Prefetching and Caching for Minimizing Service Costs: Optimal and Approximation Strategies [*]

Guocong Quan Atilla Eryilmaz Jian Tan Ness Shroff
The Ohio State University
{quan.72, eryilmaz.2, tan.252, shroff.11}@osu.edu

ABSTRACT

In practice, prefetching data strategically has been used to improve caching performance. The idea is that data items can either be cached upon request (traditional approach) or prefetched into the cache before the requests actually occur. The caching and prefetching operations compete for the limited cache space, whose size is typically much smaller than the number of data items. A key challenge is to design an optimal prefetching and caching policy, assuming that the future requests can be predicted to a certain extent. This is a non-trivial challenge even under the idealized assumption that future requests are precisely known.

To investigate the above challenge, we develop a cost-based service model. The objective is to find the optimal offline prefetching and caching policy that minimizes the accumulated cost for a given request sequence. By casting it as a min-cost flow problem, we are able to find the optimal policy for a data trace of length N in expected time $O(N^{3/2})$ via flow-based algorithms. However, this requires the entire trace for each request and cannot be applied in real time. To address this issue, we analytically characterize the optimal policy by obtaining an optimal cache eviction mechanism. We derive conditions under which proactive prefetching is a better choice than passive caching. Based on these insights, we propose a lightweight approximation policy that only exploits predictions in the near future. The approximation policy can be applied in real time and processes the entire trace in expected time $O(N)$. We prove that the competitive ratio of the approximation policy is less than $\sqrt{2}$. Extensive simulations verify its near-optimal performance, for both heavy and light-tailed popularity distributions.

1. MOTIVATION

Proactively prefetching data items instead of passively caching them has been utilized in practice to accelerate data access, e.g., for content delivery networks. However, a fundamental trade-off between prefetching and caching has not been fully understood. Because cache space is limited, loading a prefetched data item into the cache typically triggers cache eviction, which may potentially introduce more cache misses for future requests. While there have been significant efforts undertaken to approximate data statistics and prefetch the most popular data, a fundamental question remains to be answered: *even with a perfect knowledge of future requests, how can we optimally prefetch data items beforehand instead of caching them upon requests?*

[*]This paper was supported by NSF grants CMMI-SMOR-1562065, CNS-ICN-WEN-1719371, CNS-NeTS-1514260, CNS-NeTS-1717045, CNS-NeTS-1717060, CNS-NeTS-2007231, CNS-SpecEES-1824337, ONR Grant N00014-19-1-2621, and DTRA grant HDTRA1-18-1-0050.

Copyright is held by author/owner(s).

2. PROBLEM FORMULATION

We introduce a cost-based service model. Consider a sequence of requests that is assumed to be known. If the requested data item is already in the cache, then the request can be served without paying a cost. However, if it is not cached, we have two options to serve the corresponding data request at different costs. The first option is to fetch the data from the backend after the request arrives by paying a fetching cost 1. We can decide whether to load the fetched item into the cache or not. The second option is to prefetch the data item before it is requested by paying a smaller prefetching cost c, $0 \leq c \leq 1$. Note that the prefetched data item has to be loaded into the cache. If the cache space is full, other items must be evicted before storing a new one.

The objective is to make the optimal prefetching and caching decisions to minimize the accumulated cost for the given request sequence. Notably, when the optimal policy fetches a missed request, it must not load the fetched data into the cache and trigger evictions, because otherwise prefetching will be a better choice. Solving the optimal offline policy is equivalent to answering the following two questions:

Q1: *If a request is not cached, should we prefetch it beforehand or fetch it upon the request?*

Q2: *If a data item is prefetched and the cache is full, which item should be evicted?*

3. KEY RESULTS

In this section, we summarize the key results of this work. Detailed theorems can be found in the full paper [1].

3.1 Optimal Policy via Min-Cost Flow

We leverage the underlying structure of prefetching and caching, and find the optimal policy by casting it as a min-cost flow problem. Specifically, for a given request sequence, we can construct a flow network where each request is represented by some nodes in a directed graph. The detailed construction steps are described in [1]. The flow network constructed for an example trace d_1, d_2, d_1, d_3 and a cache

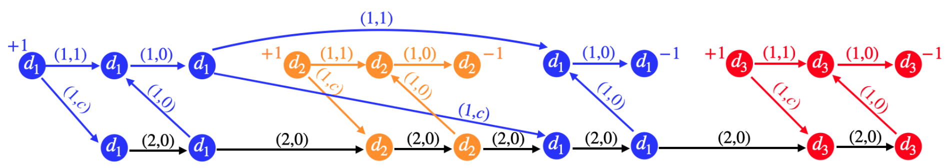

Figure 1: Flow network constructed for request sequence d_1, d_2, d_1, d_3 .

of size 2 is presented in Figure 1. We have the following key result to show the connection between the optimal offline policy and the min-cost flow on the constructed flow network.

Key Result 1. *For a given data trace, an optimal prefetching and caching policy can be developed from an integer min-cost flow on the constructed flow network.*

In the full paper [1], we present detailed steps on how to construct the optimal prefetching and caching policy from its corresponding min-cost flow solution. Based on Key Result 1, a flow-based optimal offline policy is proposed.
Flow-based optimal offline policy (π_{OPT}): Given a sequence of data requests, first construct a corresponding flow network and solve the integer min-cost flow. Then, reconstruct the optimal prefetching and caching policy from the min-cost flow by following the steps introduced in [1].

The flow-based optimal offline policy π_{OPT} is solvable in expected time $O(N^{3/2})$ for a request sequence of length N. Although π_{OPT} is offline optimal, the problem is not satisfactorily solved for the following reasons: 1) π_{OPT} does not provide analytical answers to the two questions proposed in Section 2, and therefore cannot fully reveal the underlying insights of the optimal decision; 2) π_{OPT} requires the knowledge about all future requests to find the optimal policy, and the optimal decision for a single request is unavailable unless the process for the entire data trace is completed.

These unanswered questions motivate us to 1) analytically characterize the properties of the optimal policy and explicitly answer the questions proposed in Section 2; 2) design a lightweight approximation policy that achieves near-optimal performance, requires only near-future information, and can be executed in real time.

3.2 Characteristics of The Optimal Policy

We analytically characterize the optimal policy by first answering question Q2. We show in Key Result 2 that an optimal eviction policy is to evict the data item whose next request is farthest in future.

Key Result 2. *There exists an optimal policy that evicts the farthest-in-future item for all prefetching operations.*

Assuming that the farthest-in-future eviction is adopted, we then answer question Q1 by deriving sufficient conditions under which prefetching is the optimal choice.

Key Result 3. *Assume that the upcoming request is not cached. Prefetching the upcoming request is the optimal choice, if any of the following two conditions is satisfied:*
C1: There is a request for a popular item in the near future, but that item is not cached currently;
C2: The prefetching cost c is sufficiently low.

Interestingly, even if the upcoming request is not popular, prefetching that item is still optimal, as long as C1 or C2 is satisfied. The precise mathematical expressions for these two conditions are provided in [1].

3.3 Lightweight Approximation Policy

Based on the characteristics of the optimal policy, we propose an approximation policy as follows.
Approximation Policy (π_A): Prefetch the missed request and evict the farthest-in-future item, if $c \leq \sqrt{2}/2$ or any of the conditions $C1$ and $C2$ introduced in Key Result 3 holds. Otherwise, fetch the missed item but do not cache it.

We provide performance bounds for π_A by deriving its competitive ratio r_A, which is defined as the maximum ratio of the cost achieved by π_A to the cost achieved by an optimal policy for some request sequence. We characterize the competitive ratio r_A in the following key result.

Key Result 4. *Given the prefetching cost c, we have*

$$r_A = \begin{cases} 1 & \text{for } c \in [0, 1/2], \\ 2c & \text{for } c \in (1/2, \sqrt{2}/2]. \end{cases}$$

For $c \in (\sqrt{2}/2, 1]$ and a cache size b, r_A can be bounded as $b/((b+1)c) \leq r_A \leq 1/c$.

We plot the upper bound for r_A in Figure 2. It can be observed that the proposed approximation policy achieves the optimal performance when $c \leq 0.5$. For $c > 0.5$, the competitive ratio is at most $\sqrt{2}$, which provides an effective bound for the worst-case performance of π_A. In [1], empirical experiments verify that π_A also achieves near-optimal average performance for both synthetic and real data traces.

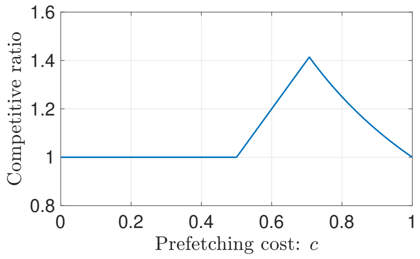

Figure 2: Upper bound for the competitive ratio r_A.

Moreover, the approximation policy π_A makes prefetching and caching decisions based only on the request information in the near future, regardless of the trace length N. As a result, π_A has an expected time complexity $O(1)$ to make decisions for a single request and $O(N)$ to process the entire trace. Unlike the flow-based policy π_{OPT} that requires all future requests to make decisions, π_A is lightweight and more practical since 1) it only requires near-future information; 2) it does not have to process the entire trace to make the decision for a single request, and therefore can be executed in real time.

4. REFERENCES

[1] G. Quan, A. Eryilmaz, J. Tan, and N. Shroff. Prefetching and caching for minimizing service costs: Optimal and approximation strategies. *Performance Evaluation*, 2020. `https://doi.org/10.1016/j.peva.2020.102149`.