HotDedup: Managing Hot Data Storage at Network Edge through Optimal Distributed Deduplication

Shijing Li, Tian Lan
ECE, George Washington University
{shijing, tlan}@gwu.edu

***Abstract*—The rapid growth of computing capabilities at network edge calls for efficient management frameworks that not only considers placing hot data on edge storage for best accessibility and performance, but also makes optimal utilization of edge storage space. In this paper, we solve a joint optimization problem by exploiting both data popularity (for optimal data access performance) and data similarity (for optimal storage space efficiency). We show that the proposed optimization is NP-hard and develop a $2\lceil 2\Gamma \rceil - 1 + \epsilon$-approximation algorithm by (i) making novel use of δ-similarity graph to capture pairwise data similarity and (ii) leveraging the k-MST algorithm to solve a Prize Collecting Steiner Tree problem on the graph. The proposed algorithm is prototyped using an open-source distributed storage system, Cassandra. We evaluate its performance extensively on a real-world testbed and with respect to real-world IoT datasets. The algorithm is shown to achieve over 55% higher edge service rate and reduces request response time by about 30%.**

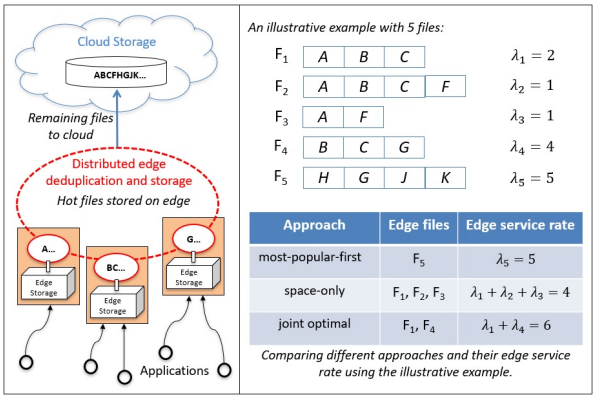


Fig. 1. An illustrative example of HotDedup at network edge.

I. Introduction

As data processing capabilities rapidly shift toward network edge in the era of edge computing, efficient management of edge storage becomes important, especially under explosive growth of Internet of Things (IoT) data. Placing hot data (i.e., popular files with frequent requests) at network edge helps increasing performance of edge applications by making hot data locally accessible, while partitioning similar files with high redundancy into the same storage tier helps improving space efficiency. These two optimizations are traditionally carried out separately, yet they are clearly coupled and need to be solved jointly to yield the best management policy. We tackle this combinatorial problem of joint optimization, and develop a novel approximation algorithm with provable performance guarantee.

Edge storage needs novel optimization for both performance and space efficiency. It is predicted that 45% of IoT-created data will be stored, processed, and acted upon at the edge of the network[1]. Modern storage systems often employ deduplication [2], [3], [4], [5], [6], [7] to substantially reduce data redundancy, e.g., for up to 70% in multimedia and traffic IoT data [8], [9]. It has recently been applied to distributed and edge systems in [3], [6]. However, these solutions only take space efficiency into account and are oblivious of data popularity and future request rates, which may vary significantly in practice. On the other hand, caching hot data at network edge have been well studied to optimize various performance objectives such as hit ratio, throughput and congestion [10], [11], [12], [13]. They aim to improve system performance but not necessarily achieves good space efficiency which is crucial for limited edge storage. Previous work has investigated how to place hot data at network edge, or select similar files for space efficiency through deduplication, but not both. Optimizing on any one dimension alone is too restrictive. Yet it remains unclear how to optimally select data that are not only popular but also amenable for space efficiency. Our paper is driven by this question.

In this paper, we propose a novel solution, HotDedup, which focuses on optimizing edge storage by exploiting both data popularity (for optimal data access performance) and similarity (for optimal storage space efficiency). The motivation of our problem is illustrated in Figure 1. Consider 5 files that are divided into fixed size chunks and have different request rates λ_i, and a distributed edge storage with a capacity of 4 chunks. To optimize the space efficiency, we can store a maximum number of 3 files $\{F_1, F_2, F_3\}$ (with a total number of 4 unique chunks after deduplication) in edge storage, whereas another policy giving higher priority to more popular (hotter) files would place only F_5 in edge storage. However, both these two solutions only serve 4 or 5 out of 13 requests per second from the network edge, i.e., $\lambda_1+\lambda_2+\lambda_3 = 4$ and $\lambda_5 = 5$, while the remaining requests must be served by the remote cloud. We show that an optimal solution of placing $\{F_1, F_4\}$ on network edge can serve the maximum of $\lambda_1 + \lambda_4 = 6$ requests, thus achieving both space efficiency and best performance.

While exploiting both degrees of freedom is undoubtedly appealing, it also presents great technical challenge. It is easy to see that even without deduplication, the problem of maximizing edge service rate (defined as the total request rate served by edge storage) under edge capacity constraint

is equivalent to the knapsack problem, which is known to be NP-hard. However, the use of deduplication further introduces a combinatorial constraint, since in order to find the best partition of files, the deduplicated storage size must be measured on every possible subset of files, yet with no closed-form characterization available. In this paper, we propose a $2\lceil 2\Gamma \rceil - 1 + \epsilon$-approximation algorithm for the problem, where Γ is the storage capacity divided by actual storage usage(Section V). Our solution leverages two key ideas: First, for any set of files, the storage space requirement after deduplication can be effectively approximated through a δ-similarity graph $G = \{V, E\}$, in which each vertex represents a file F_i and an edge weight between vertices i to j measures the number of common chunks shared by F_i and F_j (i.e., capturing pairwise file similarity). Second, we note that the partitioning problem now on δ-similarity graph is closely related to a class of problems known as the Quota version of the Prize Collecting Steiner Tree problem [14]. To this end, we make novel use of the k-Minimum-Spanning-Tree (k-MST) algorithm [15] to iteratively compute an optimal solution. The approximation ratio of our proposed algorithm is derived in closed form. The performance is evaluated using prototype implementation and real-world IoT datasets.

Our contributions are as follows:

- To manage hot data storage at network edge, we formalize a joint optimization problem to maximize edge service rate and storage space efficiency with deduplication. The problem is shown to be NP-hard.
- We propose a $2\lceil 2\Gamma \rceil - 1 + \epsilon$-approximation algorithm by making novel use of δ-similarity graph to capture pairwise file similarity and leveraging k-MST to solve a Prize Collecting Steiner Tree problem on the graph.
- The proposed algorithm is prototyped using a distributed storage system, Cassandra, and evaluated with real-world IoT datasets. It achieves over 55% higher edge service rate and reduces request response time by about 30%.

II. Related Work

IoT data such as multimedia and traffic video image sequences have high similarities due to their temporal and/or geographical correlation, which could result in more than 70% redundancy[8], [9]. Space-efficient mechanisms for managing IoT data include clustering algorithms based on data similarity that could save 55% storage space and improve system resource utilization[2], and methods leveraging temporal/geographical pattern in IoT time series data[3]. The notion of clustered deduplication [5], [4] has also been considered in general cloud storage, where systems like HYDRAstor [5] perform coarse-grained deduplication with larger chunk size and then distribute the data using DHTs or load balancers to multiple servers that perform more fine-grained deduplication. However, these work mainly focus on improving space efficiency with deduplicaiton, while not taking into account data popularity in edge processing.

Caching hot data at network edge have been well studied to optimize the hit ratio of data requests. A lazy eviction cache algorithm for cloud block storage is proposed in [10] to efficiently mitigate large reuse distances in cache blocks. Cache replacement policies for maximizing hit ratio or total throughput under the uncertainty of data arrivals are considered in [11], [12], [13], by considering peak hour characteristics [11], estimating the distribution and characteristics of Web proxy traces [12], and incorporating data locality to optimize multiple objectives including hit ratios, latency reduction and network cost reduction[13]. In contrast, our work considers not only selecting hot data for edge storage to optimize system performance, but also edge-based deduplication of IoT data to eliminate substantial redundancy and improve space efficiency of edge storage.

III. System Model and Problem Formulation

In the era of edge computing, an increasing amount of data that are produced at network edge (e.g., by smart phones and IoT sensors) will be accessed by computing nodes or edge devices that are also located at network edge. We consider a distributed storage system consisting of two-tiers - a set of n distributed edge nodes that are located close to data producers and consumers and a remote cloud with larger storage capacity and yet higher service latency.

Given a set of m files (or data objects) $\mathcal{F} = \{F_1, F_2, \ldots, F_m\}$ with known request arrival rates $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_m\}$, we consider the problem of optimally placing the files on edge and cloud storage, with the goal of maximizing the probability of serving requests from edge (or equivalently, minimizing the need to forward request to remote cloud). In order words, we partition $\mathcal{F}$ into two disjoint subsets - $\mathcal{P}$ for edge storage and $\mathcal{Q}$ for cloud storage - such that the amount of requests that are served by edge storage (i.e., $\sum_{i \in \mathcal{P}} \lambda_i$) is maximized.

Since data generated by IoT and edge applications often exhibits significant amount of redundancy, we employ a distributed storage system with edge-based deduplication similar to [6]. In particular, we maintain a distributed hash-chunk table among the edge nodes, which stores the hash value of each unique chunk as the key and can be accessed by all edge nodes to identify and deduplicate redundant chunks in new files. To store the deduplicated chunks, we also create another file-hash table to maintain an association between file names and the position indexes of corresponding chunks, while the chunks are evenly distributed among all edge nodes. To reconstruct a file, we will use the file-hash table to get the position index of each chunk, and use the hash value to link to the actual chunk in hash-chunk table. We consider each file F_i as a set of chunks, such that $|F_i|$ defines its size. We use $D(\mathcal{P}) = |\cup_i F_i|$ to denote the space required for all edge files after distributed deduplication.

Our objective is to maximize the amount of requests that are served by edge storage, which is denoted as the *edge service rate* in this paper, by optimally partitioning the files. Let C_k be the storage capacity of edge node k, so that the

$\mathcal{F}$	A set of m files
F_i	File i with file size $\|F_i\|$
λ_i	Request arrival rate for file i
$R(t)$	Chunk popularity density function
$B = \sum_k \mathcal{C}_k$	Edge storage capacity constraint
$G = (V, E)$	δ-similarity graph
$D(T)$	Deduplicated storage size of files in T
$S(T)$	Total edge plus vertex cost of T
$Q(T)$	Total vertex prize of T
μ_i	Cost assigned to vertex i
v_e	Cost assigned to edge $e = (i, j)$

TABLE I
TABLE OF KEY NOTATIONS

total edge storage capacity is $B = \sum_k C_k$. We need to solve an *partitioning problem*:

$$\max \quad \sum_{i \in \mathcal{P}} \lambda_i \tag{1}$$

$$\text{s.t.} \quad D(\mathcal{P}) \leq B = \sum_k C_k, \tag{2}$$

$$\text{var.} \quad \mathcal{P} \subseteq \mathcal{F}. \tag{3}$$

where Equation (2) is a storage capacity constraint for the edge nodes under deduplication $D(\mathcal{P})$. Despite its straightforward formulation, this problem turns out to be challenging, since the storage space requirement $D(\mathcal{P})$ depends on the redundancy existing in files $\mathcal{P}$ and does not have a closed-form quantification. Next, we will show that the problem is NP-hard even if there is no redundancy between the files and $D(\mathcal{P})$ reduces to a linear function.

Theorem 1. *The partitioning problem is NP hard.*

Proof. We prove that the problem is NP-hard by showing that the knapsack problem (which is known to be NP-hard) can be transformed into a version of the proposed partitioning problem with zero redundancy between any pair of files. More precisely, we consider a set of m items, each of size s_i and associated with a reward r_i for $i = 1, \ldots, m$, and a knapsack of size C. The knapsack problem aims to find a subset $\mathcal{P}$ of items that have a total size no greater than C and achieves maximum reward, i.e., to maximize $\sum_{i \in \mathcal{P}} r_i$ under $\sum_{i \in \mathcal{P}} s_i \leq C$.

Now we construct a set of m files, each with s_i unique chunks and a request rate of $\lambda_i = r_i$. Since there is no chunks shared by any pair of files, for any subset $\mathcal{P}$ of files we have $D(\mathcal{P}) = \sum_{i \in \mathcal{P}} s_i$. When $B = \sum_k C_k = C$, it is easy to see that the knapsack problem is exactly a special case of the proposed partitioning problem with zero redundancy between any pair of files. Thus, we conclude that any solutions to the partitioning problem solves the knapsack problem, which implies the NP-hardness.

□

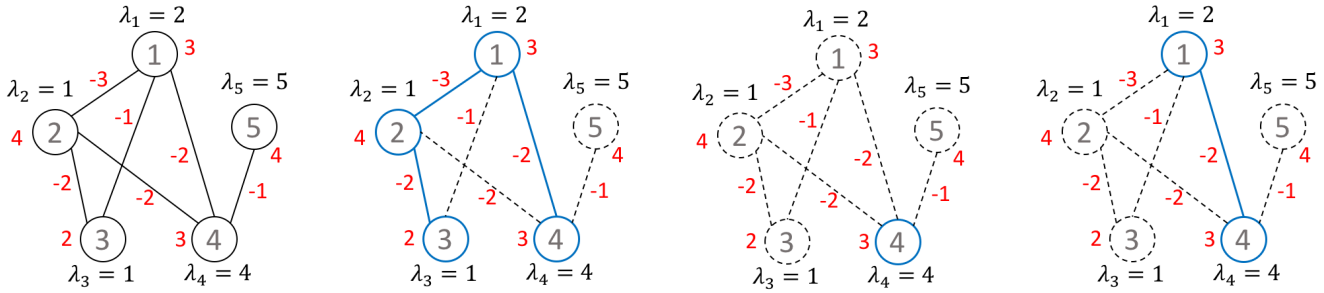
IV. OUR ALGORITHM USING DELTA STEINER TREE

Our algorithm makes novel use of two key ideas. First, for a set of files $\mathcal{P}$, the space requirement after deduplication - i.e., $D(\mathcal{P})$ - can be effectively estimated via a δ-similarity graph $G = \{V, E\}$, in which each vertex represents a file in $\mathcal{F}$ and an edge exists between vertices i to j if the two files F_i and F_j share at least one common chunk. We also assign a vertex cost $\mu_i = D(F_i)$ (which is the size of file F_i) to each vertex i and an edge cost $v_{(i,j)} = -D(F_i \cap F_j)$ (which is the negative of the size of common chunks shared by files F_i and F_j) to each edge $(i, j) \in E$. It has been shown that such δ-similarity graphs - while only capturing pairwise file similarities - provide an simple yet effective model for estimating the space requirement of deduplicating any subset of files [7]. Thus, we can consider an alternative problem of finding a subgraph on G with respect to an estimate of storage space constraint $B = \sum_k C_k$. Second, we note that this problem now is related to a class of problems known as the Quota version of the Prize Collecting Steiner Tree problem[14], a.k.a., the quota problem. Given the δ-similarity graph G, we consider a quota problem with both vertex and edge costs μ_i and $v_{(i,j)}$ in , respectively, and vertex prizes λ_i. For a given prize quota $Q > 0$, the goal of the quota problem is to find a tree T in G of minimum *vertex plus edge* cost, which spans a total prize of at least Q. We will show that the proposed partitioning problem can be efficiently solved by computing a sequence of quota problems on the δ-similarity graph and finding a proper prize quota Q under estimated storage space constraint. In particular, the vertex plus edge cost provides an estimate on the storage space required to store files in the optimal tree T, while the prize on T gives the corresponding edge service rate. Our proposed algorithm is denoted as the Delta Steiner Tree Partitioning (DSTP) in this paper.

A. Illustrative Example

We will first give an illustration of our DSTP algorithm using the example in Fig.1 with $m = 5$ files and edge storage capacity $B = \sum_k C_k = 4$. First, we construct the δ-similarity graph $G = (V, E)$ as shown in Fig .2(a). For instance, vertex 1 is assigned a cost $\mu_1 = D(F_1) = 3$ (chunks) and a prize $\lambda_1 = 2$, while edge $(1, 2)$ is assigned a cost $v_{(1,2)} = -D(F_1 \cap F_2) = -3$. There is no edge between vertices 1 and 5 because files F_1 and F_5 do not share any common chunk. Intuitively, for any subgraph of G, the sum of edge and vertex costs provide an estimate on the required storage space using deduplication, by considering only pairwise similarities. Now we start by considering a quota problem on the δ-similarity graph $G = (V, E)$ in Fig .2(a), with an initial prize quota $Q = 8$. The optimal tree that achieves the minimum vertex plus edge cost while spanning a total prize of at least $Q = 8$ is shown in Fig .2(b). We note that the vertex plus edge cost of this optimal tree - that is $\sum_{i=1}^{4} \mu_i + v_{(1,2)} + v_{(2,3)} + v_{(1,4)} = 5$ - coincide with the required storage space using deduplication, which is $D(\{F_1, F_2, F_3, F_4\}) = 5$. In general, such estimates using δ-similarity graph G may not be precise, but still provide a simple yet effective model for optimizing storage systems with deduplication, as proven later in this paper.

Next, we adjust the prize quota Q until the resulting solution to the quota problem has a vertex plus edge cost satisfying the storage capacity constraint $B = 4$. This can be achieved through a bisection search. Since the current



(a) δ-similarity graph constructed for the example in Fig.1. (b) Optimal solution (cost=$6 > B$) to the quota problem with $Q = 8$. (c) Optimal solution (cost=$3 < B$) to the quota problem with $Q = 4$ (d) Optimal solution (cost= $4 = B$) to the quota problem with $Q = 6$

Fig. 2. An illustration of our proposed DSTP algorithm. It finds a solution with the highest prize $Q = 6$ under a cost constraint $B = 4$, by leveraging a bisection search on Q and solving a sequence of quota problems accordingly. Files $\mathcal{P} = \{1, 4\}$ are placed at edge storage in the optimal solution.

cost exceeds capacity constraint, we reduce the prize quota to $Q = (8 + 0)/2 = 4$. A solution to this new quota problem on the δ-similarity graph $G = (V, E)$ is computed and the result is shown in Fig .2(c). Now, since the vertex plus edge cost is only $\mu_4 = 3$ and falls below the capacity constraint, we make another adjustment to consider $Q = (4 + 8)/2 = 6$. We show the resulting optimal tree in Fig .2(d) with cost $\mu_4+\mu_1+\upsilon_{(1,4)} = 4$. It is easy to see that any higher prize quota will lead to a solution violating the storage capacity constraint. Thus, we stop the bisection search, and get an optimal solution $P = \{F_1, F_4\}$ to the partitioning problem in (1)-(3), i.e., to store files F_1 and F_4 on network edge. In the following we will show that this DSTP algorithm is guaranteed to find a feasible solution to the partitioning problem despite using an estimate of storage space via the δ-similarity graph. Further, we will drive a closed-form approximation ratio for the DSTP algorithm.

B. The Proposed DSTP Algorithm

Given a set of m files $\mathcal{F}$, we construct a δ-similarity graph $G = dupGraph(\mathcal{F}) = (V, E)$. This can be computed efficiently with complexity $O(m^2)$ by generating a hash table for all chunks of each file F_i and comparing the hash tables of each pair of files to find the edge costs. Thus, we can run a bisection search on prize quota Q and solve a sequence of quota problems on the δ-similarity graph, until we find a value Q^* such that the tree produced by solving the quota problem for prize quota Q^* costs no more than $B = \sum_k C_k$ and any tree output by the algorithm for quota value $Q^*(1 + \epsilon)$ costs more than B for some $\epsilon > 0$. We note that a bisection search to finite precision suffices, since there exists an exponentially small η such that any two trees of graph G must have vertex plus edge cost greater than η (as there are only a finite number of trees). The bisection search is also guaranteed to converge since the minimum cost is monotonically non-decreasing over prize quota Q.

Now we only need to develop an efficient algorithm to solve the quota problem with given Q on the δ-similarity graph. To this end, we leverage the method in [14] to transform the quota problem into an equivalent k-Minimum-Spanning-Tree problem (k-MST), which finds a k-vertex, minimum edge cost spanning tree in G and is known to have quadratic-time algorithms with an approximation ratio of $(2 + \epsilon)$ [15]. We note that the method in [14] just works for graphs which only containing non-negative edge costs, whereas our quota problem has both edge and vertex costs, and the edge costs are negative. Without loss of generality, we assume that all vertex prizes have integer values. This holds because without changing the problem, we can always scale λ_i by some sufficiently large factor S, such that the quantization error due to ceiling function $\lceil \lambda_i S \rceil$ can be sufficiently small as relative to the scaled prize $\lambda_i S$.

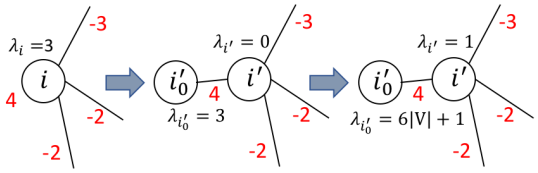


Fig. 3. An illustration of our transformation by substituting each vertex i of graph $G = (V, E)$ into 2 vertices of graph G', in which only edge costs exist. The vertex prizes are then scaled up by $\hat{\lambda}_s = 2|V|\lambda_s + 1$ for all vertex s of G'.

Our key idea is to transform graph G into G' (that only has edge costs) by substituting every vertex i by a vertex i' of zero prize and attaching a new node i'_0 with prize λ_i to vertex i'. The process is illustrated in Fig 3. If we further assign edge cost μ_i to (i', i'_0), it is easy to verify that the original quota problem on G with both edge and vertex costs is equivalent to a quota problem on G' with only edge costs. Next, to deal with zero-prize vertices we introduce in G', we will employ the technique in [14] and modify prize values in G' so as to get an equivalent instance of the quota problem with only positive prizes on the vertices. The new prize assignment is $\hat{\lambda}_s = 2|V'|\lambda_s + 1$ for all vertex s of $G' = (V', E')$. We also scale up the prize quota to be $\hat{Q} = 2Q|V'|$. The following lemma proves that any tree that is optimal for the quota problem under the modified prize assignments is optimal for the original problem before vertex prize scaling.

Lemma 1. *If tree T is an optimal solution to the quota problem with modified vertex prize $\hat{\lambda}_s$ and quota $\hat{Q}$, then T is an optimal solution to the original quota problem with vertex prize λ_s and quota Q.*

Proof. This lemma follows directly from Lemma 4.1 in [14], and having negative edge costs does not affect vertex prize. □

Finally, the quota problem on G' with modified vertex prize $\hat{\lambda}_s$ and quota $\hat{Q}$ can be solved by computing an equivalent k-MST problem. We just need to substitute every vertex s by a star with a center node s' and $\hat{\lambda}_s - 1$ leaf nodes all attached to s' by zero-cost edges [14]. Both s' and the leaf nodes are assigned unit vertex prize. Clearly, solving the quota problem is now equivalent to solving k-MST problem on the modified graph with $k = \hat{Q}$, because the modified graph only contain 1-cost vertices. More precisely, any k-MST algorithm that finds a k-vertex, minimum edge cost spanning tree achieves a collective prize of exactly $k = \hat{Q}$ as all vertices have unit prize. Based on these transformations, we can also deal with negative edge costs by adding a sufficiently large constant α on all edges to make their costs become non-negative. Since k-MST algorithm finds a k-vertex tree that has exactly $k-1$ edges, the additional constant α will only introduce a constant change $\alpha(k-1)$ in the optimal solution of k-MST. Let k-MST(G', $\hat{Q}$) be a routine that returns optimal tree T. The proposed DSTP algorithm is summarized below in Algorithm 1.

Algorithm 1: *Our DSTP Algorithm*

Input : $\mathcal{F} = \{F_1, \ldots, F_m\}$, edge storage capacity B;
Algorithm :
Construct $G = (V, E) = dupGraph(\mathcal{F})$;
//Transform G into G'
for each edge i
  Substitute i by i' and i'_0;
  Assign vertex prize: $\lambda_{i'} \leftarrow 0$ and $\lambda_{i'_0} \leftarrow \lambda_i$;
  Assign edge cost: $v_{(i',i'_0)} = \mu_i$;
end for
Scale prize: $\hat{\lambda}_s \leftarrow 2|V'|\lambda_s + 1$ for all s in G';
//Solve quota problems
Initialize Q_L and Q_U for bisection search;
do {
  $\hat{Q} \leftarrow (Q_L + Q_U)/2$;
  $T \leftarrow k$-MST(G', $\hat{Q}$);
  Estimate storage space: $S = \sum_{e \in T} v_e(G')$;
  if $S > B$ xxx Quota $\hat{Q}$ is too high
    Update: $Q_L \leftarrow \hat{Q}$;
  else Update: $Q_U \leftarrow \hat{Q}$;
  end if
} **while** $|S - B| > \epsilon$;
return T;

Theorem 2. *Algorithm DSTP finds a feasible solution for the partitioning problem.*

Proof. k-MST problem is known to have quadratic-time algorithms with an approximation ratio of $(2 + \epsilon)$ [15]. We have already shown that solving the k-MST problem with $k = \hat{Q}$ yields a solution T to the quota problem on G' with modified vertex prize $\hat{\lambda}_s$ and quota $\hat{Q}$. Lemma 1 further establishes that T is a solution to the original quota problem on G with vertex prize λ_s and quota $Q = \hat{Q}/(2|V'|)$.

Let $D(T)$ be the deduplication function in (1). To show that T is also a feasible solution to the partitioning problem, we only need to prove $D(T) \leq B$. Due to the bisection search in Algorithm DSTP, we have $\sum_{e \in T} v_e(G') \leq B$ for edge costs $v_e(G')$ on G'. This implies that $\sum_{e \in T} v_e(G) + \sum_{i \in T} \mu_i(G) = \sum_{e \in T} v_e(G') \leq B$, because of the additional edge costs that are introduced to construct G'. Using the definition of G as the δ-similarity graph, we have

$$\begin{aligned} D(\mathcal{T}) &= |\cup_{i \in T} F_i| \\ &\leq \sum_{i \in T} |F_i| - \sum_{(i,j) \in T} |F_i \cap F_j| \\ &= \sum_{e \in T} v_e(G) + \sum_{i \in T} \mu_i(G) \leq B. \end{aligned} \tag{4}$$

where the third step follows from the construction of δ-similarity graph G, and the second step holds since each unique chunk shared by t files (a.k.a popularity-t chunks) is counted t times in $\sum_i |F_i|$, while $\sum_{(i,j) \in T} |F_i \cap F_j|$ only include a subset of those popularity-t chunks that are counted at most $t-1$ times if their owner files are all adjacent and form a subtree in T. Therefore, T is a feasible solution to the partitioning problem due to $D(T) \leq B$. □

V. Approximation Ratio of DSTP

We show that DSTP is a ($2\lceil 2\Gamma \rceil$ - 1 + ϵ)-approximation algorithm, where $\Gamma = B/D(T) \geq 1$ is the edge storage capacity B divided by actual usage $D(T)$. We also provide a closed-form bound on Γ and propose a hierarchical model and quantify Γ on practical datasets.

A. Main Result.

It is easy to see that the performance of DSTP algorithm depends on two factors: (i) The optimality of using k-MST to solve the quota problem on G, and (ii) The gap of using edge plus vertex cost of δ-similarity graph to estimate deduplicated storage space in the quota problem. We will analyze these two factors and use them to jointly derive an approximation ratio for the proposed DSTP algorithm. For a set of m files $\mathcal{F} = \{F_1, \ldots, F_m\}$, we define chunk popularity function $R(t)$ as the number of unique chunks shared by exactly t files. In the following we analyze the gap Γ of estimating deduplicated storage space using a spanning tree on G.

Lemma 2. *For a set of m files $\mathcal{F}$ and corresponding δ-similarity graph $G = dupGraph(\mathcal{F})$, we have for any minimum spanning tree T:*

$$\Gamma = \frac{S(T)}{D(T)} \leq \sum_t R(t) \left[t - \frac{t^2 - t}{m} \right], \tag{5}$$

where $R(t)$ is the chunk popularity function, $S(T) = \sum_{i\in T}\mu_i + \sum_{e\in T} v_e$ is the total edge plus vertex cost of T, and $D(T)$ is the deduplication storage space.

Lemma 2 quantifies the gap Γ with respect to a chunk popularity function $R(t)$. It is similar to the result proven in [7], which however, only applies to a directed $delta$-graph. We present a different proof for undirected graph in Lemma 2, and its proof is shown in Appendix. For practical systems, such $R(t)$ can be empirically obtained for any data sources. It is evaluated in [7] that in real-world dataset, most chunks have small popularity values and thus $R(t)$ is heavily left-skewed, leading to a tight bound of Γ (very close to 1) in Lemma 2. Next, we prove another lemma that provides an important bound on splitting node-weighted trees. It plays a crucial role in analyzing the competitive ratio of DSTP algorithm. Again, the proof is collected in Appendix.

Lemma 3. *For $\forall\alpha \geq 3$, any node-weighted tree with at least α nodes, in which no node weighs more than $\frac{1}{2\alpha-1}$ of the total weight of the tree, can be split into α edge-disjoint subtrees, such that each of them contains at least $\frac{1}{2\alpha-1}$ of the total weight.*

Now we are ready to state the main theorem of this paper.

Theorem 3. *DSTP yields a polynomial-time $(2\lceil 2\Gamma\rceil - 1 + \epsilon)$-approximation algorithm for the partitioning problem.*

Proof. We prove the theorem by constructing a contradiction. Let tree T be a solution computed from DSTP algorithm. Suppose that no polynomial-time algorithm can achieve an approximation ratio less than or equal to $2\lceil 2\Gamma\rceil - 1 + \epsilon$. Then, there must exist some solution T^* that is not only feasible $D(T^*) \leq B$ but also achieve a large prize $Q(T^*)$ satisfying:

$$Q(T^*) \geq (2\lceil 2\Gamma\rceil - 1 + \epsilon)Q(T), \tag{6}$$

because otherwise $Q(T)$ would attain the desired approximation ratio. We note that T^* must be the minimum spanning tree on the subgraph containing its vertices, because we can otherwise replace T^* by the minimum spanning tree to have a feasible solution with the same vertex prize and yet smaller cost. Applying Lemma 2 to T^* we get:

$$S(T^*) \leq \Gamma D(T^*) \leq \Gamma B, \tag{7}$$

where the last step holds due to the feasibility of T^*, i.e., $D(T^*) \leq B$. Next, we choose $\alpha = \lceil 2\Gamma\rceil$ in Lemma 3 and use it to split T^* into α trees that each of them contains prize at least:

$$\frac{Q(T^*)}{2\lceil 2\Gamma\rceil - 1} \geq \frac{(2\lceil 2\Gamma\rceil - 1 + \epsilon)Q(T)}{2\lceil 2\Gamma\rceil - 1} > Q(T), \tag{8}$$

where the second step is due to (6). Since the total cost of these α split trees is less than that of T^*, there must exist one split tree with cost no more than:

$$\frac{S(T^*)}{\alpha} \leq \frac{\Gamma B}{\lceil 2\Gamma\rceil} \leq \frac{B}{2}, \tag{9}$$

where the second step uses (7) and $\alpha = \lceil 2\Gamma\rceil$. To summarize, we now find a split tree of T^*, who achieves a prize strictly higher than $Q(T)$ (due to (8)) and have a cost no more than $B/2$ (due to (9)).

However, T is a solution computed from DSTP algorithm by solving a sequence of quota problems with bisection search. Quota problems are known to have quadratic-time algorithms with an approximation ratio of $(2+\epsilon)$ [15]. It implies that any solution achieving a higher prize quota than $Q(T)$ must have cost more than $B/2$ (otherwise T would not be a $(2+\epsilon)$-approximation solution). This contradicts with the fact that we have constructed a split tree of T^* above with a prize strictly higher than $Q(T)$ and a cost no more than $B/2$. Therefore, we prove the desired result. □

Remark. We note that for datasets in practice, Γ can be very close to 1, since most data chunks have small popularity value. Empirical evaluation of popularity distribution $R(t)$ has been provided in [7]. Using the result, we can compute a bound for Γ using Lemma 2, i.e., $\Gamma \leq 1.26$. Further plugging this into our approximation ratio in Theorem 3, it becomes $2\lceil 2\Gamma\rceil - 1 + \epsilon = 2\lceil 2 \times 1.26\rceil - 1 = 5 + \epsilon$. In other words, DSTP is a $(5+\epsilon)$-approximation algorithm on the dataset.

VI. Evaluation

We implement a prototype of the proposed DSTP algorithm and evaluate its performance using real-world IoT data. In this section, we describe our system design and present the results of extensive experiments and simulations that show the efficacy of our proposed algorithm over other baselines.

A. Implementation and Experiment Setup

Our implementation of edge-based deduplication is illustrated in Fig.1. When a set of new files $(F_1, F_2, \ldots, F_5)$ arrive at the edge nodes, we split each file object into fixed-size chunks and calculate a hash value for each chunk. The unique hash values are maintained in a distributed edge storage, using Cassandra [16], [17], and allows efficient deduplication of new file objects using Duperemove [18], as well as quick construction of the δ-similarity graph as shown in Fig.III. Then, we run DSTP algorithm at edge nodes to determine the optimal set of files to store on the edge nodes. In particular, another index table in Cassandra is created to store the mapping from chunks to their original positions in file objects, to allow quick recovery at any edge node. When an access request arrives, edge files are reconstructed locally on the fly, while cloud files are retrieved from the remote cloud.

Our local storage system testbed consists of 20 VMs (*edge nodes*) created on a local Cassandra cluster, where each VM has 2 VCPUs, 2GB RAM and 5 GB virtual disk drive. We also set up four more powerful VMs in a cloud Cassandra cluster, in which each VM has 4 VCPUs, 8GB RAM, and 20GB storage. We implement a distributed Cassandra file storage on local VMs. All of the local VMs are gathered in the same Cassandra cluster. In this way, file objects stored at network edge will have their chunks evenly distributed among all nodes in the

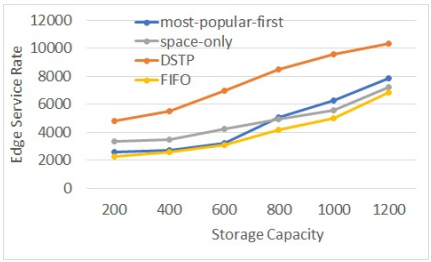


Fig. 4. We compare the performance of DSTP against the baseline algorithms. It is shown that DSTP is able to outperform the baselines by 43.4-118.5% in terms of edge service rate.

(a) Edge service rate

(b) Deduplication ratio

Fig. 5. We evaluate the performance of DSTP solution with respect to a mix of two datasets. Files are randomly selected from the datasets with varying probabilities p and $1-p$, respectively.

cluster. We maintain a table indicating the order of chunks in different file objects, which can thus be successfully recreated upon access requests.

The bandwidth among the local VMs is 1.726 Gbps with an average latency of 0.85 ms. We measured the average bandwidth between our local VMs and Amazon EC2 VMs is 0.377 Gbps with average latency of 12.2 ms. Then we use NetEm[19] to set the bandwidth and latency between local Cassandra cluster and cloud Cassandra cluster equal as the measurements. Then we use real-world IoT datasets for the evaluation. The first one consists of accelerometer data recording human walking over 25 days from 5 participants, with each data point in the size range of 80-187MB [20]. The seconds one is MIT traffic data set which is for research on activity analysis and crowded scenes. It includes video sequences captured by a stationary traffic camera[21]. All numbers presented in our evaluations are an average of 100 measurements over a real system we implemented.

B. Numerical Evaluation and Comparison

We evaluate the proposed DSTP algorithm and compare its performance with a number of baselines. The first baseline is a most-popular-first algorithm, which aims to store the hottest file objects with the highest access rate on edge nodes. This is indeed a Most-Frequently-Used policy widely employed in caching systems[22], [13]. The second baseline is a space-only algorithm, which only takes file similarity into account and maximizes the deduplication ratio of selected files, similar to those used in deduplication systems[6]. Finally, the third baseline employs the First In First Out (FIFO) heuristic, which store the latest files at network edge. In this section, we compare the performance of DSTP and the baselines, with respect to edge service rate (defined as the total request rate that are served by edge storage), response time for accessing different file objects, and the deduplication ratio that measures the space-efficiency of edge storage.

We first vary the edge storage capacity from 200MB to 1.2GB (per node) and evaluate DSTP algorithm and baselines, in terms of their optimal edge service rate. Fig. V-A shows that while all algorithms achieve better edge service rate due to higher storage capacity available for hot files, our algorithm is able to outperform the baselines by 43.4-118.5%. The maximum improvement is achieved at medium edge storage capacity, because there are either too few or too many files in low and high edge storage, leading to less room for optimization. Nevertheless, even with very limited local storage space - 200MB per edge node - our algorithm could still achieve 44.5% improvement over the baselines. It can more effectively utilize edge storage space to support hot files.

We also evaluate our DSTP algorithm on a mix of two different IoT accelerometer datasets, in which higher file similarity (and thus higher potential for deduplication) exist within each dataset, while less similarity across datasets. We randomly select files from the two datasets with probabilities p and $1-p$. Figure. V-A shows the optimal edge service rate for different values of p. It is observed that the best performance of DSTP is attained when p is close to 50% (with about 35% increase over the performance at $p = 0$ and $p = 1$). This is because when files are files are uniformly chosen from the two datasets, DSTP is able to "cherry-pick" the hot files with both high request rate and high similarity from each data set, thus achieving the best edge service rate. Fig. V-A verifies this by showing the deduplication ratio (which is defined as the storage space ratio before and after deduplication and is always greater than or equal to 1) achieved by files from each dataset for different values of p. Deduplication ratio saturates as soon as a small number of hot files from each dataset are stored on the edge. DSTP's ability to deliver better performance when the data become more hybrid makes it suitable for practical applicators.

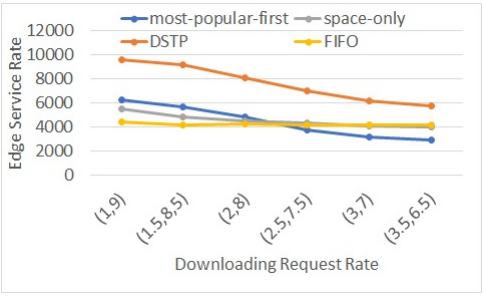
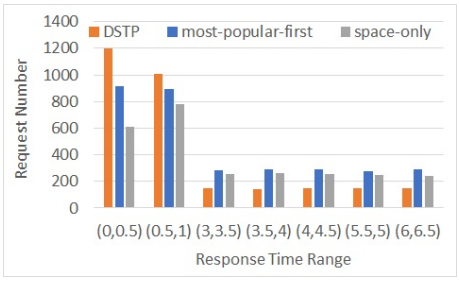
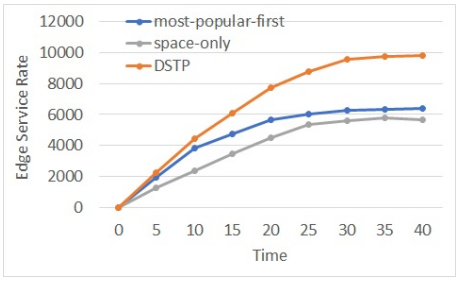


Fig. 6. Comparing DSTP and baseline algorithms with respect to different distributions of request rate (i.e., file popularity). DSTP demonstrates superior ability to adapt to different file request distributions and performs better for larger variations in file popularity.

(a) Mean response Time



(b) Distribution of response time



(c) Online re-optimization

Fig. 7. We evaluate the performance of DSTP on our testbed for randomly generated requests over a period of 100 minutes, in terms of the actual request response time and the ability of online re-optimization. DSTP can significantly reduce response time and adapt to dynamic new file arrivals.

Next, we changed the distribution of file request rates (i.e., file popularity) to compare DSTP with the baselines. Fig. 6 shows the performance when file access requests are generated according to a uniform distribution in different range (x, y). DSTP demonstrates superior ability to adapt to different file request distributions. In particular, it almost doubles the edge service rate under uniform request distribution in $(1, 9)$, which corresponds to large variation in file popularity and allows DSTP to better optimize over select hot files with high access rate. Such high variation in file popularity often hold in practical systems due to timeliness of IoT data.

In Fig. 7, we evaluate the performance of DSTP on our testbed described at the beginning of this section, in terms of the actual request response time in Fig. 7(a) and Fig. 7(b), and the ability to re-optimize file placement with online new file arrivals Fig. 7(c). The numbers shown in the figure is the sum of 500 requests (for both edge and cloud files) randomly generated according to a uniform distribution in $[1, 10]$ (requests per second). In general, downloading a file from edge nodes requires less than 1 second service time on our testbed, while 3-6.5 seconds from the cloud, due to smaller latency and higher bandwidth between clients and edge nodes. As shown in Fig. 7(a), while most-popular-first algorithm is able to store the hottest files at network edge, it results in the worst request responds time, since the hottest files may have low similarity and thus poor space efficiency for edge storage. On the other hand, space-only algorithm is able to store the maximum number of files at network edge, but it does not take file popularity into account. Our DSTP algorithm is able to achieve the minimum response time, as it is able to maximize the number of requests served by network edge and also achieve high space efficiency. DSTP is able to achieve smaller response time for a larger fraction of files and significantly reduces response time tails.

Finally, we apply DSTP to an online setting. At the beginning of each time slot, a batch of new files are randomly selected from the two datasets with equal probability and their requests are generated from a uniform distribution in $(0, 10)$. With new files arriving in each time slot, we re-optimize the storage system by solving DSTP optimization jointly for both existing files and new files. As shown in Fig. 7(c), we measure the edge service rate at the end of each time slot, for a total of 40 time slots. It is shown that as more files arrive at the system, edge service rate increases, and eventually saturates due to storage capacity constraint at network edge. DSTP achieves the largest service rate and outperforms most-popular-first and space-only by 53% and 72%, respectively. This validates the benefit of DSTP for online optimization with new file arrivals.

VII. Conclusion

Data management at network edge needs to exploit both data popularity (for optimal data access performance) and data similarity (for optimal storage space efficiency). In this paper, we formulate a joint optimization problem to maximize edge service rate and storage space efficiency with deduplication. The problem is shown to be NP-hard. Our proposed $2\lceil 2\Gamma - 1 + \epsilon$-approximation algorithm makes novel use of δ-similarity graph to capture pairwise data similarity and leverages the k-MST algorithm to solve a Prize Collecting Steiner Tree problem on the graph. Its prototype and evaluation using real-world testbed and datasets validates significant improvement in edge service rate and reduction in request response time.

Appendix

A. Proof of Lemma 2

Proof. We denote $\mathcal{T}$ as the set of all possible trees of the δ-similarity graph G. Due to the construction of G in Section IV, we have vertex cost $\mu_i = D(F_i)$ (which is the size of file F_i) and edge cost $v_{(i,j)} = -D(F_i \cap F_j)$ (which is the negative of the size of common chunks shared by files F_i and F_j). Since T is an optimal spanning tree with minimum weight $S(T)$, the weight of T can be bounded by the average weight of all spanning trees of G. We have

$$\begin{aligned} S(T) &\leq \frac{1}{|\mathcal{T}|}\sum_{T\in\mathcal{T}} S(T) \\ &= \frac{1}{|\mathcal{T}|}\sum_{T\in\mathcal{T}}\left(\sum_{i\in T}\mu_i + \sum_{(i,j)\in T} v_{(i,j)}\right) \\ &= \sum_i \mu_i - \frac{2}{m}\sum_{i,j:i<j} v_{(i,j)} \\ &= \sum_i D(F_i) - \frac{2}{m}\sum_{i,j:i<j} D(F_i \cap F_j), \end{aligned} \quad (10)$$

where the second step uses the definition of $S(T)$ in Lemma 3, the last step directly follows from the construction of of G in Section IV, and the third holds because each spanning tree traverses all vertices in G and thus has the same total vertex weight, and because there each edge shows in exactly $2/m$ spanning trees (since all edges have equal chance to be in the spanning trees due to symmetry).

Next, we consider a chunk popularity function $N(t)$, defined as the number of unique chunks shared by exactly t files. It is easy to see that we have $R(t) = N(t)/[\sum_t N(t)]$, as the fraction of chunks with popularity t. Using $N(t)$, we can rewrite rewrite deduplicated storage space $D(T)$. Sine each unique chunk of popularity t is counted exactly once in the value of $N(t)$, or in other words, the sum of $N(t)$ gives the total number of unique chunks, we have

$$D(T) \quad = D(\cup_i F_i) = \sum_t N(t). \tag{11}$$

Similarly, we can use $N(t)$ to rewrite $\sum_i D(F_i)$ and $\sum_{i,j:i<j} D(F_i \cap F_j)$ in (10). In particular, because a chunk of of popularity t exists in exactly t different files, we have

$$\textstyle\sum_i D(F_i) \quad = \sum_t t \cdot N(t). \tag{12}$$

Further, the fact that a chunk of of popularity t exists in exactly t different files also implies that the chunk will be contained in $t - choose - 2$ different pairwise intersections, $D(F_i \cap F_j)$. We have

$$\begin{aligned} \textstyle\sum_{i,j:i<j} D(F_i \cap F_j) \quad &= \sum_t \binom{t}{2} \cdot N(t), \\ &= \frac{t(t-1)}{2} \cdot N(t). \end{aligned} \tag{13}$$

Finally, plugging (11), (12), (13) into (10), we obtain:

$$\begin{aligned} \tfrac{S(T)}{D(T)} \quad &\leq \frac{\sum_i D(F_i) - \frac{2}{m}\sum_{i,j:i<j} D(F_i \cap F_j)}{D(T)}, \\ &= \frac{\sum_t t \cdot N(t) - \sum_t \frac{2}{m} \cdot \frac{t(t-1)}{2} \cdot N(t)}{\sum_t N(t)}, \\ &= \sum_t \left(t - \frac{t^2 - t}{m} \right) \cdot \left[N(t) / \sum_{t'} N(t') \right] \\ &= \sum_t \left(t - \frac{t^2 - t}{m} \right) \cdot R(t) \triangleq \Gamma \end{aligned} \tag{14}$$

where the last step uses $R(t) = N(t)/[\sum_t N(t)]$. This completes the proof. □

B. Proof of Lemma 3

Proof. Without loss of generality, we assume the total weight of all nodes is 1. Let $w(T)$ denote the total weight of tree T. We first prove the tree can be split into α subtrees($\alpha \geq 3$), where two of them weigh at least $\frac{1}{2\alpha-1}$. Then we prove that each subtree weighs at least $\frac{1}{2\alpha-1}$.

There are at least 3 nodes, so we can split them into at least 3 subtrees. First, we prove the tree can be split into α subtrees($\alpha \geq 3$), where two of them weigh at least $\frac{1}{2\alpha-1}$. If this assumption is not possible, then there exists only one subtrees T_1 whose total weight is at least $\frac{1}{2\alpha-1}$ and other $\alpha-1$ subtrees weigh less than $\frac{1}{2\alpha-1}$. Suppose T_2 is the subtree with largest total weights in those $\alpha - 1$ small subtrees. Then $w(T_2) < \frac{1}{2\alpha-1}$. The sum of all of the remaining $\alpha-2$ subtrees' weights $w_{remaining} < \frac{\alpha-2}{2\alpha-1}$.

Let v be the vertex which is shared by T_1 and T_2. If node v has only one incident edge in T_1. Then, if we move this edge from T_1 to T_2, creating two new subtrees $T_1^{'}$ and $T_2^{'}$. Then,

$$\begin{aligned} w(T_1^{'}) \quad &= \quad w(T_1) - w(v) \\ &\geq \quad 1 - \frac{\alpha-2}{2\alpha-1} - \frac{1}{2\alpha-1} - \frac{1}{2\alpha-1} \\ &> \quad \frac{\alpha-1}{2\alpha-1} > \frac{1}{2\alpha-1} \end{aligned} \tag{15}$$

$$w(T_2^{'}) \quad = \quad w(T_2) + w(v) \geq \frac{1}{2\alpha-1} \tag{16}$$

Then, both of $T_1^{'}$ and $T_2^{'}$ weigh at least $\frac{1}{2\alpha-1}$, which is contradiction.

If vertex v has at least two children in T_2, we can split T_2 into two subtrees, both rooted at v. The larger of the two subtrees must weigh at least $\frac{\alpha-1}{2\alpha-1}$. Similarly, by moving the smaller subtree of T_2 over to T_1, we have:

$$w(T_1{}') \quad > \quad \frac{1}{2\alpha-1} + \frac{\alpha-1}{2\alpha-1}) \geq \frac{1}{2\alpha-1} \tag{17}$$

$$\begin{aligned} w(T_2^{'}) \quad &\geq \quad 1 - \frac{1}{2\alpha-1} - \frac{\alpha-1}{2\alpha-1} \\ &= \quad \frac{\alpha-1}{2\alpha-1} \geq \frac{1}{2\alpha-1} \end{aligned} \tag{18}$$

Then, both of $T_1^{'}$ and $T_2^{'}$ weigh at least $\frac{1}{2\alpha-1}$, which is contradiction.

We already proved the tree can be split into α subtrees($\alpha \geq 3$), where two of them weigh at least $\frac{1}{2\alpha-1}$. So there exists an edge-disjoint split T_1 and T_2 where $w(T_1) \leq w(T_2)$ and both of them weigh at least $\frac{1}{2\alpha-1}$. Consider all the possible edge-disjoint splits into two subtree. We denote the smaller tree as T_1, and the other one as T_2. Then T_1 must weigh more than $\frac{\alpha-1}{2\alpha-1}$, otherwise T_2 could weigh at least $\frac{\alpha}{2\alpha-1}$, causing contradiction. Let v be the vertex which is shared by T_1 and T_2. If node v has only one incident edge in T_1, then if we move this edge from T_1 to T_2, we will create two new subtrees $T_1^{'}$ and $T_2^{'}$. Suppose w(T) means the total weights of a tree called T. Then, since in the original split $w(T_2) > \frac{2\alpha-2}{2\alpha-1}$, we have

$$w(T_1^{'}) = w(T_1) - w(v) > \frac{\alpha-1}{2\alpha-1} - \frac{1}{2\alpha-1} > \frac{\alpha-2}{2\alpha-1} \tag{19}$$

This is a contradiction. So this tree can be split into 2 edge-disjoint subtrees, such that one of them contains at least $\frac{1}{2\alpha-1}$ of the total weight and the other one contains at least $\frac{\alpha}{2\alpha-1}$ of the total weight.

Since no node weighs more than $\frac{1}{2\alpha-1} < \frac{1}{\alpha}$, we can take the larger subtree and split it further into α subtrees, each of weight at least $\frac{1}{\alpha}$ of the total. It is easy to see that we obtain α subtrees, each containing at least $\frac{1}{2\alpha-1}$ of the total node weight. □

REFERENCES

[1] R. Clarke, R. Westervelt, D. Vesset, M. Torchia, A. Siviero, R. Segal, K. Prouty, V. Turner, C. MacGillivray, and L. Lamy, "Idc futurescape: Worldwide internet of things 2016 predictions," *IDC FutureScape*, 2016.

[2] X. Tao and C. Ji, "Clustering massive small data for iot," in *The 2014 2nd International Conference on Systems and Informatics (ICSAI 2014)*. IEEE, 2014, pp. 974–978.

[3] S. Aljawarneh, V. Radhakrishna, P. V. Kumar, and V. Janaki, "A similarity measure for temporal pattern discovery in time series data generated by iot," in *2016 International conference on engineering & MIS (ICEMIS)*. IEEE, 2016, pp. 1–4.

[4] W. Dong, F. Douglis, K. Li, R. H. Patterson, S. Reddy, and P. Shilane, "Tradeoffs in scalable data routing for deduplication clusters." in *FAST*, vol. 11, 2011, pp. 15–29.

[5] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki, "Hydrastor: A scalable secondary storage." in *FAST*, vol. 9, 2009, pp. 197–210.

[6] B. B. M.-R. R. H. W. L. R. P. Shijing Li, Tian Lan, "Ef-dedup: Enabling collaborative data deduplication at the network edge," in *2019 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019.

[7] B. Balasubramanian, T. Lan, and M. Chiang, "Sap: Similarity-aware partitioning for efficient cloud storage," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 592–600.

[8] Y. Zhang, Y. Wu, and G. Yang, "Droplet: A distributed solution of data deduplication," in *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*. IEEE Computer Society, 2012, pp. 114–121.

[9] H. Yan, X. Li, Y. Wang, and C. Jia, "Centralized duplicate removal video storage system with privacy preservation in iot," *Sensors*, vol. 18, no. 6, p. 1814, 2018.

[10] K. Zhou, Y. Zhang, P. Huang, H. Wang, Y. Ji, B. Cheng, and Y. Liu, "Lea: A lazy eviction algorithm for ssd cache in cloud block storage," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*. IEEE, 2018, pp. 569–572.

[11] M. Ma and V. W. Wong, "An optimal peak hour content server cache update scheduling algorithm for 5g hetnets," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[12] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker *et al.*, "Web caching and zipf-like distributions: Evidence and implications," in *Ieee Infocom*, vol. 1, no. 1. INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE), 1999, pp. 126–134.

[13] P. Cao and S. Irani, "Cost-aware www proxy caching algorithms." in *Usenix symposium on internet technologies and systems*, vol. 12, no. 97, 1997, pp. 193–206.

[14] M. Minkoff, "The prize collecting steiner tree problem," Ph.D. dissertation, Massachusetts Institute of Technology, 2000.

[15] S. Arora and G. Karakostas, "A 2+ ϵ approximation algorithm for the k-mst problem," *Mathematical Programming*, vol. 107, no. 3, pp. 491–504, 2006.

[16] A. Lakshman and P. Malik, "Cassandra: structured storage system on a p2p network," in *Proceedings of the 28th ACM symposium on Principles of distributed computing*, ser. PODC '09. New York, NY, USA: ACM, 2009, pp. 5–5. [Online]. Available: http://doi.acm.org/10.1145/1582716.1582722

[17] ——, "Cassandra: a decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1773912.1773922

[18] "Duperemove," https://github.com/markfasheh/duperemove.

[19] S. Hemminger *et al.*, "Network emulation with netem," in *Linux conf au*, 2005, pp. 18–23.

[20] M. Cong, K. Kim, M. Gorlatova, J. Sarik, J. Kymissis, and G. Zussman, "CRAWDAD dataset columbia/kinetic (v. 2014-05-13)," Downloaded from https://crawdad.org/columbia/kinetic/20140513/kinetic-energy, May 2014, traceset: kinetic-energy.

[21] X. Wang, X. Ma, and W. E. L. Grimson, "Unsupervised activity perception in crowded and complicated scenes using hierarchical bayesian models," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 31, no. 3, pp. 539–555, 2008.

[22] A. A. Rocha, M. Dehghan, T. Salonidis, T. He, and D. Towsley, "Dsca: A data stream caching algorithm," in *Proceedings of the 1st Workshop on Content Caching and Delivery in Wireless Networks*. ACM, 2016, p. 8.