Access Strategies for Network Caching

Access Strategies for Network Caching

Itamar Cohen*, Gil Einziger†, Roy Friedman‡, and Gabriel Scalosub†
*Politecnico di Torino, Italy †Ben-Gurion University of the Negev, Beer Sheva, Israel ‡Technion, Haifa, Israel
itamar.cohen@polito.it, 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 as a cost minimization problem, while taking into account both access costs, the inaccuracy of the approximate summaries, as well as the penalties incurred by failed requests. We introduce practical algorithms with guaranteed approximation ratios and further show that they are optimal in various 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.**

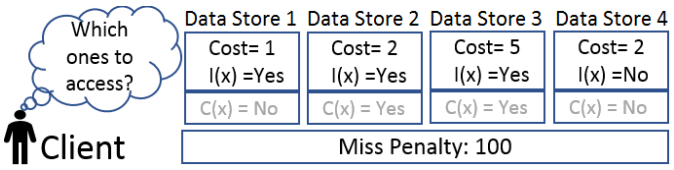
***Index Terms*—Cooperative caching, replica selection, cache sharing, access strategies, content delivery networks, network caching, information centric networks.**

I. Introduction

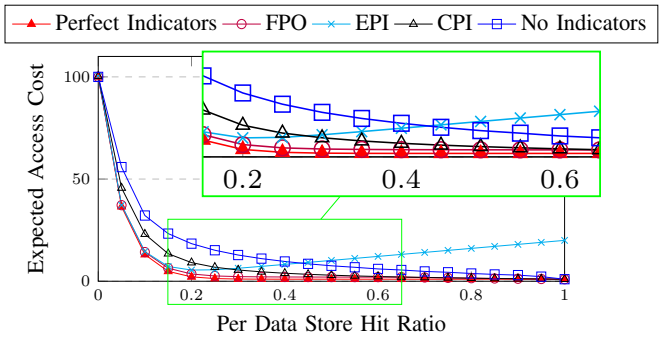
Having access to multiple network connected *data stores* is common in modern network settings such as 5G in-network caching [2], [3], content delivery networks (CDN) [4], [5], information centric networking [6], wide-area networks [7], as well as in any multi data center servie provider. Data stores 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 [8]. 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* of the content available at the various data stores [7]. Bloom filters [9] are a common implementation for such indicators, but many other space-efficient approximate membership representations can also be used [4], [10]–[15].

The shortcoming of relying on such indicators is that they may exhibit *false positives*, meaning that they may indicate that a given item is available at a particular data store while it is actually not there. Indeed, the work of [14] 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 that selects a *subset* of the data stores to access per request. Existing strategies for this problem include: (i) the *Cheapest Positive Indication (CPI)* [11], [16] 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.



(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.



sc

(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.

In the example of Fig. 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, incurring an additional miss penalty of 100, 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), implying an access cost of $1 + 2 + 5 = 8$.

An earlier version of this work was published in [1]. This work adds the DS_{PGM} access strategy and provides an extended simulation study.

*The work was done while this author was with Ben-Gurion University.

No additional miss penalty is incurred, since item x is indeed available in one of the accessed data stores (e.g., in 2). One can also consider an ideal strategy equipped with a *perfect indicator* with no false-positives. Such a strategy would require a cost of merely 2 incurred for accessing data store 2 alone.

Fig. 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 Fig. 1b. Our proposed strategies, described in Sections IV-VII, 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. In Section III, we formally model this problem in very general and heterogeneous settings with varying access costs, per data store hit ratios and miss penalties. In Section IV, we analyze the case of fully homogeneous settings. Our analysis shows that even in such highly-simplified settings, 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.

In Sections V-VII, we propose and analyze several polynomial-time approximation algorithms for the fully heterogeneous case. We evaluate our proposed algorithms via extensive simulation in Section VIII. We base our evaluation on real data with varying system parameters. Our results 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. We conclude in Section IX with a discussion of our results.

II. Related work

A. Approximate Set Membership

Approximate set membership indicators are data structures used for 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 [9] offer space-efficient encoding but do not support the removal of items. Other works [7], [12], [13], [15], [17] improve on them in various aspects, such as support for removals [13], [18], [19], a more efficient access pattern [12], [15], and lower transmission overheads [20].

Such indicators are extensively used in multiple domains, including 5G network caching [2], [3], content delivery networks (CDN) [4], [5], [21], information centric networking (ICN) [6], and wide-area networks [22]. Many systems leverage that Bloom filter variants [9], [12], [13], [18] do not exhibit false-negatives. Thus, a negative indication guarantees that the datum is not stored in the data store. Examples for such usage include Akamai's content delivery network [21], Squid Web cache [23] and Google's Bigtable [24]. We note that the usage of indicators is not restricted to network caching. E.g., such indicators are also used as part of deep packet inspection systems [25], as well as in routing mechanisms [11].

B. Access Strategies and Replica Selection

The work of [7] suggests an architecture for distributed caching in wide area networks. In this solution, caches share an approximation of their content. Clients use this information to contact only the caches with positive indications (EPI). A similar architecture is also considered in [11], [16]. There, clients access the cheapest cache with a positive indication (CPI). However, the impact of the access strategy and its optimization in the face of false-positive replies is overlooked in previous works.

The work of [8] studied access strategies to data stores in a commercial content delivery network. Access strategies to data stores have been extensively studied also in the context of data grid systems. In such systems, the problem of selecting which data store to access is commonly referred to as the *replica selection* problem. A comprehensive survey of replica selection algorithms can be found in [26]. However, all these works do not use indicators, but instead assume the existence of an exact and always-fresh list of locations of every stored datum. Maintaining such a repository incurs high overhead in terms of bandwidth consumption and synchronization mechanisms.

The work of [14] considers the special case of a single data store, equipped with a Standard Bloom Filter [9] or a Counting Bloom Filter [27]. 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 $S_{j,s}$ the set of items

TABLE I
LIST OF SYMBOLS. THE TOP PART CORRESPONDS TO OUR SYSTEM MODEL (SECTION III), THE MIDDLE PART CORRESPONDS TO THE $\mathrm{DS_{Pot}}$ AND $\mathrm{DS_{Knap}}$ ALGORITHMS (SECTIONS V-VI), AND THE BOTTOM PART CORRESPONDS TO THE $\mathrm{DS_{PGM}}$ ALGORITHM (SECTION VII).

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: $\mathrm{FP}_j = \Pr(I_j(x)=1 \mid x \notin S_j)$
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
O^*	Set of data stores used by some optimal solution OPT
$H_k(L_k)$	Access cost for the k highest (lowest) data stores in N_x
$P(k^*)$	Potential function: $P(k^*) = L_{k^*} + \beta \prod_{j=1}^{k^*} \rho_{\ell_j}$
M	$M = \min\left\{\sum_{j\in N_x} c_j, \beta\right\}$
r	$r = \log \beta$
N_j^ℓ	Partition of N_x in level ℓ
O_j^ℓ	Subset of data stores which OPT selects out of N_j^ℓ
V_j^ℓ	Candidate sub-solutions which $\mathrm{DS_{PGM}}$ considers out of N_j^ℓ

stored at data store j at time s. For every request x, issued at time s, drawn from some distribution, we let $p_{j,s}^h$ denote the probability that $x \in S_{j,s}$. We note that the average $p_{j,s}^h$ over the entire sequence is commonly referred to as the *hit ratio*, i.e., the fraction of requests in σ that were available in data store j. Our work assumes that the past hit ratio is a reasonable estimate of $p_{j,s}^h$ [28], [29]. We refer to this estimation as the probability that the next accessed item x is stored in $S_{j,s}$.[1]

Each data store j maintains an *indicator* $I_{j,s}$, which approximates the set of items in data store j at time s; given an item x, $I_{j,s}(x) = 1$ is a a *positive indication* while $I_{j,s}(x) = 0$ is a *negative indication*. Our model assumes indicators that may exhibit only one-sided errors, i.e., they never err when providing a negative indication.[2] In practice, most approximate set solutions satisfy this assumption [7], [12], [13], [15]. The *false positive ratio* of $I_{j,s}$ is defined by $\mathrm{FP}_{j,s} = \Pr(I_{j,s}(x)=1 \mid x \notin S_{j,s})$. It captures the probability that given a request for x, issued at time s, drawn from some distribution, that is not in $S_{j,s}$, the indicator would mistakenly indicate that it is in $S_{j,s}$. For every data store j, given its indicator $I_{j,s}(x)$, we let $\rho_{j,s} \in [0,1]$ denote its *misindication ratio* $\rho_{j,s} = \Pr(x \notin S_{j,s} \mid I_{j,s}(x) = 1)$. When clear from the context, we abuse notations and omit the subscript s.

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, despite 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. We note, however, that in the evaluation of our algorithms, we also consider environments where the misindication ratios need not be mutually independent (Section VIII).

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., $N_x = \{j \in N \mid I_j(x) = 1\}$, 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.,

$$\phi(D) = c_D + \beta \cdot \rho_D. \tag{1}$$

When misindication ratios are mutually independent we have

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

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,

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

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

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

To simplify expressions throughout the paper, we always use logarithms of base 2, and omit the base of the logarithm.

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

[1] We provide further details of how to obtain such an estimation in Sec. VIII-B.

[2] This means having no false-negatives, i.e., $\Pr(I_{j,s}(x)=0 \mid x \in S_{j,s}) = 0$.

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

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

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

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

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

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

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

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

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 the extension of $\tilde{\phi}$, as defined in Eq. 5, as a function defined over the reals, $\tilde{\phi}(y)$. This function is convex since its second derivative 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 $m^*(n_x)$ of data stores to access given that there are n_x positive indications satisfies $m^*(n_x) \in \{0, n_x, \lfloor y^* \rfloor, \lceil y^* \rceil\}$, where $\lfloor y^* \rfloor$ and $\lceil y^* \rceil$ should be considered only if $y^* \in [0, n_x]$. Hence, The expected overall cost of FPO is

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

Having studied the overall cost of the above policies, we may revisit Fig. 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.

In this 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. However, many systems are heterogeneous, making the minimization of $\phi(D)$ a much more challenging task. In the following sections, we describe several approximation 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.

V. A Potential-based Algorithm

In this section, we describe our first approximation algorithm for solving the DSS problem in fully heterogeneous settings and provide a rigorous analysis of its performance.

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 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 sum of the k largest access costs of data stores in N_x, respectively. Our algorithm, $\mathrm{DS}_{\mathrm{Pot}}$ (where Pot stands for *Potential*), described in Algorithm 1, considers the data stores

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

$L_k \leftarrow$ sum of k smallest c_j-s, $k = 1,\ldots,n_x$
$\ell_1,\ldots,\ell_{n_x} \leftarrow N_x$ in non-decreasing order of ρ_j
for $k = 1,\ldots,n_x$ **do**
  $D_k \leftarrow \{\ell_1,\ldots,\ell_k\}$ ▷ prefix of length k
$k^* = \arg\min_k \left\{ P(k) = L_k + \beta \prod_{j=1}^{k} \rho_{\ell_j} \right\}$
return $D = D_{k^*}$

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

$w_j \leftarrow -\log(\rho_j)$ for all $j \in N_x$
for $B \in \left\{0,1,\ldots,\min\left\{\sum_{j\in N_x} c_j, \beta\right\}\right\}$ **do**
  $D_B \leftarrow A_{\mathrm{Knap}}(B, N_x, w, c)$
return $D = \arg\min_B \{\phi(D_B)\}$

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

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

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

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 $k^* = \left|\mathrm{O}^*\right|$. Since data stores are ordered in non-decreasing order of misindication ratio, it follows that $\prod_{j=1}^{k^*} \rho_{\ell_j} \le \prod_{j\in \mathrm{O}^*} \rho_j$, 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

$$\begin{aligned} P(k^*) &= L_{k^*} + \beta \prod_{j=1}^{k^*} \rho_{\ell_j} \\ &\le \sum_{j\in \mathrm{O}^*} c_j + \beta \prod_{j\in \mathrm{O}^*} \rho_j = \phi(\mathrm{O}^*). \end{aligned} \tag{11}$$

Since D is chosen to be the set of data stores that minimizes $P(k)$, where k is the length of the prefix of 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 $\frac{H_k}{L_k} \le \max_j \{c_j\}$ 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 access costs are equal, $\mathrm{DS}_{\mathrm{Pot}}$ yields an optimal solution to the DSS problem.

VI. 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 *additive* (or 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*. The solution space of these knapsack problems 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. We will make extensive use of instances of the Knapsack problem, 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 $\sum_{j\in S} \gamma_j \le B$ 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 $(1+\varepsilon)$ factor in polynomial time by an FPTAS [30].

A. Pseudo-polynomial Algorithms for DSS

Our design of pseudo-polynomial algorithms for the DSS problem is based on defining a *collection* of Knapsack problems, and considering their solutions as provided by an approximation algorithm that solves the Knapsack problem. We assume in this section that c_j is an integer for every data store $j \in N$. 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, dubbed $\mathrm{DS}_{\mathrm{PP}}$ (where PP stands for *Pseudo-Polynomial*), formally defined in Algorithm 2, makes use of a $(1+\varepsilon)$-approximation algorithm A_{Knap} for the knapsack problem, for some $\varepsilon \ge 0$. The complexity and performance guarantee both depend 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 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 $O(nB)$ [30], 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 $O(nB)$, DS_{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 DS_{PP}, defined in Algorithm 2, finds an optimal solution to the DSS problem. Consider an optimal solution $\text{O}^* \subseteq N_x$ for the DSS problem, and let $B^* = c_{\text{O}^*}$. Since by optimality $B^* \leq M$, and by our assumption c_j is an integer for every $j \in N$, we are guaranteed that DS_{PP} considers $B = B^*$ in one of the iterations of the for-loop in lines 2-3. 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 monotony 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 $\phi(D_B) = c_{D_B} + \beta \rho_{D_B} > c_{\text{O}^*} + \beta \rho_{\text{O}^*} = \phi(\text{O}^*)$. Since $c_{\text{O}^*} = B^* = B \geq c_{D_B}$, it must follow that $\rho_{D_B} > \rho_{\text{O}^*}$, for $c_{\text{O}^*} \leq B$, which contradicts Eq. 12. For bounding the running time of DS_{PP}, note that the algorithm performs M iterations, where in each iteration it solves a knapsack problem using an algorithm that runs in $O(n_x M)$ time. It follows that the running time of DS_{PP} in this case, is $O(n_x M^2)$, as required. □

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 DS_{PP} solves the DSS problem in polynomial time.*

We now turn to study the tradeoff between the running time of DS_{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 $\delta < 1$ such that $\rho_j \leq \delta$ for all $j \in N_x$ and algorithm A_{Knap} is a $(1+\varepsilon)$-polynomial time approximation algorithm for Knapsack running in time $O(f(n_x, \varepsilon))$, then DS_{PP} is a pseudo-polynomial algorithm that finds an $O(\beta^{\frac{\varepsilon}{1+\varepsilon}})$-approximate solution for the DSS problem in time $O(f(n_x, \varepsilon) \cdot M)$.*

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

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

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

By our assumption there exists some constant $\delta < 1$ 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 $\rho_{\text{O}^*} \beta < 1$, 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 $\ell \leq \sum_{j \in \text{O}^*} w_j \leq \ell + 1$.

Consider the iteration of DS_{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 $(1+\varepsilon)$-approximation algorithm, we are guaranteed to have $\sum_{j \in D_B} w_j \geq \frac{1}{1+\varepsilon} \sum_{j \in \text{O}^*} w_j$ since O^* 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

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

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,

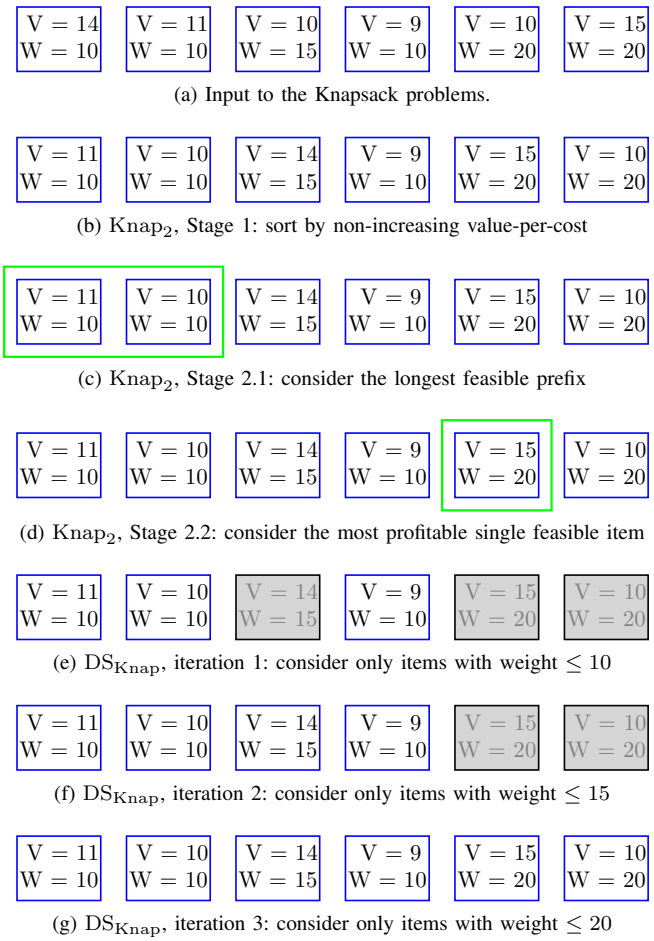
$$\begin{aligned} \phi(D_B) &= \sum\nolimits_{j \in D_B} c_j + \beta \prod\nolimits_{j \in D_B} \rho_j \\ &\leq B^* + O\left(\beta^{\frac{\varepsilon}{1+\varepsilon}}\right) \left(\beta \prod\nolimits_{j \in \text{O}^*} \rho_j\right) \\ &= \sum\nolimits_{j \in \text{O}^*} c_j + O\left(\beta^{\frac{\varepsilon}{1+\varepsilon}}\right) \left(\beta \prod\nolimits_{j \in \text{O}^*} \rho_j\right) \\ &\leq O\left(\beta^{\frac{\varepsilon}{1+\varepsilon}}\right) \left(\sum\nolimits_{j \in \text{O}^*} c_j + \beta \prod\nolimits_{j \in \text{O}^*} \rho_j\right) \\ &= O\left(\beta^{\frac{\varepsilon}{1+\varepsilon}}\right) \phi(\text{O}^*) \end{aligned} \tag{15}$$

which completes the proof. □

B. A Polynomial-time Approximation Algorithm for DSS

In what follows, we present a polynomial-time approximation algorithm for the DSS problem, which we dub DS_{Knap} (where Knap stands for *Knapsack*). We begin with a high-level overview of DS_{Knap}, and then turn to a detailed description and a formal proof of the algorithm's approximation guarantees and run time.

1) High-Level Overview of DS_{Knap}: To develop a fully-polynomial Knapsack-based solution for the DSS problem, we observe that our pseudo-polynomial solution, DS_{PP}, iterates over every possible budget (line 2), which may translate to an excessive number of iterations. We, therefore, strive to significantly decrease the number of iterations while keeping strong performance guarantees.



(a) Input to the Knapsack problems.

(b) Knap_2, Stage 1: sort by non-increasing value-per-cost

(c) Knap_2, Stage 2.1: consider the longest feasible prefix

(d) Knap_2, Stage 2.2: consider the most profitable single feasible item

(e) $\mathrm{DS}_{\mathrm{Knap}}$, iteration 1: consider only items with weight ≤ 10

(f) $\mathrm{DS}_{\mathrm{Knap}}$, iteration 2: consider only items with weight ≤ 15

(g) $\mathrm{DS}_{\mathrm{Knap}}$, iteration 3: consider only items with weight ≤ 20

Fig. 2. High-level design of Algorithm $\mathrm{DS}_{\mathrm{Knap}}$. For each item, V represents the profit, and W represents the cost. (a) shows the input collection of items. (b)-(d) exemplify a run of the 2-approximation algorithm Knap_2, when the budget is 30. (e)-(g) exemplify a run of our fully-polynomial approximation algorithm, $\mathrm{DS}_{\mathrm{Knap}}$. At each iteration, $\mathrm{DS}_{\mathrm{Knap}}$ ignores the shaded-gray items.

As an illustrative example, consider Fig. 2, which shows a list of 6 input items, where each item has some profit V, and cost W (Fig. 2a). Our solution is based upon the celebrated 2-approximation algorithm for Knapsack [30], which we hereafter dub Knap_2, for short. Knap_2 works as follows: In the first stage, it prunes all items with a cost greater than the budget B, and orders the remaining items in non-decreasing order of their profitability, captured by their profit-to-cost ratio (Fig. 2b). In the second stage, Knap_2 considers two options: (i) greedily adding elements to the solution, starting from the most profitable one, as long as their overall cost does not exceed the given budget, and (ii) taking the single, most-profitable item. For instance, if the budget is 30, Knap_2 considers taking either (i) the two items highlighted in Fig. 2c, which is the longest feasible prefix (adding the third item would increase the total cost to 35), or (ii) the item with a profit of 15, which is the most profitable single item (Fig. 2d).

A straightforward solution for the DSS problem is to run Knap_2 once for every possible budget. However, this naive approach may result in a prohibitive number of iterations. For instance, for the given input items (Fig 2a), the possible budgets (sum of costs of subsets of items) are $10, 15, 20, \ldots, 75$, and 85, namely, 15 possible budgets.

However, a careful analysis provides the following *observation: It suffices to run* Knap_2 *only once for every distinct weight of the input items*. In our example, this observation implies that it suffices to run Knap_2 only thrice: first, while considering only the items with cost up to 10 (Fig. 2e); second, considering only the items with cost up to 15 (Fig. 2f); and finally, considering only the items with cost up to 20 (Fig. 2g). Note that the order of the items in different runs may differ from each other, as captured by comparing Fig. 2e and Fig. 2f. In what follows, we provide our analysis of $\mathrm{DS}_{\mathrm{Knap}}$, which formally defines and proves the observation above.

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: **return** $D = \arg\min_{D \in \{D_t^u\}_{u,t} \cup \{\tilde{D}_t^u\}_{u,t} \cup \{\emptyset\}} \{\phi(D)\}$

2) Approximation and Run-time Analysis: Our fully-polynomial Knapsack-based algorithm, $\mathrm{DS}_{\mathrm{Knap}}$, is formally defined in Algorithm 3. The details of $\mathrm{DS}_{\mathrm{Knap}}$ are presented and discussed in the proof of Theorem 6, which provides bounds on the approximation ratio and the run-time of $\mathrm{DS}_{\mathrm{Knap}}$.

Theorem 6. *If there exists some constant* $\delta < 1$ *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 proof draws its intuition from the proof of Theorem 5, combined with the properties of the 2-approximation algorithm for Knapsack, Knap_2.

Consider a run of Knap_2, given a set of items and some budget constraint B. Knap_2 first prunes all elements with a 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 pruning 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

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

Now that Knap_2 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 $\sum_{j=1}^{t_B+1} c_{k_j} > B$. Knap_2 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 $\mathrm{O}^* \subseteq N_x$ to the DSS problem, and denote by B^* the access cost contributing to the overall cost of O^*. 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 5-3; or it will choose $\{k_{t_{B^*+1}}\}$, which is candidate $\tilde{D}_t^u$ in the iteration where $t = t_{B^*} + 1$ of lines 5-7. By the proof of Theorem 5, the best of these two candidate solutions is an $O(\beta^{\frac{\varepsilon}{1+\varepsilon}}) = O(\sqrt{\beta})$ approximate solution for the DSS problem, since we are using a 2-approximation algorithm for knapsack, implying $\varepsilon = 1$.

Since $\mathrm{DS}_{\mathrm{Knap}}$ picks the candidate solution with the 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 2-7 which has n_x iterations, where in 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. □

VII. A Partition-and-Merge Algorithmic Framework

In this section, we develop an alternative algorithm for the DSS problem and provide guarantees on its performance. We first provide a high-level description of the algorithm and then turn to a detailed description and analysis of its approximation ratio and run-time.

A. High-level Description of the Algorithm

Our proposed algorithm, $\mathrm{DS}_{\mathrm{PGM}}$ (where PGM stands for *Partition, Generate and Merge*), is based on a sequence of three basic operations: (i) *Partition* the set of data stores with positive indications N_x into disjoint subsets, based on a logarithmic scaling of the access costs, (ii) *Generate* from each subset a few candidate sets of data stores with minimal miss ratio, and (iii) *Merge* the candidate sets iteratively, until obtaining a set of candidate complete-solutions for the DSS problem.

Fig. 3 provides an illustration of the partition and generate stages of $\mathrm{DS}_{\mathrm{PGM}}$. In the *partition* stage, $\mathrm{DS}_{\mathrm{PGM}}$ partitions N_x into $r = \log \beta$ disjoint subsets: $N_0^0, N_1^0, \ldots, N_{r-1}^0$, where subset N_j^0 contains all the data stores with positive indications with an access cost that lies in the range $[2^j, 2^{j+1})$. We note that each solution $D \subseteq N_x$ to the DSS problem can be viewed as the union of r disjoint sets of the form $D_j = D \cap N_j^0$. This view will be useful in what follows as we will target bounding the cost and the miss ratio of our solution within each N_j^0 independently, for $j = 0, \ldots, r-1$. In particular, we will be considering the disjoint subsets $O_j = \mathrm{O}^* \cap N_j^0$ for some optimal solution O^* to the DSS problem.

In the *generate* stage, $\mathrm{DS}_{\mathrm{PGM}}$ sorts the data stores within each subset N_j^0 in non-decreasing order of the miss ratio and considers all possible prefixes as its initial candidate sub-solutions. We will later show that the sub-solution eventually picked by $\mathrm{DS}_{\mathrm{PGM}}$ for each $j = 1, \ldots, (r-1)$ would have cost no more than twice the cost of the said O_j, and no worse overall miss ratio than O_j.

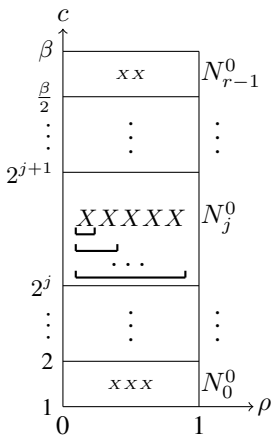


Fig. 3. $\mathrm{DS}_{\mathrm{PGM}}$'s *partition* and *generate* steps. In the y-axis, we show the possible cost ranges of the data stores contributing to each N_j^0. X represents a datastore, and within each subset N_j^0 the datastores are assumed to be sorted in non-decreasing order of miss ratio, from left to right. Marked prefixes within N_j^0 represent the candidate sub-solutions considered by the algorithm.

Fig. 4a provides and illustration of the *merge* stage, in which $\mathrm{DS}_{\mathrm{PGM}}$ iteratively merges candidate sub-solutions. The merging procedure can be viewed as generating candidate subs-solutions along a bottom-up scan of a binary tree; each leaf in this tree represent some initial candidate sub-solutions produced by the partition and generate stages, and each intermediate node represents several unions of sub-solutions available at its children. At the end of the merge stage, the algorithm picks the candidate sub-solution available at the root of the tree which minimizes the objective function $\phi(\cdot)$.

In what follows we present some preliminaries, and then use them to describe $\mathrm{DS}_{\mathrm{PGM}}$ in details.

B. Preliminaries

Denote $r = \log \beta$. We iteratively partition N_x to subsets, based on access costs, as follows. Initially, at level $\ell = 0$, we define a logarithmic-scale partitioning $N_j^0 = \left\{d \in N_x | 2^j \leq c_d < 2^{j+1}\right\}$, for $j = 0, \ldots, r-1$. In each level $\ell = 1, \ldots, \log r$ we let $N_j^\ell = N_{2j}^{\ell-1} \cup N_{2j+1}^{\ell-1}$, for $j = 0, \ldots, \frac{r}{2^\ell} - 1$. i.e., each higher-level subset is the union of two adjacent subsets in the lower level.

Given some optimal solution O^*, for each $\ell = 0, \ldots, \log r$ and $j = 0, \ldots, \frac{r}{2^\ell} - 1$, let $O_j^\ell = \mathrm{O}^* \cap N_j^\ell$, i.e., the subset of data stores which an optimal solution O^* selects out of N_j^ℓ. By the definition of N_j^ℓ, it follows that $O_j^\ell = O_{2j}^{\ell-1} \cup O_{2j+1}^{\ell-1}$ for $\ell = 1, \ldots, \log r$.

$\mathrm{DS}_{\mathrm{PGM}}$ considers *candidate subsets* of data stores $V_j^\ell \subseteq 2^{N_j^\ell}$ (i.e., a subset of the powerset of N_j^ℓ) as some specifically chosen subsets of N_j^ℓ, for each $\ell = 0, \ldots, \log r$ and $j = 0, \ldots, \frac{r}{2^\ell} - 1$.

As a data store with zero hit ratio is of no use, $\mathrm{DS}_{\mathrm{PGM}}$ assumes without loss of generality that for each $j = 1, \ldots n_x$, $p_j^h > 0$. By Eqs. 3 and 4 this implies that $\rho_j < 1$.

Algorithm 4 $\mathrm{DS}_{\mathrm{PGM}}(N_x, c, \rho, \beta)$

Partition and Generate sub-solutions

1: **for** $j = 0, 1, \ldots, r-1$ **do** ▷ $r = \log \beta$
2: $N_j^0 \leftarrow \left\{d \in N_x | 2^j \leq c_d < 2^{j+1}\right\}$
3: sort N_j^0 in non-decreasing order of miss ratio
4: $V_j^0 \leftarrow \{D | D \text{ is a prefix of } N_j^0, 0 \leq |D| \leq n_x\}$

Merge sub-solutions

5: **for** $\ell = 1, \ldots, \log r$ **do** ▷ level ℓ
6: **for** $j = 0, \ldots, \frac{r}{2^\ell} - 1$ **do** ▷ node in level ℓ
7: **for** $t = 0, 1, \ldots, r$ **do**
8: $X_t \leftarrow \arg\min\{\rho_D | D = A \cup B, \ldots$
$A \in V_{2j}^{\ell-1}, B \in V_{2j+1}^{\ell-1}, c_D \in [2^{t-1}, 2^t)\}$
9: $V_j^\ell \leftarrow V_j^\ell \cup \{X_t\}$

Pick best candidate solution

10: return $\tilde{D} = \arg\min_{X \in V_0^{\log r}} \{\phi(X)\}$

C. The $\mathrm{DS}_{\mathrm{PGM}}$ Algorithm

We now describe the details of $\mathrm{DS}_{\mathrm{PGM}}$, formally defined in Algorithm 4. In lines 1-4 $\mathrm{DS}_{\mathrm{PGM}}$ partitions the data stores and generates candidate sub-solutions as follows. First, the algorithm partitions the set of data stores with positive indications N_x into r disjoint subsets, $N_0^0, N_1^0, \ldots, N_{r-1}^0$ based on the access costs, using a logarithmic scale (line 2). Then the algorithm sorts the elements in each of the partitions by a non-decreasing order of miss ratios (line 3). Next, the algorithm generates initial candidate sub-solutions by considering all possible prefixes of each partition (line 4). Recall that the partition and generate stages are illustrated in Fig. 3. It is important to note that the *empty prefix* is also always taken as one of the candidate sub-solutions. The empty set has a cost of 0, and a miss ratio of 1.

The second part of the algorithm (lines 5-9) iteratively generates additional candidate sub-solutions across r levels. At level ℓ it merges pairs of candidate sub-solutions from level $\ell - 1$ into new candidate sub-solutions at level ℓ. This merge process essentially generates new sub-solutions across a binary tree where the leaves are the initial sets of candidate solutions $V_0^0, \ldots V_{r-1}^0$, and the root is a set of *complete* solutions for the DSS problem $V_0^{\log r}$, as depicted in Fig. 4a.

In particular, in each run of lines 7-9 $\mathrm{DS}_{\mathrm{PGM}}$ merges candidate sub-solutions corresponding to two sets of level $\ell - 1$, $V_{2j}^{\ell-1}$ and $V_{2j+1}^{\ell-1}$, into a single set V_j^ℓ of level ℓ as follows. First note that $\mathrm{DS}_{\mathrm{PGM}}$ always adds the empty set to V_j^ℓ, since for $t = 0$ we have $X_t = \emptyset$. For $t \geq 1$, the algorithm considers all the edges in the complete bipartite graph between the set of candidate sub-solutions in $V_{2j}^{\ell-1}$, on one side, and the set of candidate sub-solutions in $V_{2j+1}^{\ell-1}$, on the other side. Each edge in the bipartite graph represents the union of the two sub-solutions it connects, and is associated with (i) the *access cost* of the union of sub-solutions, which is the sum of their costs, and (ii) the *miss ratio* of the union of sub-solutions, which is the product of their miss ratios. For each $t = 1, \ldots, r$, $\mathrm{DS}_{\mathrm{PGM}}$ selects the edge (and hence union of two sub-solutions) with minimal miss ratio among all the edges with access cost in the range $\left[2^{t-1}, 2^t\right)$. This results in at most one candidate sub-solution being part of the sub-solutions in V_j^ℓ, for each of the r possible access cost ranges. After applying the merge procedure iteratively over all levels, and all nodes, the algorithm finds and returns the best candidate complete solution available at $V_0^{\log r}$ (line 10).

Fig. 4 depicts the merge stages of $\mathrm{DS}_{\mathrm{PGM}}$. In particular, Fig. 4a shows the binary merge tree. Observe that each node contains at most one sub-solution for each of the ranges $[2^{t-1}, 2^t)$, $t = 0, \ldots, r$, of the access costs. In particular, the illustration in Fig. 4 shows the details of V_0^1: the empty set (represented by an empty circle, corresponding to the range defined by $t = 0$), a single sub-solution in the range $[1, 2)$; and a single sub-solution in the range $[2, 4)$. The node V_0^1 contains no candidate sub-solution in the range $[4, 8)$. Each sub-solution is marked by some pair (a, b) where a is the sub-solution's access cost, and b is the miss ratio of the sub-solution.

Fig. 4b illustrates the merging of two sets of candidate sub-solutions into a single set in the subsequent level. The leftmost part of the figure shows the two sets of sub-solutions which $\mathrm{DS}_{\mathrm{PGM}}$ merges, $V_{2j}^{\ell-1}$ and $V_{2j+1}^{\ell-1}$. The algorithm considers all the edges in the complete bipartite graph whose vertices are the candidate sub-solutions. These edges appear as dashed edges. The middle part of the figure shows the selection of merged sub-solutions for $t = 1, 2, 3$ (since for $t = 0$ the merge simply results in the empty set).

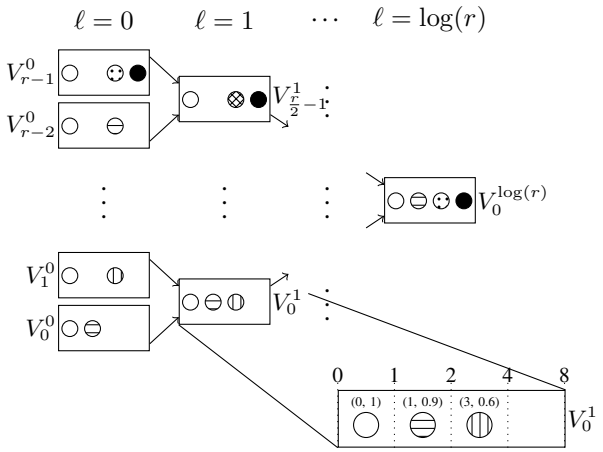
When $t = 1$, $\mathrm{DS}_{\mathrm{PGM}}$ considers all the unions of sub-solutions for which the access cost of the union is within the range $[1, 2)$. In our case, this translates to a single candidate union – the union of the empty set (represented by the empty circle) and the set of data stores with a total cost of 1. This union is hence considered as the unique candidate sub-solution corresponding to range $[1, 2)$ in V_j^ℓ, depicted in the rightmost part of the figure.

For $t = 2$, $\mathrm{DS}_{\mathrm{PGM}}$ has two edges to consider, corresponding to two sub-solutions with access cost within the range $[2, 4)$. By the definition of $\mathrm{DS}_{\mathrm{PGM}}$, it picks to retain the solution with the minimal miss ratio among the two (captured by the solid edge in the figure). This union is hence considered as the unique candidate sub-solution corresponding to range $[1, 2)$ in V_j^ℓ. We note that the chosen candidate sub-solution is actually one of the candidate sub-solutions that are part of $V_{2j+1}^{\ell-1}$ (unioned with the empty set in $V_{2j}^{\ell-1}$).

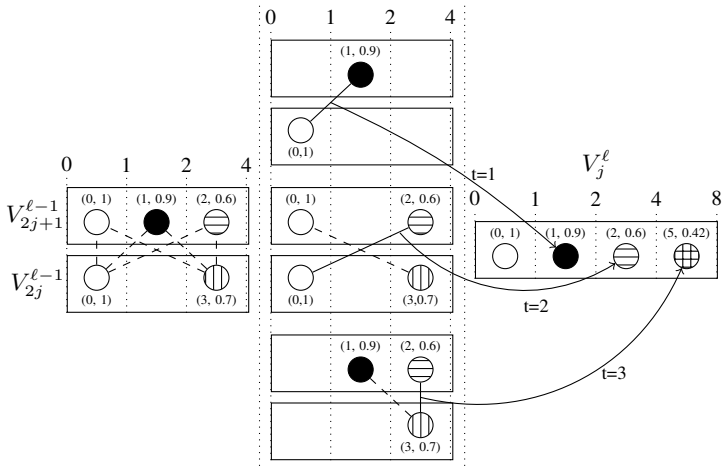
For $t = 3$, $\mathrm{DS}_{\mathrm{PGM}}$ again has two edges to consider, corresponding to two sub-solutions with access cost within the range $[4, 8)$. $\mathrm{DS}_{\mathrm{PGM}}$ maintains the sub-solution with the minimal miss ratio among the two (again captured by the solid edge in the figure). This sub-solution is the only candidate sub-solution in V_j^ℓ whose cost is in the range $[4, 8)$. We note that in this case this candidate sub-solution is taken as the union of two non-empty sets.

D. Performance and Run Time Analysis

Our performance analysis involves a careful comparison of the access cost, and miss ratio of candidate sub-solutions



(a) $\mathrm{DS}_{\mathrm{PGM}}$'s merge tree. The tree's leaves contain the initial sub-solutions (sets of data stores with positive indications), while the root contains full solutions for the DSS problem. The bottom right part shows a zoom of the node V^1_0, which contains 3 candidate sub-solutions, captured by 3 circles. The pair of values above each circle represents the access cost, miss ratio of this sub-solution. The dotted vertical lines capture log-scale ranges of the access costs. $\mathrm{DS}_{\mathrm{PGM}}$ stores at most a single sub-solution per each such range.



(b) An example of a merge of two sets of candidate sub-solutions, $V^{\ell-1}_{2j}$ and $V^{\ell-1}_{2j+1}$, into the set V^{ℓ}_j. Each edge represents a union of sets which the algorithm considers. In particular, a solid line represents a union taken for the next level, while a dashed line represents a union which the algorithm considers but decides to dismiss. Note that the algorithm takes (at most) one union of sets for each of the log-scale ranges of the access costs.

Fig. 4. $\mathrm{DS}_{\mathrm{PGM}}$'s merge step. Each circle represents a candidate sub-solution for the DSS problem.

considered by $\mathrm{DS}_{\mathrm{PGM}}$ out of every partition N^{ℓ}_j with the access cost, and miss ratio of a respective optimal sub-solution.

The following proposition shows that in each level ℓ, $\left\{N^{\ell}_j\right\}_j$ is a partition of N_x into disjoint sets, based on the access costs of the data stores.

Proposition 7. *In each level ℓ, $\left\{N^{\ell}_j\right\}_j$ is a partition of N_x, satisfying*

$$N^{\ell}_j = \left\{d \in N_x | 2^{j \cdot 2^{\ell}} \leq c_d < 2^{(j+1)2^{\ell}}\right\}. \tag{17}$$

Proof. We prove the claim by induction on ℓ. Setting $\ell = 0$ in Equation 17 simply yields the definition $N^0_j = \left\{d \in N_x | 2^j \leq c_d < 2^{j+1}\right\}$, which is a partition by definition, thus completing the base case. For the induction step, assuming that for each j in level ℓ, the set N^{ℓ}_j is part of a partition of level ℓ which satisfies Equation 17, we conclude that

$$\begin{aligned} N^{\ell+1}_j &= N^{\ell}_{2j} \cup N^{\ell}_{2j+1} = \\ &\left\{d \in N_x | 2^{2j \cdot 2^{\ell}} \leq c_d < 2^{(2j+1)2^{\ell}}\right\} \cup \\ &\left\{d \in N_x | 2^{(2j+1) \cdot 2^{\ell}} \leq c_d < 2^{(2j+2)2^{\ell}}\right\} = \\ &\left\{d \in N_x | 2^{j \cdot 2^{\ell+1}} \leq c_d < 2^{(j+1)2^{\ell+1}}\right\}, \end{aligned}$$

which therefore forms a partition of level $\ell+1$, thus completing the proof. □

In what follows, we let O^* denote some optimal solution, and let $O^{\ell}_j = \mathrm{O}^* \cap N^{\ell}_j$. We note that the proof of Proposition 7 implies that

$$N^{\log r}_0 = \left\{d \in N_x | 2^0 \leq c_d < 2^{2^{\log r}}\right\} = N_x.$$

The following corollary is therefore an immediate consequence of Proposition 7.

Corollary 8. *Given any optimal solution O^*, for any level ℓ, the set $\left\{O^{\ell}_j\right\}_j$ forms a partition of O^*, and*

$$O^{\log r}_0 = \mathrm{O}^* \cap N_x = \mathrm{O}^* \tag{18}$$

The following lemma shows that at each level ℓ, one can bound the cost and miss ratio of some candidate sub-solution available to $\mathrm{DS}_{\mathrm{PGM}}$, compared to those of an optimal sub-solution.

Lemma 9. *If $c_{\mathrm{O}^*} < \frac{\beta}{2 \cdot \log \beta}$ then for each $\ell = 0, \ldots, \log r$ and $j = 0, \ldots, \frac{r}{2^{\ell}} - 1$, V^{ℓ}_j contains a set X s.t. $c_X \leq 2^{\ell+1} \cdot c_{O^{\ell}_j}$ and $\rho_X \leq \rho_{O^{\ell}_j}$.*

Proof. Note first that for each ℓ and j s.t. $O^{\ell}_j = \emptyset$ the claim holds true since we can take $X = \emptyset$ which is always a member of V^{ℓ}_j (added in the iteration considering $t = 0$). We may therefore focus our attention only on ℓ and j for which $O^{\ell}_j \neq \emptyset$.

We prove the claim by induction over ℓ. For the base case ($\ell = 0$) we have to prove that if $c_{\mathrm{O}^*} < \frac{\beta}{2 \cdot \log \beta}$ then for each $j = 0, 1, \ldots r-1$ there exists a set of data stores $X \in V^0_j$ s.t. $c_X \leq 2 \cdot c_{O^0_j}$ and $\rho_X \leq \rho_{O^0_j}$. Denote $k_j = \left|O^0_j\right|$. As the access cost of each item in O^0_j is in $\left[2^j, 2^{j+1}\right)$, we have

$$k_j \cdot 2^j \leq c_{O^0_j} \tag{19}$$

Denote by D_j the k_j-size prefix of items in N^0_j, when sorted in non-decreasing order of miss ratio. Note that $\mathrm{DS}_{\mathrm{PGM}}$ inserts D_j to V^0_j (line 4). The access cost of each item in D_j is within $\left[2^j, 2^{j+1}\right)$, and hence

$$c_{D_j} < k_j \cdot 2^{j+1}. \tag{20}$$

Combining Equations 19 and 20, we obtain $c_{D_j} \leq 2 \cdot c_{O^0_j}$. Furthermore, as D_j is a prefix of N^0_j when sorted in a non-decreasing order of miss ratio and $|D_j| = k_j = |O^0_j|$, we have $\rho_{D_j} \leq \rho_{O^0_j}$, thus completing the proof of the induction's base.

For the induction step, we assume that the claim holds for level ℓ, and prove it for level $\ell+1$. Assume $c_{\mathrm{O}^*} < \frac{\beta}{2\log\beta}$. We have to show that there exists a set $X \in V_j^{\ell+1}$ that satisfies $c_X \leq 2^{\ell+2}\cdot c_{O_j^{\ell+1}}$ and $\rho_X \leq \rho_{O_j^{\ell+1}}$.

By the induction hypothesis, there exist sets $A^* \in V_{2j}^{\ell}$ and $B^* \in V_{2j+1}^{\ell}$ s.t. $c_{A^*} \leq 2^{\ell+1}\cdot c_{O_{2j}^{\ell}}$ and $c_{B^*} \leq 2^{\ell+1}\cdot c_{O_{2j+1}^{\ell}}$. Consider the set $D^* = A^* \cup B^*$. We first show that $D^* \neq \emptyset$. Recall that $O_j^{\ell+1} = O_{2j}^{\ell} \cup O_{2j+1}^{\ell}$. Since it suffices to focus on the case where $O_j^{\ell} \neq \emptyset$, we have either $O_{2j}^{\ell} \neq \emptyset$ or $O_{2j+1}^{\ell} \neq \emptyset$, and therefore either $\rho_{O_{2j}^{\ell}} < 1$ or $\rho_{O_{2j+1}^{\ell}} < 1$. By the induction hypothesis $\rho_{A^*} \leq \rho_{O_{2j+1}^{\ell}}$ and $\rho_{B^*} \leq \rho_{O_{2j}^{\ell}}$. Therefore, either $\rho_{A^*} < 1$ or $\rho_{B^*} < 1$. As a result, either $A^* \neq \emptyset$ or $B^* \neq \emptyset$, and hence $D^* \neq \emptyset$.

Recall that $D^* = A^* \cup B^*$, and by definition $O_j^{\ell+1} = O_{2j}^{\ell} \cup O_{2j+1}^{\ell}$. Therefore,

$$\begin{aligned} c_{D^*} &\leq c_{A^*} + c_{B^*} \\ &\leq 2^{\ell+1}\left(c_{O_{2j}^{\ell}} + c_{O_{2j+1}^{\ell}}\right) \\ &= 2^{\ell+1}\cdot c_{O_j^{\ell+1}} \\ &\leq 2^{\ell+1}\cdot c_{\mathrm{O}^*} \\ &< 2^{\ell+1}\cdot\frac{\beta}{2\cdot\log\beta}, \end{aligned} \tag{21}$$

where the second inequality is by the induction hypothesis.

Recalling that $\ell \leq \log r$ and $r = \log\beta$ we also have $2^{\ell+1} \leq 2\cdot 2^{\log(\log\beta)} = 2\cdot\log\beta$. Combining the reasoning above we have $c_{D^*} < \beta$. Therefore (and recalling that $D^* \neq \emptyset$), there exists some $t \in \{1, 2\ldots, r\}$ s.t.

$$2^{t-1} \leq c_{D^*} < 2^t. \tag{22}$$

As a result, one of the iterations of the merge loop (lines 7-9) inserts to $V_j^{\ell+1}$ a set X_t s.t.

$$2^{t-1} \leq c_{X_t} < 2^t. \tag{23}$$

Combining Equations 21, 22, and 23, we have

$$c_{X_t} < 2\cdot c_{D^*} \leq 2^{\ell+2}\cdot c_{O_j^{\ell+1}}.$$

Furthermore, by lines 8-9, X_t minimizes the miss ratio; and by the induction hypothesis we have that $\rho_{A^*} \leq \rho_{O_{2j}^{\ell}}$ and $\rho_{B^*} \leq \rho_{O_{2j+1}^{\ell}}$. We conclude that

$$\rho_{X_t} \leq \rho_{D^*} = \rho_{A^*}\cdot\rho_{B^*} \leq \rho_{O_{2j}^{\ell}}\cdot\rho_{O_{2j+1}^{\ell}} = \rho_{O_j^{\ell+1}},$$

which completes the proof. □

We are now in a position to prove an upper-bound the approximation ratio and run-time of $\mathrm{DS}_{\mathrm{PGM}}$.

Theorem 10. *$\mathrm{DS}_{\mathrm{PGM}}$ is a $(2\cdot\log\beta)$-approximation algorithm, running in time $O\left(n_x^2 + \log^3\beta\right)$.*

Proof. For proving the approximation ratio of $\mathrm{DS}_{\mathrm{PGM}}$, consider some optimal solution O^*. We recall that in every iteration – and, in particular, in the last iteration – $\mathrm{DS}_{\mathrm{PGM}}$ considers using the empty set. Hence, $\phi(\tilde{D}) \leq \phi(\emptyset) = \beta$. Therefore, if $c_{\mathrm{O}^*} \geq \frac{\beta}{2\cdot\log\beta}$, the claim is true.

If $c_{\mathrm{O}^*} < \frac{\beta}{2\cdot\log\beta}$, then by Lemma 9 there must exist a set $X \in V_0^{\log r}$ such that

$$c_X \leq 2^{\log r+1}\cdot c_{O_0^{\log r}} = 2\cdot\log\beta\cdot c_{O_0^{\log r}}.$$

Using Equation 18, we obtain

$$c_X \leq 2\log\beta\cdot c_{\mathrm{O}^*}. \tag{24}$$

Furthermore, by Lemma 9 we have

$$\rho_X \leq \rho_{\mathrm{O}^*}. \tag{25}$$

Combining Equations 24 and 25 and recalling that $\tilde{D}$ minimizes ϕ over $V_0^{\log r}$, we obtain that

$$\phi(\tilde{D}) \leq \phi(X) \leq 2\log\beta\cdot\phi(\mathrm{O}^*).$$

We now turn to analyze the runtime of algorithm $\mathrm{DS}_{\mathrm{PGM}}$. Lines 1-4 require only a single sort of the n_x data stores with positive indications, which takes $O(n_x\log n_x)$ time.

When $\ell = 1$, the worst-case run time of lines 6-9 occurs when there exists some j such that $|N_{2j}^0| = \Theta(n_x)$ and $|N_{2j+1}^0| = \Theta(n_x)$, in which case considering all the edges of the full bipartite graph between N_{2j}^0 and N_{2j+1}^0 requires $\Theta(n_x^2)$ steps.

For each $\ell = 2, 3, \ldots, \log(r)$, $\mathrm{DS}_{\mathrm{PGM}}$ inserts to V_j^{ℓ} at most one candidate sub-solution for every $t = 1, 2, \ldots, r$. Therefore when $\ell > 1$, each iteration of the merge block (lines 7-9) requires considering the complete bipartite graph where the number of nodes in each side is at most $O(r)$. Hence, each iteration of the merge block requires $\mathrm{DS}_{\mathrm{PGM}}$ $O(r^2)$ steps. As the merge tree contains $O(r)$ nodes (recall Fig. 4a), $\mathrm{DS}_{\mathrm{PGM}}$ performs $O(r)$ such merge operations. Thus, the time required to run lines 5-9 when $\ell > 1$ is $O(r^3)$. It follows that the runtime of $\mathrm{DS}_{\mathrm{PGM}}$ is $O(n_x^2 + r^3)$, which completes the proof. □

TABLE II
APPROXIMATION GUARANTEES AND RUN-TIME OF PROPOSED ALGORITHMS

Alg	Approx.	Run time	Based on
$\mathrm{DS}_{\mathrm{PP}}$	$O\left(\beta^{\frac{\varepsilon}{1+\varepsilon}}\right)$	$O\left(\left(n_x + \varepsilon^{-2.4}\right)\cdot M\right)$	Knapsack FPTAS
$\mathrm{DS}_{\mathrm{Knap}}$	$O\left(\sqrt{\beta}\right)$	$O\left(n_x^2\log n_x\right)$	Knapsack 2-approx.
$\mathrm{DS}_{\mathrm{Pot}}$	$\max_j\{C_j\}$	$O\left(n_x\log n_x\right)$	Potential function
$\mathrm{DS}_{\mathrm{PGM}}$	$O(\log(\beta))$	$O\left(n_x^2 + \log^3(\beta)\right)$	Divide & conquer

Table II compares the approximation guarantees and the run-times of our algorithms. For $\mathrm{DS}_{\mathrm{PP}}$ we use the state-of-the-art $(1+\varepsilon)$-approximation for the Knapsack problem, whose running time is $O\left(n_x + \varepsilon^{-2.4}\right)$ [31]. Our algorithms suggest various trade-offs between run-time and approximation guarantees. In particular we note that none of our algorithms strictly dominates another in terms of our analytic guarantees. The choice of the best algorithm therefore depends upon the relations between the parameters β, n_x and $\{c_j\}_j$ in a specific system, and any constraints imposed in such a system.

VIII. Simulation Study

This section provides the results of our simulation study, which uses real-life access trace, and a system deployment based on a real content distribution network topology. Our results provide insights into the performance of various access strategies in versatile settings.

A. System Topology and Costs

We use the topology of the OVH [32] content distribution network. The OVH network [32] includes 19 *Points of Presence*

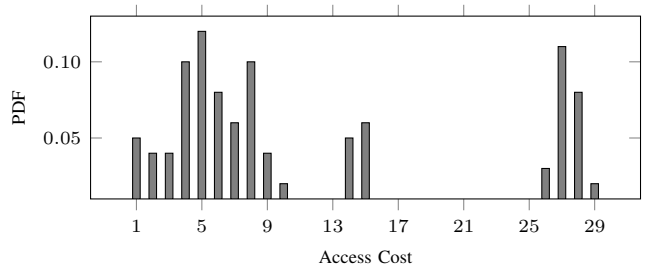


Fig. 5. Histogram of $c_{i,j}$ values for the OVH network, based on Eq. 26, using $\alpha = 0.5$ and $T = 500$.

(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:

$$c_{i,j} = \lceil 1 + \alpha \cdot \mathrm{dist}(i,j) + (1-\alpha) \cdot \frac{T}{\mathrm{BW}(i,j)} \rceil, \qquad (26)$$

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 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 $\alpha = 0.5$ and $T = \max_{i \neq j} \mathrm{BW}(i,j)$. Specifically, $T = 500$ for the OVH network.

Fig. 5 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 [27], 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 [10]. 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 [20], or by only transmitting the changes as in [4].

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

We consider 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 [33]. We consider values of k ranging from 1 to 5, where the latter value implies that an item may be placed in 26% of the 19 data stores in the system. Notice that in these experiments, the misindication ratios of different data stores are not mutually-independent, as our model assumes (see Sec. III). Depending upon the varying value of k, these misindication ratios may be either mutually independent, positively correlated, or negatively correlated. Hence, we evaluate our algorithms in versatile domains and, in particular, in domains where the misindication ratios are mutually dependent.

C. Traffic Trace, Metrics, and Simulated Scenarios

We used a publicly available Wikipedia trace [34] consisting of 357K read requests to Wikipedia pages during 5 minutes [4]. We assign each request to a random client, and the 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) $\mathrm{DS}_{\mathrm{Knap}}$, (iv) $\mathrm{DS}_{\mathrm{Pot}}$, and (v) $\mathrm{DS}_{\mathrm{PGM}}$.[5] 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. We measure the *total cost* (TC) incurred by each access strategy for serving the entire trace. We normalize the TC of each access strategy by the TC of the perfect indicator PI. This normalization aims to compare the performance in various settings while alleviating some of the exogenous effects specific to the evaluated scenario.

D. Heterogeneous Case (OVH network)

In our first experiment, we compare the performance of various access strategies when varying the number of locations per item k, and the miss penalty β. For each configuration, we measure the normalized total cost (TC). Recall that the total cost (TC) is the sum of the access cost and the miss cost.

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

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

[5]We focused our attention on the fully-polynomial approximation algorithms due to the exorbitant runtime of our pseudo-polynomial scheme.

TABLE III
OVH NETWORK SIMULATION. RESULTS PRESENT FOR EVERY SCENARIO AND EVERY POLICY THE ACCESS COST (AC) AND TOTAL COST (TC). THE VALUES ARE NORMALIZED BY THE TC OF THE PERFECT INDICATORS POLICY.

β	Policy	1 location		3 locations		5 locations	
		AC	TC	AC	TC	AC	TC
10^2	CPI	0.16	1.20	0.10	1.11	0.08	1.08
	EPI	0.23	1.09	0.43	1.39	0.49	1.55
	DS_{Knap}	0.19	1.10	0.16	1.10	0.13	1.09
	DS_{Pot}	0.20	1.11	0.18	1.13	0.20	1.16
	DS_{PGM}	0.19	1.10	0.16	1.11	0.14	1.09
10^3	CPI	0.02	1.22	0.01	1.10	0.01	1.07
	EPI	0.03	1.01	0.05	1.05	0.07	1.08
	DS_{Knap}	0.03	1.01	0.03	1.04	0.02	1.03
	DS_{Pot}	0.03	1.01	0.04	1.04	0.03	1.04
	DS_{PGM}	0.03	1.01	0.03	1.04	0.03	1.03
10^4	CPI	0.00	1.22	0.00	1.10	0.00	1.07
	EPI	0.00	1.00	0.00	1.02	0.01	1.02
	DS_{Knap}	0.00	1.00	0.00	1.02	0.00	1.02
	DS_{Pot}	0.00	1.00	0.00	1.02	0.00	1.02
	DS_{PGM}	0.00	1.00	0.00	1.02	0.00	1.02

Hence, to obtain better insights into the dominant source for the cost of access strategies, we show for each access strategy also its *access cost* (AC). We normalize both the AC and the TC of each access strategy by the TC of the perfect indicator PI. The outcome of this evaluation is provided in Table III, where we present the PI-normalized results for various β and k values.

The results show that that CPI has minimal AC across the board as it always selects the data store with the minimal access cost. When k is high, CPI obtains low total cost because when there are many true-positives, CPI is less likely to follow a false-positive indication. Also, when β is low ($\beta = 100$), the relative part of the AC within the TC is high (recall that the total cost of the PI normalizes all costs). Hence, CPI obtains the lowest TC in the combination where $k = 5$ and $\beta = 100$. However, CPI is highly sensitive to false-positives as it only accesses a single data store. Thus when k is low, there are fewer true-positive indications, which makes CPI more likely to pay the miss penalty of β.

EPI, on the other hand, is very useful for $k = 1$ but becomes less attractive as we increase k, due to the fact it ends up accessing too many data stores, which is captured by increasing AC. This effect is mitigated when β is high because a high miss penalty implies that one should better access multiple data stores in aim to minimize the probability of a miss, even at the cost of some unnecessary accesses.

Our proposed algorithms – DS_{Pot}, DS_{Knap} and DS_{PGM} – outperform the two heuristics (CPI and EPI) in all configurations, except for $\beta = 100$ and $k = 1$, where CPI obtains a slightly lower cost. Focusing on our three algorithms, DS_{Pot} does slightly worse than DS_{Knap} and DS_{PGM}, which can be explained by the higher AC of DS_{Pot}. Intuitively, DS_{Pot} optimizes for reducing the miss cost, even at the cost of a slightly higher access cost. DS_{Knap} and DS_{PGM} are the best strategies in almost all scenarios, but most importantly, they

TABLE IV
OVH NETWORK SIMULATION WITH VARYING FALSE POSITIVE (FP) RATIOS. THE MISS PENALTY IS SET TO $\beta = 100$.

Policy	1 location, varying FP				5 locations, varying FP			
	0.01	0.02	0.03	0.04	0.01	0.02	0.03	0.04
CPI	1.11	1.20	1.29	1.35	1.04	1.08	1.12	1.15
EPI	1.04	1.09	1.13	1.17	1.51	1.55	1.60	1.64
DS_{Knap}	1.06	1.10	1.14	1.18	1.05	1.09	1.12	1.14
DS_{Pot}	1.06	1.11	1.16	1.21	1.12	1.16	1.19	1.23
DS_{PGM}	1.06	1.10	1.14	1.18	1.06	1.09	1.12	1.15

are never bad strategies: even when underperforming compared to some other strategy, the difference is marginal.

Our next experiment explores the effect of the False Positive (FP) ratio on the performance of different access strategies. As seen in the results of our previous experiment, the differences are better pronounced for smaller values of β, so we consider the settings where $\beta = 100$. Furthermore, as our previous experiment implied that CPI and EPI do best when $k = 5$ and $k = 1$, respectively, we focus hereafter on these two values of k. The results are shown in Table IV. The results show that CPI and EPI are highly sensitive to the FP ratio. For instance, when $k = 1$ and $FP = 0.04$, CPI incurs an excessive cost of 35% above OPT. When $k = 5$ and $FP = 0.04$, EPI incurs an excessive cost of 64% above OPT. In contrast, all our proposed algorithms – DS_{Knap}, DS_{Pot} and DS_{PGM} – show only a mild increase in the cost when incrementing FP. In particular, DS_{Knap} and DS_{PGM} obtain again minimal, or close to minimal, costs across the board.

E. Homogeneous Case: Varying Data Store Size

Our next experiment considers homogeneous settings, where the access costs of all 19 data stores in fixed 1. We aim at studying the effect of the data store size in these settings. We examine the system's performance with a miss penalty $\beta = 100$, and a false positive ratio of 0.02. Fig. 6 shows the results for $k = 1$ and $k = 5$ locations per item, where we vary the size of each data store from 200 up to 1600. In these homogeneous cost settings, all our algorithms – namely, DS_{Pot}, DS_{Knap} and DS_{PGM} – are equivalent to the scheme which minimizes the expected overall cost, FPO. Furthermore, their

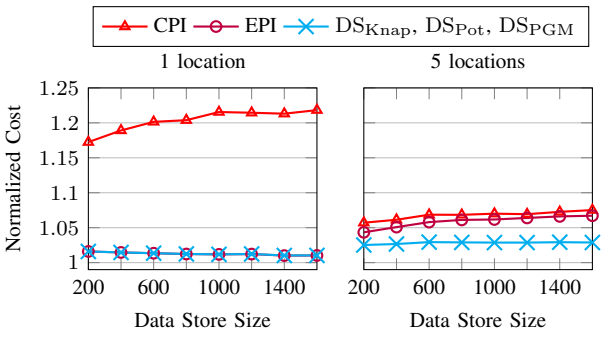


Fig. 6. Homogeneous network with varying data store size. The miss penalty is set to $\beta = 100$, and the target false positive ratio is 0.02.

performance is always very close to the one achieved with perfect indicators. In contrast to our previous experiments, CPI does not do very well, even with $k = 5$ locations per item. The reason is that in such homogeneous settings, when there exist multiple positive indications, none of them is "cheapest". As a result, CPI randomly selects a single data store. That is, CPI always accesses a data store, which neither minimizes the miss ratio nor minimizes the access cost. The homogeneous case results show again that the existing heuristics are too simplistic to fit all system configurations, whereas our algorithms are extremely robust to various system settings.

IX. 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 shows 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 lifetime, and may not be known in advance. In contrast, we present algorithms for which we provide provable performance guarantees that are robust to the various system parameters, and further demonstrate via extensive simulation that our proposed solutions exhibit near-optimal performance in various system settings.

As a future work, we aim to combine our access strategies with schemes for periodically advertising the indicators while complying constraints on the bandwidth consumption, or on the freshness of the indicators [35], [36].

Acknowledgements

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

References

[1] I. Cohen, G. Einziger, R. Friedman, and G. Scalosub, "Access strategies for network caching," in *IEEE INFOCOM*, 2019, pp. 28–36.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[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] X. Guo, T. Wang, and S. Wang, "Joint optimization of caching and routing strategies in content delivery networks: A big data case," in *IEEE ICC*, 2019, pp. 1–6.

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

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

[11] 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.

[12] 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.

[13] ——, "Counting with tinytable: Every bit counts!" in *IEEE Access*, 2019.

[14] 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.

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

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

[17] L. Luo, D. Guo, R. T. Ma, O. Rottenstreich, and X. Luo, "Optimizing bloom filter: Challenges, solutions, and comparisons," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1912–1949, 2018.

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

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

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

[21] B. M. Maggs and R. K. Sitaraman, "Algorithmic nuggets in content delivery," *ACM SIGCOMM Comp. Comm. Rev.*, vol. 45, no. 3, pp. 52–66, 2015.

[22] 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.

[23] "Squid cache wiki." [Online]. Available: https://wiki.squid-cache.org/SquidFaq/CacheDigests

[24] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, pp. 1–26, 2008.

[25] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood, "Deep packet inspection using parallel bloom filters," in *Hot Interconnects*, 2003, pp. 44–51.

[26] R. K. Grace and R. Manimegalai, "Dynamic replica placement and selection strategies in data gridsa comprehensive survey," *Journal of Parallel and Distributed Computing*, vol. 74, no. 2, pp. 2099–2108, 2014.

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

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

[29] 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.

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

[31] T. M. Chan, "Approximation schemes for 0-1 knapsack," in *Symposium on Simplicity in Algorithms (SOSA)*, 2018, pp. 5:1–5:12.

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

[33] G. Einziger, R. Friedman, and E. Kibbar, "Kaleidoscope: Adding colors to kademlia," in *IEEE P2P*, 2013, pp. 1–10.

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

[35] Y. Zhu and H. Jiang, "False rate analysis of bloom filter replicas in distributed systems," in *ICPP*, 2006, pp. 255–262.

[36] I. Cohen, G. Einziger, and G. Scalosub, "Self-adjusting advertisement of cache indicators with bandwidth constraints," in *IEEE INFOCOM*, 2021.