2017 46th International Conference on Parallel Processing

Data Caching in Next Generation Mobile Cloud Services, Online vs. Off-line

Yang Wang[†], Shuibing He[‡], Xiaopeng Fan[†], Chengzhong Xu[†♮], Joseph Culberson[§], and Joseph Horton[§]
[†]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, China
[§]Faculty of Computer Science, University of New Brunswick, Canada
[‡]School of Computer, Wuhan University, China
[♮]Department of Electrical and Computer Engineering, Wayne State University, Detroit, USA
{yang.wang1,xp.fan,cz.xu}@siat.ac.cn, heshuibing@whu.edu.cn, jdh@unb.ca

***Abstract*—In this paper we consider the data caching problem in next generation data services in the cloud, which is characterized by using monetary cost and access trajectory information to control cache replacements, instead of exploiting capacity-oriented strategies as in traditional research. In particular, given a stream of requests to a shared data item with respect to a homogeneous cost model, we first propose a fast off-line algorithm using dynamic programming techniques. The proposed algorithm can generate optimal schedule within $O(mn)$ time-space complexity to cache, migrate as well as replicate the shared data item to serve an n-length request sequence with minimum cost in a fully connected m-node network, substantially improving the previous results. Additionally, we also study this problem in its online form, and present a 3-competitive online algorithm by leveraging a speculative caching idea. The algorithm can serve an online request in constant time, and is space efficient in $O(m)$ as well, rendering it to be more practical in reality. Our research complements the shortage of similar research in literature on this problem.**

***Index Terms*—Data caching; data service; mobile cloud computing; dynamic programming; speculative caching; competitive ratio**

I. Introduction

With mobile devices (e.g., smartphone and table PC) gaining popularity, data service, as a mainstream service to the mobile cloud users, has been inspiring great interest from both academia and industries. A shared common feature of such a service is the maximization of data availability [1], which is one of the pressing needs of the cloud service providers (CSPs) to offer high-quality mobile data services in shortest possible time. To achieve the maximization, data caching is an often-used technique due to its effectiveness in minimizing access latency and reducing network traffics.

However, when considering data caching as a service to facilitate mobile accesses in the cloud, there are two main challenges on caching replacement policies that would characterize the next generation mobile cloud services. First, the cache replacement policy is usually cost driven in the cloud, instead of being capacity-oriented as in classic network caching. This is because the cloud-based caching generally does not impose a limit on the cache capacity, instead, the size of cache as a resource could be virtually infinite provided that the users of the cache can afford it based on the cost model. Second, as apposed to classic network caching, whose design is typically to exploit the spatial and temporal localities in access sequences, the data caching in the cloud is usually required to facilitate the mobile accesses that often exhibit spatial-temporal trajectory patterns. This requirement is of importance to the data caching since big data studies reveal that more than 93% human behavior is predictable [2], including the accesses to data items in cloud services, which are highly predictable in both time and space.

The above challenges profoundly affect the design of cache policies on how to utilize the resources in a cost-effective way. Traditionally, the capacity-oriented replacements like *LRU* or *LFU* are typically designed to maximize the *hit ratio* of the cache by selecting a victim item wisely to evict from the cache so that a newly accessed item could be brought in. In contrast, for the data caching in the cloud, the goal is, instead, changed to minimize the cost of accessing the data items since the cache capacity is not an issue anymore, it is neither fixed nor bounded.

In this paper, we study the data caching problem by efficiently moving around a shared data item, with possible multiple copies, in a fully connected cloud network so that a sequence of requests to the item could be satisfied with minimum service cost. In the study, the request sequences could be online or off-line, signifying different application scenarios in reality. The off-line sequence is embedded with the spatial-temporal trajectory information of the requests, which could be secured in advance by mining the data service logs or exploiting some spatial-temporal trajectory model [3], while for online sequence, we assume nothing known about it in order to make competitive analysis, which could give a theoretical bound to the worst case of the algorithm.

For comparison purpose with the classic caching problem that has homogeneous miss penalty cost, we also assume that the cost model in the data caching is homogeneous, which means the transfer cost between any pair of servers in the network is identical and the caching cost per time unit at each individual server is also identical. The model is also practical in the sense that the provisioned cloud infrastructure for a particular data service is always organized as a subset of homogeneous resources, which in turn results in homogeneous computations and communications in the cloud [4]. To make our research goal more clear, a detailed comparison between

2332-5690/17 $31.00 © 2017 IEEE
DOI 10.1109/ICPP.2017.50

IEEE computer society

Authorized licensed use limited to: Zhejiang University. Downloaded on March 13,2023 at 01:45:54 UTC from IEEE Xplore. Restrictions apply.

TABLE I: Classic network caching vs. Cloud data caching.

	Classic Caching	Cloud Data Caching
Network	Fully Connected	Fully Connected
Cost Model	Transfer Cost	Caching&Transfer Costs
Operation	Page Fault	Caching, Transfer&Replication
Cache Size	Fixed Number k	Dynamic Number
Opt. Goal	Total Fault Cost	Total Service Cost
Locality	Spatial-Temporal	Spatial-Temporal Trajectory
Opt. Off-line	Belady's Alg. [5]	**O(mn) Fast Opt. Alg.**
Comp. Online	k-competitive	**3-competitive**

classic network caching and the cloud-based data caching is listed in Table I where fast optimal off-line and competitive online algorithms for the raised data caching problem are the focus of this paper.

First, we consider off-line algorithms with respect to a given request sequence characterized by its spatial-temporal trajectory information. To this end, we use dynamic programming technique to design a fast optimal algorithm that can cache, migrate and replicate the shared data item in a fully connected m-node network to serve an n-request sequence within $O(mn)$ time and space complexity. The result is $O(m \log m)$ times faster than the previous algorithms [4], [6] (**Contribution1**). Second, we conduct competitive analysis on the online version of this problem. By leveraging a concept of *speculative caching* idea, we propose a 3-competitive online algorithm, which not only guarantees the performance of this algorithm in the worst case, but is also efficient in both time and space. To the best of our knowledge, this is the first competitive analysis to show this new caching paradigm has a *constant* competitive ratio, a surprising result when comparing it with those in the *classic caching* problem (**Contribution2**).

We provably achieve these results by making several key observations on how the request sequence is served in both online and off-line cases, and thereby conducting a strict analysis on the schedules in both algorithms. Our algorithms are not only practical to the data caching problem, but also appear to be of theoretical significance to a natural new caching paradigm in the realm of online algorithms.

The remainder of this paper is organized as follows: Section II surveys some related work. The problem notation is introduced in Section III, and the off-line algorithm, together with its critical analysis, is presented in Section IV. We then conduct competitive analysis in Section V, and conclude the paper in the last section.

II. Related Work

The caching problem, due to its pervasiveness for performance optimization in distributed computing, has been endowed with extensive studies in literature. Chockler et al. [7] study the data caching problem in the cloud service, and propose *BLAZE*, a simple multi-tenant caching scheme to guarantee minimum QoS for each tenant. However, the scheme is still oriented to the cache resources with limited capacity, which is not necessary in the next generation data services. With *BLAZE* as a basis, Chockler et al. [8] present a new cloud-based caching service called *Simple Cache for Cloud* (SC2), which can optimize the global use of cache resources while simultaneously guaranteeing minimum service quality for all users according to their stated requirements. Although SC2 is distributed in network for shared uses among tenants as in our case, it, like *BLAZE*, still takes the maximization of overall *hit ratio* as a goal.

Unlike previous studies, Scouarnec et al. [9] investigate cache policies for cloud-based caching from a different angle that views cloud resources to be potentially infinite and only paid when used. To deal with this new context, they design and evaluate a new caching policy that minimizes the cost of a cloud-based system in online fashion. We adopt this point of view to design the optimal off-line and competitive online data caching algorithms in the cloud. Particularly, the off-line sequence is assumed to be available in advance when considering the trajectory information inherent in access patterns.

In the off-line setting, the work on optimal caching was first conducted by Belady in 1966 [5], since then the follow-up studies on this problem are few and far between, each being in different contexts with different goals [10]–[12]. However, none of them is applicable to our caching model. A highly related one is Veeravalli's work [6] which deals with the network caching problem with respect to sharing a data item in a set of fully networked servers. With a homogeneous cost model, he obtains an optimal algorithm using dynamic programming technique within $O(nm^2 \log m)$ time, which can automatically generate multiple copies to minimize the total service cost of request sequence. Compared to this algorithm, our off-line algorithm can reach the same goal in $O(mn)$ time and space complexity. Veeravalli's work is later extended by Wang *et al.* [4] to the context of clouds with some practical constraints so that a balance between the caching costs and the transfer costs of multiple shared data items can be optimally struck.

In the online setting, there are substantial researches in literature. However, most of them are capacity oriented [13]–[15], and among which a few leverage competitive analysis to evaluate their performance [10], [16]. Charikar et al. [17] propose the *dynamic servers problem* (DSP), which can be viewed as a generalization of the data caching problem in our case in terms of its request pattern and metric space. For the online DSP in arbitrary metric spaces, they present an $O(\min\{\log n, \log \rho\})$-competitive algorithm where n is the number of requests and ρ, the (normalized) diameter of the metric space. This is an improvement to the previous result of Halperin et al. in [18] where the DSP has an $O(\log n)$ competitive algorithm in a special case of paths.

III. Problem Notation on Data Caching

In this section, we describe the problem in details under a homogeneous cost model. We first define some useful concepts and then give the standard form of the solution to the problem. Some used symbols are listed in Tab. II for easy reference.

Suppose in a cloud environment, the server set is $\mathcal{S} = \{s^1, \ldots, s^m\}$ that are fully connected by a network, and a

Authorized licensed use limited to: Zhejiang University. Downloaded on March 13,2023 at 01:45:54 UTC from IEEE Xplore. Restrictions apply.

TABLE II: Frequently used notation

Symbol	Meaning
r_{-j}	$r_{-j} = (s^j, -\infty), 1 \leq j \leq n$
$\delta t_{i,j}$	$\delta t_{i,j} = t_j - t_i$, time diff. btw r_i and r_j
$p(i)$	the previous req. before t_i (same server)
$p'(i)$	the most recent event before t_i (same server)
σ_i	$\sigma_i = t_i - t_{p(i)}$
$Tr(s_i, s_j, x)$	data transfer from s_i to s_j at t_x
$H(s, x, y)$	data is held in cache on server s from t_x to t_y
μ	uniform caching cost per time unit
λ	uniform transfer cost between servers
ω_j^i	the ith speculative caching cost on s^j
Ω_j	the set of SC costs on s^j
β	uploading cost to servers
$\Psi^*(n)$	optimal schedule for up to r_n
$\Pi(\Psi(i))$	the cost of schedule $\Psi(i)$
$\Psi^{(-1)}(i)$	sub-schedule of $\Psi(i)$ for up to r_{i-1}
b_i	$b_i = \min\{\lambda, \mu\sigma_i\}, 1 \leq i \leq n$.
B_i	$B_i = \sum_{j=1}^{i} b_j$
$\Psi'(i)$	conditional opt. sched. with $H(s_i, t_{p(i)}, t_i)$ as the final cache H
κ	pivot index

shared data item is initially located at a certain server, say s^1. The request for the data item is generated online at each server, which could be made by users outside the cloud. The request vector is denoted as $\mathcal{R} = \langle r_1, \ldots, r_n \rangle$, where each $r_i = (s_i, t_i)$, with $t_i < t_{i+1}$ and $s_i \in \mathcal{S}$, represents that r_i is made from s_i at t_i. Note that the use of superscripts for the server indexes (e.g., s^i) should be distinguished from the reference labels (e.g., s_i). For example, the ith request r_i's server s_i could be s^j. To satisfy the request sequence, the shared item should be moved around between the servers, replicated at or cached and then deleted at certain servers to satisfy each request on time only if the total cost is minimum (cost model is discussed later).

To simplify boundary conditions, we define $r_0 = (s^1, 0)$ and $r_{-j} = (s^j, -\infty), 1 \leq j \leq n$. Note that the requests at $-\infty$ will never be included in a solution, and are only defined to make notation on the intervals on a server easier. We assume at least one request for each server (i.e. we ignore those servers without requests).

For $i < j$, we define $\delta t_{i,j} = t_j - t_i$, which is the time difference between requests r_i and r_j. For $r_i = (s_i, t_i), 1 \leq i < n$, we define the previous request on the same server by $p(i) = \arg\max_{j<i}\{s_j = s_i\}$. Then, we can define the *server interval on request* r_i as $\sigma_i = t_i - t_{p(i)}$. Moreover, we can define a data item *transfer* $Tr(s_i, s_j, x)$ from server s_i to s_j at time x, and say the data item is cached, or *held in cache* on server s from time x to y using the notation of $H(s, x, y)$. In particular, if there is no confusion incurred, we would also use $H(s, x, y)$ or H to refer to the cached copy or its cost in corresponding server.

The cost model for this process is homogeneous in the sense that the cost of one unit of time caching on each server is the same across all the servers, denoted by μ, and the transfer cost from any server to any other server, denoted by λ, is also identical no matter which servers are being used. As a result, the transfer cost between servers s^j and s^k is λ, $\forall j \neq k$ at any time x, and the caching cost from time x to y for any server is $\mu(y - x)$. This model is corresponding to the traditional caching model where the penalty of cache miss is always assumed a constant. On the other hand, the homogeneous cost model is also available in reality [4].

The problem is to find the set of cache intervals and transfers for the data item so that

1) at least one server is caching the data item at any time $t, t_0 \leq t \leq t_n$.
2) The data item is available for r_j on s_j at time $t_j, 1 \leq j \leq n$, the time instance when r_j is made (discuss later).
3) The total transfer and caching costs are minimized.

To this end, multiple copies of the data item could be automatically generated during the service process. Except the first one each of the other copies is caused by a transfer, and eventually deleted when it is no longer used. Therefore, without loss of accuracy, we assume that the replication cost and the deletion cost are free since these costs are always constants and can be included in the transfer cost.

Fig. 1 shows an example to illustrate the problem notation where three servers are fully connected by a network. A shared data item is initially located at s^1, it is migrated, replicated or cached among the servers to satisfy the request sequence in time order with minimized total cost as a goal. Note that the red color indicates that the corresponding cached data item is deleted after being accessed. As such, the next request at the same server should be served by a transfer (e.g., $r_7@s_3$).

Definition 1 (Schedule)**.** We say that a *schedule* $\Psi(i)$ is any *minimal* set of caches and transfers satisfying 1) and 2) for online requests $r_0, \ldots, r_i$. A schedule is *optimal* if it also satisfies 3). However, it is not achievable for online algorithms.

There could be many feasible (off-line) schedules for $r_0, \ldots, r_i$, we use $\Gamma(i)$ to represent the space of the feasible schedules for up to r_i, each $\Psi(i)$ having a cost, denoted by $\Pi(\Psi(i))$. The goal is to find an optimal schedule $\Psi^*(n)$:

$$\Psi^*(n) = \arg\min_{\Psi(n)\in\Gamma(n)} \{\Pi(\Psi(n))\}$$

We can view a schedule in a *space-time diagram* as shown in [19], where the edges are caching intervals or transfers, and the vertices are requests or end points of transfers. More formally, we have

Definition 2 (Space-Time Graph)**.** We define a *space-time graph* as a weighted directed graph $G = (V, E, W)$ where $V = \{v_{ji} : 0 \leq j \leq m, 0 \leq i \leq n\}$. Vertex v_{ji} corresponds to time t_i on server s^j when $1 \leq j \leq m$, and $v_{0,i}$ represents the external storage at t_i. The edge set E consists of three subsets:

1) a set of *cache edge* $C = \{(v_{j,i-1}, v_{ji}) : 0 \leq i \leq n, 1 \leq j \leq m\}$,

Authorized licensed use limited to: Zhejiang University. Downloaded on March 13,2023 at 01:45:54 UTC from IEEE Xplore. Restrictions apply.

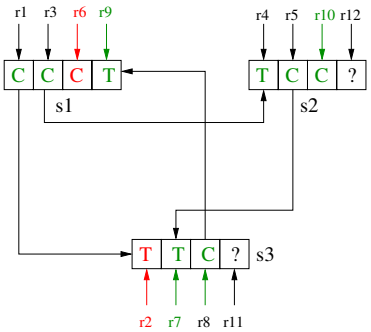


Fig. 1: Three servers are fully connected by a network (only caches are depicted as squares), and a shared data item is initially located at s^1, which could be migrated, replicated, or cached to satisfy request sequence $\mathcal{R} = \langle r_1, \ldots, r_{12} \rangle$ in time order. The characters in each square specify how the data item is created in the cache, say "C" is by caching, and "T" is by transferring, each being charged by its own rate. After the requests are served, the corresponding cached items could be either kept in the cache for future demands (in green color) or deleted for cost saving (in red color).

2) a set of *transfer edge* $T = \{(v_{ki}, v_{ji}), (v_{ji}, v_{ki}) : j \neq k, \text{and } (s_k, t_i) \in \mathcal{R}\}$, and

The edge weights W are defined as $W(e) = \lambda$ for edges $e \in T$, and $W(e) = \mu(t_{i-1} - t_i)$ for edges $(v_{j,i-1}, v_{ji}) \in C$.

Notice that a request r_i in the instance will correspond to vertex $v_{s_i,i}$ in the graph. For convenience we will often refer to request vertex r_i. All other vertices we call *intermediate vertices*. The set of vertices v_{*i} induce a subgraph that is a biconnected star centred on the request vertex r_i. According to the graph, the transfer time is negligible, we thus can satisfy r_i by a transfer at time t_i. This assumption can be validated by tweaking the graph as shown in [4], and is thus often adopted in previous studies [4], [6], [17].

Since a schedule is minimal, it implies that it is tree-like. If there is more than one path from s^1 to r_i then at the last juncture of paths, at least one of the entries must be either a transfer or an upload which can be deleted without loss of service since such a path cannot be minimal. Also, a schedule will contain no dead-end caches, that is cache on a server beyond the last request or transfer time from that server.

The data staging problem in its general form is a variant of the Rectilinear Steiner Arborescence problem [20]. As such, it is believed to be NP-complete [6]. However, its formal proof still remains open. Fortunately, in some realistic settings as in our case, when the cost model is homogeneous, we can expect optimal solution to this problem. The following observation indicates that we only need to consider the schedules where the transfers end on requests.

Observation 1 (Standard Form)**.** *For any instance, there exists at least one optimal schedule in which every transfer occurs at a request time t_i with its output ends on server s_i.*

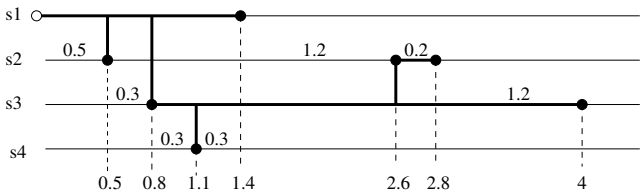


Fig. 2: An example of a standard schedule (shown in bold lines) for an off-line request sequence (solid dots along timeline). The vertical (horizontal) lines represent transfers (caching) that end on requests.

This observation can be directly obtained from Theorem 1 in Veeravalli [6]. Fig. 2 shows an optimal schedule in the standard form in a space-time diagram where all transfers end up with the requests at different servers. In the figure, the caching cost and the transfer cost are $1.4\mu + 0.2\mu + 1.6\mu = 3.2$ and $4\lambda = 4.0$, respectively, when $\mu = 1$ and $\lambda = 1$.

The following observation implies that the optimal schedule is a directed tree rooted at s^1 (see Fig. 2).

Observation 2. *In any optimal schedule, each request r_i will be served: 1) by the cache (i.e., the cached copy) on s_i, or 2) by a single transfer ending at r_i (see Fig. 2 again).*

Given the standard form of schedules, we can define sub-schedule as follows:

Definition 3 (Sub-schedule)**.** The *primary sub-schedule*, hereafter referred to as the *sub-schedule*, $\Psi^{(-1)}(i)$ of $\Psi(i)$ is a schedule for r_{i-1} that consists of the set of caching intervals and transfers from $\Psi(i)$ required to satisfy all requests $r_0, \ldots, r_{i-1}$.

Note that even if $\Psi(i)$ is optimal, the sub-schedule $\Psi^{(-1)}(i)$ may not be an optimal schedule for $r_0 \ldots r_{i-1}$. Moreover, $\Psi^{(-1)}(i)$ may not be unique.

Since Observation 1 applies to every transfer in an optimal schedule $\Psi^*(i)$, it will also hold in the sub-schedule $\Psi^{(-1)}(i)$. This justifies that from now on we only need to consider those schedules in the standard form indicated by Observation 1, which dramatically simplifies the feasible schedule space as it is not necessary to consider the proactive data transfer to some vantage servers for the subsequent requests. Note that the last caching interval in $\Psi^*(i)$ may be truncated to the last transfer point or request prior to i on that server in $\Psi^{(-1)}(i)$ (e.g., $r_7@s_3$ in Fig. 2).

IV. An Optimal $O(mn)$ Off-line Algorithm

Given the problem notation, in this section, we conduct strict analysis on the problem and give our optimal off-line algorithm with $O(mn)$ time and space complexities.

To this end, we first obtain a lower bound on the marginal costs to satisfy each individual request by combining above observations, which is defined by

Definition 4 (Marginal Cost Bound)**.** The marginal cost bound of request r_i is $b_i = \min\{\lambda, \mu\sigma_i\}, 1 \leq i \leq n$.

Authorized licensed use limited to: Zhejiang University. Downloaded on March 13,2023 at 01:45:54 UTC from IEEE Xplore. Restrictions apply.

Based on the marginal cost bound, we can further have a lower bound on the total costs to satisfy a request sequence, which is defined by

Definition 5 (Running Bound)**.** The running bound of the marginal costs up to request i is $B_i = \sum_{j=1}^{i} b_j$.

Definition 6 (Optimal Cost $C(i)$)**.** We define $C(i), 1 \leq i \leq n$ to be the cost of the optimal schedule $\Psi^*(i)$. Recall that r_0 is a boundary variable we created, with $C(0) = 0$, so $C(i)$ is defined for $0 \leq i \leq n$.

Clearly, given the definitions of B_i and $C(i)$ for $1 \leq i \leq n$, we can have the following observation as B_i is only a lower bound of the optimal cost, that is $B_i \leq C(i), 1 \leq i \leq n$.

Our goal is to create a recurrence for $C(i)$ that we can solve dynamically. To this end, we first prove the following lemma,

Lemma 1. *If $\Psi^*(i)$ is an optimal schedule in which the last operation is a transfer $Tr(s_j, s_i, t_i)$ then $\Psi^{(-1)}(i)$ is an optimal schedule up to request r_{i-1}(i.e., $\Psi^{(-1)}(i) = \Psi^*(i-1)$).*

Proof: If the optimal $\Psi^*(i)$ ends in a transfer, it must cache the unique data copy on the interval $[t_{i-1}, t_i]$ on some server other than s_i. All transfers are of equal cost, so one optimal extension to the sub-schedule is to cache $H(s_{i-1}, t_{i-1}, t_i)$ and then transfer $Tr(s_{i-1}, s_i, t_i)$. If $\Psi^{(-1)}(i)$ were not optimal this would lead to a contradiction. ■

Given Observation 2, we only need to consider two cases that r_i is served, either by the cache on s_i or by the immediate transfer from r_{i-1}. The next lemma covers the easy case of our recurrence.

Lemma 2. *If the conditions of Lemma 1 hold, then*

$$C(i) = C(i-1) + \mu\delta t_{i-1,i} + \lambda. \tag{1}$$

Proof: This is just the sum of the optimal cost up to r_{i-1} plus the cost of caching and transfer. ■

Now, we consider the non-trivial case that the optimal schedule $\Psi^*(i)$ uses the cached data copy on server s_i to satisfy r_i. Unlike the transfer case where the last data transfer $Tr(s_j, s_i, t_i)$ does not impact the optimality of $\Psi^{(-1)}(i)$, in this case, the last $H(s_i, t_{p(i)}, t_i)$ may impact all the requests made in $[t_{p(i)}, t_{i-1}]$ since a cache is extended from $t_{p(i)}$ to t_i which allows the requests to re-adjust the sources of the data item (e.g., a cache may be changed to transfer for cost reduction). As a consequence, no request $r_j, 0 < j < i$ is guaranteed to be optimal anymore for the sub-schedule of $\Psi^*(i)$ with respect to the interval $[t_1, t_{i-1}]$. To deal with this, we define an auxiliary recurrence that helps compute $C(i)$ in this case.

Definition 7 (Semi-Optimal Cost $D(i)$)**.** We define $D(i)$ to be the semi-optimal cost of a schedule $\Psi(i)$ in standard form (see Observation 1) under the condition that r_i is served by cache on server s_i. Clearly, $C(i) \leq D(i)$.

To see the efficacy of this definition we note the following.

Observation 3. *In a $\Psi(i)$, if s_i has a cached copy at t_i, then the cache extends from time $t_{p(i)}$, that is, the cache on s_i is $H(s_i, t_{p(i)}, t_i)$.*

Observation 3 follows from the standard form requirement that no transfer ends at point that is not a request.

We can now complete the recurrence for $C(i)$ in terms of the not yet completed $D(i)$ since the optimal will either use cache or transfer (Observation 2).

$$C(i) = \begin{cases} 0 & i = 0 \\ \min\{D(i),\ C(i-1) + \mu\delta t_{i-1,i} + \lambda\} & \\ & 1 \leq i \leq n \end{cases} \tag{2}$$

Recall that we added boundary points to our problem definition, and we extend here by defining base cases $D(i) = +\infty, i < 1$. These together with the infinite negative starting values of these intervals prevent us from using $D(i)$ as the cost of the first request on any server. That is, the first request on any server except s^1 will have to be served by either a transfer or an upload. Recall that the first request on s^1 is r_0 with cost 0.

The basic idea of auxiliary recurrence is to establish the relationships between $D(i)$ and certain $C(\kappa)$ that has been available whereby the most recent $C(i)$ can be computed. To this end, we first denote the conditional optimal schedule with $H(s_i, t_{p(i)}, t_i)$ as the final cache H by $\Psi'(i)$, and then look for the last cache that could cover $r_{p(i)}$ with respect to $\Psi'(i)$ (again, such a cache has potentials to satisfy $r_{p(i)}$ by a transfer). Specifically, we have the following definitions:

Definition 8 (Cover Index Set)**.** We define *cover index set* $\pi(i)$ as those potential caches that could satisfy $r_{p(i)}$ by a transfer.

$$\pi(i) = \{k | p(k) < p(i) \leq k < i\}$$

Definition 9 (Pivot Index κ)**.** The *pivot index* κ is defined by either 0 or the maximum in $\pi(i)$ (i.e., the most recent cache in $\pi(i)$ relative to r_i), depending on whether or not $\pi(i) = \varnothing$, i.e.,

$$\kappa = \begin{cases} 0 & \pi(i) = \varnothing \\ \max\{\pi(i)\} & \text{Otherwise} \end{cases}$$

The case of $\kappa \neq 0$ is important as it signifies the last request in $[t_{p(i)}, t_{i-1}]$ that is served by the cache $H(s_\kappa, t_{p(\kappa)}, t_\kappa)$ other than the transfer from $H(s_i, t_{p(i)}, t_i)$ in $\Psi'(i)$, which forms the basis for the $D(i)$ recurrence. There could be two distinguished cases: 1) $\kappa \leq p(i)$, and 2) $\kappa > p(i)$. The first is the boundary case, which is trivial.

Lemma 3. *For the pivot index κ as defined in definition 9, if $\kappa \leq p(i)$ then the optimal restricted cost*

$$D(i) = C(p(i)) + \mu\sigma_i + B_{i-1} - B_{p(i)}. \tag{3}$$

Proof: When $\kappa \leq p(i)$, there is no cache spanning across $t_{p(i)}$. As a result, all requests $r_j, p(i) < j < i$ have to be satisfied by the cache H using transfers or short cache intervals by a cost of $B_{i-1} - B_{p(i)}$. On the other hand, all requests up to $t_{p(i)}$ must still be satisfied, and $C(p(i))$ is a lower bound on any schedule satisfying this. As such, given the cache cost

Authorized licensed use limited to: Zhejiang University. Downloaded on March 13,2023 at 01:45:54 UTC from IEEE Xplore. Restrictions apply.

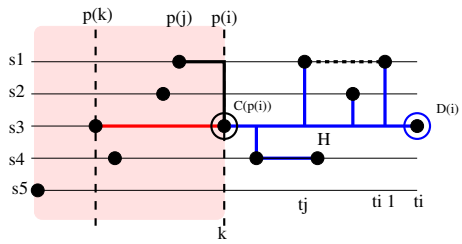


Fig. 3: An example of the trivial case when $\kappa \leq p(i)$. $H(s_i, t_{p(i)}, t_i)$ as the final cache H impacts how the requests in $[t_{p(i)}, t_{i-1}]$ are served (shown in bold blue line). The shadow area is optimal.

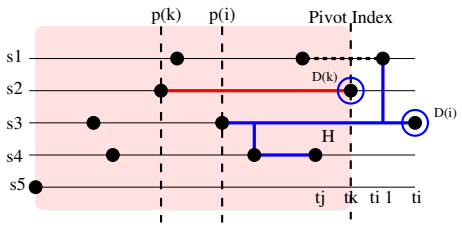


Fig. 4: An example of the non-trivial case when there are some caches spanning across $t_{p(i)}$ ($\pi(i) \neq \varnothing$). The shadow area is conditionally optimal.

of $\mu\sigma_i$, we have the lemma. Since this difference is a lower bound on serving these requests, the total is optimal under the stated conditions of the lemma. ■

An illustrative example of the trivial case is shown in Fig. 3 where $\kappa \leq p(i)$. According to Observation 2, $r_{p(i)}$ could be served by a cache on s_i (in bold red line) or a single transfer ending at $r_{p(i)}$ (in bold black line), say the cache on s_1 in the example. In both cases, the optimality of the cost to serve the request sequence up to $t_{p(i)}$ is independent of the subsequent requests until r_i since the service before (including) $t_{p(i)}$ has no impact on the later requests.

Now let's examine the non-trivial case that $\kappa > p(i)$. In this case, both $H(s_\kappa, t_{p(\kappa)}, t_\kappa)$ and $H(s_i, t_{p(i)}, t_i)$ are in the final schedule $\Psi'(i)$ as shown in Fig. 4 as an example, then we have

Lemma 4. *For κ as defined in definition 9, if $\kappa \neq 0$ then the optimal restricted cost,*

$$D(i) = D(\kappa) + \mu\sigma_i + B_{i-1} - B_\kappa. \tag{4}$$

Proof: When $\kappa > p(i)$, $H(s_\kappa, t_{p(\kappa)}, t_\kappa)$ in $\Psi'(i)$ must span across $t_{p(i)}$. As such, all requests up to t_κ are satisfied. Since $D(\kappa)$ is a lower bound on the cost of the schedule up to t_κ, $B_{i-1} - B_\kappa$ is a lower bound on adding the requests $r_h, \kappa < h < i$, and $\mu\sigma_i$ must be added to cover the cache interval $[t_{p(i)}, t_i]$, we see that $D(\kappa) + \mu\sigma_i + B_{i-1} - B_\kappa \leq \Pi(\Psi'(i)) = D(i)$.

If we start with a restricted optimal schedule to r_κ with cost $D(\kappa)$, then we can similarly construct a restricted schedule to r_i with cost $D(\kappa) + \mu\sigma_i + B_{i-1} - B_\kappa \geq \Pi(\Psi'(i)) = D(i)$, and thus the lemma follows. ■

By combining these lemmas, we enumerate all the request indexes on the interval $[t_{p(i)}, t_{i-1}]$ to derive $D(()i)$ recurrence as follows:

$$D(i) = \begin{cases} +\infty & -m \leq i \leq 0 \\ \min \begin{cases} C(p(i)) + \mu\sigma_i + B_{i-1} - B_{p(i)} \\ \min\limits_{\kappa \in \pi(i)} \{D(\kappa) + \mu\sigma_i + B_{i-1} - B_\kappa\} \end{cases} \end{cases} \tag{5}$$

Theorem 1. *With a homogeneous cost model, Recurrences (2) and (5) can correctly compute the minimum cost of the data caching problem.*

Proof: The optimality of the algorithm can be directly derived by combining Lemma 2 to Lemma 4. In particular, we examine all the caches in $\pi(i)$ in (5) to ensure the semi-optimal cost of $D(i)$ since we do not know the pivot index in $D(i)$ in advance. ■

Recurrences (2) and (5) define a recurrence system that allows us to compute the optimal cost. Using a sweep algorithm, we can compute this value by incrementally indexing through the requests from 1 to n, storing $C(i)$ and $D(i)$ for each request. A straightforward implementation should run in $O(n^2)$ time, which is dominated by the needs to check at most $O(n)$ previous values in the computation of $D(i)$ as indicated in Recurrence (5).

However, a closer look at Recurrence (5) indicates that at each i we need check at most one interval on each server (since $|\pi(i)| \leq m$, we do not need to compute $\pi(i)$ for $1 \leq i \leq n$ in our algorithm), provided we can efficiently find the interval on each server containing time $t_{p(i)}$. Based on this observation, we can have the following result:

Theorem 2. *The time and space complexity of the proposed algorithm is $O(mn)$.*

Proof: We create the following structures in a pre-scan of the requests. For each server, $s^j, 1 \leq j \leq m$ we create a doubly linked list Q_j which is initialized by the dummy boundary requests, and a matrix $A[n, m]$ of pointers. As $r_i, 1 \leq i \leq n$ is considered, it is added to the list Q_{s_i}, and $A_{i,j}$ is assigned to the current last element of Q_j for each $1 \leq j \leq m$. Then, for each request node $r_i = (s_i, t_i)$ in Q_j, a pointer is set up for each other server s_k that points to its most recent request node $r_k = (s_k, t_k)$, which could be obtained from $A[i, k], k \neq j$. Given hat each node in $Q_j, 1 \leq j \leq m$ has $O(m)$ space, the total data space in pre-scan thus takes $O(mn)$ time and space.

During the next pass over the requests to compute the recurrences, these pointers can be used to precisely identify each of the intervals required by Recurrence (5) in $O(m)$ time per request. Thus this pass also takes $O(mn)$ time. ■

Fig. 5 is an example to how the data structures are organized in efficient implementation of the proposed algorithm for the data staging problem in Fig. 2. During the computation of the recurrences, the algorithm follows the pointer of recent request r_i in $A[i, j]$ (e.g., $A[7, 3]$) to find the current last element of Q_j

Authorized licensed use limited to: Zhejiang University. Downloaded on March 13,2023 at 01:45:54 UTC from IEEE Xplore. Restrictions apply.

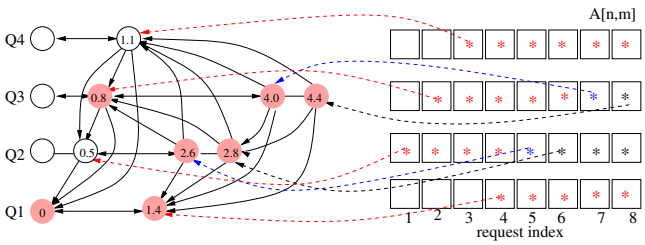
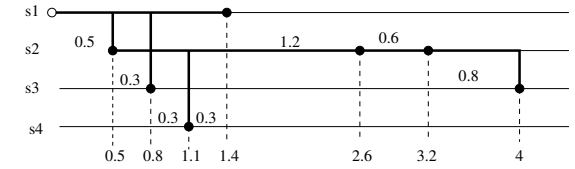


Fig. 5: An example to show how the proposed algorithm efficiently addresses the data caching problem in Fig. 2 where the computed cache intervals on each server are marked, and each pointer, represented by "*" in different colors, is kept up to t_8 (updated when a new request made on that server is processed).



i	0	1	2	3	4	5	6	7
b_i	0	1	1	1	1	1	0.6	1
B_i	0	1	2	3	4	5	5.6	6.6
$C(i)$	0	1.5	2.8	4.1	4.4	6.5	7.1	8.9
$D(i)$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	4.4	6.5	7.1	9.2

Fig. 6: An optimal schedule for an off-line request sequence (solid dots) is shown in bold lines, and the Marginal Cost Bounds (b_i), Running Bounds (B_i), as well as costs $C(i)$ and $D(i)$ are also presented in the table at bottom.

(e.g., request node 4.0) and then go back along the backward link to get its previous request node which records $t_{p(i)}$ (e.g., 0.8). Then, by following the $m = 1$ pointers, each for one server, the required interval on each server by Recurrence (5) can be identified in $O(1)$ (e.g., $\{[0, 1.4], [0.5, 2.6], \varnothing, \varnothing\}$ in our example).

For illustration purpose, we present a running example of the algorithm for an off-line demand sequence shown in the space-time diagram Fig. 6 where $m = 4$, $n = 8$ and each time instance for the requests is also marked. The data item is initially located at s^1 given $\lambda = 1$ and $\mu = 1$.

To facilitate the computation, we can pre-scan the sequence and compute the marginal cost bound b_i, the running bound B_i as well as $\pi(i)$, for each individual request r_i, $1 \le i \le n$, in advance as we illustrated before. With the information of B_i, we can further compute the $C(i)$ and $D(i)$.

At $t_0 = 0$, $C(0)$ and $D(0)$ are initialized by 0 and $+\infty$, respectively. Since the first request on any server except s^1 will have to be served by a transfer, $D(1) - D(3)$ are set by $+\infty$, while $C(1) = \min\{D(1), C(0)+1+0.5\} = 1.5$, $C(2) = \min\{D(2), C(1)+0.3+1\} = 2.8$, $C(3) = \min\{D(3), C(2)+0.3+1\} = 4.1$. In order to compute $C(4)$, we have to compute $D(4)$ first. $D(4) = C(0) + 1.4 + 3 - 0 = 4.4$ and $C(4) = \min\{D(4), C(3)+0.3+1\} = 4.4$. Now we consider to compute the final optimal value $C(7)$. To this end, according to Recurrence (5), we have $D(7) = 9.2$ and $C(7) = 8.9$ because

$$D(7) = \begin{cases} C(2) + 3.2 + 5.6 - 2 = 9.6 \\ \min \begin{cases} 4.4 + 3.2 + 5.6 - 4 = 9.2, \\ 6.5 + 3.2 + 5.6 - 5 = 10.03, \\ 7.1 + 3.2 + 5.6 - 5.6 = 10.03 \end{cases} \end{cases}$$

and $C(7) = \min\{D(7), C(6) + 0.8 + 1\}$. The full vectors of C and D are listed in the table of Fig. 6.

The optimal schedule $\Psi^*(7)$ can be reconstructed by recursively backtracking the vectors of C and D up to the initial configuration at $t = 0$. As such, we can phase by phase steer to the final optimal results as shown in Fig. 6. Since the transfer cost is a constant, we can only store the cache schedule by marking the responding request nodes in Q_j, $1 \le j \le m$ (Fig. 6).

V. A 3-Competitive Online Algorithm

In this section, we describe an 3-competitive algorithm for the online version of this problem. The algorithm is built on a concept of *speculative caching* that allows the copy migrated to a sever to speculatively keep active for another period of $\Delta t = \lambda/\mu$ after it serves the most recent request at time t. The rationale behind this idea is that if the next request is coming no later than $t + \Delta t$, it should be served by caching as the caching cost is not more than λ; otherwise, the copy is not worthwhile to be kept, and the request is served by a transfer from other server, instead. In this way, we can enable the online algorithm to mimic the optimal off-line algorithm as close as possible. The algorithm operates on a per-epoch basis along the time-line, and each *epoch* is composed of n transfers. We call this online algorithm *Speculative Caching (SC)* algorithm, which operates as follows in each epoch.

1) use variables c and r to record the number of active copies and the number of transfers in current epoch, respectively. Initially, $c \leftarrow 1$ and $r \leftarrow 0$, and the data is located at s^1;
2) use a counter array of $C[m]$, initialized by zero, to maintain the copy expiration information of each server in current epoch, e.g., $C[i] \leftarrow t_i$ indicates the copy on s^i will expire at t_i, $1 \le i \le m$.
3) when a new request r_i on s^j is coming at t_i:
   - for s^j, if $t_i \in [t_{p'(i)}, t_{p'(i)} + \Delta t]$ and $C[j] \neq 0$, then serve r_i by the copy on s^j, and then update $C[j] \leftarrow t_i + \Delta t$. Otherwise, serve r_i by a transfer from $s^k, k \neq j$ where r_{i-1} is made, and update $C[j] \leftarrow t_i + \Delta t$ and $r \leftarrow r + 1$;
   - for $s^k, k \neq j$, if $s^k (C[k] \neq 0)$ performs a transfer at t_i, then update $C[k] \leftarrow t_i + \Delta t$.
   - if $r = n$ then the current epoch is completed, and the next epoch is started with $c \leftarrow 1$ and $r \leftarrow 0$, $C[m] \leftarrow 0, 1 \le i \le m$, and the data located at s^j.
4) when events of copy expiration happen at t_i:[1]

[1]According to SC, there are at most two expiration events resulted from a transfer that could occur at the same time.

Authorized licensed use limited to: Zhejiang University. Downloaded on March 13,2023 at 01:45:54 UTC from IEEE Xplore. Restrictions apply.

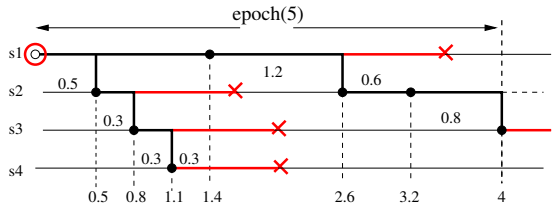


Fig. 7: An example of the online *SC* algorithm where the schedule for an epoch with size of 5 is illustrated.

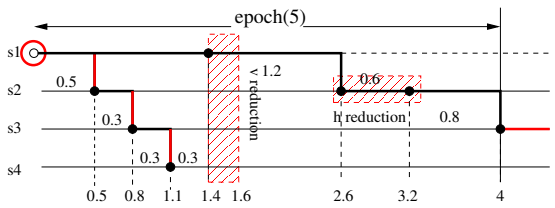


Fig. 8: An example of the DT schedule where the red circle and transfer lines represent the initial cost and the transfer cost of the data that are increased by corresponding $\omega_j^i \in \Omega_j, 1 \leq j \leq 4$ ($\omega_2^2 = 0$ and $\omega_3^2 = 0$ starting at t_4). Additionaly, the two types of reductions are also showed in shaded rectangles.

- $c \leftarrow$ the number of active copies,
- if there are two events on s^j and s^k at the same time, and $c > 2$, then $c \leftarrow c - 2$ and delete the copies on s^j and s^k (i.e., $C[j] \leftarrow 0, C[k] \leftarrow 0$).
- if there are two events on s^j and s^k, but $c = 2$ (the last two copies), then delete the copy in source server, say s^j, to break the tie, keep the copy in target server s^k (i.e., $C[j] \leftarrow 0, C[k] \leftarrow t_i + \Delta t$), and finally set $c \leftarrow 1$,
- in other cases, if there is a single event on s^j and $c > 1$, just delete the copy on s^j (i.e., $C[j] \leftarrow 0$) and set $c \leftarrow c - 1$. Otherwise if there is a single event on s^j but $c = 1$, then extend the copy expiration time on s^j to $t_i + \Delta t$ (i.e, $C[j] \leftarrow t_i + \Delta t$).

An illustrative example of this algorithm for a single epoch with 5 transfers is shown in Fig. 7 where each copy survives another speculative period of time at most $\Delta t = \lambda/\mu$ for incoming requests. Based on the algorithm and this example, we can easily make the following observation:

Observation 4. *For each request r_i at t_i on s^j, the online* SC *algorithm satisfies the following properties:*

1) *when $\mu\delta t_{p'(i),i} \leq \lambda$, r_i is always served by caching on s^j;*
2) *when $\mu\delta t_{p'(i),i} > \lambda$,*
   - *if $t_{p'(i)} < t_{i-1}$, r_i is served by the copy created at t_{i-1} on $s^k, k \neq j$ via a transfer where $t_{p'(i)} < t_{i-1}$.*
   - *otherwise, when $t_{p'(i)} = t_{i-1}$, which also implies $t_{p(i)} = t_{i-1}$, r_i is served by the copy created at t_{i-1} on s^j.*

here, $t_{p'(i)}$ is the most recent time instance that a request or a transfer (to other server) happen before r_i at t_i on the same server, say s^j in this observation (e.g., $p'(6) = 3$ and $t_{p'(6)} = 0.8$ on s^2 in Fig. 7). Clearly, $p(i) \leq p'(i) \leq i-1$. 2) is correct since according to the algorithm, the latest copy created at t_{i-1} is always available to r_i by continuously expanding its active periods.

For the sake of easy analysis, we transform the SC schedule in an epoch into an equivalent schedule, called *Double Transfer (DT)* schedule, that has exactly the same cost with the SC schedule. To this end, we denote the set of SC costs on s^j as Ω_j, and have the following definition:

Definition 10 (Double Transfer Schedule)**.** The double transfer schedule can be obtained from the SC schedule by performing the following transformations for each SC cost $\omega_j^i \in \Omega_j$ on $s^j, 1 \leq j \leq m$ (note that $\omega_j^i \leq \lambda$):

1) if $j = 1$ and $i = 1$, increase the initial cost on s^1 from 0 to ω_1^1;
2) otherwise, remove ω_j^i and add it to the weight of the most recent transfer edge to s^j, whose value is increased to $\lambda + \omega_j^i$ which is less than or equal to 2λ.

The transformation is reasonable as each ω_j^i on s^j corresponds to an incoming transfer edge in the schedule. Therefore, we can have $\Pi(DT) = \Pi(SC)$ in $O(mn)$ time.

An example of the schedule produced by the *DT* algorithm after the transformation is shown in Fig. 8. Given ω_2^2 and ω_3^2 starting at t_4 are equal to zero in the example, one can verify they have the same scheduling strategies and the total costs.

With these results, we have the following lemma that shows if a time interval is greater than λ, we can only consider a single caching location for both *DT* and any optimal algorithm (*OPT*).

Lemma 5. *In both* DT *and* OPT *schedules, if $\mu\delta t_{i-1,i} > \lambda$ then only one server will cache the data in $[t_{i-1}, t_i]$.*

Proof: We first consider the *OPT* schedule. Suppose there are two or more caching intervals covering $[t_{i-1}, t_i]$ in some optimal solution. By Observation 2, r_i will be served by exactly one of the following choices,

1) the cached copy on server s_i
2) a transfer $Tr(s^j, s_i, t_i)$ where s^j holds one of the overlapping caching intervals and $s^j \neq s_i$.

In either case, there will be another cached copy on another server s^k, $H(s_k, t_k, t_j)$ which spans the interval $[t_{i-1}, t_i]$. Since there are no requests between r_{i-1} and r_i, $t_k < t_{i-1}$ and $t_j > t_i$. If we transfer $Tr(s_i, s_k, t_i)$ and eliminate the interval $[t_{i-1}, t_i]$ on server s_k, we will reduce the cost by $\mu\sigma_{i-1,i} - \lambda > 0$, contradicting the assumption that the solution was optimal.

According to the *DT* schedule, if $\mu\delta t_{i-1,i} > \lambda$, then $t_{p'(i),i} \leq i - 1$, and $\delta t_{p'(i),i} > \lambda$. According to *DT*, r_i is served by the copy created at t_{i-1} on $s^k, k \neq j$ via a transfer. If there is another cached copy on another server that spans the interval $[t_{i-1}, t_i]$, it must contradict the algorithm by following

Authorized licensed use limited to: Zhejiang University. Downloaded on March 13,2023 at 01:45:54 UTC from IEEE Xplore. Restrictions apply.

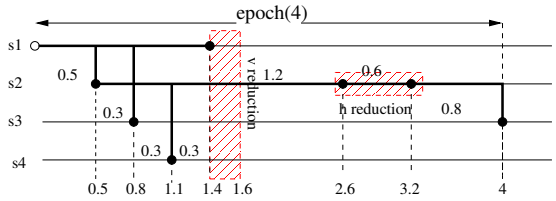
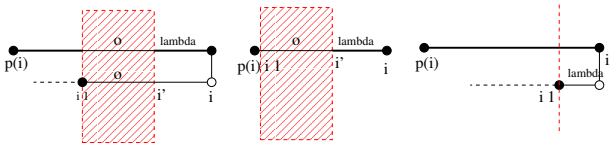


Fig. 9: The optimal schedule with 4 transfers after the two types of reductions.



(a) Weight(i 1,i) lambda (b) Weight(p(i),i) lambda (c) Weight(i 1,i) lambda

Fig. 10: How the $\mu\sigma'_i$ is computed for reduced schedule after the reductions? (a) when $p(i) < i-1$ and $\mu\delta t_{i-1,i} > \lambda$, (b) when $p(i) = i-1$ and $\mu\delta t_{i-1,i} > \lambda$, (c) when $\mu\delta t_{i-1,i} \leq \lambda$.

the same arguments for *OPT*. As a result, no more than one copies in *DT* will be active in parallel in $[t_{i-1}, t_i]$. ■

Based on Lemma 5, we can make the following reduction on both schedules.

Definition 11 (V-Reduction)**.** For each intervals $[t_{i-1}, t_i], i \in [1, n]$ that satisfy $\mu\delta t_{i-1,i} > \lambda$ in both schedules, we can reduce its weight to $\mu\delta t_{i-1,i} = \lambda$ by setting $\mu\delta t_{i-1,i'} = 0$ where $t_{i'} < t_i$. We call it *V-Reduction*.

As such, for any $r_i \in \mathcal{R}$ in an epoch, we have $\mu\delta t_{i-1,i} \leq \lambda$ in both *DT* and *OPT*. An example of the *v-reduction* is shown in Fig. 8 and Fig. 9.

Moreover, we have the following lemma to show that each request in $SR = \{r_i : \mu\sigma_i < \lambda, i > 0\}$ is satisfied in the same way in both *DT* and any *OPT* schedules.

Lemma 6. *For any i where $\mu\sigma_i < \lambda$, $H(s_i, t_{p(i)}, t_i)$ is a part of the* DT *and any* OPT *schedules.*

Proof: First, consider a request r_i in *OPT* where $\sigma_i < \lambda$. Any solution which uses a transfer to supply request r_i can be improved by $\lambda - \sigma_i$ by replacing the transfer with the caching on s_i for the interval $[t_{p(i)}, t_i]$. As for *DT*, this is a direct conclusion from Observation 4. ■

Based on Lemma 6, we can make the following reduction on both schedules.

Definition 12 (H-Reduction)**.** The caching cost of each request in SR can be removed by setting the cost to zero for both schedules. We call it *H-Reduction*.

As a result, we have for any $r_i \in \mathcal{R}$ in an epoch, $\mu\sigma_i \geq \lambda$ in both *DT* and *OPT*. We can observe it by comparing the *h-reductions* in Fig. 8 and Fig. 9.

After the above reductions, we can reduce both *DT* and *OPT* schedules by the same amount of weights in serving the requests in $\mathcal{R}$, and then have $\frac{\Pi(DT)}{\Pi(OPT)} \leq \frac{\Pi(DT')}{\Pi(OPT')}$ where DT' and OPT' represent the *reduced DT* and *OPT* schedules, respectively, after the reductions.

With these results, we now have our main theorem:

Theorem 3. *The speculative caching (*SC*) algorithm is* 3*-competitive.*

To prove the theorem, let $\mathcal{R}' = \mathcal{R} \setminus SR$ since we can equivalently view the reduced schedules as those working on $\mathcal{R}'$ given the *h-reduction*. In view of this, we then have the following lemma to show an upper bound of $\Pi(DT')$.

Lemma 7. *For an online sequence $\mathcal{R}$ in an epoch, $\Pi(DT')$ is upper bounded by $3n'\lambda$ where $n' = |\mathcal{R}'|$.*

Proof: Since for DT', the schedule reduction results in $\mu\delta t_{i-1,i} \leq \lambda$ for any request $r_i \in \mathcal{R}'$, we then have

- if $\mu\delta t_{p'(i),i} \leq \lambda$, r_i is served by caching at cost of no more than λ.
- Otherwise, r_i is served by the copy created at t_{i-1} on $s^k, k \neq j$ via a transfer at most cost of 2λ. Since $\mu\delta t_{i-1,i} \leq \lambda$, the cost to serve r_i would be at most 3λ.

Overall, the total cost to serve the whole sequence $\mathcal{R}'$ in an epoch is at most $3n'\lambda$. ■

Now, we estimate the lower bound of $\Pi(OPT')$. According to Definition 5, running bound B' is an lower bound of $\Pi(OPT')$. To this end, we let $B' = \sum_{i=1}^{n'} b'_i = \sum_{i=1}^{n'} \min\{\lambda, \mu\sigma'_i\}$ with respect to $\mathcal{R}'$ after the schedule reductions. Due to the impact of *v-reduction* on $\mu\sigma'_i$, we have to refine $\mu\sigma'_i$ to compute B' as we show in Fig. 10 where three cases are considered to obtain:

$$\mu\sigma'_i = \begin{cases} \mu\sigma_i - (\mu\delta t_{i-1,i} - \lambda) & \text{if } p(i) \leq i-1 \text{ (case 1\&2)} \\ \mu\sigma_i & \text{Otherwise (case 3)} \end{cases} \tag{6}$$

Lemma 8. *For online sequence $\mathcal{R}$, $\Pi(OPT')$ is lower bounded by $n'\lambda$ where $n' = |\mathcal{R}'|$.*

Proof: Given definition in (6), we have $\mu\sigma'_i \geq \lambda$ since 1) $\mu\sigma_i \geq \mu\delta t_{i-1,i}$ when $p(i) \leq i-1$, and 2) $\mu\sigma_i \geq \lambda$ in OPT' after *h-reduction*. Then, $\Pi(OPT') \geq B' = \sum_{i=1}^{n'} b_i = \sum_{i=1}^{n'} \min\{\lambda, \mu\sigma'_i\} = n'\lambda$. ■

With the above results, we can prove Theorem 3 as follows:

$$\frac{\Pi(SC)}{\Pi(OPT)} = \frac{\Pi(DT)}{\Pi(OPT)} \leq \frac{\Pi(DT')}{\Pi(OPT')} \leq \frac{3n'\lambda}{n'\lambda} = 3.$$

Finally, $\Pi(SC) \leq 3 \cdot \Pi(OPT)$ in an epoch. Since it can be repeated on each epoch, the *SC* algorithm is 3-competitive.

VI. Conclusions

In this paper, we studied the data caching problem driven by next generation cloud-based data services, which is characterized by using monetary cost and access trajectory information to derive the cache replacements. With homogeneous cost model, we first proposed a fast optimal algorithm that can

Authorized licensed use limited to: Zhejiang University. Downloaded on March 13,2023 at 01:45:54 UTC from IEEE Xplore. Restrictions apply.

serve an off-line request sequence in a fully connected network within $O(mn)$ time-space. Then, we investigated online algorithms for this problem, and presented a 3-competitive algorithm that leverages an idea of *speculative caching*. Our algorithms are not only practical to the data caching problem, but also appear to be of theoretical significance to a natural new paradigm in the realm of online algorithms. We provably achieve these results with our deep insights into the problem and the careful analysis.

ACKNOWLEDGMENTS

This work was supported in part by the China National Basic Research Program (973 Program, No. 2015CB352400), NSFC under grant No. 61672513, 61572487, and 61572377. Science and Technology Planning Project of Guangdong Province (2015B010129011, 2016A030313183).

REFERENCES

[1] A. Juels and A. Oprea, "New approaches to security and availability for cloud data," *Commun. ACM*, vol. 56, no. 2, pp. 64–73, Feb. 2013.

[2] C. Song, Z. Qu, N. Blumm, and A.-L. Barabási, "Limits of predictability in human mobility," *Science*, vol. 327, no. 5968, pp. 1018–1021, 2010.

[3] P. R. Lei, T. J. Shen, W. C. Peng, and I. J. Su, "Exploring spatial-temporal trajectory model for location prediction," in *2011 IEEE 12th International Conference on Mobile Data Management*, vol. 1, June 2011, pp. 58–67.

[4] Y. Wang, B. Veeravalli, and C.-K. Tham, "On data staging algorithms for shared data accesses in clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 99, no. PrePrints, 2012.

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

[6] B. Veeravalli, "Network caching strategies for a shared data distribution for a predefined service demand sequence," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 6, Nov. 2003.

[7] G. Chockler, G. Laden, and Y. Vigfusson, "Data caching as a cloud service," in *Proceedings of the 4th International Workshop on Large Scale Distributed Systems and Middleware*, ser. LADIS '10, 2010, pp. 18–21.

[8] ——, "Design and implementation of caching services in the cloud," *IBM Journal of Research and Development*, vol. 55, no. 6, pp. 9:1–9:11, Nov 2011.

[9] N. L. Scouarnec, C. Neumann, and G. Straub, "Cache policies for cloud-based systems: To keep or not to keep," in *2014 IEEE 7th International Conference on Cloud Computing*, June 2014, pp. 1–8.

[10] M. Kallahalla and P. J. Varman, "Pc-opt: Optimal offline prefetching and caching for parallel i/o systems," *IEEE Trans. Comput.*, vol. 51, no. 11, pp. 1333–1344, Nov. 2002.

[11] B. S. Gill, "On multi-level exclusive caching: Offline optimality and why promotions are better than demotions," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, ser. FAST'08, 2008, pp. 4:1–4:17.

[12] A. Jain and C. Lin, "Back to the future: Leveraging belady's algorithm for improved cache replacement," in *the 43th International Symposium on Computer Architecture (ISCA)*, 2016.

[13] S. Jiang and X. Zhang, "Lirs: An efficient low inter-reference recency set replacement policy to improve buffer cache performance," in *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '02, 2002, pp. 31–42.

[14] M. Chaudhuri, "Pseudo-lifo: The foundation of a new family of replacement policies for last-level caches," in *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 42, 2009, pp. 401–412.

[15] D. A. Jiménez, "Insertion and promotion for tree-based pseudolru last-level caches," in *Proc. of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46, 2013, pp. 284–296.

[16] D. Sleator and R. Tarjan, "Amortized efficiency of list update and paging rules," *Communication of ACM*, vol. 28, pp. 202–208, 1985.

[17] M. Charikar, D. Halperin, and R. Motwani, "The dynamic servers problem," in *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, ser. SODA '98, Philadelphia, PA, USA, 1998, pp. 410–419.

[18] D. Halperin, J. Latombe, and R. Motwani, "Dynamic maintenance of kinematic structures," Stanford, CA, USA, Tech. Rep., 1995.

[19] B. Veeravalli and E. Yew, "Network caching strategies for reservation-based multimedia serives on high-speed networks," *Data&Knowledge Engineering*, vol. 41, no. 1, Apr. 2002.

[20] W. Shi and C. Su, "The rectilinear steiner arborescence problem is np-complete," in *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, ser. SODA '00, Philadelphia, PA, USA, 2000, pp. 780–787.

Authorized licensed use limited to: Zhejiang University. Downloaded on March 13,2023 at 01:45:54 UTC from IEEE Xplore. Restrictions apply.