Caching Static and Transient Data

Rudrabhotla Sri Prakash
Indian Institute of Technology Bombay
sriprakash@ee.iitb.ac.in

Sharayu Moharir
Indian Institute of Technology Bombay
sharayum@ee.iitb.ac.in

ABSTRACT

Motivated by applications like Information Centric Networking for the Internet of Things, we study caching policies for the setting where the data being cached is heterogeneous in nature. This heterogeneity is in two aspects, namely, the lifetime of the data and the size of the data. We propose a caching policy which divides the cache into sub-caches, such that each sub-cache is reserved for data of a specific size and lifetime. Via analytical results and simulations, we show that our policy outperforms existing caching policies for heterogeneous data.

ACM Reference Format:
Rudrabhotla Sri Prakash and Sharayu Moharir. 2018. Caching Static and Transient Data. In *The 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18), October 29-November 2, 2018, New Delhi, India.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3241539.3267770

1 INTRODUCTION

This work is motivated by the communication paradigm of Information Centric Networking (ICN). In ICN, each piece of data is assigned a unique name and an interested receiver requests for a specific piece of data instead of requesting to communicate with the source of the data. To facilitate ICN, nodes in the communication network are equipped with storage capabilities so that data can be cached at intermediate nodes. When a request for a specific data is received, it can be served by any node in the network which has that data in its cache. Potential benefits of ICN over IP-based communication are lower delay, reduction in bandwidth consumption, and lowered energy consumption [1].

This work was supported by the Bharti Centre for Communications at IIT Bombay. Sharayu Moharir's research was funded in part by a seed grant from IIT Bombay, and an Indo-French grant "Machine learning for network analytics".

MobiCom '18, October 29-November 2, 2018, New Delhi, India

ACM ISBN 978-1-4503-5903-0/18/10.
https://doi.org/10.1145/3241539.3267770

The Internet of Things (IoT) is an application which benefits from ICN based communication as many IoT applications are information-centric in nature [1, 5]. In [5], it was observed that the performance of ICN for IoT depends heavily on the caching policies used by various nodes in the network. Since the data produced in such networks can be heterogeneous in nature, the focus of this work is on designing caching strategies for heterogeneous data. We capture this heterogeneity via two attributes.

The first attribute is the lifetime of the data, i.e., the duration for which this data is useful and can be used to serve requests. This duration could be finite (transient data) or infinite (static data). While the classical caching literature focuses on static data, more recent works have looked at the problem of caching transient data [4]. Our setting allows for the data to be cached to be of either type, thus generalizing the existing body of work on caching.

The second attribute is the size of the data being cached. While the problem of caching static data of different sizes has been studied in [2, 7], to the best of our knowledge, this work is the first attempt at designing universal caching policies that work for both transient and static data of different sizes.

In this work, we consider a system consisting of a single cache which stores and serves data of different sizes and lifetimes generated by multiple data producers. In an IoT system like a smart home, these producers are sensors which continuously sense potentially time-varying quantities. When the data of such a producer is requested and is available in the cache, the request can be served using the cached data only if its lifetime hasn't expired. In all other cases, fresh data is fetched from the producer to serve the request. The goal is to design caching policies which maximize the fraction of requests served using the cache, popularly known as the cache hit-rate.

1.1 Our Contributions

We focus on a class of policies called Cache-Split policies. Any policy in this class divides the available cache into sub-caches with each sub-cache reserved for data of a specific size and lifetime. We characterize the upper bound on the hit rate of each such sub-cache as a function of the size and lifetime of the data it caches (Theorem 3.1). Next we provide an approximation of the performance of the First In First Out (FIFO) policy for each sub-cache as a function of the size and lifetime of the data it caches (Proposition 3.2). Under

this approximation, we show that FIFO is an optimal caching policy for each sub-cache (Corollary 3.3). If each sub-cache implements FIFO, we characterize the optimal cache split across various data types (Theorem 3.4). In addition, via simulations, we show that our cache split followed by FIFO policy outperforms existing policies designed for static data of heterogeneous sizes.

2 SETTING

We consider a system of N producers, where each producer produces time varying data. Any request for the producers' data is forwarded through a router which has a local cache with limited memory. In addition, if needed, the router can fetch data directly from the producers to serve requests. Each producer (say Producer i) has three attributes associated with it, namely, its popularity ($\tilde{p}_i$), i.e., the fraction of arriving requests that are for the producer's data, lifetime ($\tilde{F}_i$), i.e., the duration for which the data of the producer can be used to serve incoming requests, and the size of the data ($\tilde{S}_i$) of the producer.

Multi-Class Producer Model: We consider a multi-class producer model, where the N producers are divided K ($\leq N$) classes. The number of producers in Class j is denoted by N_j. The data of producers belonging to a class have the same popularity, lifetime, and size. We denote the probability for a request, lifetime, and size of data of a producer in Class j by p_j, F_j, and S_j respectively, i.e., if Producer i belongs to Class j, then $\tilde{p}_i = p_j$, $\tilde{F}_i = F_j$, and $\tilde{S}_i = S_j$.

Request Model: Requests arrive at the router according to a Poisson process with parameter 1. For each incoming request, $\tilde{p}_i$ is the probability of the request being for Producer i.

Storage and Service Model: The router's cache has memory of C units. A request for Producer i's data is served via the cache if the lifetime of the cached data of Producer i has not expired, i.e., a request for Producer i arriving at time t can be served by the router only if the cached data of Producer i was fetched from the producer after time $t - \tilde{F}_i$. If either Producer i's cached data has expired or Producer i's data is not cached at all, the request is served by fetching data directly from the producer. On fetching data from the producer, the router can decide whether to cache it or not. Note that the router fetches data from a producer only when an incoming request cannot be served via its cache.

Goal: We call the event of a request being served by the router as a cache hit and its complement as a cache miss. Our goal is to design caching policies which maximize the cache hit-rate, i.e., the fraction of requests that result in cache hits.

3 MAIN RESULTS AND DISCUSSION

Our key contribution is a caching policy which divides the cache into multiple sub-caches with each sub-cache being reserved for a class of producers. We focus on this class of policies because of their simplicity and because they are known to perform well in the setting where all producers are static, but their data is of different sizes [3]. In Section 4, we compare the performance of our policy with other policies which don't split the cache into sub-caches.

3.1 Converse for each Sub-cache

Our first result provides an upper bound on the hit-rate for a sub-cache as a function of the popularity, lifetime and size of the data of the class of producers corresponding to the sub-cache. This result follows from Proposition 1 in [4].

Theorem 3.1. *Let the sub-cache reserved for Class w have memory C_w and let h_w^* be the maximum hit rate for the sub-cache, then we have that,*

$$h_w^* \leq \min\left\{p_w \frac{C_w}{S_w}, N_w p_w \frac{p_w F_w}{1 + p_w F_w}\right\}.$$

This result provides a benchmark against which the performance of specific caching policies can be compared.

3.2 First In First Out (FIFO) Policy

As the name suggests, First In First Out (FIFO) is a caching policy in which the contents are evicted from the cache in the order in which they are brought into the cache. Formally, on a cache miss, the requested data is fetched and stored in the cache and in order to make space for this newly fetched data, that cached content which was brought in least recently is evicted from the cache. The computational complexity for FIFO is $O(n)$ for search, and $O(1)$ for each insertion and eviction.

An approximation for the hit rate of the FIFO policy for static data of equal sizes is given in [6]. This approximation is obtained by mapping the evolution of the FIFO cache to that of an M/G/1/1 queue such that the cache hit-rate is equal to the blocking probability of the queue. For the next result we extend this technique to our setting where the data being cached is transient.

Proposition 3.2. *Let h_w^{FIFO} be the hit-rate of the FIFO policy for a sub-cache with size C_w reserved of data of producer class w, then we have that,*

$$h_w^{FIFO} \approx \min\left\{p_w \frac{C_w}{S_w}, N_w p_w \frac{p_w F_w}{1 + p_w F_w}\right\}.$$

We evaluate the accuracy of this approximation by comparing it with the simulated hit-rate of the FIFO policy in Figure 1. Here, we consider a class of 100 producers with popularity of 0.01 and data size of 1 unit. We vary the cache size and lifetime of the data. We note that the approximation is quite accurate for all cases considered.

The following result is a consequence of Theorem 3.1 and Proposition 3.2.

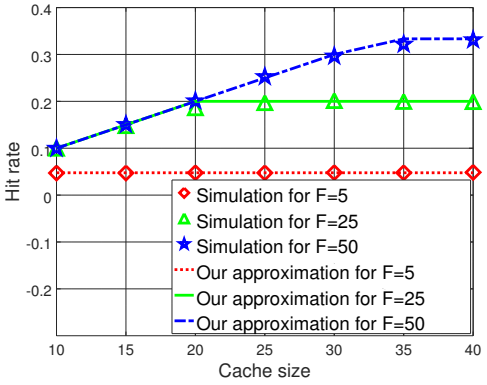

Figure 1: Comparison of the simulated performance of FIFO with the approximation derived in Proposition 3.2

Corollary 3.3. *Under the approximation given in Proposition 3.2, FIFO is an optimal policy for each sub-cache.*

3.3 Our Policy: Split Cache + FIFO

As mentioned above, our policy is to divide the cache into sub-caches with each sub-cache being reserved for a class of producers. Since FIFO is an optimal policy for each sub-cache, we use FIFO within each sub-cache.

We next discuss how we determine the split of the cache into sub-caches.

Theorem 3.4. *Let C_w be the memory allocated to Sub-cache w which is reserved for Class w producers. If each sub-cache uses the FIFO policy, the optimal split of the cache is the solution to the following optimization problem.*

$$\underset{C_1,C_2,\ldots,C_K}{\text{maximize}} \quad \sum_{w=1}^{K} \frac{p_w}{S_w} \min\left\{C_w, N_w S_w \frac{p_w F_w}{1+p_w F_w}\right\}$$

$$\text{subject to} \quad \sum_{w=1}^{K} C_w \le C.$$

Without loss of generality, let the producer classes be indexed in decreasing order of $\frac{p_w}{S_w}$, i.e., $\frac{p_1}{S_1} \ge \frac{p_2}{S_2} \ge \cdots \ge \frac{p_K}{S_K}$. Let $C_{w,sat} = N_w S_w \frac{p_w F_w}{1+p_w F_w}$ and j^* be such that $\sum_{w=1}^{j^*-1} C_{w,sat} < C$ and $\sum_{w=1}^{j^*} C_{w,sat} \ge C$. The solution to the problem is

$$C_w = \begin{cases} C_{w,sat} & w < j^* \\ C - \sum_{i=1}^{j^*-1} C_{i,sat} & w = j^* \\ 0 & otherwise. \end{cases}$$

Theorem 3.4 thus gives the optimal split of the cache into sub-caches when each sub-cache implements the FIFO caching policy. The computational complexity of finding the optimal cache split is $O(K \log K)$. Once this split is determined, on each request arrival, the request is forwarded to the relevant sub-cache, thus simplifying the search operation.

4 SIMULATION RESULTS

In this section we compare the performance of our policy with three other policies. The policies we compare against are the Adapt size policy proposed in [2], the LRU-S policy proposed in [7] and the classical LRU policy. The first two policies are designed for caching static data of different sizes.

The parameters used for the simulation are $K = 2$, $N_1 = 50$, $N_2 = 50$, $p_1 = 0.014$, $p_2 = 0.006$, $F_1 = 50$, $F_2 = \infty$, $S_1 = 1$, $S_2 = 8$ and we vary cache size from 10 to 35 units. We see that our policy outperforms the three other policies.

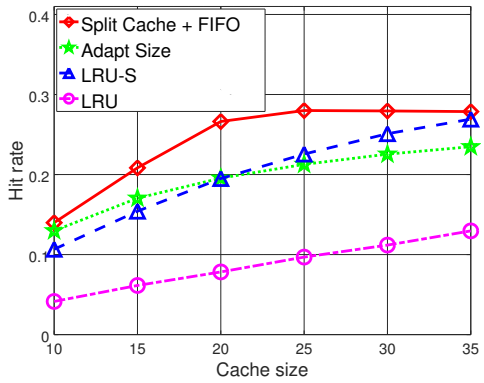

Figure 2: Comparison of the performance of the Cache Split + FIFO policy with other policies

REFERENCES

[1] Marica Amadeo, Claudia Campolo, Jose Quevedo, Daniel Corujo, Antonella Molinaro, Antonio Iera, Rui L Aguiar, and Athanasios V Vasilakos. 2016. Information-centric networking for the internet of things: challenges and opportunities. *IEEE Network* 30, 2 (2016), 92–100.

[2] Daniel S Berger, Ramesh K Sitaraman, and Mor Harchol-Balter. 2017. AdaptSize: Orchestrating the Hot Object Memory Cache in a Content Delivery Network.. In *NSDI*. 483–498.

[3] Asaf Cidon, Assaf Eisenman, Mohammad Alizadeh, and Sachin Katti. 2016. Cliffhanger: Scaling Performance Cliffs in Web Memory Caches.. In *NSDI*. 379–392.

[4] Santosh Fatale, Sri Prakash, and Sharayu Moharir. 2018. Caching Polices for Transient Data. (2018). https://www.dropbox.com/s/mw7tziaumbflxg8/main.pdf?dl=0

[5] Mohamed Ahmed M Hail, Marica Amadeo, Antonella Molinaro, and Stefan Fischer. 2015. On the performance of caching and forwarding in information-centric networking for the IoT. In *International Conference on Wired/Wireless Internet Communication*. Springer, 313–326.

[6] Valentina Martina, Michele Garetto, and Emilio Leonardi. 2014. A unified approach to the performance analysis of caching systems. In *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2040–2048.

[7] David Starobinski and David Tse. 2001. Probabilistic methods for web caching. *Performance evaluation* 46, 2-3 (2001), 125–137.