Access Strategies for Network Caching

Itamar Cohen*, Gil Einziger*, Roy Friedman†, and Gabriel Scalosub*
*Ben-Gurion University of the Negev, Beer Sheva, Israel
†Technion, Haifa, Israel
Email: itamarq@post.bgu.ac.il, gilein@bgu.ac.il, roy@cs.technion.ac.il, sgabriel@bgu.ac.il

***Abstract*—Having multiple data stores that can potentially serve content is common in modern networked applications. Data stores often publish approximate summaries of their content to enable effective utilization. Since these summaries are not entirely accurate, forming an efficient access strategy to multiple data stores becomes a complex risk management problem.**

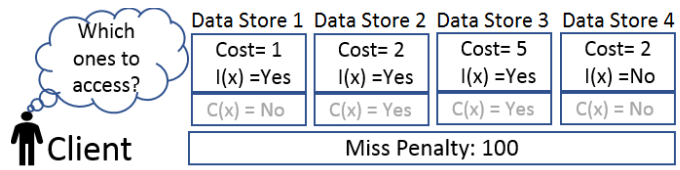
This paper formally models this problem, and introduces practical algorithms with guaranteed approximation ratios, and in particular we show that our algorithms are optimal in a variety of settings. We also perform an extensive simulation study based on real data, and show that our algorithms are more robust than existing heuristics. That is, they exhibit near optimal performance in various settings whereas the efficiency of existing approaches depends upon system parameters that may change over time, or be otherwise unknown.

I. Introduction

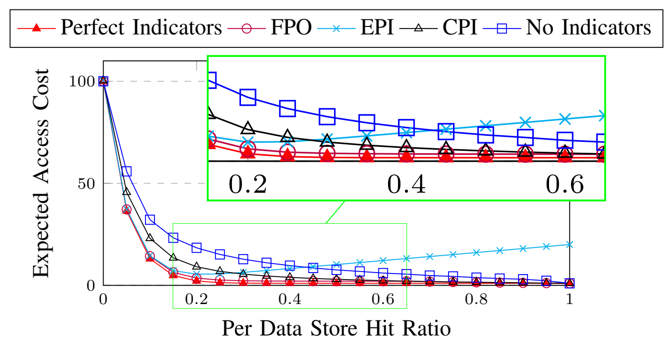
Having access to multiple network connected *data stores* is common in modern network settings such as 5G in-network caching [1], [2], content delivery networks (CDN) [3], [4], information centric networking [5], [6], wide-area networks [7], as well as in any multi data center Internet company. Data stores can be cache enabled network devices, memory layers within a server, virtual machines, physical hosts, remote data centers or any combination of the above examples. In such settings, each data store acts as a network cache by holding a potentially overlapping fraction of the entire data that may be accessed by applications and services hosted in the network.

Accessing a data store incurs a certain cost in terms of latency, bandwidth, and energy. Hence, smart utilization of data stores may reduce the operational costs of such systems and improve their users' experience. Naturally, knowing which item is stored in each data store at any given moment is a key enabler for efficient utilization, but maintaining such knowledge may not be feasible. Instead, it is more practical to occasionally exchange space efficient *indicators* for the content of the data stores [7]. Bloom filters [8] are a common implementation for such indicators, but many other space-efficient approximate membership representations can also be used [3], [9]–[14].

The shortcoming of relying on such indicators is that they may exhibit *false positives*, meaning that they may indicate that a given item is held by a certain data store while it is actually not there. Indeed, the work of [13] formally showed that naively relying on indicators for accessing even a single data store may do more harm than good. In this work, we are interested in the general case of accessing multiple data stores. The difference is that we require an access strategy



(a) The client is looking for item x and needs to select which data stores to access. Data stores provide an indication ($I(x)$) if they store x. The grayed content ($C(x)$) indicates if they actually store x. Notice the false positive in Data store 1. The client pays the cost for the selected data stores, and in case of a failure to find x in any of the accessed data stores, it incurs a miss penalty.



(b) An example of the average access cost of different strategies (lower is better), when varying the cache hit ratio. The number of data stores here is 20, the access cost to each of them is 1 while the miss penalty is 100 and the false positive ratio is 0.02.

Fig. 1. Motivation for the access strategy problem.

that selects a *subset* of the data stores to access per request. Existing strategies for this problem include: (i) the *Cheapest Positive Indication* (CPI) [10], [15] strategy that accesses the cheapest data store with a positive indication for the requested item, and (ii) the *Every Positive Indication* (EPI) [7] strategy that accesses every data store with a positive indication. The access is considered successful if the item is stored in one of the accessed data stores, and incurs no further cost. Otherwise, we pay a *miss penalty* for retrieving the requested item, e.g., due to the need to fetch it from an external remote site.

In the example of Figure 1a, CPI accesses only data store 1, which is the cheapest with a positive indication (captured by $I(x) = \text{Yes}$), and incurs a cost of 1 for this. However, since x is not in data store 1 (captured by $C(x) = \text{No}$), this indication is a false positive, and an additional miss penalty of 100 is incurred for the request, for a total cost of 101 imposed on CPI. Alternatively, the EPI policy accesses every data store with a positive indication (data stores 1, 2, and 3). This implies an access cost of $1+2+5=8$. In this case, no additional miss penalty is incurred, since item x is indeed available in one of the accessed data stores, e.g., in data store 2. One can also consider an ideal strategy equipped with a *perfect indicator* with no false positives. Such an ideal strategy would require a cost of merely 2 incurred for accessing data store 2 alone.

Figure 1b provides a numerical example motivating this work (see Section IV for the exact settings). The figure illustrates the expected access cost for varying strategies with a false positive ratio of $\mathrm{FP}=0.02$. The strategies are compared to the performance of two baseline scenarios. The No Indicators (blue) line illustrates the best that can be obtained without indicators (which can be viewed as using indicators with $\mathrm{FP}=1$, or equivalently, using indicators that always return 'Yes'). In contrast, the Perfect Indicators (red) line corresponds to having no false positives ($\mathrm{FP}=0$) in any of the indicators.

The area between the plots describing the performance of the two baseline scenarios (blue and red) exhibits the potential gains of employing indicator based access policies. Specifically, we observe that EPI is near optimal when the per data store hit ratio is low but becomes highly inefficient when it is high. In fact, even the No Indicators approach outperforms EPI once the hit ratio is above a certain threshold (in our plot, this occurs at a hit-ratio of around 0.45). In contrast, CPI is near optimal when the hit ratio is very high but performs poorly when it is low. Between these two extremes, there is a gap where both strategies are inefficient, as highlighted in the magnified area of Figure 1b. Our proposed strategies, described in Sections IV-V, aim at providing near-optimal performance, independent of the actual hit ratio. In particular, the performance of our false-positive-aware optimal policy, FPO, depicted by the pink line, comes extremely close to the Perfect Indicators (red) line despite relying on indicators whose $\mathrm{FP}=0.02$.

Our Contribution: As mentioned, despite the popularity of indicators, the problem of efficiently working with indicators and of forming a successful access strategy has remained unexplored. Our work formally models this problem in very general and heterogeneous settings with varying access costs, per data store hit ratios and miss penalties. We show that previously suggested strategies are too simplistic and implicitly rely on specific assumptions about the workload, or the underlying system. Thus, in general, an access strategy that works well in one scenario may be inefficient for another.

Our work suggests and analyzes two practical approximation algorithms that work in polynomial time. Through an extensive evaluation with varying system parameters, we show that our algorithms are more stable than existing approaches. That is, they outperform or achieve very similar access costs to the best competitor for any tested system configuration.

II. Related work

A. Approximate Set Membership

Approximate set membership is about encoding a set of items, such as the content of a data store, in a space efficient manner. Intuitively, an accurate representation requires storing all identifiers which may be prohibitively expensive. Alternatively, space can be conserved by allowing a small number of false positives. Bloom filters [8] offer space-efficient encoding but do not support the removal of items. Other works [7], [11], [12], [14], [16] improve on them in various aspects, such as support for removals [12], [17], [18], a more efficient access pattern [11], [14], and lower transmission overheads [19].

TABLE I
LIST OF SYMBOLS

Symbol	Meaning
N	All data stores
n	Number of data stores, $n=\|N\|$
N_x	Data stores with positive indications for requested datum x
n_x	Number of positive indications for requested datum x ($\|N_x\|$).
S_j	The set of data items in data store j
p_j^h	Hit ratio of data store j
$I_j(x)$	Indication of data store j for datum x.
q_j	Probability of positive indication by I_j: $\Pr(I_j(x)=1)$
FP_j	False positive ratio for I_j: $\mathrm{FP}_j=\Pr(I_j(x)=1 \| x\notin S_j)$
ρ_j	Misindication ratio for a data store j
ρ_D	Misindication ratio for a set of data stores D
c_i	Access cost for data store i.
c_D	Total access cost (sum of costs of all data stores in set D)
ϕ	Cost function: $\phi(D)=\sum_{i\in D} c_i+\beta\prod_{i\in D}\rho_i$
β	Miss penalty
M	$M=\min\left\{\sum_{j\in N_x} c_j,\beta\right\}$.
$H_k(L_k)$	Access cost for the k highest (lowest) data stores in N_x.

B. Applicability Examples

Bloom filter variants are extensively used in multiple domains [9], [10]. Most notable is their use in front of a cache or a slow memory hierarchy. Such usage leverages that Bloom filters do not exhibit false negatives. Thus, there is no need to access the data store on a negative indication.

The work of [7] suggests an architecture for distributed caching on wide area networks. In this solution, caches share an approximation of their content. Clients use this information to only contact the caches with positive indications (EPI). A similar architecture is also considered in [10], [15]. There, clients access the cheapest cache with a positive indication (CPI). Let us note that the impact of the access strategy and its optimization is overlooked in previous works.

The work of [13] considers the special case of a single data store, equipped with a Standard Bloom Filter [8] or a Counting Bloom Filter [20]. They identify cases where following a positive indication may increase the overall cost. Thus, they suggest that in those cases the data store should be ignored, regardless of its indicator value. We, on the other hand, address the more general problem, which involves any number of data stores, equipped with any kind of indicators.

III. System Model and Preliminaries

This section formally defines our system model and notations. For ease of reference, our notation is summarized in Table I. We consider a set N of n *data stores*, containing possibly overlapping subsets of items. We denote by S_j the set of items stored at data store j. Given a sequence of requests for items σ (with possible repetitions), the *hit ratio* of a data store j is the fraction of requests in σ that were available in data store j (when requested). Our work assumes that past hit ratio is a good indication for the near future [21], [22]. We denote by p_j^h the hit ratio of data store j, i.e., the probability that the next accessed item x is stored in S_j.

Each data store j maintains an *indicator* I_j, which approximates S_j; given an item x, $I_j(x) = 1$ indicates that x is likely to be in S_j while $I_j(x) = 0$ indicates that it is surely not in S_j. These are referred to as a *positive indication* and a *negative indication*, respectively. Our model assumes indicators that may exhibit only one-sided errors, i.e., they never err when providing a negative indication[1]. In practice, most implementations satisfy this assumption [7], [11], [12], [14]. The *false positive ratio* of I_j is defined by $\mathrm{FP}_j = \Pr(I_j(x) = 1 | x \notin S_j)$. It captures the probability that given a uniformly random item that is not in S_j, the indicator would mistakenly indicate that it is in S_j. For every data store j, given its indicator I_j, we let $\rho_j \in [0, 1]$ denote its *misindication ratio* $\rho_j = \Pr(x \notin S_j | I_j(x) = 1)$, where x is uniformly selected from the entire domain S_j.

Given an item x within sequence σ, a query for x triggers a *data access* which consists of selecting a subset of the data stores D and accessing this subset in parallel. The data access is considered successful, or a *hit*, if the item x is found in at least one of the data stores being accessed and is considered unsuccessful, or a *miss*, otherwise. Since by our assumption all indicators might have a one-sided error, we focus our attention only on subsets of data stores which all provide a positive indication. Given such a subset of the data stores D all providing a positive indication, we denote by ρ_D the misindication ratio of D, i.e., the probability that an item is not available in any of the data stores in D, in spite of their positive indications. Note, that if $D = \emptyset$, then $\rho_D = 1$. We make no assumptions on the sharing policy among the data stores. Yet, in the analysis sections we assume that the misindication ratios are mutually independent, that is, $\rho_D = \prod_{j \in D} \rho_j$. Under this assumption our analysis provides a baseline for understanding the performance of such systems.

Each data store has some predefined *access cost*, c_j, which is incurred whenever data store j is being accessed. These access costs induce the overall cost for accessing a set D of data stores, defined by $c_D = \sum_{j \in D} c_j$. We assume without loss of generality that $\min_j c_j = 1$. In case the data access results in a miss, it incurs a *miss penalty* of β, for some $\beta \geq 1$. For a subset of data stores D, which all provide a positive indication, we define its (expected) *miss cost* by $\beta \cdot \rho_D$.

For any query item x, let $N_x \subseteq N$ denote the subset of data stores with a positive indication, i.e., $N_x = \{j \in N | I_j(x) = 1\}$, and denote the size of this set by $n_x = |N_x|$. The expected *cost* of accessing any $D \subseteq N_x$ is defined to be the sum of its access cost and its expected miss cost, i.e.,

$$\phi(D) = c_D + \beta \cdot \rho_D. \qquad (1)$$

When misindication ratios are mutually independent we have

$$\phi(D) = c_D + \beta \cdot \rho_D = \sum\nolimits_{j \in D} c_j + \beta \prod\nolimits_{j \in D} \rho_j. \qquad (2)$$

[1] This means having no false negatives, i.e., $\Pr(I_j(x) = 0 | x \in S_j) = 0$.

The *Data Store Selection (DSS) problem* is to find a subset of data stores $D \subseteq N_x$ that minimizes the expected cost $\phi(D)$.

We denote by q_j the probability that indicator j positively replies to a query for an item x. This happens when either $x \in S_j$; or $x \notin S_j$, and a false positive occurs. Therefore,

$$q_j = \Pr(I_j(x) = 1) = p_j^h + (1 - p_j^h)\,\mathrm{FP}_j\,. \qquad (3)$$

Using Bayes' theorem and Eq. 3, the misindication ratio ρ_j is

$$\begin{aligned}\rho_j &\equiv \Pr(x \notin S_j | I_j(x) = 1)\\ &= \mathrm{FP}_j(1 - p_j^h)/[p_j^h + (1 - p_j^h)\,\mathrm{FP}_j]. \qquad (4)\end{aligned}$$

IV. The Fully Homogeneous Case

To gain some insight about the challenges in developing an access strategy, we start with a simplified fully-homogeneous case. In this setting, the cost of accessing each data store is the same ($c = 1$). The per data store hit ratios and false positive ratios are uniform, i.e., for each j, $p_j^h = p^h$ and $\mathrm{FP}_j = \mathrm{FP}$, for some constants $p^h, \mathrm{FP} \in [0, 1]$. Consequently, the per data store misindication ratios, captured by Eq. 4, are also uniform, i.e., for each j, $\rho_j = \rho$ for some constant $\rho \in [0, 1]$. Recall that our objective is to pick a subset of data stores with positive indications, $D \subseteq N_x$, so as to minimize the overall expected cost of a query, $\phi(D) = \sum_{j \in D} c_j + \beta \prod_{j \in D} \rho_j$. In the fully-homogeneous case considered here, the expected cost reduces to $\phi(D) = |D| + \beta \rho^{|D|}$, which merely depends on the *size* of the chosen set D of data stores to be accessed. The task of choosing which subset of data stores to access is reduced to deciding on the number $0 \leq k \leq n_x$ of data stores one should access. For any such potential number k, we denote the expected cost of accessing k data stores by

$$\tilde{\phi}(k) = k + \beta \rho^k, \qquad (5)$$

and focus our attention on studying the cost $\tilde{\phi}(\cdot)$ incurred by different data store selection schemes.

The size of the selected subset is clearly upper-bounded by the number of positive indications, n_x. So we start by calculating the distribution of n_x. Ideally, one can interpret each positive indication as a result of an independent Bernoulli trial with success probability q. By Eq. 3, $q = p^h + (1 - p^h)\,\mathrm{FP}$. Hence, n_x is binomially distributed such that

$$\Pr(n_x = k) = \binom{n}{k} q^k (1 - q)^{n-k}. \qquad (6)$$

Using equations 5 and 6 we now derive the expected costs of several selection schemes, where we let D_X denote the set of data stores selected by selection scheme X.

The EPI policy accesses all the data stores with positive indications, and therefore its expected overall cost is

$$\begin{aligned}\phi(D_{EPI}) &= \sum_{k=0}^{n} \left[\Pr(n_x = k) \cdot \tilde{\phi}(k)\right]\\ &= \sum_{k=0}^{n} \left[\Pr(n_x = k) \cdot k\right] + \beta \cdot \sum_{k=0}^{n} \left[\Pr(n_x = k) \cdot \rho^k\right]\\ &= E[n_x] + \beta \cdot \mathrm{PGF}_{n_x}(\rho) \qquad (7)\\ &= n \cdot q + \beta \left(1 - q + q \cdot \rho\right)^n,\end{aligned}$$

where $\mathrm{PGF}_X(t)$ denotes the probability generating function for random variable X at point t.

CPI accesses either a single data store with a positive indication, if one exists, or no data store if there are no positive indications. The expected overall cost of CPI is therefore

$$\begin{aligned}\phi(D_{CPI}) &= \Pr(n_x = 0) \cdot \tilde{\phi}(0) + \Pr(n_x > 0) \cdot \tilde{\phi}(1) \\ &= (1-q)^n \beta + [1-(1-q)^n](1+\beta\rho).\end{aligned} \tag{8}$$

We now turn to analyze the false-positive-aware optimal policy, FPO, which minimizes the expected overall cost, given the false positive ratio, FP. In the fully homogeneous case, this translates to finding $\arg\min_k \tilde{\phi}(k)$. Consider $\tilde{\phi}(y)$ defined in Eq. 5 as a function defined over the reals. This function is convex since its second derivatives is non-negative, and it obtains its minimum at $y^* = -\ln(-\beta\ln(\rho))/\ln(\rho)$ for $0 < \rho < 1$. In practice, the number of data stores accessed must be an integer between 0 and n_x. The optimal number $m^*(k)$ of data stores to access given that there are k positive indications satisfies $m^*(k) \in \{0, k, \lfloor y^* \rfloor, \lceil y^* \rceil\}$, where $\lfloor y^* \rfloor$ and $\lceil y^* \rceil$ should be considered only if $y^* \in [0, k]$. Hence, The expected overall cost of FPO is

$$\phi(D_{FPO}) = \sum_{k=0}^{n} \left[\binom{n}{k} q^k (1-q)^{n-k} m^*(k) \right]. \tag{9}$$

Having studied the overall cost of the above policies, we may revisit Figure 1b. The expected costs of each of the policies are presented as a function of p^h, using Equations 7-9. In particular, in the special case where $\mathrm{FP} = 0$, the expected overall costs of CPI, FPO and the perfect indicators benchmark are identical. This fits our intuition that when there are no false indications, the optimal policy is to access a single data store among those with positive indications if such a data store exists. At the other extreme, we have the case where $\mathrm{FP} = 1$, in which we always have $n_x = n$, i.e., all the indicators are positive. This extreme case renders the indicators useless and is thus equivalent to not having indicators at all. In particular, note that depending on the values of n and β, EPI might end up being worse than not having any indicators at all.

V. The Heterogeneous Case

In the previous section, we addressed the fully homogeneous case, in which minimizing our objective function $\phi(D)$ was made tractable due to the uniformity of the settings. In general, many systems are heterogeneous, making the minimization of $\phi(D)$ a much more challenging task.

Recall that our goal is to select a subset $D \subseteq N_x$ of data stores with positive indications minimizing the expected cost

$$\phi(D) = c_D + \beta\rho_D = \sum\nolimits_{i \in D} c_j + \beta \prod\nolimits_{j \in D} \rho_j,$$

as defined in Eq. 2. This can be viewed as a combined bi-criteria optimization problem, of minimizing two objectives simultaneously: (i) c_D, which is monotone *non-decreasing* as we pick more data stores to include in D, and (ii) ρ_D, which is monotone *non-increasing* as we pick more data stores to include in D, where the latter objective is "regularized" by β.

In this section, we describe several algorithms for solving the DSS problem in fully heterogeneous settings and provide a rigorous analysis of their performance. In particular, we also study trade-offs between the time complexity and the performance guarantees of our proposed solutions.

Algorithm 1 $\mathrm{DS}_{\mathrm{Pot}}(N_x, c, \rho, \beta)$

```
ℓ_1, …, ℓ_{n_x} ← N_x in non-decreasing order of ρ_j
for k = 1, …, n_x do
    D_k ← {ℓ_1, …, ℓ_k}
end for
return D = argmin_k { P(k) = L_k + β ∏_{j=1}^{k} ρ_{ℓ_j} }
```

A. A Potential-based Algorithm

In the special case where the non-decreasing orderings of data stores by access costs and by misindication ratios are the same, a simple substitution argument shows that a greedy approach will yield an optimal solution D which consists of a prefix of this ordering.

In what follows we generalize the above observation and suggest an algorithm for the general case based on the special case described above. We denote by L_k and H_k the sum of the k smallest access costs of data stores in N_x and the k largest access costs of data stores in N_x, respectively. Our algorithm, $\mathrm{DS}_{\mathrm{Pot}}$, described in Algorithm 1, considers the data stores ordered in non-decreasing order of miss-ratio, $\ell_1, \ldots, \ell_{n_x}$, such that $\rho_{\ell_j} \le \rho_{\ell_{j+1}}$ for all $j = 1, \ldots, n_x - 1$. The algorithm iterates over all prefixes of indices in this order, and picks a subset of data stores corresponding to a prefix which minimizes the *potential function* $P(k) = L_k + \beta \prod_{j=1}^{k} \rho_{\ell_j}$.

We now turn to analyze the performance of our proposed algorithm $\mathrm{DS}_{\mathrm{Pot}}$. In particular, we show the following theorem:

Theorem 1. *Let D^* be an optimal set of data stores for the DSS problem, and let D be the solution found by* $\mathrm{DS}_{\mathrm{Pot}}$. *Then* $\phi(D) \le \frac{H_{|D|}}{L_{|D|}} \phi(D^*)$.

Proof. Let $k = |D|$. We therefore have

$$\begin{aligned}\phi(D) &= \sum\nolimits_{j=1}^{k} c_{\ell_j} + \beta \prod\nolimits_{j=1}^{k} \rho_{\ell_j} \le H_k + \beta \prod\nolimits_{j=1}^{k} \rho_{\ell_j} \\ &\le \frac{H_k}{L_k} \left(L_k + \beta \prod\nolimits_{j=1}^{k} \rho_{\ell_j} \right) = \frac{H_k}{L_k} P(k),\end{aligned} \tag{10}$$

where the penultimate inequality follows from the definitions of L_k and H_k, and the last equality follows from the definition of the potential function $P(k)$. Let $k^* = |D^*|$. Since data stores are ordered in non-decreasing order of misindication ratio, it follows that $\prod_{j=1}^{k^*} \rho_{\ell_j} \le \prod_{j \in D^*} \rho_j$, and by the definition of L_{k^*} as the sum of the k^* smallest access costs of data stores in N_x, it follows that

$$\begin{aligned}P(k^*) &= L_{k^*} + \beta \prod\nolimits_{j=1}^{k^*} \rho_{\ell_j} \\ &\le \sum\nolimits_{j \in D^*} c_j + \beta \prod\nolimits_{j \in D^*} \rho_j = \phi(D^*).\end{aligned} \tag{11}$$

Since D is chosen to be the set of data stores that minimizes $P(k)$, where k is the length of the prefix N_x considered in non-decreasing order of miss-ratio, we have $P(k) \le P(k^*)$. Combining this with Eqs. 10 and 11, the result follows. □

Since for every k we have $\frac{H_k}{L_k} \leq \max_j \{c_j\}$ and the running time of $\mathrm{DS}_{\mathrm{Pot}}$ is dominated by the time required to sort the data stores, we obtain the following corollary:

Corollary 2. $\mathrm{DS}_{\mathrm{Pot}}$ *is a* $(\max_j \{c_j\})$*-approximation algorithm, running in time* $O(n_x \log n_x)$.

In particular, Corollary 2 implies that for the case where all accesses costs are equal, $\mathrm{DS}_{\mathrm{Pot}}$ yields an optimal solution to the DSS problem.

B. A Knapsack-based Algorithmic Framework

In this section, we develop an alternative algorithm for the DSS problem and provide guarantees on its performance. We begin by recalling that the main difficulty in solving the DSS problem stems from the fact that our objective function is composed of a *linear* component (the access cost) and a *multiplicative* component (the miss cost). The algorithmic framework we propose in the sequel is based on carefully linearizing the multiplicative component, and defining a collection of *knapsack problems* for which their solution space contains a good approximate solution to the DSS problem.

We associate each data store j with its *log-hit weight*, defined by $w_j = -\log(\rho_j)$. We therefore have for every subset of data stores $D \subseteq N$, $-\log(\rho_D) = \sum_{j \in D} w_j$. Therefore, any set of data stores has a minimal miss cost if and only if it has a maximal log-hit weight. In what follows we define a collection of Knapsack problems, where the Knapsack problem is defined as follows: Given a budget B, and collection of items U, such that each item $j \in U$ has some profit π_j and cost γ_j, the goal is to find a subset of items $S \subseteq U$ such that $\sum_{j \in S} \gamma_j \leq B$ and $\sum_{j \in S} \pi_j$ is maximized. We refer to such an instance as the (B, U, π, γ)-Knapsack problem, and denote by $A_{\mathrm{Knap}}(B, U, \pi, \gamma)$ the set of items produced as output by an algorithm A_{Knap} for the Knapsack problem. The Knapsack problem is known to be NP-hard, but it can be solved exactly by dynamic programming in pseudo-polynomial time, and can be approximated to within a $(1 + \epsilon)$ factor in polynomial time by an FPTAS [23].

We now turn to define our collection of knapsack problems, to be used by our algorithm for solving the DSS problem. We recall that given a query x, $N_x \subseteq N$ denotes the subset of data stores for which their indicator is positive. In the following we let $M = \min \left\{ \sum_{j \in N_x} c_j, \beta \right\}$. Clearly, M is an upper bound on the access cost of any optimal solution for the DSS problem. For any $B \in \{0, 1, \ldots, M\}$, consider the (B, N_x, w, c)-Knapsack problem, i.e., the Knapsack problem with budget B over a collection of items N_x, such that each item $j \in N_x$ has profit w_j (the log-hit weight of data store j) and cost c_j (the access cost of data store j).

Our algorithm named $\mathrm{DS}_{\mathrm{PP}}$, formally defined in Algorithm 2, makes use of a $(1 + \epsilon)$-approximation algorithm A_{Knap} for the knapsack problem, for some $\epsilon \geq 0$. The complexity and performance guarantee depends upon the value of ϵ. $\mathrm{DS}_{\mathrm{PP}}$ essentially iterates over all possible values for the access cost, and solves the associated Knapsack problem using the algorithm A_{Knap} as a subroutine for each such value. $\mathrm{DS}_{\mathrm{PP}}$ then selects

Algorithm 2 $\mathrm{DS}_{\mathrm{PP}}(N_x, c, \rho, \beta,$ $(1+\epsilon)$-approximation algorithm A_{Knap} for Knapsack$)$

1: $w_j \leftarrow -\log(\rho_j)$ for all $j \in N_x$
2: **for** $B \in \left\{0, 1, \ldots, \min\left\{\sum_{j \in N_x} c_j, \beta\right\}\right\}$ **do**
3: $\triangleright$ c_j in this algorithm is assumed to be an integer
4: $D_B \leftarrow A_{\mathrm{Knap}}(B, N_x, w, c)$
5: **end for**
6: **return** $D = \arg\min_B \{\phi(D_B)\}$

the subset of data stores $D \subseteq N_x$ which minimizes $\phi(D)$ over all Knapsack solutions calculated by A_{Knap} in all iterations.

We first show that if A_{Knap} finds an *optimal* solution to the Knapsack problem in each iteration, then our algorithm finds an *optimal* solution to the DSS problem. In terms of running time, since the best exact algorithm for the Knapsack problem over n items with budget B runs in pseudo-polynomial time of $O(nB)$ [23], our algorithm also runs in pseudo-polynomial time. These properties are formalized in the following theorem:

Theorem 3. *When using the pseudo-polynomial algorithm* A_{Knap} *which finds an optimal solution to the Knapsack problem over* n *items with budget* B *in time* $O(nB)$*,* $\mathrm{DS}_{\mathrm{PP}}$ *is a pseudo-polynomial algorithm that finds an optimal solution to the DSS problem in time* $O(n_x M^2)$.

Proof. We first show that $\mathrm{DS}_{\mathrm{PP}}$, defined in Algorithm 2, finds an optimal solution to the DSS problem. Consider an optimal solution $D^* \subseteq N_x$ for the DSS problem, and let $B^* = c_{D^*}$. Since by optimality $B^* \leq M$, we are guaranteed that $\mathrm{DS}_{\mathrm{PP}}$ considers $B = B^*$ in one of the iterations of the for-loop in lines 2-5. Let D_B denote the solution of the knapsack problem being solved in that iteration, where the knapsack budget is B. Since algorithm A_{Knap} finds an optimal solution for the knapsack problem in this iteration

$$D_B = \arg \max_{D \subseteq N_x | c_D \leq B} \left\{ \sum\nolimits_{j \in D} w_j \right\}.$$

By the definition of w_j and the monotonicity of the log function, such a D_B also satisfies

$$D_B = \arg \min_{D \subseteq N_x | c_D \leq B} \{\rho_D\}. \tag{12}$$

Assume by contradiction that D_B is not optimal for the DSS problem, i.e., that $\phi(D_B) = c_{D_B} + \beta \rho_{D_B} > c_{D^*} + \beta \rho_{D^*} = \phi(D^*)$. Since $c_{D^*} = B^* = B \geq c_{D_B}$, it must follow that $\rho_{D_B} > \rho_{D^*}$, for $c_{D^*} \leq B$, which contradicts Eq. 12.

Running time: $\mathrm{DS}_{\mathrm{PP}}$ performs M iterations, where in each iteration it solves a knapsack problem using an algorithm which runs in $O(n_x M)$ time. It follows that the running time of $\mathrm{DS}_{\mathrm{PP}}$ in this case, is $O(n_x M^2)$, as required. $\square$

In many cases, the value of $M = \min \left\{ \sum_{j \in N_x} c_j, \beta \right\}$ is polynomially bounded by n_x. The following is an immediate corollary of Theorem 3 in such cases:

Corollary 4. *If* $M = \min \left\{ \sum_{j \in N_x} c_j, \beta \right\}$ *is polynomially bounded by* n_x*, then* $\mathrm{DS}_{\mathrm{PP}}$ *solves the DSS problem in polynomial time.*

We now turn to study the tradeoff between the running time of $\mathrm{DS}_{\mathrm{PP}}$ and its performance guarantee, when using a polynomial time approximation algorithm for Knapsack instead of the pseudo-polynomial time exact algorithm. We first show in Theorem 5 how the approximation guarantee of an algorithm for Knapsack translates to an approximation guarantee for the DSS problem, while still in pseudo-polynomial time.

Theorem 5. *If there exists some constant δ such that $\rho_j \leq \delta$ for all $j \in N_x$ and algorithm A_{Knap} is a $(1+\epsilon)$-polynomial time approximation algorithm for Knapsack running in time $O(f(n_x, \epsilon))$, then $\mathrm{DS}_{\mathrm{PP}}$ is a pseudo-polynomial algorithm that finds an $O(\beta^{\frac{\epsilon}{1+\epsilon}})$-approximate solution for the DSS problem in time $O(f(n_x, \epsilon) \cdot M)$.*

Proof. First, note that by its definition, the running time of $\mathrm{DS}_{\mathrm{PP}}$ is as required since it makes M iterations, and in every iteration solves an instance of Knapsack in time $O(f(n_x, \epsilon))$. It remains to bound the approximation ratio of $\mathrm{DS}_{\mathrm{PP}}$.

Consider an optimal solution $D^* \subseteq N_x$ to the DSS problem, and let $B^* = c_{D^*}$ and ℓ be an integer such that

$$2^{-(\ell+1)} \leq \rho_{D^*} \leq 2^{-\ell}. \tag{13}$$

By our assumption there exists some constant δ such that for all $j \in N_x$ we have $\rho_j \leq \delta$. We are therefore guaranteed to have $\ell = O(\log \beta)$, since for $\ell > \log_{1/\delta} \beta$ we have $\rho_{D^*}\beta < 1$, in which case the optimal solution would not benefit from accessing more data stores than it currently does. By the definition of the log-hit weight, we therefore have $\ell \leq \sum_{j \in D^*} w_j \leq \ell + 1$.

Consider the iteration of $\mathrm{DS}_{\mathrm{PP}}$ where $B = B^*$, and let D_B denote the solution obtained by algorithm A_{Knap} for solving the Knapsack problem in this iteration. Since A_{Knap} is a $(1+\epsilon)$-approximation algorithm we are guaranteed to have $\sum_{j \in D_B} w_j \geq \frac{1}{1+\epsilon} \sum_{j \in D^*} w_j$ since D^* is an optimal solution with an access cost of B^*, and therefore maximizes the objective function in the Knapsack problem being solved in this iteration. It follows that

$$\begin{aligned}\prod_{j \in D_B} \rho_j &\leq \prod_{j \in D^*} \rho_j^{\frac{1}{1+\epsilon}} \leq 2^{\frac{-\ell}{1+\epsilon}} \\ &= 2^{-\ell + \frac{\epsilon\ell}{1+\epsilon}} = 2^{-(\ell+1)+(1+\frac{\epsilon\ell}{1+\epsilon})} \\ &\leq 2^{1+\frac{\epsilon\ell}{1+\epsilon}} \prod_{j \in D^*} \rho_j \leq O\left(\beta^{\frac{\epsilon}{1+\epsilon}}\right) \prod_{j \in D^*} \rho_j,\end{aligned} \tag{14}$$

where the first inequality follows from our Knapsack approximation guarantee, the following two inequalities follow from Eq. 13, and the last inequality follows from the fact that $\ell = O(\log \beta)$. For $B = B^*$ we are guaranteed to have $\sum_{j \in D_B} c_j \leq B^*$. Hence,

$$\begin{aligned}\phi(D_B) &= \sum_{j \in D_B} c_j + \beta \prod_{j \in D_B} \rho_j \\ &\leq B^* + O\left(\beta^{\frac{\epsilon}{1+\epsilon}}\right)\left(\beta \prod_{j \in D^*} \rho_j\right) \\ &= \sum_{j \in D^*} c_j + O\left(\beta^{\frac{\epsilon}{1+\epsilon}}\right)\left(\beta \prod_{j \in D^*} \rho_j\right) \\ &\leq O\left(\beta^{\frac{\epsilon}{1+\epsilon}}\right)\left(\sum_{j \in D^*} c_j + \beta \prod_{j \in D^*} \rho_j\right) \\ &= O\left(\beta^{\frac{\epsilon}{1+\epsilon}}\right)\phi(D^*)\end{aligned} \tag{15}$$

which completes the proof. □

Algorithm 3 $\mathrm{DS}_{\mathrm{Knap}}(N_x, c, \rho, \beta)$

1: $w_j \leftarrow -\log(\rho_j)$ for all $j \in N_x$
2: **for** $u \in \{c_j | j \in N_x\}$ **do**
3: $\quad N_x^u \leftarrow \{j \in N_x | c_j \leq u\}$, let $n_x^u = |N_x^u|$
4: $\quad k_1, \ldots, k_{n_x^u} \leftarrow N_x^u$ in non-increasing order of w_j / c_j
5: $\quad$**for** all $1 \leq t \leq n_x^u$ **do**
6: $\quad\quad D_t^u \leftarrow \{k_1, \ldots, k_t\}$
7: $\quad\quad \tilde{D}_t^u \leftarrow \{k_t\}$
8: $\quad$**end for**
9: **end for**
10: **return** $D = \arg\min_{D \in \{D_t^u\} \cup \{\tilde{D}_t^u\} \cup \{\emptyset\}} \{\phi(D)\}$

In what follows, we present a polynomial-time approximation algorithm, $\mathrm{DS}_{\mathrm{Knap}}$ for the problem, formally defined in Algorithm 3. The algorithm is based on $\mathrm{DS}_{\mathrm{PP}}$ but avoids the need to iterate over all possible budgets. In particular, $\mathrm{DS}_{\mathrm{Knap}}$ does not make use of a general $(1+\epsilon)$-approximation algorithm for solving the Knapsack problem. Instead, $\mathrm{DS}_{\mathrm{Knap}}$ incorporates within its design the specifics of a 2-approximation algorithm for the Knapsack problem, the details of which are presented and discussed in the proof of Theorem 6.

Theorem 6. *If there exists some constant δ such that $\rho_j \leq \delta$ for all $j \in N_x$, then Algorithm $\mathrm{DS}_{\mathrm{Knap}}$ is a polynomial $O(\sqrt{\beta})$-approximation algorithm running in time $O(n_x^2 \log n_x)$.*

Proof. The algorithm is based on the 2-approximation algorithm for Knapsack [23], which works as follows: given budget B, prune all elements with a cost greater than B. Order all elements in non-increasing order of their profitability, captured by their profit-to-cost ratio. Greedily add elements to the solution, starting from the most profitable one, as long as their overall cost does not exceed the given budget. Once adding an additional element causes a violation of the budget constraint, pick the best out of two candidate solutions: the set of elements accumulated which satisfy the budget constraint, and the first element that caused the violation of the constraint.[2]

The remainder of the proof draws its intuition from the proof of Theorem 5, combined with the properties of the 2-approximation algorithm for Knapsack.

Given some budget constraint B on the access cost of a solution, consider the 2-approximation algorithm for knapsack when given B as its budget constraint.

The algorithm first prunes all elements with cost greater than the budget. In particular, there exists some element j such that c_j is the maximal cost of an element not violating the budget. $\mathrm{DS}_{\mathrm{Knap}}$ simulates the same prunning by iterating over all potential values for this maximal cost, and maintaining only the data stores with cost not exceeding this maximal cost (lines 2-3). It follows that there is a $u \in \{c_j | j \in N_x\}$ for which

$$N_x^u = \{j \in N_x | c_j \leq B\}. \tag{16}$$

[2]Most common implementations consider the element with maximum profit instead of the first element causing the violation of the budget constraint. However, such an amended choice has no effect on the analysis of the algorithm's performance.

Now that the knapsack approximation algorithm only considers items with cost not violating the budget B, it orders the items in non-increasing order of w_j/c_j, and scans the items in this order, starting from the most profitable, until reaching the first item in this order, k_{t_B}, such that $\sum_{j=1}^{t_B} c_{k_j} \leq B$, but $\sum_{j=1}^{t_B+1} c_{k_j} > B$. The algorithm then picks the best between two possible candidate solutions: the set $\{1, \ldots, k_{t_B}\}$, and the set $\{k_{t_B+1}\}$.

Our algorithm iterates over *all* potential candidates of this form, namely, all sets of data stores $\{1, \ldots, k_t\}$, and all sets of data stores $\{k_t\}$. Consider an optimal solution $D^* \subseteq N_x$ to the DSS problem, and denote by B^* the access cost contributing to the overall cost of D^*. Consider the iteration of $\mathrm{DS}_{\mathrm{Knap}}$ where $N_x^u = \{j \in N_x | c_j \leq B^*\}$ (as shown in the argument leading to Eq. 16 such a cost u necessarily exists).

Consider the items in N_x^u ordered in non-increasing order of w_j/c_j, and let t_{B^*} be the first item in the order for which $\sum_{j=1}^{t_{B^*}} c_{k_j} \leq B^*$, but $\sum_{j=1}^{t_{B^*}+1} c_{k_j} > B^*$. The algorithm will choose either $\{1, \ldots, k_{t_{B^*}}\}$, which is candidate D_t^u in the iteration where $t = t_{B^*}$ of lines 5-5; or it will choose $\{t_{B^*}+1\}$, which is candidate $\tilde{D}_t^u$ in the iteration where $t = t_{B^*}+1$ of lines 5-8. By the proof of Theorem 5, the best of these two candidate solutions is an $O(\beta^{\frac{\epsilon}{1+\epsilon}}) = O(\sqrt{\beta})$ approximate solution for the DSS problem, since we are using a 2-approximation algorithm for knapsack, implying $\epsilon = 1$.

Since $\mathrm{DS}_{\mathrm{Knap}}$ picks the candidate solution with minimal overall cost, the solution returned by the algorithm is itself an $O(\sqrt{\beta})$-approximate solution for the DSS problem. The running time of the algorithm is dominated by the outer for-loop in lines 2-9 which has n_x iterations, wherein each iteration we order all elements in N_x^u, which takes $O(n_x \log n_x)$ time. Hence, the overall running time of the algorithm is $O(n_x^2 \log n_x)$, which completes the proof. $\square$

VI. Simulation Study

This section uses a real access trace and a real content distribution network topology to provide insights into the performance of various access strategies in versatile settings.

A. System Topology and Costs

We use the topology of the OVH [24] content distribution network. The OVH network [24] includes 19 *Points of Presence (PoPs)* in Europe and North America along with the available bandwidth between PoPs. We interpret each PoP as containing both a data store and a co-located client. Queries are generated at clients and each such query triggers an access to a subset of the data stores according to the prescribed policy.

We assume that clients use the shortest hop-count path between their location and the data store they access. Ties are broken by picking the path with maximal bottleneck link bandwidth. The cost for a client located at node i to access a data store at node j is:

$$c_{i,j} = \lceil 1 + \alpha \cdot \mathrm{dist}(i,j) + (1-\alpha) \cdot \frac{T}{\mathrm{BW}(i,j)} \rceil, \tag{17}$$

where (i) $\mathrm{dist}(i,j)$ is the hop-count between node i and node j, where $\mathrm{dist}(i,i) = 0$, (ii) $\mathrm{BW}(i,j)$ is the maximum bottleneck

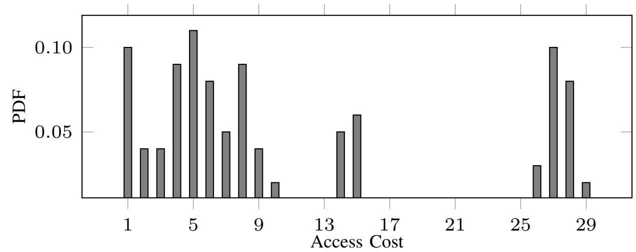


Fig. 2. Histogram of $c_{i,j}$ values for the OVH network, based on Eq. 17, using $\alpha = 0.5$ and $T = 500$.

bandwidth of a minimum length path from node i to node j, where $\mathrm{BW}(i,i) = \infty$, (iii) T is a design parameter satisfying $T \geq \max_{i,j} \mathrm{BW}(i,j)$, that relates the increased cost of having a smaller bandwidth with the increased cost due to having a higher hop-count. Lastly, (iv) α is a design parameter that helps balance the effects of hop-count distance and bottleneck bandwidth on the cost. In particular, for $\alpha = 1$ the cost is fully dominated by the hop-count distance and for $\alpha = 0$ it is fully dominated by the bottleneck bandwidth, regularized by the parameter T. Unless stated otherwise, throughout our simulations we set $T = \max_{i \neq j} \mathrm{BW}(i,j)$. Specifically, $T = 500$ for the OVH network.

Figure 2 presents the histogram of the default access cost used in our evaluation between all pairs of clients and data stores in the OVH network.

B. Data Store Characteristics

Data stores are initially empty, and each can contain a maximum of S data elements. Once an item is added to a full data store, it evicts an item according to the Least Recently Used (LRU) policy. The indicators are implemented using Counting Bloom Filters [20], each consisting of $B(S)$ 8-bit counters and 5 hash functions, where $B(S)$ is chosen as the number of counters required to obtain a target false positive ratio of 0.02 [9]. For example, in most of our simulations we set $S = 1000$, which implies $B(S) = 8181$. We assume that up-to-date indicators are available at all time as can be efficiently realized by compressed Bloom filters [19], or by only transmitting the changes as in [3].

Each data store estimates its own misindication ratio by evaluating an exponential moving average over epochs of R requests made to the data store. Formally, let $m_j(s,t)$ denote the number of misses occurring at data store j during the requests $s+1, \ldots, t$ made to data store j[3]. For any $t \leq R$ we let the estimated misindication ratio after handling request t be $\rho_j(t) = \frac{m_j(0,t)}{t}$. For $t > R$, we let $\rho_j(t)$ be the most recent estimate over epochs of R requests, $\rho_j(\lfloor t/R \rfloor \cdot R)$, where for every non-negative integer k this estimate is updated after handling request $(k+1)R$ such that $\rho_j((k+1)R) = \delta \cdot m_j(kR,(k+1)R)/R + (1-\delta) \cdot \rho_j(kR)$. In our simulations, we take $\delta = 0.1$ and $R = 100$, as we found this configuration

[3]Recall that we only access a data store if it has provided a positive indication.

TABLE II
OVH NETWORK SIMULATION. RESULTS PRESENT FOR EVERY SCENARIO AND EVERY POLICY THE NORMALIZED COST OF THE METRICS: NON-COMPULSORY MISS PENALTY (NCMP), ACCESS COST (AC), AND TOTAL COST (TC). THE VALUES FOR NCMP AND AC ARE NORMALIZED BY THE AC OF THE PERFECT INDICATORS POLICY AND THE VALUES FOR TC ARE NORMALIZED BY ITS TC.

β	Policy	1 location			3 locations			5 locations		
		NCMP	AC	TC	NCMP	AC	TC	NCMP	AC	TC
10^2	CPI	1.32	1.08	1.21	1.12	1.31	1.12	0.81	1.27	1.09
	EPI	0.00	1.58	1.09	0.23	5.65	1.39	1.72	7.29	1.56
	$\mathrm{DS_{Knap}}$	0.35	1.31	1.10	0.21	2.05	1.11	0.17	2.06	1.09
	$\mathrm{DS_{Pot}}$	0.02	1.56	1.09	0.02	4.43	1.28	0.01	4.91	1.28
10^3	CPI	13.20	1.08	1.23	11.15	1.31	1.11	8.09	1.27	1.07
	EPI	0.00	1.58	1.02	0.00	5.93	1.06	0.00	9.55	1.08
	$\mathrm{DS_{Knap}}$	0.01	1.58	1.02	0.10	4.11	1.04	0.17	3.64	1.03
	$\mathrm{DS_{Pot}}$	0.00	1.58	1.02	0.01	5.61	1.05	0.02	6.38	1.05
10^4	CPI	132.00	1.08	1.24	111.50	1.31	1.11	80.86	1.27	1.07
	EPI	0.00	1.58	1.01	0.00	5.93	1.02	0.00	9.55	1.02
	$\mathrm{DS_{Knap}}$	0.00	1.58	1.01	0.01	5.66	1.02	0.24	5.22	1.02
	$\mathrm{DS_{Pot}}$	0.00	1.58	1.01	0.00	5.89	1.02	0.02	7.89	1.02

to yield a stable ρ at each data store and to work well in practice.

C. Traffic Trace, Metrics, and Simulated Scenarios

We used a publicly available Wikipedia trace [25] consisting of 357K read requests to Wikipedia pages during a 5 minute period[4]. Each request in this trace is assigned to a random client issuing the request, and requests appear according to their order in the trace. For handling the requests, we consider the following access policies applied by the clients for choosing the set of data stores to access: (i) CPI, (ii) EPI, (iii) $\mathrm{DS_{Knap}}$, and (iv) $\mathrm{DS_{Pot}}$. The evaluation factors the total cost, where all clients are running the same algorithms. We also considered the benchmark performance provided by using perfect indicators (PI). This benchmark is used to normalize the costs of the various policies considered.

In terms of metrics, we measure the following three metrics: First and foremost, the *total cost* (TC) incurred by each access strategy for serving the entire trace, normalized by the total cost of PI. This is further refined into the total *access cost* (AC) and the *Non Compulsory Miss Penalty* (NCMP). These two measures are normalized by the total access cost of PI. The former (AC) captures the cost of accessing the data stores and is likely to be higher for access strategies that access multiple data stores for each item request. The latter (NCMP) accounts for miss penalties incurred by an access strategy despite the fact that the item already exists in one of the data stores. This can happen due to the combined effect of false positives and strategies that do not access all data stores whose indicator is positive, such as CPI. We note that normalizing these performance measures by the total access cost of PI allows us to compare the performance in various settings, while alleviating some of the exogenous effects specific to the scenario being evaluated.

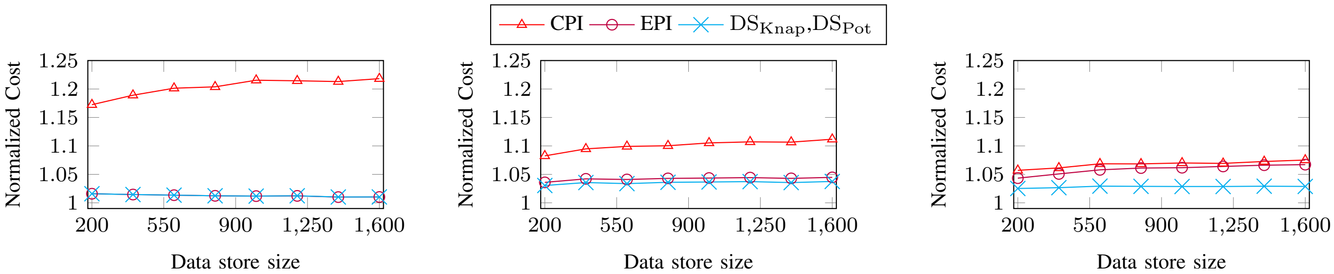
D. Heterogeneous Case (OVH network)

Our first experiment considers a *system-wide request distribution policy* where an item can only be placed in k data stores that are chosen by a hash function based on the requests' content. Such a policy is inspired by ideas such as replication and partitioning to increase the hit ratio [26]. The outcome of this evaluation is provided in Table II, where we present the PI normalized results for various β and k values. We increased k up to 25% of the 19 data stores in the system. Notice that CPI has the minimal AC in all scenarios, as could be expected by its definition. However, CPI is extremely sensitive to false positives, which are translated to a high NCMP value. EPI, on the other hand, is very effective for $k = 1$ but becomes less attractive as we increase k, due to the fact it ends up accessing too many data stores. It has the minimal NCMP but pays too much for access costs. Note that for $\beta = 100$ and $k = 5$ EPI has NCMP > 0 as the total access cost of all positive data stores is often larger than β and thus EPI would rather avoid accessing any of the data stores and reverts to paying the miss-penalty β. The $\mathrm{DS_{Pot}}$ strategy is the most efficient (by a very small margin) for $\beta = 100$ and $k = 1$, and is equal or inferior to $\mathrm{DS_{Knap}}$ in all other cases. Intuitively, $\mathrm{DS_{Pot}}$ optimizes for reducing the miss penalty which in turn results in an increased AC. It always outperforms EPI but is inefficient in cases where the access cost is the dominant part of the cost (similarly to EPI). In contrast, $\mathrm{DS_{Knap}}$ exhibits the all-around best performance. It is the best strategy in most scenarios, but most importantly it is never a bad strategy. Thus, even when it underperforms compared to some other strategy, the differences tend to be marginal. Furthermore, when considering the costs incurred by its potential errors (i.e., its AC and NCMP costs), it demonstrates the best performance compared to the other policies in almost all scenarios, and by a significant margin.

E. Homogeneous Case: Varying Data Store Size

Figure 3 shows an experiment with 19 locations, where we vary the data store sizes. The results are shown for $k = 1$, $k = 3$ and $k = 5$ data stores, with homogeneous access costs throughout the network. In these homogeneous cost settings $\mathrm{DS_{Knap}}$ and $\mathrm{DS_{Pot}}$ are equivalent to the scheme which minimizes the expected overall cost, FPO. Furthermore, their performance is very close to the one achieved with perfect

[4]The trace includes requests made on Sep. 22, 2007, from 06:12 to 06:17



(a) 1 location (b) 3 locations (c) 5 locations

Fig. 3. Homogeneous network with varying data store size. The miss penalty is set to $\beta = 100$, and the target false positive ratio is 0.02.

indicators. When $k = 1$, $\mathrm{DS_{Knap}}$ and $\mathrm{DS_{Pot}}$ behave like EPI, while CPI is inefficient. When k increases, CPI improves (due to fewer non-compulsory misses) while EPI worsens (due to higher access cost). This shows that the existing heuristics are too simplistic to fit all system configurations, thus motivating the need for our algorithms.

VII. Discussion

Our work closes an important knowledge gap concerning indicator based caching in network systems. Namely, it answers the fundamental question of providing a stable access strategy that achieves near-optimal results in a wide variety of scenarios.

Our work starts by showing that the access strategy problem was roughly ignored until now and that the existing solutions are only attractive for some system parameters. That is, their effectiveness is determined by uncontrolled variables that may change throughout the system's life, and may not be known in advance. In contrast, the algorithms suggested in this work provide provable approximation ratios to the optimal solution and are shown to be near optimal in a variety of system settings.

Acknowledgements

The work was supported by the Israel Science Foundation (grants No. 1036/14 and 1505/16).

References

[1] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. M. Leung, "Cache in the air: exploiting content caching and delivery techniques for 5g systems," *IEEE Comm. Mag.*, vol. 52, no. 2, pp. 131–139, 2014.

[2] F. Boccardi, R. W. Heath, A. Lozano, T. L. Marzetta, and P. Popovski, "Five disruptive technology directions for 5g," *IEEE Comm. Mag.*, vol. 52, no. 2, pp. 74–80, 2014.

[3] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An architecture for global-scale persistent storage," *SIGPLAN Not.*, vol. 35, no. 11, pp. 190–201, 2000.

[4] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, "Adaptsize: Orchestrating the hot object memory cache in a content delivery network," in *NSDI*, 2017, pp. 483–498.

[5] M. Bilal and S. G. Kang, "A cache management scheme for efficient content eviction and replication in cache networks," *IEEE Access*, vol. 5, pp. 1692–1701, 2017.

[6] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in *ICN*, 2012, pp. 55–60.

[7] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, 2000.

[8] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[9] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

[10] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of bloom filters for distributed systems," *IEEE Commun. Surveys Tuts.*, vol. 14, no. 1, pp. 131–155, 2012.

[11] G. Einziger and R. Friedman, "Tinyset: An access efficient self adjusting bloom filter construction," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 2295–2307, 2017.

[12] ——, "Counting with tinytable: Every bit counts!" in *ICDCN*, 2016, p. 27.

[13] O. Rottenstreich and I. Keslassy, "The bloom paradox: When not to use a bloom filter," *IEEE/ACM Trans. Netw.*, vol. 23, no. 3, pp. 703–716, 2015.

[14] Y. Kanizo, D. Hay, and I. Keslassy, "Access-efficient balanced bloom filters," *Comput. Commun.*, vol. 36, no. 4, pp. 373–385, 2013.

[15] A. Rousskov and D. Wessels, "Cache digests," *Computer Networks and ISDN Systems*, vol. 30, no. 22-23, pp. 2155–2168, 1998.

[16] L. Luo, D. Guo, R. T. Ma, O. Rottenstreich, and X. Luo, "Optimizing bloom filter: Challenges, solutions, and comparisons," *arXiv preprint*, 2018. [Online]. Available: https://arxiv.org/abs/1804.04777

[17] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An improved construction for counting bloom filters," in *ESA*, 2006, pp. 684–695.

[18] W. Li, K. Huang, D. Zhang, and Z. Qin, "Accurate counting bloom filters for large-scale data processing," *Mathematical Problems in Engineering*, 2013.

[19] M. Mitzenmacher, "Compressed bloom filters," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 604–612, 2002.

[20] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An improved construction for counting bloom filters," in *ESA*, 2006, pp. 684–695.

[21] G. Einziger, O. Eytan, R. Friedman, and B. Manes, "Adaptive software cache management," in *ACM Middleware*, 2018, pp. 94–106.

[22] G. Einziger, R. Friedman, and B. Manes, "Tinylfu: A highly efficient cache admission policy," *TOS*, vol. 13, no. 4, pp. 35:1–35:31, 2017.

[23] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[24] "The OVH CDN network," 2018. [Online]. Available: https://www.ovh.co.uk/cdn/cdn-network-map.xml

[25] G. Urdaneta, G. Pierre, and M. van Steen, "Wikipedia workload analysis for decentralized hosting," *Computer Networks*, vol. 53, no. 11, pp. 1830–1845, 2009.

[26] G. Einziger, R. Friedman, and E. Kibbar, "Kaleidoscope: Adding colors to kademlia," in *IEEE P2P 2013*, 2013.