Final version appears in Proceedings of IEEE INFOCOM 2017.

Proactive Retention Aware Caching

Samta Shukla and Alhussein A. Abouzeid

Department of Electrical, Computer and Systems Engineering
Rensselaer Polytechnic Institute, USA
{shukls, abouza}@rpi.edu

***Abstract*—We consider the problem of proactive (i.e. predictive) content caching that is aware of the costs of retention of the content in the cache. Prior work on caching (whether proactive or reactive) does not explicitly take into account the storage cost due to the duration of time for which a content is cached. This new problem, which we call retention aware caching, is motivated by two recent technological developments that are described in the paper: cloud storage rental costs and flash memory damage. We consider a hierarchical network consisting of a server connected to a number of cache-enabled nodes, located either at the edge of a network (e.g. base stations) or in the core of a data center. There are two types of network costs: storage cost at the caches and download cost from the server. We formulate the problem of proactive retention aware data caching (PRAC), which minimizes the total cost subject to the node capacity constraints. We first prove that PRAC is NP-Hard in general and then analyze PRAC for two cases: (1) linear storage cost, (2) convex storage cost. We show that PRAC admits efficient polynomial time algorithms when the storage cost is linear in retention times and caches have a large capacity. Furthermore, we derive bounds on the performance of PRAC for the case when the storage cost is a practically motivated convex function. Numerical evaluations demonstrate that PRAC outperforms other state-of-the-art caching policies for a wide range of parameters of interest.**

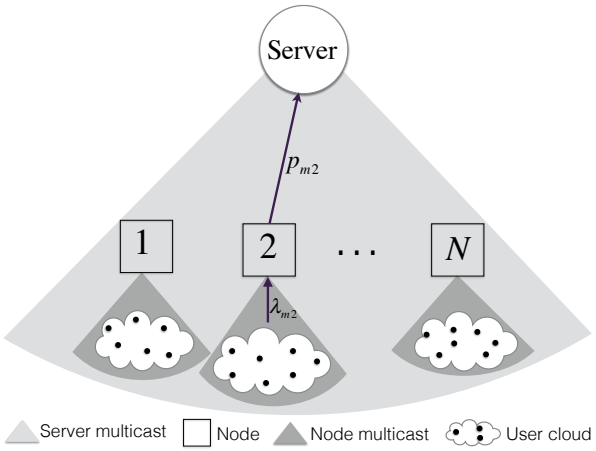

Fig. 1: The network consists of a server housing a library of M files and a set of N nodes equipped with caches. A multicast transmission from the server is received by every node and user cloud, whereas the multicast from a node can locally serve the demands of the associated user cloud only.

I. Introduction

Data caching is known to provide significant gains in reducing download delays for delay-intolerant, data-intensive applications such as in CDNs, data centers [1], [2], information-centric/mobile networks [3]–[5], social networks [6] and cloud computing frameworks [7]. A recent line of work, proactive data caching [3], [4], [6], [7], is shown to further boost the performance compared to reactive data caching. Caching decisions in proactive caching are made ahead of time. The content demands, although uncertain, can either be predicted [8], [9] or are natively available, such as in information-centric network (ICN) due to its receiver-driven architecture [4].

In this paper, we investigate the problem of Proactive Retention Aware content Caching (PRAC) in a hierarchical network where every content is stored in the cache for a specific duration, called *retention time* henceforth. This model for having retention times is motivated by two very different practical problems, first is the storage cost for renting disk space in cloud networks (e.g. data centers) and second is the cost incurred due to retaining data in the underlying hardware memory. These two factors contribute to the storage cost and play a critical role in a data-intensive environment where reliability is a prime concern. Traditionally, cache network models have largely focused on minimizing the content download delay while overlooking the storage attributes.

The first motivation behind considering storage cost can be explained as follows. Cloud service providers (like Amazon AWS, Google, Microsoft Azure) charge users for the actual usage of disk space as well as file downloads. Many cloud providers offer CDN as a service. Caching/replicating a popular file in different data centers minimizes the download cost. This has motivated the prior work [10] of studying caching in data centers. However, caching a file at multiple data centers incurs a storage cost at each of these data centers. This cost, which typically depends on the duration for which a content is stored, has not been considered by previous works.

The second motivation behind considering storage cost is explained as follows. Storing a file involves writing it to an underlying memory device which could be costly depending on the way it is written. NAND Flash – the most widely used memory hardware for caching applications – is marketed with an endurance/lifetime specified in the form of the number of P/E cycles it can sustain [11], [12]. Thus, writing a content to the memory for a specified retention duration incurs a physical damage that reduces its lifetime. Right before retention expiration, the content is rewritten by the virtue of a scrubbing

algorithm at the expense of a P/E cycle at each write, thus causing substantial flash damage with each write/re-write. (See [13], [14] to read more about P/E cycles, retention times, flash failures and reliability.) Prior work [11] minimized the device damage for an isolated content-centric cache where the authors found that the optimal retention times are proportional to content popularity.

In this work, we study proactive content caching to minimize the sum of download and storage costs in a hierarchical multicast-enabled cache network as shown in Figure 1. Typically, the servers in such a cache network relay data to the leaf node either by employing a unicast transmission, such as in a wired network (e.g. data centers [1], [2]) or by multicasting data to all the leaf nodes at once. The multicasting mode of transmission becomes a natural choice in wireless edge caches [15]–[17]; it is also used in hybrid networks, such as data centers, where both wireless and wired links coexist [18].

Earlier works in this area did not consider storage cost. Our work may seem related to [3] which proposes proactive data caching to minimize the content-retrieval delay (download cost), however, it does not consider multicast for serving files. Authors in [16] propose multicast-aware proactive caching to minimize the download cost but they did not consider storage cost. Another related work in [17] assumes that every content can be stored only for a fixed duration time frame and there is a cost associated with reloading the cache in the beginning of a time frame. While [17] may appear very similar to our work, there are notable differences. First, their assumption regarding the lifetime of a content chunk being a time frame is too stringent; we generalize this by allowing contents to be stored for any duration *in the multiple of time slots* during a time frame. Second, their assumption of cache reload cost being a random variable is not in accordance with the recent literature that suggests that a deterministic increasing linear/moderately-convex function appropriately fits the storage cost [13].

Summary of Contributions:

1) We formulate PRAC as an Integer Program and prove that it is NP-Hard (see Theorem 1). We observe that PRAC exhibits coupling among files (due to cache capacity constraints[1]) as well as nodes (due to multicast).
2) We investigate PRAC by first decoupling it across files by assuming very large cache capacities (i.e., essentially removing cache capacity constraints). With this assumption, PRAC is reduced to optimizing the objective function for each individual file. For a given file (which can be requested on multiple nodes), we prove that the sequence of retention durations across the nodes follow a simple characterization: higher request probability implies higher retention time (see Theorem 2). This eliminates the combinatorial terms and simplifies the objective function for each file to be an Integer Linear Program (ILP). Note that the resulting ILP is still coupled across nodes.
3) With a large cache capacity assumption, we analyze ILP for linear as well as convex storage costs while keeping the download cost linear in both the cases. We prove that with linear costs, it can be solved by its corresponding LP relaxation, as the LP always returns integral solutions (see Theorem 3) compatible with our model assumption that the content can be stored only in the multiples of time slots. Further, when the storage cost is a convex increasing polynomial in retention times, we solve the problem again with the help of its LP relaxation and by subsequently rounding the fractional solutions. We derive analytical bound on the expected cost of rounded solution for reasonably convex storage cost functions (see Theorem 4).
4) Further, we consider the cache capacity constraints and propose a heuristic to fill the caches based on the optimal retention times obtained for the uncapacitated caches without significantly altering the costs. The simulations indicate that this heuristic outperforms a naïve popularity-based cache filling algorithm (see Section V-B). We synthetically generate data traces from uniform and ZipFian traffic distributions and conduct performance evaluation to study the impact of the parameters used (i.e. the slot size, the time frame duration, coefficient of ZipF distribution, weight of storage cost, choice of convex function and the unicast and multicast transmission modes (see Section V-C)). We conclude by establishing that a retention aware policy (such as, PRAC) significantly outperforms a retention unaware policy (see Section V-C).

[1]The observation that capacity constraints complicate the problem by coupling the formulation among the files also led to the recent work in TTL-caches in [19] or DARE caches in [12] where the authors have relaxed the cache capacity constraints and have instead associated timers with files.

II. System model and preliminaries

In this section, we introduce the system model and assumptions, formulate PRAC and conclude with a discussion on the hardness of the problem.

A. System Model

We consider a time-slotted operation where each time slot is of duration d_T units. Starting from time $t = 0$, a time *frame* consists of T slots, $t = 1, 2, ..., T$. Files are requested in every slot whereas they are proactively fetched only at the beginning of every time frame. Let p_{mn} denote the probability of requesting file m at node n in a given slot (assumed independent over slots) and let $[T]$ denote the set of slots in the frame, then the model allows proactively caching files at $t = 0$ for any number of slots $\leq T$. The length T of the frame is usually determined by the periodicity in the request patterns. Our model is motivated by [3] which exploits the periodicity in user demands to proactively cache the content ahead of time (typically when the server is not busy). Owing to the temporal periodicity in user demand fluctuations, it is reasonable to consider a time frame equal to a day [3].

The cache network consists of a server and N nodes (see Figure 1).[2] The server contains M files, where for the ease

[2]We will use nodes and caches, content and files interchangeably.

of exposition every file is of unit size (as in [17], [20]). We denote the set of N caches by $[N]$ and set of M files by $[M]$. The caches have finite capacity; cache $n \in [N]$ can contain at most B_n files.

We now state the key assumptions used in our model.

- *Content demands*: We assume that file request probabilities are known/can be predicted ahead of a time frame [4], [8], [9]. We assume that file m is requested at node n in each time slot independently with probability p_{mn}.
- *Delay intolerant service*: We assume delay intolerant service, i.e. a file requested in a slot will be served in the same slot. This assumption is relevant for media files or video content as outlined in [3], [17].
- *Proactive caching*: We assume that the nodes can proactively cache files in the beginning of each time frame for any number of slots $\leq T$ (and not during the frame, i.e. $1 \leq t \leq T$). This assumption is motivated from [3] which advocates proactively caching files when the server is not busy (say, at midnight) owing to periodicity in demands.
- *Multicast mode of transmission*: If a file requested at slot $t \in [T]$ is present in the cache then it is served instantaneously with no additional cost; otherwise, the node forwards the request to the server. A server multicasts all the files that are requested in a slot as the traffic is delay-intolerant. A multicast transmission of a file is received by all caches including the ones that have not requested the file (see Figure 1).
- *Storage cost*: Let $y_{mn} \in [T]$ denote the retention duration defined as the number of slots for which file m is stored at cache n starting from $t = 0$. Storing a file for duration y in cache incurs a storage cost $g(y) \in \mathbb{R}^+, g(0) := 0$. We assume $g(.)$ to be an increasing linear/convex polynomial function in this work motivated by [11], [13].
- *Download cost*: As all the files are unit-sized, we assume that a unit download cost is incurred on multicasting every file over a time frame of T slots.

B. PRAC formulation

As the cache network timeline is divided in time-frames of duration T time slots and the file request probabilities are assumed to be stationary, we formulate the objective only for the duration of a time frame.

The objective function consists of download and storage costs. The download cost is calculated as follows: In the t^{th} slot of a time frame, the probability that the server receives at least one request for file m from any of the caches becomes,

$$P(m, n, t) = 1 - \prod_{n: y_{mn} < t} (1 - p_{mn}), \tag{1}$$

where the product is over all nodes that do not contain the requested file at time t. Here $y_{nm} < t$ implies that the file m is no longer present in the node n as it had been stored only for the beginning y_{mn} time slots in a time frame. $P(m, n, t)$ is the probability with which the server multicasts file m in the t^{th} time slot. Thus $P(m, n, t)$ summed over all the time slots in a frame also represents the total download cost since we assume that each multicast carries unit cost. Furthermore, the storage cost during a time frame can be expressed as the summation of $\alpha \cdot g(y_{mn})$ over the files and nodes where $\alpha \in [0, 1]$ is the weight of the storage cost relative to the download cost. Then the objective of PRAC, assumed to be the average of the storage cost and the download cost over a time frame, becomes the following:

$$L(\mathbf{y}) = \frac{1}{T} \sum_{m \in [M]} \sum_{n \in [N]} \Big[\underbrace{\alpha \cdot g(y_{mn})}_{\text{storage cost}} + \underbrace{\sum_{t=1}^{T} P(m, n, t)}_{\text{download cost}} \Big]. \tag{2}$$

In the above equation, $\mathbf{y}$ denotes the matrix consisting of y_{mn} for all $n \in [N], m \in [M]$. Thus the optimization problem PRAC can be expressed as:

$$\begin{aligned} \textbf{PRAC}: \quad & \min_{y_{mn} \in \mathbf{y}} \; L(\mathbf{y}) \\ & \text{s. t.} \sum_{m \in [M]} 1_{y_{mn} > 0} \leq B_n \quad \forall n \in [N], \qquad (3) \\ & \qquad y_{mn} \in \{0, 1, 2, \ldots, T\}, \qquad (4) \end{aligned}$$

where $1_{y_{mn}>0}$ is an indicator function which equals one if the retention time of file m an node n is non-zero and zero otherwise. In the above formulation, the first constraint accounts for cache capacities and second constraint models the integrality requirements on retention times.

C. The hardness of PRAC

We prove that PRAC is NP-Hard even when the storage cost $g(.)$ is linear in retention times.

Theorem 1. PRAC *is NP-Hard.*

Proof sketch. We show that 3D-matching problem, which is known to be NP-hard, is a special case of PRAC. In 3D-matching, we are given a set of 3-tuples $\{(q_i, r_i, s_i) | q \in Q, r \in R, s \in S\}$ with Q, R, S being disjoint sets. The objective is to select the largest collection of 3-tuples such that no element is a part of more than one tuple in this collection. In the Appendix of [21], we show that an instance of 3D-matching problem can easily be mapped to PRAC under straightforward setting of the caches being divided into 3 disjoint sets with each file being requested at exactly three nodes (along with some more simplifying conditions). ■

Remark: The above result is not surprising since various problems on optimal caching exhibit the property of being NP-hard [16], [22], [23] or even inapproximable [16].

We now move on to Section III-A where we discuss PRAC for a linear storage cost. We show that PRAC admits polynomial time solutions in this case when *nodes do not have cache capacity constraints*.

III. Analytical results for linear storage cost

In this section we formulate PRAC under the assumption that the storage cost of file m at cache n is linear in retention

time, i.e. $g(y_{mn}) = y_{mn}$[3]. Then the cost objective with linear $g(.)$ can now be expressed as:

$$L_a(\mathbf{y}) := \frac{1}{T} \sum_{m\in[M]} \sum_{n\in[N]} \left[\alpha y_{nm} + \sum_{t=1}^{T} P(m,n,t) \right].$$

Thus, PRAC with linear storage costs becomes:

$$\mathbf{IP_a}: \min_{y_{mn}\in\mathbf{y}} \; L_a(\mathbf{y}) \text{ subject to } (3) \text{ and } (4)$$

We investigate $\mathrm{IP_a}$ in two steps. First, we remove the cache capacity constraint (3). Under this setting, $\mathrm{IP_a}$ decouples over files and is equivalent to minimizing objective function individually for each file. We then prove a key structural property of optimal retentions. This property enables us in reducing $\mathrm{IP_a}$ to an Integer Linear Problem (ILP). We characterize that this ILP can be efficiently solved using a simple threshold based rule for large caches. Finally, for the capacity-constrained caches, we propose a heuristic algorithm to assign files to caches respecting their capacity constraints, given the optimal file retentions for the corresponding uncapacitated case.

A. Optimal retentions for a large cache

In what follows we assume that all caches are large (or uncapacitated). Typically data centers have huge storage units so from a practical perspective one can consider that the storage capacity is unlimited [1], [2].

We remove constraint (3) from problem $\mathrm{IP_a}$ thus assuming that the caches may contain as many files where each file is stored for their retention time. Removing the capacity constraint simplifies the analysis as now the problem decouples over files and can be independently solved for each file $m \in [M]$. Under this setting, the objective function for a given file $m \in [M]$ can be expressed (where we slightly abuse the notation by reusing $p_{nm} = p_n, y_{nm} = y_n$) as:

$$L_b(\mathbf{y}) := \frac{1}{T}[\sum_{n\in[N]} \alpha y_n + \sum_{t=1}^{T} (1 - \prod_{n:y_n<t} (1-p_n))].$$

In the above equation, $\mathbf{y} = \{y_n\}, n \in [N]$ denotes the array of retention times for N caches. Adding the constraint that retention times must be integral in terms of time slots, the optimization problem $\mathrm{IP_b}$ (for a file under consideration) can be expressed as:

$$\mathbf{IP_b}: \min_{\mathbf{y}} \; L_b(y) \quad \text{subject to } (4)$$

The above formulation remains combinatorial as before. Note that in IP_b the retention times can take any values in the set $\{0, 1, \ldots, T\}$. However, as we prove next in Theorem 2, the optimal retention times (for a given file across the nodes) exhibit a key structural property:

Theorem 2. *W.l.o.g. let* $0 \le p_1 \le p_2 \cdots \le p_N \le 1$. *Then the optimal retention times* $y_1, y_2, \ldots, y_N$ *are ordered as* $0 \le y_1 \le y_2 \le \cdots \le y_N \le T$.

[3]Note that even with a generalized linear cost of $g(y_{mn}) = ay_{mn} + b$, the objective can be shown to be equivalent to $L_a(\mathbf{y})$.

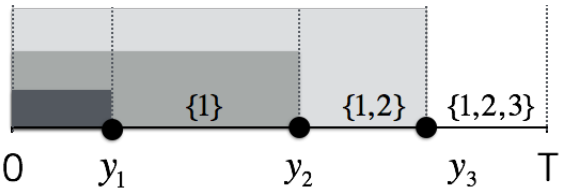

Fig. 2: Timeline of cache evolution for $N = 3$, $M = 1$.

Proof sketch. We prove this by making use of exchange argument where we show that in the optimal solution if $y_i > y_j$ for some pair $i < j$ then by exchanging y_i and y_j the overall cost can be reduced leading to a contradiction. The full proof is available in the Appendix of [21]. ■

Theorem 2 proves that the optimal retention times (for a particular file across the caches) preserve the order on request probabilities, i.e. higher the request probability, more likely is the file to be retained for a longer time in the cache. We emphasize that the order specified in Theorem 2 depends only on the request probabilities and is independent of other parameters such as α, T. Moreover, this characterization also applies for convex polynomial storage cost considered in Section IV.

Note: *Following Theorem 2, we assume that* $0 \le p_1 \le p_2 \cdots \le p_N \le 1$ *for a given file without loss of generality for the rest of this section until Section III-B.*

Theorem 2 simplifies the objective function $L_b(\mathbf{y})$ even further. We explain it with the help of a graphical illustration in Figure 2. Here the time between $t = 0$ to $t = T$ slots is partitioned in $N + 1$ regions where the probability that the server receives at least one file request during region i (i.e. from slot $y_{i-1} + 1$ to y_i) is given by,

$$A_i = 1 - \prod_{n=1}^{i} (1 - p_n). \tag{5}$$

Note that from slot $t = 1$ to y_1 the download cost is zero as all the caches have the file; on the contrary, from slot $t = y_N + 1$ to $t = T$ none of the caches have the file so the server gets a file request with probability A_N. Thus, the download cost over the frame duration can be expressed in terms of A_i, $i \in \{1, 2, \ldots, N\}$ as:

$$\begin{aligned}
&\sum_{t=1}^{T} (1 - \prod_{n:y_n<t} (1-p_n)) \\
&= \sum_{y_1+1}^{y_2} A_1 + \sum_{y_2+1}^{y_3} A_2 + \cdots + \sum_{y_{N-1}+1}^{y_N} A_{N-1} + \sum_{y_N+1}^{T} A_N \\
&= \sum_{n=1}^{N-1} (y_{n+1} - y_n) A_n + (T - y_N) A_N \\
&= \sum_{n=1}^{N} (y_{n+1} - y_n) A_n
\end{aligned} \tag{6}$$

where y_{N+1} is a dummy variable defined as T. Thus, due to

(6), $L_b(y)$ gets simplified to:

$$L_c(\mathbf{y}) := \sum_{n=1}^{N} \alpha \frac{y_n}{T} + \sum_{n=1}^{N} \frac{(y_{n+1} - y_n)}{T} A_n. \tag{7}$$

Note that the objective $L_c(\mathbf{y})$ is linear in retention time variables y_n as the A_n's are known. Thus we obtain an Integer LP formulation. We solve it by relaxing the integral constraints on the variables and call the problem $\mathrm{IP_c}$.

$$\mathbf{IP_c}: \min_{\mathbf{y}} L_c(\mathbf{y}) \quad \text{subject to} \quad 0 \leq y_n \leq T \ \ \forall n \in [N].$$

$\mathrm{IP_c}$ is a linear program that can be efficiently solved. In fact, by the virtue of linearity, we observe that the optimal solution y_n^* will have the following characterization:

Theorem 3. *The optimal solution of* $\mathrm{IP_c}$ *has the following threshold-based characterization:*

$$\mathbf{y}^* = [\underbrace{0, 0, \ldots, 0}_{k}, \underbrace{T, T, \ldots, T}_{N\text{-}k}]$$

where the first k *retention times (i.e.,* $\{y_1^*, y_2^*, \ldots, y_k^*\}$*) are set to* 0 *and the next* $N-k$ *retention times are set to* T *and where* k *is defined as:*

$$k := \arg\max_i \{\alpha + A_i - A_{i-1} > 0\}$$

Proof sketch. The above theorem is a consequence of linearity of the objective function $\mathrm{IP_c}$ and Theorem 2. Complete proof and the characterization of k is in the Appendix of [21]. ■

The optimal solution of $\mathrm{IP_c}$ for each file can be computed in $O(N \log N)$ time from Theorem 3, where sorting the request probabilities take $O(N \log N)$ time. As discussed in the beginning of Section III-A, with linear storage cost and large caches, the formulation of PRAC decouples over files. Thus, an optimal solution of PRAC, for all files, with linear storage cost and unbounded cache capacities can be computed in $O(MN \log N)$ time.

Recall that so far we have solved the problem of determining optimal retention times assuming large cache capacities. Next, we propose a heuristic to compute retention times for the case with linear storage cost and finite cache capacities.

B. Cache capacitated prefetching with optimal retentions

Since PRAC is NP-hard in general and even for linear storage cost with finite cache capacities (See Section II-C), we cannot guarantee efficient solutions for such a case. We thus present a heuristic in Algorithm 1, which we call Fill-Cache, which proceeds via iterations on the set of nodes that violate the cache capacity constraints.

Each iteration of Fill-Cache can be intuitively explained as follows: If the set of nodes U violating cache capacity constraints is non-empty then remove that particular file from the nodes in U (i.e. set its retention to 0 for all the nodes in U) such that the cost $L_a(\mathbf{y})$ is the least after removing it (or, in other words, agrees the most with the overall cost $L_a(\mathbf{y})$ of the prior state). Note that computing $L_a(\mathbf{y})$ is equivalent to computing the sum of $L_c(y)$ for each file.

Algorithm 1 Fill-Cache

Input: The optimal retention matrix $\mathbf{y}$ consisting of y_{mn} (computed assuming no limit on cache capacities) for all $m \in [M], n \in [N]$ and the cache capacities B_n, $n \in [N]$.
Define U as the set of nodes violating cache capacity constraints.
while U is not empty **do**
  for each file m' cached in at least one node in U **do**
    Compute $cost(m') = L_a(\mathbf{y})$ by temporarily setting $y_{m'n} = 0$ for all $n \in U$.
  end for
  Find $\tilde{m} = \arg\min_{m'} cost(m')$.
  Set the retention time $y_{\tilde{m}n} = 0$ for all $n \in U$ in the retention matrix.
  Update U.
end while
Output: The retention matrix.

Fill-Cache is polynomial time, $O(MN)$, as each iteration requires computing the effect of removing at most M files and there are at most N iterations.

Thus far we analyzed the problem of minimizing the sum of linear storage and download cost for a caching application where caches can prefetch data for serving requests over the time frame. Although the problem is NP-hard for linear storage cost, we proved that when caches have large capacities, an optimal solution can be computed in polynomial time and characterized the structural properties of resulting retention times. Further, we proposed a heuristic algorithm for the case of finite capacity caches. We now move on to the analysis when storage cost is given by a convex, increasing polynomial function of retention times.

IV. Analytical results for convex storage cost

We now analyze PRAC when the storage cost is a convex, increasing polynomial function $f(.)$ of retention times with $f(0) := 0$ and degree d [11]. Similar to the linear case, we first remove the capacity constraints and invoke Theorem 2 which ensures that the optimal retention times preserve the order of request probabilities. Thus, similar to Equation (7), the objective for a given file becomes:

$$L_d(y) = \sum_{n=1}^{N} \frac{\alpha}{T} f(y_n) + \sum_{n=1}^{N} \frac{(y_{n+1} - y_n)}{T} A_n, \tag{8}$$

thus leading to a convex integer problem $\mathrm{CIP_a}$ defined as:

$$\mathrm{CIP_a}: \min_{y} L_d(y) \text{ subject to } y_n \in \{0, 1, 2, \ldots, T\}^N, \forall m.$$

We solve $\mathrm{CIP_a}$ for all $m \in [M]$ by relaxing the integral constraints in (4) as $0 \leq y_n \leq T$, for all $n \in [N]$. The resulting formulation is a convex program which can be efficiently solved. Let $h(\cdot) := f'^{-1}(\cdot)$ be the inverse of the derivative of the convex function, then the optimal solution is:

$$y_n^* = \min\left\{\max\left\{0, h\left(\frac{A_n - A_{n-1}}{\alpha}\right)\right\}, T\right\}, \forall n \in [N].$$

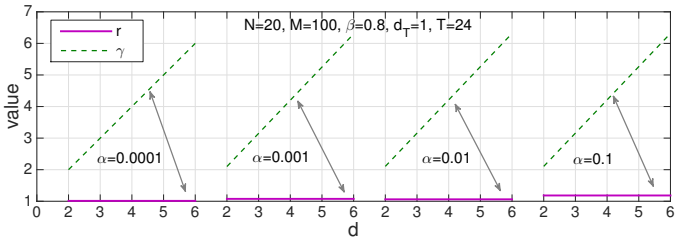

Fig. 3: Impact of varying d and α in the values of r and γ.

This characterization ensures that the the optimal solution always belongs to the range $[0, T]$ and that the optimal values may not be at extreme points (0 or T) anymore (unlike the linear case) and could be fractional, given by $h\left(\frac{A_n - A_{n-1}}{\alpha}\right)$ for y_n^*. Now we propose an approach to round the solution to integral values as required by the constraint of having retention times as multiples of time slots.

A. A rounding algorithm

Let y^* denote the optimal retention vector obtained from solving the relaxed convex program for a given file. Our approach to round an element $y_n^* \in y^*$ to an element in $\{\lfloor y_n^* \rfloor, \lfloor y_n^* \rfloor + 1\}$ as outlined in Algorithm 2 (where $\lfloor z \rfloor$ is the floor of z). Let $\widehat{y}$ denote a solution obtained after rounding y^*.

Algorithm 2 Mix-Rounding

Input: The retention duration $y_n^* \in y^*$ from CPopt.
for each n in 1 to N **do**
  Find $K = \lfloor y_n^* \rfloor$
  if $K = 0$ **then**
    deterministically assign $\widehat{y}_n = 0$
  else
    probabilistically assign $\widehat{y}_n$ as:

$$\widehat{y}_n = \begin{cases} K & \text{w. p. } 1 - (y_n^* - K) \\ K+1 & \text{w. p. } y_n^* - K. \end{cases}$$

  end if
end for
Output: The rounded solution $\widehat{y}_n,\ n \in [N]$.

The deterministic part is straightforward as it rounds $y_n^* \in [0, 1)$ to 0[4]. For the probabilistic part, when $y_n^* \geq 1$, let $K = \lfloor y_n^* \rfloor$. Then the rounding probability $y_n^* - K$ can be intuitively justified as the distance of the optimal solution y_n^* from K. The smaller the distance, the less is the probability of it being rounded to $K+1$ and vice versa. From the rounding algorithm, we have:

$$y_n^* \geq 1 \Rightarrow E[\widehat{y}_n] = (K+1)(y_n^* - K) + K(1 - y_n^* + K) = y_n^*. \tag{9}$$

[4]rounding $y_n^* \in [0, 1)$ to 0 is instrumental in proving the bound in Theorem 4 as explained in the proof in [21].

B. Bounding the performance of rounding algorithm

Let the cost in Equation (8) with y^* be denoted as CPopt. Let the expected cost of the rounded solution be given by $E[\text{CIPopt}]$. Obviously, $E[\text{CIPopt}] \geq \text{CPopt}$ (for every file) as relaxing the integrality constraints would only result in a lower bound on the value of the optimal. We now move on to our main result in this section (Theorem 4) in which we give an upper bound, γ, on $r = \frac{E[\text{CIPopt}]}{\text{CPopt}}$.

Theorem 4. *Suppose $f(.)$ is a convex increasing polynomial of degree d and the time frame consists of T slots, then*

$$r = \frac{E[\text{CIPopt}]}{\text{CPopt}} \leq \gamma = \max\left\{\frac{T}{T-1}, d, \frac{2^d}{d(e-1)}\right\}, \tag{10}$$

where e is the exponent of natural logarithm.

Proof sketch. When the optimal solution $\mathbf{y}^*$ is such that all $y_i^* \in [0, 1)$, the rounded solution $\widehat{\mathbf{y}} = \mathbf{0}$. Thus the rounded solution stores no files, and we have $E[\text{CIPopt}] = A_N$ (recall $y_{N+1} = T$). We also have $\text{CPopt} \geq \frac{y_{N+1}^* - y_N^*}{T} A_N \geq \frac{T-1}{T} A_N$. From these expressions of $E[\text{CIPopt}]$ and CPopt, we obtain the bound to be $\frac{T}{T-1}$. When the optimal solution $\mathbf{y}^*$ is such that all $y^* \geq 1$, we have the following expression for CPopt:

$$\text{CPopt} = \sum_{n=1}^{N} \left[\frac{\alpha}{T} f(y_n^*) + \frac{y_{n+1}^* - y_n^*}{T} A_n\right] \tag{11}$$

Also, with the help of Equation (9) the expected cost $E[\text{CIPopt}]$ of rounded solution $\widehat{\mathbf{y}}$ can be expressed as:

$$\begin{aligned} E[\text{CIPopt}] &= \sum_{n=1}^{N} \mathbb{E}\left[\frac{\alpha f(\widehat{y}_n)}{T} + \sum_{n=1}^{N} \frac{\widehat{y}_{n+1} - \widehat{y}_n}{T} A_n\right] \\ &= \sum_{n=1}^{N} \left[\frac{\alpha \mathbb{E}(f(\widehat{y}_n))}{T} + \sum_{n=1}^{N} \frac{y_{n+1}^* - y_n^*}{T} A_n\right] \end{aligned} \tag{12}$$

Equations (11)-(12) and simple ratio arithmetic gives:

$$\frac{\text{E[CIPopt]}}{\text{CPopt}} \leq \frac{\sum_{n=1}^{N} \mathbb{E}(f(\widehat{y}_n))}{\sum_{n=1}^{N} f(y_n^*)} \tag{13}$$

Note that the above bound holds when $y_i^* \geq 1,\ \forall i \in [N]$. In the Appendix of [21], we show that $\frac{\mathbb{E}(f(\widehat{y}_n))}{f(y_n^*)} \leq \frac{2^d}{d(e-1)}$ for a convex polynomial of degree d. Combining this the result for $y_i^* \in [0, 1)$, we get a simplified bound of $\max(\frac{T}{T-1}, \frac{2^d}{d(e-1)})$. Please see the full proof in the Appendix of [21]. ■

Observe that Theorem 4 gives an upper bound, γ, which is a function of only T and d. Figure 3 shows that the bound is close to 2 for convex polynomials of degree 2; for convex polynomials of small degree it increases approximately linearly. The values do not vary significantly across α. Note that our bound is very good (a factor of 2 or 3) for practical cases when α is small and the storage function is not very convex (has a degree at most $d = 2$ or 3) [13]. Further note that the bound evaluated in simulations is close to 1 for all the cases presented in Figure 3 illustrating that our rounding algorithm produces an integral solution with cost very close to the cost of the optimal solution in expectation. This is because in the experiments, the bound tends to be close to $\frac{\sum_{n=1}^{N} \mathbb{E}(f(\widehat{y}_n))}{\sum_{n=1}^{N} f(y_n^*)}$ (see

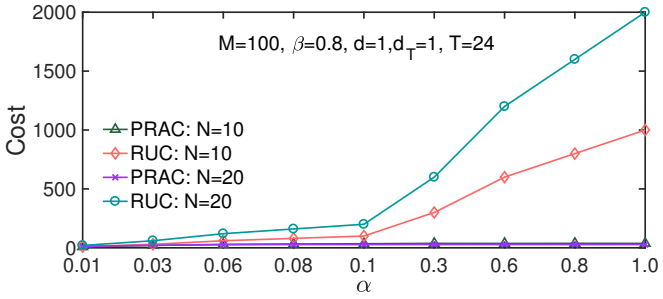

Fig. 4: Retention aware caching (PRAC) versus retention unaware caching (RUC) with varying α and N.

the proof of Theorem 4), which for well-spread values of y^* and large values of T tends to be close to 1.

C. Cache capacitated prefetching with optimal retentions

Unlike the linear storage case, where the retention times were either 0 or T, with the convex storage cost function, the optimal retention times may take values in the set $\{0, 1, \ldots, T\}$. However, we continue with (the same) Algorithm 1 as proposed in Section III-B for filling the caches respecting their capacity constraints. See further results from numerical evaluations in Section V-B.

V. Numerical Results

In this section, we conduct numerical evaluations on a synthetically generated data trace. We first construct a proactive multicast-enabled retention unaware caching policy motivated by MACP in [16] to compare its performance with PRAC (see Section V-A). We then study the impact of cache size on PRAC for different kinds of file arrivals (see Section V-B). Finally, we carry out a set of evaluations to study the impact of the various parameters used (see Section V-C).

We now explain the file request arrival process used. We assume that the inter-arrival times of file requests are exponentially distributed with rate parameter λ'_{mn} for file m at node n generated uniformly at random in $[0, 1]$ such that $\sum_{n\in[N]} \lambda'_{mn} = 1$. The arrival process across nodes (for a file) conform to an independent and homogeneous Poisson process. The arrivals are modulated using a ZipF-β popularity law under which the modulated process rate becomes $\lambda_{mn} = \lambda'_{mn} \frac{1}{R^\beta}$ where R is the rank of file m at node n obtained by sorting λ'_{mn} in descending order for each m. Moreover, the probability of a node n getting at least one request for file m is given by $p_{mn} = 1 - e^{-\lambda_{mn} d_T}$ (normalized such that $\sum_{n\in[N]} p_{mn} = 1, \forall m \in [M]$). The parameters used in our evaluations are $M = 100, N = 14, d_T = 1, T = 24, \beta = 0.8$ unless otherwise mentioned.

A. Effect of retention awareness on cost

We compare PRAC with a proactive multicast-enabled Retention Unaware Caching policy, say RUC, motivated by Multicast Aware Caching Policy (MACP) in [16]. RUC employs the same network model as PRAC except that (a) the retention times either correspond to storing a file for T slots or not storing it at all; (b) the optimal retention times minimize the download cost only (and not the storage cost) (hence retention unaware). The total cost incurred with RUC will be the sum of the storage and download costs obtained as a function of retentions determined through (a) and (b).

Figure 4 compares PRAC with RUC when the storage cost is a linear in retention times. We observe that for small values of α (up to 0.1) the cost of RUC is slightly more than that of PRAC, however, as α increases (from 0.1 to 1) RUC costs significantly higher, and more so for a higher N or d. This advocates the need for retention awareness in caching policies.

B. Effect of varying the cache size

We consider two Algorithms – Fill-Cache (Algorithm 1) and popularity-based caching – to populate caches given the optimal retention times obtained for uncapacitated cache. We calculate multicast cost with two settings: (1) uniform file arrivals, (2) Poisson arrivals with ZipF-0.8. The uncapacitated costs for the multicast-uniform and multicast with ZipF-0.8 is referred as MU and MZ, respectively. $\mathrm{MU} - (\mathcal{I})$ or $\mathrm{MZ} - (\mathcal{I})$ refers to the cache capacitated cost on MU (or MZ) when algorithm $\mathcal{I}$ (FillCache or popularity-based) is used for populating the caches. We vary the cache size with respect to number of files, i.e. B/M from 0 to 1 and obtain the costs as shown in Figure 5. We observe that the pairs $\{\mathrm{MU}, \mathrm{MZ}\}$, $\{\mathrm{MU} - \mathrm{Popularity}, \mathrm{MU} - \mathrm{FillCache}\}$ and $\{\mathrm{MZ} - \mathrm{Popularity}, \mathrm{MZ} - \mathrm{FillCache}\}$ equal each other in Figure 5 (a) when the storage cost is linear. This is mainly due to the binary retention times with linear storage, suggesting that locally populating caches based on optimal retentions is as good as FillCache for linear storage function. Next we plot the cost variations with convex storage functions in Figure 5 (b)) and observe that MU is significantly higher than MZ. We also observe that for uniform traffic ($\beta = 0$) FillCache significantly outperforms popularity-based caching, and the difference in their costs decreases as β increases. We note that at $\beta = 0.8$ the two algorithms perform almost the same, with FillCache only marginally lower than popularity-based caching.

C. Effect of varying the slot size relative to the frame size

We vary the slot size d_T and the number of slots per frame (T) for a day long prediction window as shown in Table I. As we go from ID-a through ID-j the slot interval increases; this could capture various types of requests, for example, a new Facebook page is requested every thirty seconds from a user but a movie is requested only once in two hours.

We observe that the cache occupancy (i.e. the number of files stored in each node) decreases as d_T increases. This can be explained as follows. Since $p_{mn} = 1 - e^{-\lambda_{mn} d_T}$, for a fixed λ_{mn} as d_T increases, p_{mn} will decrease. Thus, for $d_T = 4$ hours it is only reasonable to cache all the files because the probability of getting a request for any of them is high and vice versa compared to that with $d_T = 2$ minutes.

Now, to compare the impact of other parameters, we also consider a unicast transmission model where the server indi-

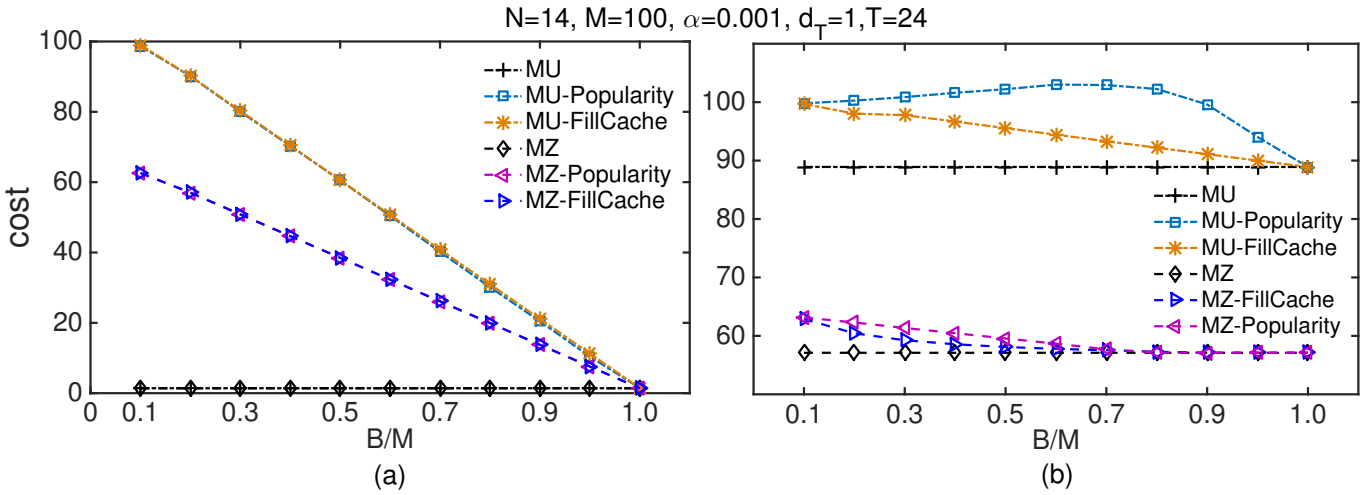

Fig. 5: Impact of varying B/M on the cost when caches are filled on the basis of file popularity versus when the caches are filled as per the Algorithm Fill-cache when (a) the storage cost is linear, (b) storage cost is convex.

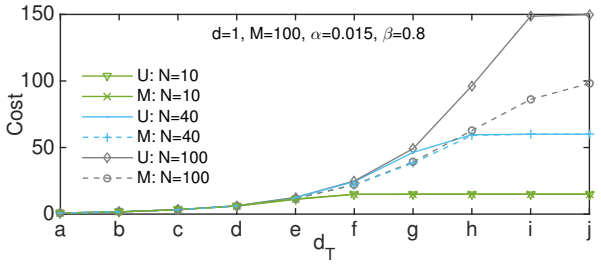

Fig. 6: Impact of increasing d_T with varying N.

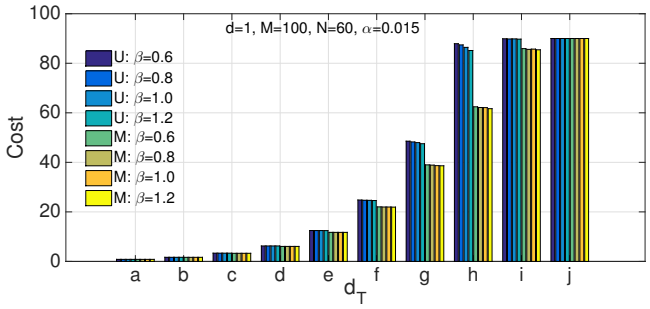

Fig. 7: Impact of increasing d_T with varying β.

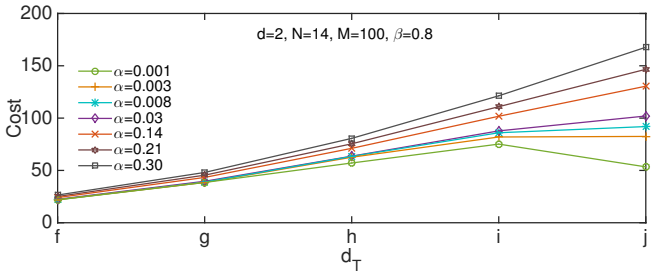

Fig. 8: Impact of increasing d_T with varying α.

vidually transmits all the requests from nodes that do not have the file cached at a given time t. Thus the unicast cost of file m, $cost_U$, for a linear storage function becomes:

$$cost_U = \sum_{i=1}^{N} \frac{\alpha}{T} y_i + \sum_{t=1}^{T} \sum_{n:y_{mn}<t} p_{mn}.$$

1) Effect of increasing N: We vary N and plot the cost incurred with linear storage costs in Figure 6. Figure shows that as d_T increases, the cost increases for both unicast as well as multicast transmissions. We observe that multicast cost is upper bounded by the unicast cost and the disparity between the two costs grows with N. This is reasonable since a single multicast can satisfy requests from many caches (thus incurring less cost) with N large. Similar conclusions were observed with convex storage costs.

TABLE I: Values of slot size d_T (in hours) and time frame T used for a day (note $T \times d_T = 24$ hours).

ID	a	b	c	d	e	f	g	h	i	j
T	2880	1440	720	284	192	96	48	24	12	6
d_T	$\frac{1}{120}$	$\frac{1}{60}$	$\frac{1}{30}$	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{2}$	1	2	4

2) Effect of increasing β: We vary the ZipF coefficient β for the parameters in Table I and plot the results in Figure 7. We note that the costs decrease with increasing β, which is reasonable as with a higher β the file requests are more skewed and thus can be served by caching fewer files. Similar results were observed with convex storage costs.

As multicast always outperformed unicast in the evaluations so far, in what follows, we will exclude unicast simulations.

3) Effect of increasing α and d: We now consider convex storage costs with parameters in Table I to study the impact of cost parameter α (see Figure 8) and degree d (Figure 9).

We observe that from ID-a to ID-f (in Table I) the various α result in the same cost (and thus not shown in Figure 8). From ID-g to ID-i cost increases with increasing α; at ID-j

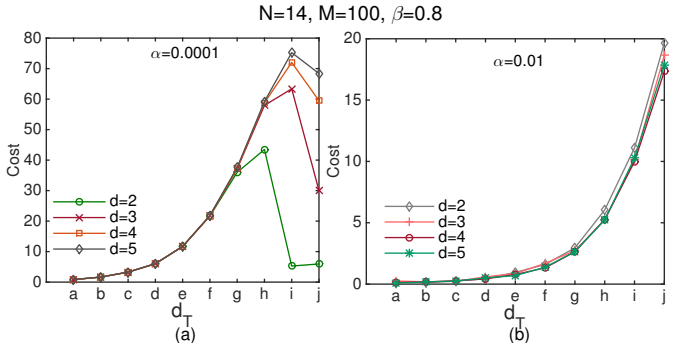

Fig. 9: Impact of increasing d_T with varying d.

cost decreases for $\alpha = 0.001$ while increases for others.

We observe a similar behavior with varying d. We observe that a higher degree results in a higher cost (Figure 9 (b)), however, for very small α the cost first increases then decreases (see Figure 9 (a)); this cost inversion occurs at increasing IDs as d increases for such an α.

The observations with increasing α and d can be described as follows. When $\alpha = 0.0001$, $d = 2$, all the retentions are between 0 and 1 from ID-a to ID-h (in Table I) and after that at least one retention variable is greater than 1. Recall that a retention value between 0 and 1 is rounded to 0 implying that the file is not stored and thus incurs a download cost upon request; on the other hand, with a retention variable greater than 1 the download cost reduces as more files are stored, but at the cost of a small storage cost as α is small. Thus the cost transition at ID-h for $d = 2$ is due to the change in the structure of optimal retention times. On the contrary, with a higher α, for all d, the retentions times are between 0 and 1 for all IDs hence no transition.

VI. Conclusion

We considered the problem of proactive retention aware caching (PRAC) motivated by the applications where storage cost is critical to the performance of the cache network. We formulated PRAC for a hierarchical network, with the objective to minimize the total cost subject to the node(s) capacity constraint(s). We first proved that PRAC is NP-Hard; and then analyzed it for both linear and convex storage costs. We showed that PRAC admits efficient polynomial time algorithms for large caches when the storage cost is linear in data retentions. When cost is a convex function, we derived bounds on the solution, that proved that our solution is close to the optimal. Our numerical evaluations demonstrated that PRAC outperforms caching policies that are retention unaware for parameters of interest.

One of the future directions we are currently working on is to consider the further generalization that a user can request files from not one but several caches on a multi-hop network. In this case, the decision where whether a node should store a file not only depends on storage cost but also on how content is routed through the network. The analysis under a continuous time model is another open direction.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 1422153. The authors thank Koushik Kar, Elliot Anshelevich, Onkar Bhardwaj and Siva Theja Maguluri for their insightful comments.

References

[1] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center." in *USENIX NSDI*, 2011.

[2] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *ACM SIGCOMM Computer Communication Review*, 2009.

[3] J. Tadrous and A. Eryilmaz, "On Optimal Proactive Caching for Mobile Networks With Demand Uncertainties," *IEEE/ACM Transactions on Networking*, 2016.

[4] V. A. Siris, X. Vasilakos, and G. C. Polyzos, "Efficient Proactive Caching for Supporting Seamless Mobility," *CoRR*, 2014.

[5] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "FemtoCaching: Wireless Content Delivery Through Distributed Caching Helpers," *IEEE Transactions on Information Theory*, 2013.

[6] K. Hamidouche, W. Saad, and M. Debbah, "Many-to-many Matching Games for Proactive Social-caching in Wireless Small Cell Networks," in *IEEE WiOpt*, 2014.

[7] H. Zhang, G. Jiang, K. Yoshihira, and H. Chen, "Proactive Workload Management in Hybrid Cloud Computing," *IEEE Transactions on Network and Service Management*, 2014.

[8] S. Dernbach, N. Taft, J. Kurose, U. Weinsberg, C. Diot, and A. Ashkan, "Cache Content-Selection Policies for Streaming Video Services," *IEEE INFOCOM*, 2016.

[9] M. Leconte, G. Paschos, L. Gkatzikis, M. Draief, S. Vassilaras, and S. Chouvardas, "Placing Dynamic Content in Caches with Small Population," *CoRR*, 2016.

[10] S. Ioannidis and E. Yeh, "Adaptive Caching Networks with Optimality Guarantees," in *ACM SIGMETRICS*, 2016.

[11] S. Shukla and A. A. Abouzeid, "On Designing Optimal Memory Damage Aware Caching Policies for Content-Centric Networks," in *IEEE WiOpt*, 2016.

[12] S. Shukla and A. Abouzeid, "Optimal Device Aware Caching," *IEEE Transactions on Mobile Computing*, 2016.

[13] B. Schroeder, R. Lagisetty, and A. Merchant, "Flash Reliability in Production: The Expected and the Unexpected," in *USENIX, FAST*, 2016.

[14] I. Narayanan, D. Wang, M. Jeon, B. Sharma, L. Caulfield, A. Sivasubramaniam, B. Cutler, J. Liu, B. Khessib, and K. Vaid, "SSD Failures in Datacenters: What, When and Why?" in *ACM SIGMETRICS*, 2016.

[15] U. Niesen and M. A. Maddah-Ali, "Coded Caching for Delay-Sensitive Content," in *IEEE ICC*, 2015.

[16] K. Poularakis, G. Iosifidis, V. Sourlas, and L. Tassiulas, "Exploiting Caching and Multicast for 5G Wireless Networks," *IEEE Transactions on Wireless Communications*, 2016.

[17] N. Abedini and S. Shakkottai, "Content Caching and Scheduling in Wireless Networks With Elastic and Inelastic Traffic," *IEEE/ACM Transactions on Networking*, 2014.

[18] Y. Cui and I. Stojmenovic, "Wireless Datacenter Networks," *Encyclopedia of Cloud Computing*, 2016.

[19] N. C. Fofack, M. Dehghan, D. Towsley, M. Badov, and D. L. Goeckel, "On the Performance of General Cache Networks," in *ACM VALUETOOLS*, 2014.

[20] M. Dehghan, A. Seetharam, B. Jiang, T. He, T. Salonidis, J. Kurose, D. Towsley, and R. Sitaraman, "On the Complexity of Optimal Routing and Content Caching in Heterogeneous Networks," in *IEEE INFOCOM*, 2015.

[21] Technical-Report, "Proactive Retention Aware Caching," https://www.dropbox.com/s/juujftdrza57c45/Full-version.pdf?dl=0, 2016.

[22] P. Blasco and D. Gündüz, "Learning-based Optimization of Cache Content in a Small Cell Base Station," in *IEEE ICC*, 2014.

[23] A. Gharaibeh, A. Khreishah, B. Ji, and M. Ayyash, "A Provably Efficient Online Collaborative Caching Algorithm for Multicell-Coordinated Systems," *CoRR*, 2015.