)$. OPT needs to forward requests for these services to the back-end cloud. We will count the number of requests for these services.

When RL downloads a service S_j, it also needs to delete a service. If RL deletes the service hosted by the space L_k, then we say that S_j is hosted by L_k. We use $L_k[r]$ to denote the r-th service hosted at space L_k between the n-th arrival and the m-th arrival. We use $d_k[r]$ to denote the time after which $L_k[r]$ is deleted and $L_k[r+1]$ is downloaded. We set $d_k[0] = n-1$, for all k. If $L_k[r]$ is the last service hosted by L_k before the m-th arrival, we set $d_k[r] = m$. Therefore, the service $L_k[r]$ is hosted at the edge-cloud between the $(d_k[r-1]+1)$-th arrival and the $d_k[r]$-th arrival, for all r.

LEMMA 3. *Suppose* $L_k[r] = S_j$, *then* $\sum_{l=d_k[r-1]+1}^{d_k[r]} x_j[l] \geq \sum_{l=d_k[r-1]+1}^{d_k[r]} x_i[l] - 2M$.

PROOF. We prove this by contradiction. By the assumption of Lemma 1, $S_i \notin RL(l)$, for all $d_k[r-1]+1 \leq l \leq d_k[r]$. If $\sum_{l=d_k[r-1]+1}^{d_k[r]} x_j < \sum_{l=d_k[r-1]+1}^{d_k[r]} x_i - 2M$, then RL would have downloaded S_i before the $d_k[r]$-th arrival. □

We separate all services that are in $RL(l)$ but not in $OPT(l)$, for some $l \in [n, m]$, into two types:

DEFINITION 6. *Suppose* $L_k[r] = S_j \notin OPT(n)$, *then we say*

*that S_j is **type I** if*

$$\sum_{l=n}^{d_k[r]} x_j(l) \geq \sum_{l=n}^{d_k[r]} x_i(l) - 4M, \tag{12}$$

*and say that S_j is **type II** if there exists some $\tau \geq n$ such that*

$$\sum_{l=\tau}^{d_k[r]} x_j(l) \geq \sum_{l=d_k[r-1]+1}^{d_k[r]} x_i(l). \tag{13}$$

By this definition, OPT needs to forward at least $\sum_{l=n}^{d_k[r]} x_i(l) - 4M$ requests for $L_k[r]$ if it is type I, and at least $\sum_{l=d_k[r-1]+1}^{d_k[r]} x_i(l)$ requests for $L_k[r]$ if it is type II. It is possible for a service to be both type I and type II. We first prove that a service $L_k[r] \notin OPT(n)$ is either type I or type II.

LEMMA 4. *Suppose $L_k[r] = S_j \notin OPT(n)$, then S_j is either type I or type II.*

PROOF. If $r = 1$, then $d_k[r-1]+1 = n$, and we have $\sum_{l=n}^{d_k[r]} x_j(l) \geq \sum_{l=n}^{d_k[r]} x_i(l) - 2M > \sum_{l=n}^{d_k[r]} x_i(l) - 4M$ by Lemma 3. In this case, S_j is type I.

Next, we consider the case $r > 1$. S_j is downloaded by RL after the $(d_k[r-1])$-th arrival. By the design of RL, there exists a service S_{j^*} and τ such that $S_{j^*} \in RL(l)$, for all $\tau \leq l \leq d_k[r-1]$, and

$$\sum_{l=\tau}^{d_k[r-1]} x_j(l) \geq \sum_{l=\tau}^{d_k[r-1]} x_{j^*}(l) + 2M. \tag{14}$$

If $\tau \geq n$, then we have

$$\begin{aligned}
\sum_{l=\tau}^{d_k[r]} x_j(l) &= \sum_{l=\tau}^{d_k[r-1]} x_j(l) + \sum_{l=d_k[r-1]+1}^{d_k[r]} x_j(l) \\
&\geq \sum_{l=\tau}^{d_k[r-1]} x_{j^*}(l) + 2M + \sum_{l=d_k[r-1]+1}^{d_k[r]} x_i(l) - 2M \\
&\geq \sum_{l=d_k[r-1]+1}^{d_k[r]} x_i(l),
\end{aligned}$$

and S_j is type II. On the other hand, if $\tau < n$, then we have

$$\sum_{l=\tau}^{n-1} x_j(l) < \sum_{l=\tau}^{n-1} x_{j^*}(l) + 2M, \tag{15}$$

or S_j would have been downloaded earlier. Combining (14) and (15) yields $\sum_{l=n}^{d_k[r-1]} x_j(l) \geq \sum_{l=n}^{d_k[r-1]} x_{j^*}(l)$. Therefore,

$$\begin{aligned}
\sum_{l=n}^{d_k[r]} x_j(l) &= \sum_{l=n}^{d_k[r-1]} x_j(l) + \sum_{l=d_k[r-1]+1}^{d_k[r]} x_j(l) \\
&\geq \sum_{l=n}^{d_k[r-1]} x_{j^*}(l) + \sum_{l=d_k[r-1]+1}^{d_k[r]} x_i(l) - 2M \\
&\geq \sum_{l=d_k[r-1]+1}^{d_k[r]} x_i(l) - 4M,
\end{aligned}$$

and S_j is type I. □

We are now ready to prove Lemma 1.

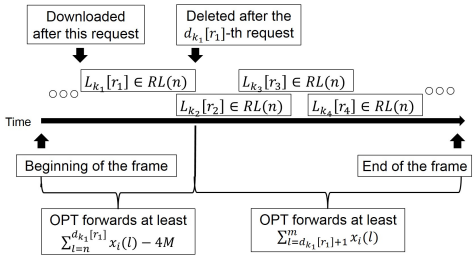


Figure 8: An example for the proof of Lemma 1.

PROOF OF LEMMA 1. Let $L_{k_1}[r_1]$ be the type I service with the largest $d_k[r]$. Since $OPT(l) \neq RL(l), \forall d_{k_1}[r_1]+1 \leq l \leq m$, we can find a set of type II services $\{L_{k_2}[r_2], L_{k_3}[r_3] \dots\}$ such that the union of $[d_{k_j}[r_j - 1]+1, d_{k_j}[r_j]]$ covers $[d_{k_1}[r_1]+1, m]$. Fig. 8 illustrates an example of finding $L_{k_1}[r_1], L_{k_2}[r_2], \dots$.

By the definition of type II service, the total number of requests that OPT needs to forward for these services is at least

$$\sum_{l=d_{k_1}[r_1]+1}^{m} x_i(l).$$

Also, OPT needs to forward at least

$$\sum_{l=n}^{d_{k_1}[r_1]} x_i(l) - 4M$$

requests for $L_{k_1}[r_1]$. Therefore, in total, OPT needs to forward at least $\sum_{l=n}^{m} x_i(l) - 4M$ requests. □

B. PROOF OF LEMMA 2

PROOF OF LEMMA 2. By the first condition of RD, there are at least $2M$ requests for S_i between the n-th arrival and the m-th arrival. Otherwise, S_i cannot be downloaded between the n-th arrival and the m-th arrival, and therefore cannot be deleted at the m-th arrival.

When S_i is to be deleted at the m-th arrival, it must have the largest τ_i among all services currently hosted by RL, where τ_i is defined in Def. 3. Since there are at least $2M$ requests for S_i between arrival $[n, m]$, all services in $RL(m)$ have at least $2M$ requests between arrival $[n, m]$.

First, consider the case $RL(m-1) \neq OPT(m-1)$. There must exist a service S_j such that $S_j \in RL(m-1)$, but $S_j \notin OPT(m-1)$. OPT needs to forward all requests for S_j between arrival $[n, m]$, and there are at least $2M$ of them.

Next, consider the case $RL(m-1) = OPT(m-1)$. At the m-th arrival, RL deletes S_i in order to download another service $S_j \notin RL(m-1) = OPT(m-1)$. By the design of RL, there exists τ and another service $S_{i^*} \in RL(m-1)$ such that the conditions in Def. 2 are satisfied. In particular, $\sum_{l=m-\tau}^{m} x_j(l) \geq \sum_{l=m-\tau}^{m} x_{i^*}(l) + 2M$. If $m - \tau \geq n$, then there are at least $2M$ requests for S_j between arrival $[n, m]$, and OPT needs to forward all of them. On the other hand, consider the case when $m - \tau < n$. Since RL does not download S_j until the m-th arrival, we have $\sum_{l=m-\tau}^{n-1} x_j(l) < \sum_{l=m-\tau}^{n-1} x_{i^*}(l) + 2M$, and therefore $\sum_{l=n}^{m} x_j(l) \geq \sum_{l=n}^{m} x_{i^*}(l) \geq 2M$. OPT still needs to forward at least $2M$ requests. □