Partial Service Caching at the Edge

Rudrabhotla Sri Prakash, Nikhil Karamchandani, Veeraruna Kavitha, and Sharayu Moharir
Indian Institute of Technology Bombay

***Abstract*—We consider the problem of service caching at the edge, in which an application provider can rent resources at the edge to cache its codes/libraries to serve incoming requests. A key characteristic of current edge computing platforms is that they provide pay-as-you-go flexibility via short term contracts. This implies that the client can make significant cost benefits by dynamically adjusting its caching decisions depending on the intensity of service requests. A novel feature of this work in contrast to the literature is that we allow the service to be partially cached at the edge. The probability that the edge is able to serve an incoming request is assumed to be an increasing function of the fraction of service cached. The focus of this work is on characterizing the benefits of partial service caching at the edge. The key insight from this work is that partial service caching can bring down the cost of service only if there is at least one intermediate caching level at which the probability that an incoming request can be served at the edge is strictly more than the fraction of service cached and the cost of renting edge storage resources falls within a range of values which depends on the statistics of the request arrival process. We use these insights to design near-optimal service caching policies.**

I. Introduction

Software as a Service (SaaS) instances like social networking platforms, online shopping platforms, and navigation apps have had a huge impact on our lives in recent times and continue to grow in popularity. This phenomenon has in part been fueled by cloud-computing which allows service developers to use computational and storage resources in the cloud to serve their customers [1]. Even though the computational resources in the cloud are unparalleled in their processing power, serving requests via the cloud increases latency due to the time required to transfer data between the users and the cloud. Latency leads to deterioration in the quality of service.

A promising solution to this roadblock is edge computing [2]. Edge computing refers to serving requests using storage and computation resources at the edge of the network, i.e., close to the users. Broadly speaking, the edge is defined as any point between the end-users and the cloud data centers. In this work, we consider the setting where third-party edge resources (storage/computation) can be rented at a cost via short-term contracts [3], [4]. We say a service is *cached at the edge* if the code and data/libraries required to serve requests is stored on the edge server.

Usage patterns show that the request arrival rates for many services are time-inhomogeneous. For instance, for a navigation service like Google Maps, congestion on a particular route after an accident causes an increase in the request arrival rate. This is because drivers either want to get an estimate of their expected time of arrival at the destination or attempt to find alternative routes. As this congestion resolves, the request arrival rate reduces. Since the resources at the edge can be rented via short-term contracts, the cache state can be changed as the request arrival rate changes. This flexibility is exploited to design dynamic caching policies.

The key novelty of this work is the option to partially cache the service at the edge. Many services are naturally suited for partial caching. For instance, for a translation service like Google Translate, the edge can be equipped to handle translation tasks for certain language pairs and all other requests are served by the cloud. Similarly, for a navigation service like Google Maps, the route search algorithm and road information of certain geographical areas can be cached at the edge. Requests that can be answered by running the search algorithm on the partial road information cached at the edge are served at the edge and the rest are forwarded to the cloud. In addition, designing services as a collection of independent microservices instead of a monolith is a paradigm which is growing in popularity [5]. All services designed as collections of microservices are also candidates for partial caching.

We model the overall cost of service as the cumulative cost incurred due to latency in service, bandwidth consumption, and renting edge resources. When the service is partially cached, the probability that the edge is able to serve an incoming request is assumed to be a non-decreasing function of the fraction of service cached. The main takeaway from this work is that partial caching can reduce the average cost of service only if there exists at least one intermediate caching level for which the probability that an incoming request can be served at the edge is more than the fraction of service cached. This insight is useful in two ways: *(i)* it serves as a guide to design partitions of a service into microservices, and *(ii)* it assists in designing caching policies.

A. Our Contributions

We first formulate the dynamic caching problem as a Markov Decision Process (MDP) and the optimal policy corresponds to the solution of this MDP, which decides what fraction of the service to cache in the next time-slot based on the current state information. We also study a variant of the MDP which restricts the action set at each step to be binary (cache/don't cache). We propose another caching policy called Arrival Based Caching (ABC) which, unlike the MDP, makes the caching decisions based on partial state information.

We show that for the case when there is no intermediate caching level at which the probability that an incoming request can be served at the edge is strictly more than the fraction of service cached, all the three policies are equivalent in terms of their long-term average cost (Theorems 2, 3, and

ISBN 978-3-903176-29-4

4). Furthermore, for this case, none of the policies (including the optimal policy corresponding to the solution of the MDP) use partial caching.

For the complementary case, i.e., when there exists at least one intermediate caching level at which the probability that an incoming request can be served at the edge is strictly more than the fraction of service cached, finding an analytical solution to the MDP proves elusive. On the other hand, the ABC policy framework which bases decisions on partial state information (and thus can potentially be sub-optimal) turns out to be more tractable and we are able to explicit characterizations in a few cases. We also conduct simulations for this case and show that the performance of ABC is indistinguishable from the optimal policy. Finally, we also observe that partial caching can indeed bring down the cost of service in this case.

B. Related work

The advent of new bandwidth-intensive and latency-sensitive applications such as Augmented/Virtual Reality, the Internet of Things, and autonomous vehicles has been accompanied by the emergence of various edge computing systems and architectures [6]–[8]. We now briefly review some of the academic works which attempt to model and analyze such systems, focusing primarily on works closest to ours in spirit.

One line of work [9]–[13] poses the problem of designing efficient edge computing systems as a one-shot static optimization problem to decide which services to cache at the edge and which computation tasks to offload. Our work differs from this approach in that we are interested in designing online algorithms which adapt their service placement decisions over time, depending on the varying number of requests. There are broadly two classes of works which look at the dynamic setting: *1) adversarial requests*: the focus here is on arbitrary request arrival processes and the goal is to provide worst-case guarantees on the performance of the proposed service caching schemes, see for example [14], [15]; *2) stochastic arrivals*: here, the request process is assumed to be generated from a stochastic model and the measure of efficiency is the 'average' cost of serving the requests, see for example [16] which assumes that the requests follow a Poisson process. In this work, we take the latter approach; however, unlike most previous work we consider temporal changes in the request process and model requests as a time-inhomogeneous stochastic process whose intensity is modulated by an underlying discrete-time Markov chain. One work which does consider such temporal variations in the request process, albeit in a different context, is [17] which considers a Markovian model for user mobility and uses a Markov Decision Process (MDP) framework to decide when and which services to migrate between different edge servers as the users move around. Finally, a key distinguishing feature of our work with respect to the literature is that we allow for partial caching of the service and show that this flexibility can provide significant benefits in terms of the efficiency of service caching policies.

Most of the literature on caching policies has dealt with *content caching* problem, which considers the problem of delivering multimedia content such as music or videos by deploying storage caches close to the end users. See for example [18]–[23]. While the problem of service caching does resemble the content caching problem, there are some key differences. In the content caching problem, upon any cache miss, the requested content has to be fetched entirely from the cloud server. On the other hand, in the service caching problem, if a request for a service arrives when it is not cached, there are two options: *(a) request forwarding* which simply forwards the service request to the back-end server, which then carries out all the relevant computation for addressing the request; and *(b) service download* which downloads all the data and code needed for running the service from the back-end server and caches it on the edge server. The cost for these two options differ, depending for example on the amount of network bandwidth needed or the latency incurred for each of them. Motivated by empirical evidence [2], [24], a natural assumption is that the cost of forwarding a single request to the back-end server is lower than the cost of downloading the entire service to cache it on the edge server. These differences are critical and in fact it can be shown that schemes which work well in the content caching setting turn out to be highly inefficient for the service caching problem [14], [15].

C. Organization

The rest of the paper is organized as follows. Section II describes our problem setting. Section III details some natural classes of policies for our setting and presents our results on the structure as well as performance of the optimal policy in each class. We propose and study an alternate, more tractable policy in Section IV. We present simulation results in Section V. The proofs are discussed in the appendix of [25].

II. System Setup

A. Arrival Process

We consider a time-slotted system with stochastic request arrivals. At the beginning of time-slot t, requests arrive in a batch according to a stochastic process, such that the expected number of requests in the batch is $\lambda(t)$, where $\lambda(t)$ can take K distinct values, namely, λ_k for $1 \leq k \leq K$. Without loss of generality, $\lambda_i < \lambda_j$ for $i < j$. We consider the setting where $\lambda(t)$ evolves in a Markovian manner. We define transition probabilities $r_{i,j}$ for $1 \leq i,\ j \leq K$, where $r_{i,j} = \mathbb{P}(\lambda(t) = \lambda_j | \lambda(t-1) = \lambda_i)$.

For some of the results presented in this paper, we consider a special case of this Markovian process with $K = 2$.

Assumption 1 (Two-state Arrival Process): The request arrival rate $\lambda(t)$ takes on two values λ_H and λ_L, with $\lambda_L < \lambda_H$. The transition probabilities are as follows:

$$\begin{aligned}
&\mathbb{P}(\lambda(t) = \lambda_L | \lambda(t-1) = \lambda_H) = p,\\
&\mathbb{P}(\lambda(t) = \lambda_H | \lambda(t-1) = \lambda_H) = 1-p,\\
&\mathbb{P}(\lambda(t) = \lambda_H | \lambda(t-1) = \lambda_L) = q,\\
&\mathbb{P}(\lambda(t) = \lambda_L | \lambda(t-1) = \lambda_L) = 1-q.
\end{aligned}$$

B. Service Model

We study a system consisting of a back-end (cloud) server and an edge server in proximity to the end-user. The back-end server always stores the service and can serve all requests that are routed to it. In addition, the service can be cached on the edge server either in its entirety or in part by paying the appropriate renting cost.

Let ρ_t denote the fraction of service cached at the edge-server at the beginning of time-slot t. The probability that the edge is able to serve an incoming request is a non-decreasing function of ρ_t. We use a function $g(.)$ to quantify this probability, where $g : [0,1] \to [0,1]$ is a non-increasing function with $g(0) = 1$ and $g(1) = 0$. Each incoming request in time-slot t can be served by the edge server with probability $1 - g(\rho_t)$, independent of all other requests.

C. Sequence of Events in a Time-slot

The following sequence of events occurs in each time-slot. We first have request arrivals. Each incoming request is served by the edge server if possible. Requests which cannot be served by the edge server are routed to the back-end server. The system then makes a caching decision for the next time-slot.

D. Cost Model and Constraints

Our cost model builds on the model proposed in [14] to incorporate partial service caching.

For a policy $\mathcal{P}$, the total cost incurred in time-slot t, denoted by $C_t^{\mathcal{P}}$, is the sum of the following three costs.

- *Service cost* ($C_{S,t}^{\mathcal{P}}$): Each request forwarded to the back-end server is served at the cost of one unit.
- *Fetch cost* ($C_{F,t}^{\mathcal{P}}$): On each fetch of the entire service from the back-end server to cache on the edge-server, a fetch cost of M (> 1) units is incurred. The cost incurred to fetch parts of the service from the bank-end server to cache on the edge-server is proportional to the fraction of service fetched.
- *Rent cost* ($C_{R,t}^{\mathcal{P}}$): A renting cost of c units is incurred to cache the entire service on the edge server for a time-slot. The cost incurred to store parts of the service on the edge-server is proportional to the fraction of service cached.

Let the number of requests arriving in time-slot t be denoted by X_t. Let $Y_t(i)$ be the indicator random variable denoting if the i^{th} request arriving in time-slot t can be served by the edge-server. Then, $C_t^{\mathcal{P}} = C_{S,t}^{\mathcal{P}} + C_{F,t}^{\mathcal{P}} + C_{R,t}^{\mathcal{P}}$, such that $C_{S,t}^{\mathcal{P}} = \sum_{i=1}^{X_t}(1-Y_t(i))$, $C_{F,t}^{\mathcal{P}} = M(\rho_t - \rho_{t-1})^+$, $C_{R,t}^{\mathcal{P}} = c\rho_t$.

E. Algorithmic Challenge and Goal

The algorithmic challenge is to design a policy which decides what fraction of the service to cache on the edge server in each time-slot. We consider the setting where the policy knows the value of λ_t before making the caching decision at the end of time-slot t. Note that this caching decision determines the fraction ρ_{t+1} of the service that will be cached in the next time-slot $t+1$.

The goal is to design an online caching policy which makes caching decisions based on the knowledge or estimate of the current request arrival rate and various costs, i.e., the rent cost c, the fetch cost M, and the function g, to minimize the infinite horizon time-average of the expected cost incurred in per time-slot, i.e.,

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[C_t^{\mathcal{P}}\right]. \tag{1}$$

III. Optimal Policies

In this section, we characterize optimal static/dynamic caching policies with/without partial caching.

A. Optimal Policy with Partial Caching

We formulate our caching problem as a Markov Decision Process (MDP). We refer to this MDP as *MDP w/ PartialCaching*. Under assumptions discussed in [26], optimizing the average cost (1) is equivalent to optimizing the discounted cost with discount factor γ close to 1. In view of this, we instead optimize for the discounted cost, i.e., $\min_{\mathcal{P}} \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{(t-1)} C_t^{\mathcal{P}}\right]$. The MDP w/ PartialCaching has the following components.

- States: The state of the system at time t, denoted by $s(t)$, is given by $(\Lambda_{t-1}, \rho_{t-1})$. Note that $s \in \{\lambda_1, \cdots, \lambda_K\} \times [0,1]$. For simplicity we denote ρ_{t-1} as ρ, ρ_t as ρ', Λ_{t-1} as Λ, and Λ_t as Λ'. We initialize $\rho_1 = 0$.
- Actions: Let $a(t) \in [0,1]$ be the action of the MDP at time t. Any stationary Markov policy is represented by $\mathcal{P} = \{a(\Lambda, \rho); \forall(\Lambda, \rho)\}$. At any time t, the policy specifies a state dependent action $a(t) = a(\Lambda_{t-1}, \rho_{t-1}) = \rho_t$, which is the fraction of service to be cached given the state.
- Transition probabilities: The transition probability from state s to s' under action a is

$$\mathbb{P}(s'|s,a) = \mathbb{1}_{\{\rho'=a\}} \mathbb{P}(\Lambda_t|\Lambda_{t-1}).$$

  Note that the evolution of arrival rates is independent of the actions taken.
- Reward: If action a is chosen in a time-slot, the expected immediate reward is

$$\mathbb{E}[r(s'|s,a)] = ca + M(a-\rho)^+ + g(a)\mathbb{E}[\Lambda'|\Lambda].$$

  This is the sum of the three types of costs (service, fetch, and rent costs) discussed in Section II.
- Discount factor: γ.

The dynamic programming (DP) equations for the MDP are

$$\begin{aligned} v(\lambda_H, \rho) = \min_a \{ & ca + M(a-\rho)^+ + g(a)\tilde{\lambda}_H + \\ & \gamma(1-p)v(\lambda_H, a) + \gamma p v(\lambda_L, a)\}, \\ v(\lambda_L, \rho) = \min_a \{ & ca + M(a-\rho)^+ + a(a)\tilde{\lambda}_L + \\ & \gamma(1-q)v(\lambda_L, a) + \gamma q v(\lambda_H, a)\}. \end{aligned}$$

For the special case when $g(\rho) = 1 - \rho$, the optimal policy can be obtained in closed form, as shown in our first result.

Theorem 1: For the arrival process satisfying Assumption 1, let $a_H^*(\rho)$, $a_L^*(\rho) \in [0,1]$ be optimal actions when system is in state (λ_H, ρ), (λ_L, ρ) respectively. Let $\bar{\lambda} = \frac{q}{p+q}\tilde{\lambda}_H + \frac{p}{p+q}\tilde{\lambda}_L$, where $\tilde{\lambda}_H = p\lambda_L + (1-p)\lambda_H$ and $\tilde{\lambda}_L = q\lambda_H + (1-q)\lambda_L$. Then for $g(\rho) = 1-\rho$, the values of $a_H^*(\rho)$, $a_L^*(\rho)$ are as shown in Table I.

Cases	$\lim_{\gamma\to 1}(a_H^*(\rho), a_L^*(\rho))$
1. $c > \max\{\tilde{\lambda}_H, \tilde{\lambda}_L\}$	$(0,0)$
2. $c < \min\{\tilde{\lambda}_H, \tilde{\lambda}_L\}$	$(1,1)$
3. $M < \frac{\tilde{\lambda}_H - c}{p}$ & $M < \frac{c - \tilde{\lambda}_L}{q}$	$(1,0)$
4. $M > \frac{\tilde{\lambda}_H - c}{p}$ & $\bar{\lambda} < c < \tilde{\lambda}_H$	$(\rho,0)$
5. $M > \frac{c - \tilde{\lambda}_L}{q}$ & $\tilde{\lambda}_L < c < \bar{\lambda}$	$(1,\rho)$
6. $M > \frac{\tilde{\lambda}_L - c}{q}$ & $c > \bar{\lambda}$ & $c < \tilde{\lambda}_L$	$(0,\rho)$
7. $M > \frac{c - \tilde{\lambda}_H}{p}$ & $c > \tilde{\lambda}_H$ & $c < \bar{\lambda}$	$(\rho,1)$
8. $M < \frac{\tilde{\lambda}_L - c}{q}$ & $M < \frac{c - \tilde{\lambda}_H}{p}$	$(0,1)$

TABLE I: Solution of MDP w/ PartialCaching for arrival processes satisfying Assumption 1, with $g(\rho) \geq 1-\rho$ for all $\rho \in [0,1]$. This is also the solution of the MDP w/o PartialCaching for arrival processes satisfying Assumption 1.

Refer to Figure 1 for a pictorial representation of the solution in Table I for three cases: $p+q<1$, $p+q=1$ and $p+q>1$.

We use Theorem 1 to obtain the optimal policy in closed form for more general cases. Specifically, our next result characterizes the optimal caching policy for the arrival process satisfying Assumption 1 in the case when there is no intermediate caching level for which the probability that an incoming request can be served at the edge is more than the fraction of service cached, i.e., $1-g(\rho) \leq \rho$ for all $\rho \in [0,1]$.

Theorem 2: For the arrival process satisfying Assumption 1, let $a_H^*(\rho)$, $a_L^*(\rho) \in [0,1]$ be the optimal actions when the system is in state (λ_H, ρ), (λ_L, ρ) respectively. Let $\bar{\lambda} = \frac{q}{p+q}\tilde{\lambda}_H + \frac{p}{p+q}\tilde{\lambda}_L$, where $\tilde{\lambda}_H = p\lambda_L + (1-p)\lambda_H$ and $\tilde{\lambda}_L = q\lambda_H + (1-q)\lambda_L$. If $g(\rho) \geq 1-\rho$ for all $\rho \in [0,1]$, the values of $a_H^*(\rho)$, $a_L^*(\rho)$ are as shown in Table I.

Remark 1: We conclude the following from Theorem 2.

(i) For arrival processes satisfying Assumption 1, if $g(\rho) \geq 1-\rho$ for all $\rho \in [0,1]$, and the initial cache state $\rho_1 = 0$, the optimal policy does not use partial caching. To see this, note that when $(a_H^*(\rho), a_L^*(\rho))$ is $(\rho, 0)$, the optimal action will be $(0,0)$ since $\rho_1 = 0$. Similarly, if $(a_H^*(\rho), a_L^*(\rho))$ is $(1, \rho)$, the optimal action will be $(1,0)$ until the time when the request arrival rate makes a transition from λ_H to λ_L, beyond which the value of (a_H^*, a_L^*) becomes $(1,1)$.

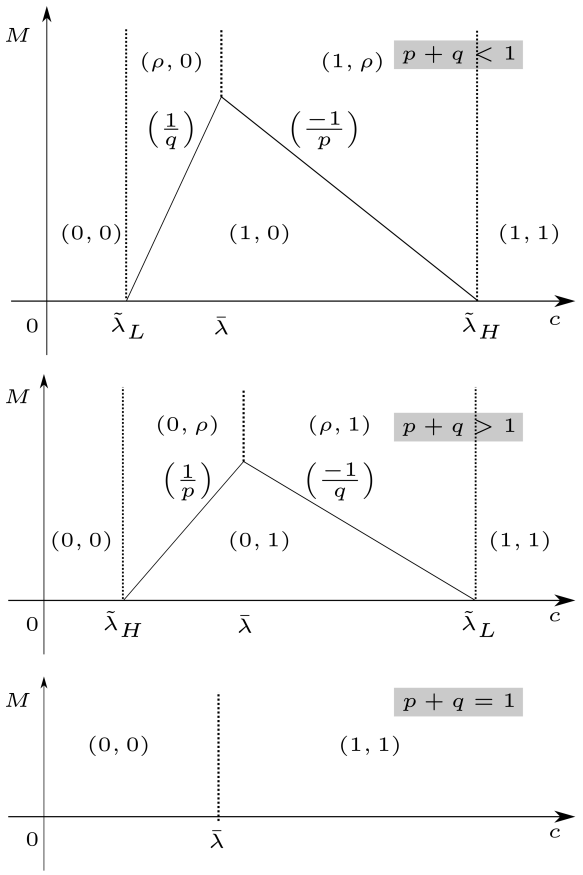

Fig. 1: Solution $(a_H^*(\rho), a_L^*(\rho))$ to the MDP w/ PartialCaching for arrival processes satisfying Assumption 1 with $g(\rho) \geq 1-\rho$ for all $\rho \in [0,1]$ for three cases, namely, $p+q<1$, $p+q>1$, $p+q=1$, as a function of the values of the rent cost c and the fetch cost M. For each region, we show the values of $(a_H^*(\rho), a_L^*(\rho))$ as defined in Theorem 1.

(ii) For $p+q<1$ and $p+q>1$, for very low and very high values of c, the storage policy is static, the fraction of service cached does not change as the request arrival rate changes. As expected, for very low/high values of c, the service is always/never cached. In addition, for high values of M, the fraction of service cached does not change as the request arrival rate changes. For $p+q=1$, the optimal caching policy is static for all values of M and the service is always cached when c is less than the average request arrival rate and never cached otherwise.

A closed form characterization of the optimal policy for the case where $g(\rho) < 1-\rho$ for some $\rho \in [0,1]$ remains an open problem. We present numerical results for this case in the simulations section.

B. Optimal Policy without Partial Caching

Next, we focus on the setting where, at all times, the service is either cached completely or not cached at all, i.e., $\rho_t = \{0,1\}$ for all t. We again formulate our caching problem as a Markov Decision Process (MDP). We refer to this MDP as

MDP w/o PartialCaching and it is identical to the MDP w/ PartialCaching except that the action $a(t) \in \{0,1\}$ and the expected immediate reward is $\mathbb{E}[r(s'|s,a)] = ca + M(a-\rho)^+ + \mathbb{1}_{\{a=0\}}\mathbb{E}[\Lambda'|\Lambda]$.

Our next result characterizes the optimal policy for the MDP w/o Partial Caching when the arrival process satisfies Assumption 1.

Theorem 3: For the arrival process satisfying Assumption 1, let $a_H^*(\rho)$, $a_L^*(\rho) \in \{0,1\}$ be the optimal actions when the system is in state (λ_H, ρ), (λ_L, ρ) respectively. Let $\bar{\lambda} = \frac{q}{p+q}\tilde{\lambda}_H + \frac{p}{p+q}\tilde{\lambda}_L$, where $\tilde{\lambda}_H = p\lambda_L + (1-p)\lambda_H$ and $\tilde{\lambda}_L = q\lambda_H + (1-q)\lambda_L$. Then, the values of $a_H^*(\rho)$, $a_L^*(\rho)$ are as shown in Table I.

Remark 2: The solution to the MDP w/o PartialCaching is the same as that of the MDP w/ PartialCaching for the case where $g(\rho) \geq 1-\rho$ for all $\rho \in [0,1]$. Furthermore, it does not involve partial caching for any choice of system parameters. However, when $g(\cdot)$ does not satisfy the above constraint, partial caching can indeed provide significant benefits, as illustrated via simulations in Section V.

C. Optimal Static Policy with Partial Caching

We say that a policy is static if the fraction of service cached does not change with time. The optimal static policy without partial caching has the following structure.

$$\rho_t = \arg\min_{\rho\in[0,1]} c\rho + g(\rho)\bar{\lambda}, \tag{2}$$

where $\bar{\lambda}$ is the time-average of the request arrival rate.

D. Optimal Static Policy without Partial Caching

Restricting ρ to be in $\{0,1\}$ in (2), we find that the optimal static policy without partial caching has the following structure.

$$\rho_t = \begin{cases} 1 & \text{if } c < \bar{\lambda}, \\ 0 & \text{otherwise,} \end{cases}$$

where $\bar{\lambda}$ is the time-average of the request arrival rate.

IV. Arrival Based Caching (ABC)

In this section we propose a policy called Arrival Based Caching (ABC). ABC makes caching decisions based only on the current request arrival rate, i.e., the caching decision at the end of time-slot t is made based on the value of λ_t. Note that this caching decision determines the fraction ρ_{t+1} of the service that will be cached in the next time-slot $t+1$. Unlike the MDP w/ PartialCaching (Section III-A) which makes decisions based on the current arrival rate and cache state, ABC does not take into account the current cache state ρ_t while making decisions. This makes the ABC policy analytically tractable even in cases in which the MDP is not.

To implement ABC, we choose values of ρ_k for $1 \leq k \leq K$ such that $\rho_{t+1} = \rho_k$ if $\lambda_t = \lambda_k$. Let $\mathcal{S} = \{(u,v,w) : 1 \leq u,v,w \leq K\}$, where u, v, w represent the index of the arrival rates in time-slots $t-2$, $t-1$, and t respectively. Under the model discussed in Section II, $r_{u,v}$ denotes the probability of transition from λ_u to λ_v. Let π_u be the steady state probability of the arrival rate being λ_u. Then, the steady-state probability of state $s = (u,v,w) \in \mathcal{S}$ is given by by $\mathbb{P}(s) = \pi_u r_{uv} r_{vw}$.

Under ABC, for $s = (u,v,w) \in \mathcal{S}$, the expected cost incurred in slot t is given by $C(s) = c\rho_v + M(\rho_v - \rho_u)^+ + \lambda_w g(\rho_v)$. ABC uses the values of $\{\rho_1, \cdots, \rho_K\}$ which minimize the expected cost. Formally,

$$\{\rho_k^* : 1 \leq k \leq K\} = \arg\min_{\{\rho_k : 1\leq k\leq K\}} \sum_{s\in\mathcal{S}} \mathbb{P}(s)C(s)$$
$$\text{s.t } 0 \leq \rho_k \leq 1, \forall\, 1 \leq k \leq K.$$

Example 1: In Fig. 2, we illustrate the ABC policy for the case when $g(\rho) \geq 1-\rho$ for all $\rho \in [0,1]$ and the arrival process satisfies Assumption 1 with $p+q<1$, $p+q=1$, or $p+q>1$.

Remark 3: We conclude from Fig. 2 that for arrival processes satisfying Assumption 1, if $g(\rho) \geq 1-\rho$ for all $\rho \in [0,1]$, the ABC policy does not use partial caching, i.e., it either caches the whole service or does not cache it at all. This property is also satisfied by the optimal policy, see Remark 1.(i).

Our next result compares the performance of ABC to the optimal policy defined in Section III-A for the above class of $g(\cdot)$ functions.

Theorem 4: For arrival processes satisfying Assumption 1, the ABC policy is optimal if $g(\rho) \geq 1-\rho$ for all $\rho \in [0,1]$.

The above result follows by comparing the optimal actions under MDP w/ Partial Caching and ABC shown in Fig. 1 and Fig. 2 respectively, and the argument in Remark 1.(i). A key consequence of this result is that when $g(\rho) \geq 1-\rho$ for all $\rho \in [0,1]$, the ABC policy performs as well as the optimal policy even though it only uses partial state information to make caching decisions.

Next, we study the ABC policy for some other classes of g functions.

Example 2: In Fig. 3, we illustrate the ABC policy for the case when $g(\rho)$ is differentiable and strictly convex and the arrival process satisfies Assumption 1 with $p+q<1$. Caching decisions under ABC for $p+q \geq 1$ can be obtained similarly.

Example 3: In Fig. 4, we illustrate the ABC policy for the case when $g(\rho) = 1$ for $0 \leq \rho < \alpha$, $g(\rho) = \beta < 1-\alpha$ for $\alpha \leq \rho < 1$ and $g(1) = 0$ and the arrival process satisfies Assumption 1 with $p+q<1$. Caching decisions under ABC for $p+q \geq 1$ can be obtained similarly.

Remark 4: We note that for the cases considered in Examples 2 and 3, under ABC, partial caching is used for intermediate values of the rent cost c. For very low values of c, the entire service is always cached and for very high values of c, the service is not cached.

While a closed form characterization of the ABC policy for the general case where $g(\rho) < 1-\rho$ for some $\rho \in [0,1]$ is not available, we present numerical results for this case in the simulations section.

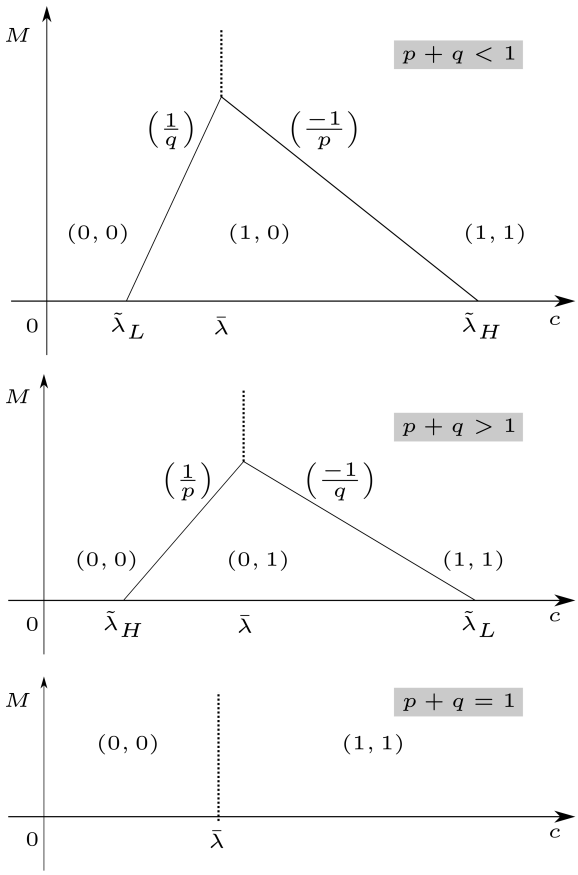

Fig. 2: Caching under the ABC policy for arrival processes satisfying Assumption 1 and $g(\rho) \geq 1-\rho$ for all $\rho \in [0,1]$ for the three cases, namely, $p+q<1$, $p+q>1$, $p+q=1$, as a function of the values of the rent cost c and the fetch cost M. For each region, we show the values of $(\rho_H,\, \rho_L)$ as defined in Section IV. Here $\bar{\lambda}$, $\tilde{\lambda}_H$ and $\tilde{\lambda}_L$ are as defined in Theorem 1.

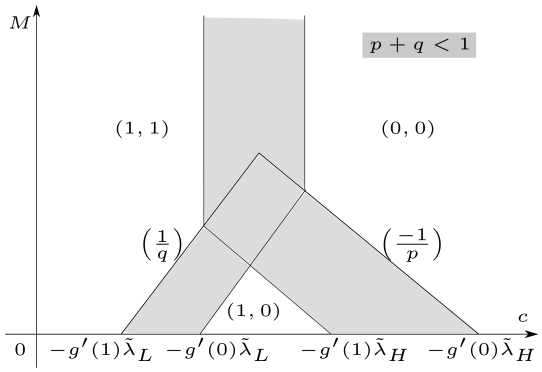

Fig. 3: Caching under the ABC policy for arrival processes satisfying Assumption 1 for strictly convex $g(\rho)$ for $p+q<1$ as a function of the values of the rent cost c and the fetch cost M. For each region, we show the values of $(\rho_H,\, \rho_L)$ as defined in Section IV. Here $\tilde{\lambda}_H$ and $\tilde{\lambda}_L$ are as defined in Theorem 1. The shaded region represents the values of M and c for which ABC uses partial caching.

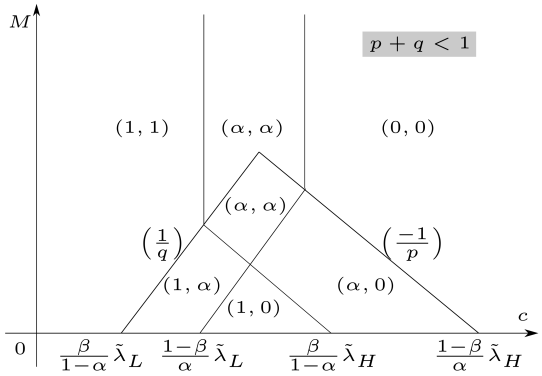

Fig. 4: Caching under the ABC policy for arrival processes satisfying Assumption 1 with $p+q<1$ for the case when $g(\rho)=1$ for $0 \leq \rho < \alpha$, $g(\rho)=\beta<1-\alpha$ for $\alpha \leq \rho < 1$ and $g(1)=0$ as a function of the values of the rent cost c and the fetch cost M. For each region, we show the values of $(\rho_H,\, \rho_L)$ as defined in Section IV. Here $\tilde{\lambda}_H$ and $\tilde{\lambda}_L$ are as defined in Theorem 1.

V. Simulation Results

In this section, we compare the performance of ABC with other policies via simulations for arrival processes satisfying Assumption 1. The other polices we consider include the optimal policy with partial caching (OPT w/ PC) which is the solution of the MDP w/ PartialCaching discussed in Section III-A, the optimal policy without partial caching (OPT w/o PC) which is the solution of the MDP w/o PartialCaching discussed in Section III-B, and the optimal static policies with and without partial caching (OPT Static w/ PC and OPT Static w/o PC) discussed in Sections III-C and III-D respectively.

The parameters we consider for the simulations are $p=0.2$, $q=0.1$, $\lambda_H=8$, $\lambda_L=1$, and $g(\rho)=(1-e^{(1-\rho)\mu})/(1-e^{\mu})$. Note that for $\mu \leq 0$, $g(\rho) \geq 1-\rho$ for all $\rho \in [0,1]$ and if $\mu>0$, $g(\rho)$ is strictly convex w.r.t ρ and $g(\rho)<1-\rho$ for all $\rho \in (0,1)$. Each data point is obtained by averaging over 10^4 time-slots and 100 iterations.

We begin by studying the impact of μ, M, and c on the average cost of the five policies mentioned above. In Fig.5, we fix $M=6$, $c=3$ and vary μ. Note that for values of $\mu \leq 0$, the cost for OPT w/ PC, OPT w/o PC, and ABC are identical. This is as expected from Theorems 3 and 4 since $g(\rho) \geq 1-\rho$ for all $\rho \in [0,1]$ in this case. On the other hand, for $\mu>0$, we note the the performance of ABC and OPT w/ PC is indistinguishable, while all other polices are sub-optimal for certain parameter values. We note that the performance gap between policies which use partial caching and those which do not increases with the value of μ. This is

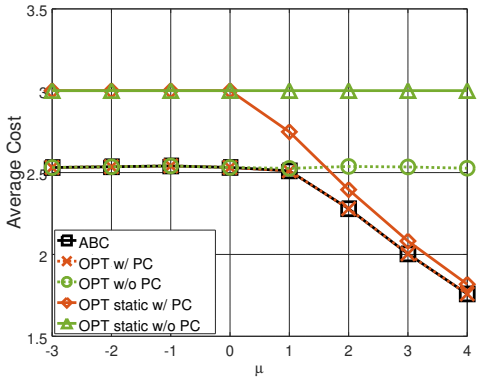

Fig. 5: Performance of various caching policies for $M = 6$ and $c = 3$. The performance of ABC and OPT w/ PC is indistinguishable and all other polices are sub-optimal. The performance gap between policies which uses partial caching and those which do not increases with the value of μ.

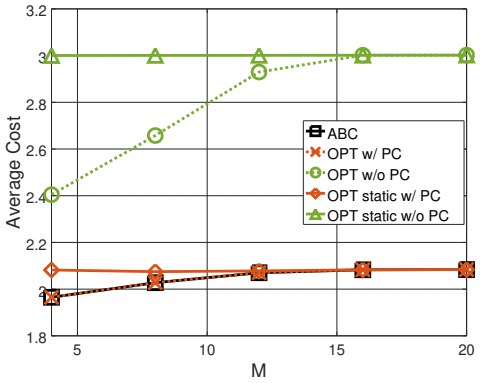

Fig. 7: Performance of various caching policies for $\mu = 3$ and $c = 3$. The performance of ABC and OPT w/ PC is indistinguishable and all other polices are sub-optimal for certain parameter values. With increase in the value of M, the performance of OPT Static w/o PC and OPT Static w/ PC approaches that of OPT w/o PC and OPT w/ PC respectively.

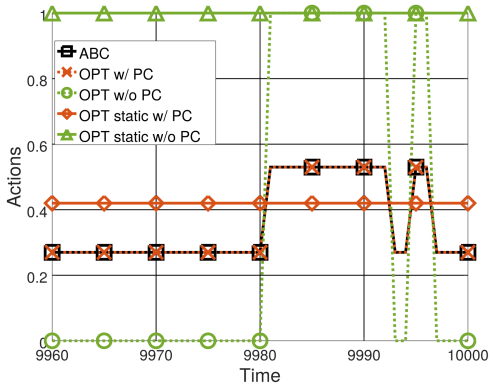

Fig. 6: Caching decisions under various policies as a function of time for $\mu = 3$ and $c = 3$, $M = 6$ for a sample path. Both ABC and OPT w/ PC use partial caching for the entire time-interval and their caching decisions are indistinguishable. Caching decisions under OPT w/o PC oscillate between 0 and 1. Caching decisions are time-invariant for the two static policies (OPT static w/ PC and OPT static w/o PC).

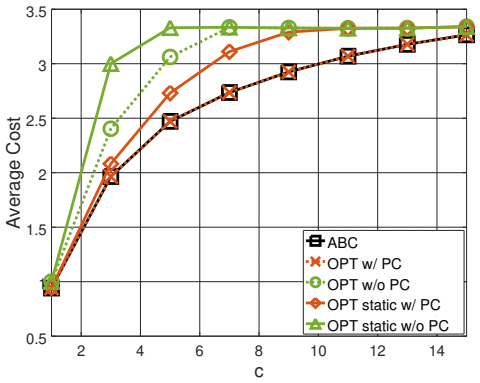

Fig. 8: Performance of various caching policies for $\mu = 3$ and $M = 6$. The performance of ABC and OPT w/ PC is indistinguishable and all other polices are sub-optimal for certain parameter values. The performance of all policies is close for very low/high values of c.

due to the fact that as μ increases, service cost for the same level of caching goes down and this is exploited by partial caching. In Fig.6, we fix $M = 6$, $c = 3$, and $\mu = 3$ and show the caching decisions made by the five policies for the last 40 time-slots of a particular run of the experiment. Again, both ABC and OPT w/ PC make the same partial caching decisions. Caching decisions for the two static policies (OPT static w/ PC and OPT static w/o PC) are time-invariant, while the caching decision under OPT w/o PC oscillates between 0 and 1. In Fig.7, we set $\mu = 3$, $c = 3$ and vary the value of M, while in Fig.8, we consider the setting where $\mu = 3$, $M = 6$ and vary the value of c. As expected, with increase in the value of M, the performance of OPT Static w/o PC and OPT Static w/ PC approaches that of OPT w/o PC and OPT w/ PC respectively. Further, the performance of all policies is close for very low/high values of c since the optimal decision under all policies is to always/never cache respectively.

VI. Conclusions

We consider the setting where a service can rent resources at the edge to cache its codes/libraries to serve incoming requests. Notably, we allow the service to be partially cached at the edge. The probability of the edge being equipped to serve an incoming request is an increasing function of the fraction of service cached. The focus of this work is on characterizing the benefits of partial service caching at the edge. The key insight from this work is that partial caching can be useful when *(i)* there is at least one intermediate caching level at which the

probability that an incoming request can be served at the edge is strictly more than the fraction of service cached, and *(ii)* the cost of renting edge storage resources falls within a range of values which depends on the statistics of the request arrival process. We use these insights to design near-optimal service caching policies.

VII. ACKNOWLEDGEMENTS

This work was supported in part by a SERB grant on "Content Caching and Delivery over Wireless Networks" and seed grants from IIT Bombay.

REFERENCES

[1] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future generation computer systems*, vol. 29, no. 1, pp. 84–106, 2013.

[2] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[3] 2019, aWS: `https://aws.amazon.com`.

[4] 2019, azure: `https://azure.microsoft.com`.

[5] S. Newman, *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.", 2015.

[6] C. Puliafito, E. Mingozzi, F. Longo, A. Puliafito, and O. Rana, "Fog computing for the internet of things: A survey," *ACM Trans. Internet Technol.*, vol. 19, no. 2, pp. 18:1–18:41, Apr. 2019. [Online]. Available: http://doi.acm.org/10.1145/3301443

[7] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[8] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[9] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 514–522.

[10] S. Bi, L. Huang, and Y.-J. A. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing system," *arXiv preprint arXiv:1906.00711*, 2019.

[11] L. Chen and J. Xu, "Collaborative service caching for edge computing in dense small cell networks," *arXiv preprint arXiv:1709.08662*, 2017.

[12] T. X. Tran, K. Chan, and D. Pompili, "COSTA: cost-aware service caching and task offloading assignment in mobile-edge computing," in *16th Annual IEEE International Conference on Sensing, Communication, and Networking, SECON 2019, Boston, MA, USA, June 10-13, 2019*, 2019, pp. 1–9. [Online]. Available: https://doi.org/10.1109/SAHCN.2019.8824854

[13] L. Yang, J. Cao, G. Liang, and X. Han, "Cost aware service placement and load dispatching in mobile cloud systems," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1440–1452, 2015.

[14] T. Zhao, I.-H. Hou, S. Wang, and K. Chan, "Red/led: An asymptotically optimal and scalable online algorithm for service caching at the edge," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 8, pp. 1857–1870, 2018.

[15] L. Narayana, S. Moharir, and N. Karamachandani, "Retrorenting: An online policy for service caching at the edge," *arXiv preprint arXiv:1912.11300*, 2019.

[16] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 207–215.

[17] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *2015 IFIP Networking Conference (IFIP Networking)*. IEEE, 2015, pp. 1–9.

[18] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," Mar. 2010, pp. 1478–1486.

[19] B. Tan and L. Massoulié, "Optimal content placement for peer-to-peer video-on-demand systems," vol. 21, no. 2, pp. 566–579, Apr. 2013. [Online]. Available: http://dx.doi.org/10.1109/TNET.2012.2208199

[20] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy, "On the scale and performance of cooperative web proxy caching," in *Proc. ACM SOSP*, 1999, pp. 16–31.

[21] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," Mar. 1999, pp. 126–134.

[22] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Communications of the ACM*, vol. 28, no. 2, pp. 202–208, 1985.

[23] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems journal*, vol. 5, no. 2, pp. 78–101, 1966.

[24] K. Ha, Y. Abe, T. Eiszler, Z. Chen, W. Hu, B. Amos, R. Upadhyaya, P. Pillai, and M. Satyanarayanan, "You can teach elephants to dance: agile vm handoff for edge computing," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 12.

[25] R. S. Prakash, N. Karamchandani, V. Kavitha, and S. Moharir, "Partial service caching at the edge," Tech. Rep., 2020. [Online]. Available: https://www.dropbox.com/s/ca9ipgkyq3otax7/main.pdf?dl=0

[26] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.