Joint source selection and transfer optimization for erasure coding storage system

Han Zhang*‡, Xingang Shi†‡, YingYa Guo*‡,Haijun Geng§, Zhiliang Wang†‡,Xia Yin*‡ *Department of Computer Science and Technology, Tsinghua University
†Institute for Network Sciences and Cyberspace, Tsinghua University
‡Tsinghua National Laboratory for Information Science and Technology (TNLIST)
Beijing, P.R. China
§school of software engineering, shanxi university
Email:{zhanghan, wzl, yxia,guoyingya}@csnet1.cs.tsinghua.edu.cn,
{shixg}@cernet.edu.cn,{genghaijun}@sxu.edu.cn

***Abstract*—With the deployment of big data applications, more and more data are stored in the online storage. Erasure coding storage system has been widely used by companies such as Google and Facebook, since it provides space-optimal data redundancy to protect against data loss. In erasure coding storage system, (n, k) MDS erasure code is used to divide file into n chunks. When a user want to access the file, any subset of k out of n chunks will be needed to reconstruct the file. In this case, how to select k out of n chunks and how to let the trunks transfer quickly become important problems. In this paper, we joint the two problems together to optimize. Our optimization goal is to minimize average file access time (FAT). To achieve this, we propose smallest load first heuristic to do source selection and design an online algorithm to reduce trunk transfer latency. Base on this, we design and implement D-Target, a centralized scheduler that tries to minimize average FAT in distributed erasure coding storage system. We then test D-Target's performance by trace-driven simulation. Results show that, for the trace of AT&T, D-Target performs 2.5×, 1.7×, 1.8×, 3.6× better than TCP, Aalo, Barrat and pFabric respectively.**

I. Introduction

Social networking and e-commerce activities are popular these days and more and more data are stored in the online storage. Also, businesses are relying on big data analytics for business intelligence and are migrating their traditional IT infrastructure to the cloud. In this trend, cloud storage services like Amazon's Cloud drive, Apple's iCloud, DropBox, Google Drive, Microsoft's SkyDrive, AT&T Locker stores redundant information on distributed servers to increase reliability for storage systems.

Erasure coding has been widely used by companies such as Google and Facebook [23] [27], since it provides space-optimal data redundancy to protect against data loss. In erasure coding storage system, (n, k) MDS erasure code is used to divide file into n chunks. When a user fetches the file, any subset of k out of n chunks are needed to reconstruct it. As Fig. 1 shown, chunks are stored at cloud servers. Clients receive requests from customers and the scheduler selects sources for each request. Then the client receives the chunks, reconstructs the file and sends the file to users. Critical factor that affects the service quality is the delay in accessing the stored file.

We can see two processes affect the accessing latency of the distributed erasure coding storage system. Firstly, inefficient source selection can lead to flows conflict at server nodes. Modern distributed erasure coding storage systems always use random source selection for requests. This is simple, however, random source selection will make some nodes have heavy load and this will magnify latency. To prevent this from happening, servers that have small load should be chosen to reduce the conflicts. Secondly, tcp is not a good choice for distributed storage system. TCP is fair transfer method, but in erasure coding storage system, a file can be reconstructed only when the client receives all the chunks. For TCP transfer, different chunks may get different bandwidth and the last finish chunk may finish late. This indeed wastes bandwidth as the early finish one should wait for the late ones to reconstruct the file.

In this paper, we try to minimize average file access time (FAT) for distributed erasure coding storage system. We joint source selection and trunk transfer together to optimize. At first, we formulate the Idealized File Access Time Minimization (IFATM) Problem. Then we ignore the source selection and propose the Simple Idealized File Access Time Minimization (SIFATM) problem . We then prove even the Simple Idealized File Access Time Minimization (SIFATM) problem is NP-hard. After this, we propose *smallest load first* heuristic

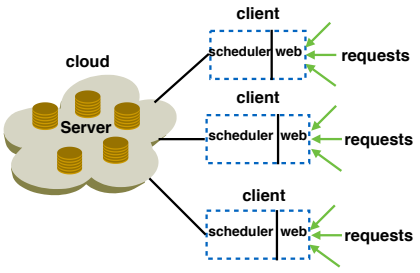


Fig. 1: Structure of distribute erasure coding storage system. Chunks are stored at the servers in the cloud and the clients receive requests and select sources for each request.

to select sources for requests. For the trunk transfer, we use the 2-approximate online heuristic to determine the scheduling order of trunk transfer and uses *Minimum-Allocation-for-Desired-Duration* (MADD) to compute the bandwidth to be allocated. Then we evaluate D-Target, a scheduler that selects sources and allocate bandwidth for trunk transfer. We at last use the trace of AT&T to test the performance of D-Target. Results show that D-Target performs D-Target performs 2.5×, 1.7×, 1.8×, 3.6× better than TCP, Aalo [8], Barrat [14] and pFabric [5] respectively.

Thus the contribution we make in this paper:

- We are the first to joint source selection and trunk transfer together to optimize in distributed erasure coding storage system.
- We formulate Idealized File Access Time Minimization (IFATM) problem, study its hardness, and derive a non-preemptive scheduling algorithm that is 2-approximate optimal.
- We further design D-Target, a scheduler to select efficient sources and allocate bandwidth to trunks.
- Trace-driven simulation shows that D-Target performs D-Target performs 2.5×, 1.7×, 1.8×, 3.6× better than TCP, Barrat [14], Aalo [8], pFabric [5] respectively.

The rest of the paper is organized as follows. At Section II, we use a small example to show the necessity of source selection and task-level transfer optimization. Section III introduces the related work. Section IV formulates the Idealized File Access Time Minimization (IFATM) Problem and discusses its hardness. Section V presents the source selection and the design of D-Target. Section VI evaluates D-Target against several task scheduling methods, and Section VII concludes the paper.

II. MOTIVATION

Erasure coding has been widely used by distributed storage system. Files are encoded into chunks and chunks are stored at different machines. When accessing a file, machine set should be chosen from the chunk nodes to reconstruct the file. A critical factor that affects the user experiences is the delay in accessing the stored file. However, the bandwidth between different nodes is frequently limited and so is the bandwidth from a user to different storage nodes, which can cause significant delay in data access and perceived as poor quality of service. In this section, we show the necessity of source selection and task-level transfer optimization for erasure coded storage system.

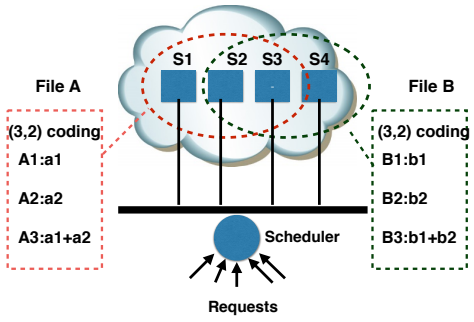


Fig. 2: An erasure-coding storage system contains 2 files and they are partitioned with (3, 2) MDS codes.

In erasure coding storage system, files are encoded into blocks using maximum distance separable (MDS) codes. Under an (n, k) MDS code, a file is encoded and stored in n storage nodes such that the chunks stored in any k of these n nodes suffice to recover the entire file. As Fig. 2 shown, we consider two files A and B, both encoded and stored at $n = 3$ machines. A request to retrieve file A can be completed after it is successfully processed by 2 distinct nodes chosen from $\{S1, S2, S3\}$. It is similar for file B, which can be chosen from $\{S2, S3, S4\}$. To simplify the problem, we assume capacity of all links is 1 and chunk size of file A and file B is also 1. Firstly, we consider two requests R_A and R_B that arrive simultaneously at $t = 0$ to fetch file A and file B respectively. For R_A, file A has 3 source selection options: $(S1, S2), (S1, S3), (S2, S3)$, while file B has $(S2, S3), (S2, S4), (S3, S4)$. If the scheduler uses random source selection, a common case maybe R_A chooses $(S2, S3)$ and R_B chooses $(S3, S4)$. If the transfer policy is TCP, then the file access time for R_A is $t_A = 2$ and for R_B is $t_B = 2$. The AFAT(average file access time) is 2. However, if R_A chooses $(S1, S2)$ and R_B chooses $(S3, S4)$, the AFAT is 1. From the simple example, we can see, for the erasure coding storage system, efficient source selection for request can reduce AFAT, thus user experience will be better.

What's more, only tcp-fair transfer is not enough for the distributed storage system. Just think the simple case, there are two request for file A, R_{A_1} arrives at $t = 0$ and R_{A_2} arrives at $t = 0.1$. R_{A_1} chooses$(S1, S2)$ as the sources and R_{A_2} chooses $(S2, S3)$ as the sources. For TCP transfer, FAT(file access time) for R_{A_1} is $t = 1.9$ and FAT for R_{A_2} is $t = 2.0$, so that AFAT(average file access time) is $(1.9 + 2.0)/2 = 1.95$. However, if we let R_{A_1} have higher priority than R_{A_2}. Then FAT for R_{A_1} is $t = 1$ and FAT for R_{A_2} is $t = 2$, so that AFAT is $(1 + 2)/2 = 1.5$. We can see, incorporating task level optimization can improve much on reducing AFAT for file transfer in distributed storage system.

From the two simple examples, we can see that both source selection and file transfer optimization play important roles in reducing FAT(file access time). In this paper, we join the two problems together to optimize.

III. BACKGROUND AND RELATED WORK

It has been reported that the storage space used for photo storage only in Facebook has been over 20 PB in 2011 and is increasing by 60 TB every week [6] [17]. As massive data stores on disk, there can be a large number of failures everyday. To compensate for the data loss and thus to guarantee the integrity of data stored in the cloud, it is natural to store replicas in multiple disks, such that data losses can be tolerated as long as there is at least one replica available [17]. However, just storing the reputation of data can reduce the efficiency of storage space. Erasure coding is used to encode data into a set

of trunks and a file can be reconstructed by a subset of total trunks. This method can compensate for the data loss and thus to guarantee the integrity of data stored in the cloud.

Although erasure coding storage system can guarantee the integrity of data, however, as reading or writing data, the system needs to encode or decode data, leading to a high access latency and a low access throughput due to the CPU limit, so that previous work [18], [11], [13], [12] mainly focus on efficient encoding and decoding. By efficient encoding technology, overhead of CPU will significantly reduce. However, though erasure coding stores data as multiple coded blocks, when accessing file or one coded block getting lost, the system must get multiple coded blocks that are sufficient to recover all the data. This process can add large burden to network. Indeed, latency optimization has attracted more and more attention, even Google and Amazon have published that every additional 500ms can lead to 1.2% user loss [24]. Optimizing the transfer latency of shares is very important, as this can improve user experience, thus revenue. Still now, lots of methods have been proposed to reduce applications' transfer latency in data center. According to schedule granularity, we can divide them into two kinds: flow level optimization and task level optimization.

DCTCP [4], D^{2}TCP [25], L^{2}DCT [21], PDQ [15], pFabric [5], LPD [29], D^3 [26] are flow level methods. DCTCP is fair sharing method and it tries to maintain the switch's queue shallow to reduce queue delay. However, in data center, applications have different demands of bandwidth and fair sharing is not a good choice [30]. D^{2}TCP, LPD, D^3 are deadline-aware bandwidth allocation methods. They set explicit deadline to flows which have different emergency, so that tight flows will get more bandwidth than the lax ones. As a result, applications' demand will be satisfied. L^{2}DCT, PDQ, pFabric try to minimize average flow completion time. They assume short flows are always the emergency ones which need to finish transfer as fast as possible. They throttle the bandwidth of large flows and spare the bandwidth to the short ones. Then the short flows transfer faster than the large ones. As a result, average flow completion time reduces. Although flow-level optimization methods perform better than TCP on latency optimization, however, in data center network, distributed applications always have parallel flows and just flow-level optimization is not enough. Task-level schedule methods such as Barrat [14], Varys [9], Aalo [8], sunflow [16] regard flows of applications as a whole. Barrat [14] schedules tasks in FIFO order but avoids head-of-line blocking by dynamically changing the level of multiplexing in the network. Varys [9] regards applications' parallel flows as coflow and it uses SEBF to schedule coflows and MADD to perform rate control, as a result, average task completion time will reduce. Aalo [8] ,sunflows [16], and CODA [28] do not need to know coflow information beforehand. In erasure coding storage system, file access will generate parallel flows. We think just flow level optimization is not enough, task level schedule methods should be considered to make the transfer efficiency.

IV. Network model and Analysis

In this section, we first introduce the data center non-blocking model, and then propose Idealized File Access Time Minimization (IFATM) Problem. At last, we simplify the problem and prove even the simplified IFATM problem is NP-hard.

A. Network model

Recent studies [5], [9], [16], [8], regard the data center as a big switch, where all ports have normalized united capacity. All flows compete for the ingress and egress bandwidth. Such abstraction is reasonable and matches with recent full bisection bandwidth topologies widely used in current production data centers [19]. In this paper, we use this assumption and only take the contention of ingress and egress ports into consideration.

B. Problem formulation

We consider a non-blocking data center that consists of n storage nodes, denoted by $\mathcal{S} = \{s_1, s_2, ...s_n\}$. Then r files, denoted by $\mathcal{F} = \{f_1, f_2, f_3..f_r\}$ are stored in the server set. For each file f_i, we partition it into k_i fixed-size trunks and then encode it using an (n_i, k_i) MDS erasure code to generate n_i distinct trunks of the same size for f_i. The n_i distinct shares are stored at n_i machines. The use of (n_i, k_i) MDS erasure code allows the file to be reconstructed from any subset of k_i-out-of-n_i trunks, whereas it also introduces a redundancy factor of n_i/k_i. The m clients, denoted by $\mathcal{C} = \{c_1, c_2...c_m\}$. Assume all the L requests arrive at time 0 and the $k-th$ request denoted by $T_{f^k}^{(k)}$ means requesting file $f^k (f^k \in \mathcal{F})$. The k-th request $T_{f^k}^{(k)}$ consists of parallel subtasks, each subtask is a flow from a server to the client. $T_{f^k}^{(k)} = \{t_{i,j}^{(k,f^k)} | 1 \le i \le n, 1 \le j \le m\}$, where $t_{i,j}^{(k,f^k)}$ presents a sub-task whose size is $t_{i,j}^{(k,f^k)}$. $x_{i,j}^{(k,f^k)} \in \{0,1\}$, where 1 denotes that there is a flow from server i to client j and 0 denotes there is no flow from server i to client j. As the non-blocking model's ingress and egress ports have unit capacity, so the transfer time for sub-task $t_{i,j}^{(k,f^k)}$ is $t_{i,j}^{(k,f^k)}$. The problem of non-preemptive Idealized File Access Time Minimization (IFATM) Problem can be defined as:

$$\text{minimize} \quad \sum_{g=1}^{L} C_g \tag{1}$$

$$\text{s.t.} \quad \forall g, j \sum_{\forall l: C_l \le C_g} \sum_{i=1}^{n} t_{i,j}^{(l,f^l)} * x_{i,j}^{(l,f^l)} \le C_g \tag{2}$$

$$\forall g, i \sum_{\forall l: C_l \le C_g} \sum_{j=1}^{m} t_{i,j}^{(l,f^l)} * x_{i,j}^{(l,f^l)} \le C_g \tag{3}$$

$$\forall l, j \sum_{i=1}^{n} x_{i,j}^{(l,f^l)} = k_{f^l} \tag{4}$$

Our goal is to minimize the average file access time. The constraints (2) and (3) are due to the competition on the capacity of each port. For a request $T_{f^k}^{(k)}$ with completion time

C_k, consider the set of file transfer that finish before it, i.e., $T_{f^l}^{(l)}$: $C_l \leq C_k$. For any ingress port i (or egress port j), the file access time on this port is at least $\sum_{\forall l:C_l \leq C_g} \sum_{i=1}^{n} t_{i,j}^{(l,f^l)} * x_{i,j}^{(l,f^l)}$(or $\sum_{\forall l:C_l \leq C_g} \sum_{j=1}^{m} t_{i,j}^{(l,f^l)} * x_{i,j}^{(l,f^l)}$, correspondingly), which must no longer than C_k. (4) describes source selection, which means that the scheduler should selects k_{f^l} servers for file f^l.

C. NP-hard proof

In this part, we prove Idealized File Access Time Minimization (IFATM) problem is NP-hard. To prove this, we firstly consider a simple case, in which $n_i = k_i$ and n_i equals to total number of servers. That means that every server stores one chunk and all chunks are needed when accessing the file. We further assume all the requests arrive simultaneously. As a result, the Simple idealized File Access Time Minimization (SIFATM) problem can be defined as :

$$\text{minimize} \quad \sum_{g=1}^{L} C_g \tag{5}$$

$$\text{s.t.} \quad \forall g, j \sum_{\forall l:C_l \leq C_g} \sum_{i=1}^{n} t_{i,j}^{(l,f^l)} \leq C_g \tag{6}$$

$$\forall g, i \sum_{\forall l:C_l \leq C_g} \sum_{j=1}^{m} t_{i,j}^{(l,f^l)} \leq C_g \tag{7}$$

we can prove even the Simple idealized File Access Time Minimization (SIFATM) problem is NP-hard:

proposition *1:* Simple idealized File Access Time Minimization (SIFATM) problem is equivalent to the problem of minimizing the sum of job completion time in a concurrent open shop.

Proof: For the $n \times m$ network fabric, we mark ingress ports as $1..n$ and egress ports as $n+1...n+m$. We consider the transfer time of request $T_{f^k}^{(k)}$ through port p ($1 \leq p \leq$m+n):

$$T_{f^k}^{(k,p)} = \begin{cases} \sum_{j=1}^{m} t_{i,j}^{(k,f^k)} & 1 \leq p \leq \text{n} \\ \sum_{i=1}^{n} t_{i,j}^{(k,f^k)} & \text{n} < p \leq \text{n+m} \end{cases} \tag{8}$$

For the given $T_{f^k}^{(k)}$, there are n+m sub-transfers and each sub-transfer's time is computed as (8). Now, we consider the concurrent open shop scheduling problem with n+m identical machines (hence the same capacity). L jobs arrive at time 0 and each job has n+m types of operations on the n+m machines. Each file transfer can be regarded as a job and each port can be regarded as one machine. file transfer has sub-transfer through port p can be regarded as job has an operation on the machine labeled by p. The two problems are equivalent. ■

As the problem of minimize completion time of concurrent open shop problem is NP-hard [20], [7], [22], so SIFATM problem is also NP-hard. As a result, with source selection process, the more complex problem IFATM problem is NP-hard.

V. Algorithm and System design

In this section, we first introduce a 2-approximate algorithm to solve SIFATM problem. Then, we propose a heuristic source to optimize IFATM problem. At last, we extend the algorithm to an online one that can be used in practice.

A. Algorithm to solve SIFATM

The SIFATM problem is equivalent to minimize completion time of the concurrent open shop. The best solution to the problem of minimizing completion time of concurrent open shop problem is a 2-approximate solution that is proposed at [20]. According to the relationship between concurrent open shop problem and SIFATM problem, we change the algorithm to adapt to SIFATM problem just as Algorithm 1 shown.

Algorithm 1 takes a list $\mathcal{T}$ of n requests as its input. It outputs γ, a permutation of {1,...,n} that indicates the scheduling order of the n requests. The Algorithm first composes a port list P = {1, ... 2m}, corresponding to m ingress and m egress ports, and computes the total load on each port (line 2 $\sim$ 3). Then its iteratively finds the request to be scheduled in the i-th round, nonetheless in a reverse order. In each iteration, it finds the port with the heaviest load, chooses the request which has the minimal ratio of weight to its load on that port, and saves its index in $\gamma[i]$(line 7). It then updates the set of request remaining to be scheduled, and the port load (line 10), before it goes to the next iteration.

Algorithm 1 SIFATM algorithm

Input: Require set $\mathcal{T}$; chunk size $t_{i,j}^{(k,f^k)}$ for the k-th require from server i to client j, where $1 \leqslant i \leqslant n$,$1 \leqslant j \leqslant m$

Output: γ

1: $\gamma : \{1, 2, ...l\} \leftarrow \mathcal{T}$
   $UT \leftarrow \{1, 2, 3...l\}$
   $P \leftarrow \{1, 2, 3...m + n\}$
   $W\{1, 2, ...l\} \leftarrow \{1, 1...1\}$
2: $L_i^{(k)} = \sum_{j=1}^{m} t_{i,j}^{(k,f^k)}$ for all k $\leq l$ and i $\leq$ n
3: $L_{j+n}^{(k)} = \sum_{i=1}^{n} t_{i,j}^{(k,f^k)}$ for all k $\leq l$ and j $\leq$ m
4: $L_i = \sum_{k \leq l} L_i^k$ for all i $\in$ P
5: **for** i $\in \{l, l-1, l-2...1\}$ **do**
6: u=arg $\max_{k \in P} L_k$
7: $\gamma[i]$=arg $\min_{F \in UT} W[F]/L_u^{(F)}$
8: $\theta = W[\gamma[i]]/L_u^{\gamma[i]}$
9: W[j]=W[j]-θ*$L_u^{(j)}$ for all j $\in UT$
10: $L_j = L_j - L_j^{\gamma[i]}$ for all j $\in$ P
11: $UT = UT \setminus \{\gamma[i]\}$
12: **end for**

B. from offline to online

Indeed, Algorithm 1 is an ideal case as it assumes all the requests arrive simultaneously and each server stores one chunk for each file. Indeed, in real world, requests can arrive at any time and chunks for a file only stored at a set of machines. In this condition, Algorithm 1 is not a good choice.

In practice, file request occurs online and the load of each port varies with time. On average, all ports would have the same load since today's data centers generally assign jobs with load balancing [14], [19], [10]. In this case, the scheduler does not need to take the load diversity of ports into account (**relaxation 1**) [19]. What's more, for the distributed erasure coding storage system, each client knows the chunk information of all files. Based on this, we can compute the load of server side (**fact 1**). Also, for the erasure coding storage system, a file's trunks have the same size (**fact 2**). We can define file f_b's compress ratio as $\alpha_{f_b} = \frac{trunk\ size\ of\ f_b}{file\ size\ of\ f_b}$ and compute chunk size according to this. Then we get the online schedule Algorithm 2.

Algorithm 2 Online schedule algorithm

Input: Active require set $\mathcal{T}$, selected sources set $\theta_{f^k}^{(k)} = \{x_{1,1}^{(k,f^k)}, x_{1,2}^{(k,f^k)} ... x_{i,j}^{(k,f^k)} ...\}, \forall T_{f^k}^{(k)} \in \mathcal{T}, \forall x_{i,j}^{(k,f^k)} \in \{0,1\}, i \leq n, j \leq m$, compression set for files $\alpha = \{\alpha_{f_1}, \alpha_{f_2} .. \alpha_{f_r}\}$, file size set $\beta = \{\beta_{f_1}, \beta_{f_2}, .. \beta_{f_r}\}$, new arrival fetch T_{f^c}

Output: γ

1: $\theta_{f^c} = SourceSelection(\mathcal{T}, \theta, \alpha, \beta, T_{f^c})$
2: $\mathcal{T} = \mathcal{T} \cup \{T_{f^c}\}$
3: $L_i^{(b)} = \sum_{j=1}^{m} \alpha[f^b] * \beta[f^b] * x_{i,j}^{(b,f^b)}$ for $\forall b \in \mathcal{T}, i \leq n$
4: $L_{j+n}^{(b)} = \sum_{i=1}^{n} \alpha[f^b] * \beta[f^b] * x_{i,j}^{(b,f^b)}$ for $\forall b \in \mathcal{T}, j \leq m$
5: $l^{(b)} = \max_{1 \leq i \leq n+m} L_i^{(b)}$ for $\forall b \in \mathcal{T}$
6: $\pi^{(b)} = 1/l^{(b)}$ for $\forall b \in \mathcal{T}$
7: sort[$\pi^{(1)}$,$\pi^{(2)}$,$\pi^{(3)}$...] in non-decreasing order and set the sequence result to γ

Algorithm 2 is called when a new file request arrives. Inputs of Algorithm 2 are $\mathcal{T}, \theta_{f^k}^{(k)}, \alpha, \beta, T_{f^c}$, where $\mathcal{T}$ is the active file request set, $\theta_{f^k}^{(k)}$ contains the source servers and destination client of fetch $T_{f^k}^{(k)}$, α is the set that stores the size of every files in the system,β is the compress ratio set of each file, and T_{f^c} is the new request. Line 1 chooses sources for the new request. Line 3-Line 5 computes the load for each active requst in set $\mathcal{T}$. Line 3 computes the load of the server sides and Line 4 computes the client sides. Specially, we use $\alpha[f^b] * \beta[f^b] * x_{i,j}^{(b,f^b)}$ to compute the chunk size of file f^b. Line 5 decides the load of the request and the request is decided by the last completion flow. Then Line 6 computes the completion time of request $T_{f^c}^{(c)}$. The completion time of it is decided by the last one. Line 7 sorts $\pi^{(1)}$,$\pi^{(2)}$,$\pi^{(3)}$... in non-decreasing order and set the sequence result to γ.

C. Source selection

Different from Algorithm 1, Algorithm 2 is an online method and is called when a new request arrives. A very important step of Algorithm 2 is source selection, which selects k_i out of the total n_i servers for f_i. In the modern storage system such as Tahoe [3] and Ceph [1], random source selection is always used. However, as Section II shown, random source selection can make some nodes have heavy load and this will enlarge FAT. To prevent this, we propose the *smallest load first* heuristic to choose sources for the new request. Details of source selection is shown at Algorithm 3.

Algorithm 3 Smallest load first heuristic algorithm

Input: Active request set $\mathcal{T}$, source selected server set $\theta_{f^k}^{(k)} = \{x_{1,1}^{(k,f^k)}, x_{1,2}^{(k,f^k)} ... x_{i,j}^{(k,f^k)} ...\}, \forall T_{f^k}^{(k)} \in \mathcal{T}, \forall x_{i,j}^{(k,f^k)} \in \{0,1\}, i \leq n, j \leq m$, compression set for files $\alpha = \{\alpha_{f_1}, \alpha_{f_2} .. \alpha_{f_r}\}$, file size set $\beta = \{\beta_{f_1}, \beta_{f_2}, .. \beta_{f_r}\}$, new arrival fetch T_{f^c}

Output: θ_{f^c}

1: $B_i = \sum_{\forall b \in \mathcal{T}} \sum_{j=1}^{m} \alpha[f^b] * \beta[f^b] * x_{i,j}^{(b,f^b)}$, for $i \leq n$ and server b stores T_{f^c}'s chunk.
2: Rank B in non-decreasing order.
3: Select the first k_{f^c} ones and set the corresponding $x_{i,j}^{(c,f^c)}$ to 1, others set to 0, then add all $x_{i,j}^{(c,f^c)}$ to θ_{f^c}.
4: RETURN θ_{f^c}.

In Algorithm 3, Line 1 computes the load of ingress ports. Then, Step 2 ranks the load in non-decreasing order. Step 3 selects the smallest k_{f^c} ones and set the corresponding $x_{i,j}^{(c,f^c)}$ to 1, for other ports, just set to 0. At last, just add all $x_{i,j}^{(c,f^c)}$ to θ_{f^c} and return θ_{f^c}.

Algorithm 3 tries to find sources with smallest load. This makes sense, because file access time is decided by the slowest trunk. The smallest load first heuristic indeed tries to *minimize the maximum transfer time of trunks*. As FAT is decided by the maximum transfer time of trunks, so that FAT reduces.

D. Design of D-Target

In this part, we show the design of D-Target, a centralized scheduler that tries to minimize average file access time (FAT). As the server sides' load can be computed by the information from clients, so that the scheduler only manages all the clients. D-Target has three key components: client manager, client information collector, server rate allocator.

1) client manager: In D-Target, client manager is the heart of the system. When a new file request arrives, it computes the priority (Algorithm 2) and the chooses sources (Algorithm 3) for each request. After computing the priority of each file request, it calculates the bandwidth of each trunk. Procedure of client manager is shown at Algorithm 4.

In Algorithm 4, when a new file request arrives, client manager sorts the all the active file request according to Algorithm 2 (Line 1). Then it computes the bandwidth for every active trunk according to Algorithm 5 (Line 2). At last if distributes remaining bandwidth to all the trunks and sends the result to the corresponding server (Line3∼Line4).

$$g^{(k)} = \max(\max_i \frac{\sum_{j=1}^{m} t_{i,j}^{(k,f^k)}}{Rem(P_i^{(in)})}, \max_j \frac{\sum_{i=1}^{n} t_{i,j}^{(k,f^k)}}{Rem(P_j^{(out)})}) \quad (9)$$

Algorithm 5 shows the details of bandwidth computation. For each sorted request, (9) calculates out the bottleneck time (Line2). Other trunks of the file, use the bottleneck finishing

Algorithm 4 Procedure of the client manager

Input: Active request set $\mathcal{T}$, source selected server set $\theta_{f^k}^{(k)} = \{x_{1,1}^{(k,f^k)}, x_{1,2}^{(k,f^k)} ... x_{i,j}^{(k,f^k)} ...\}, \forall T_{f^k}^{(k)} \in \mathcal{T}, \forall x_{i,j}^{(k,f^k)} \in \{0,1\}, i \leq n, j \leq m$, compression set for files $\alpha = \{\alpha_{f_1}, \alpha_{f_2} .. \alpha_{f_r}\}$, file size set $\beta = \{\beta_{f_1}, \beta_{f_2}, .. \beta_{f_r}\}$, new arrival fetch T_{f^c}

Output: bandwidth of all active trunks

1: Sort $\mathcal{T}$ and the new arrival $T_{f^k}^{(k)}$ according to Algorithm 2
2: Allocate bandwidth to $\mathcal{T}$ according to Algorithm 5
3: Distribute unused bandwidth to $\mathcal{T}$ for all ports
4: Send $b_{i,j}$ to corresponding Server

time as its finishing time (Line4). At last, update the remaining capacity of each ports (Line6 $\sim$ Line7).

2) client: Clients store file information, including trunk positions, file size, etc. Clients interact with customers and collect file require information. At last they send the information to client manager.

3) server: Servers store all the trunks. They receive clients' fetching request and throttle bandwidth according to clients manager's computation results.

Algorithm 5 Procedure of bandwidth computation

Input: Sorted require set γ, Remaining capacity for ports set $Rem(.)$

Output: bandwidth of each trunk, Remaining capacity for ports set $Rem(.)$

1: **for** k $\in \gamma$ **do**
2: Compute Load $g^{(k)}$ according to (9)
3: **for** $t_{i,j}^{(k,f^k)} \in k$ **do**
4: $b_{i,j} = t_{i,j}^{(k,f^k)} / g^{(k)}$
5: $Rem(P_i^{(in)}) -= b_{i,j}$
6: $Rem(P_j^{(out)}) -= b_{i,j}$
7: **end for**
8: **end for**

VI. Evaluation

In this section, we thoroughly evaluate the performance of D-Target by trace-driven simulation. Main results of our evaluation can be summarized as:

- For the trace of AT&T, D-Target performs 2.5×, 1.7×, 1.8×, 3.6× better than TCP, Aalo [8], Barrat [14] and pFabric [5], respectively.
- Without *smallest load first* (SLF) heuristic source selection, D-Target performs about 1.9×, 1.4×,1.6×, 2× better than TCP, Barrat, Aalo, pFabric, respectively. With *smallest load first* (SLF) heuristic source selection, TCP, Barrat [14], Aalo [8], pFabric [5] perform about 30%, 27%, 33%, 20% better than the corresponding methods that are without SLF, respectively.
- Compared with the 2-approximation algorithm, the online algorithm suffers less than 15% of performance loss.

A. Trace-driven Simulation Methodology

We use the trace of AT&T in most of experiments. The trace is collected from 720 servers of 60 racks in the data center of AT&T and it includes file size, trunk size. trunk position of each file as well as 30days' requests.

Metric. We use File Access Time (FAT) to evaluate the effectiveness of different scheduling methods. We consider the average FAT of specific categories for files. FAT covers the time from scheduler chooses the sources to clients receive all the chunks. We compare D-Target against four typical scheduling algorithm: TCP, Barrat [14],Aalo [8],pFabric [5]. In some group of experiments, we use TCP as the baseline and we define the factor of improvement = $\frac{baseline\ of\ avg\ FAT}{current\ avg\ FAT}$.

Open source. To make our experiment repeatable, we publish main codes of D-Target. Sources of D-Target can be downloaded at [2], so that interested readers can repeat them.

B. Real traffic trace-driven

In this part, we use the trace of AT&T to test the performance of D-Target. For random source selection, sources are different every time and sources for SLF are same. To eliminate accidental fluctuation, we run the trace 100 times for each group of experiments. Then Fig. 3 shows the result. Note, in each picture, error bar indicates average, maximum, minimum value.

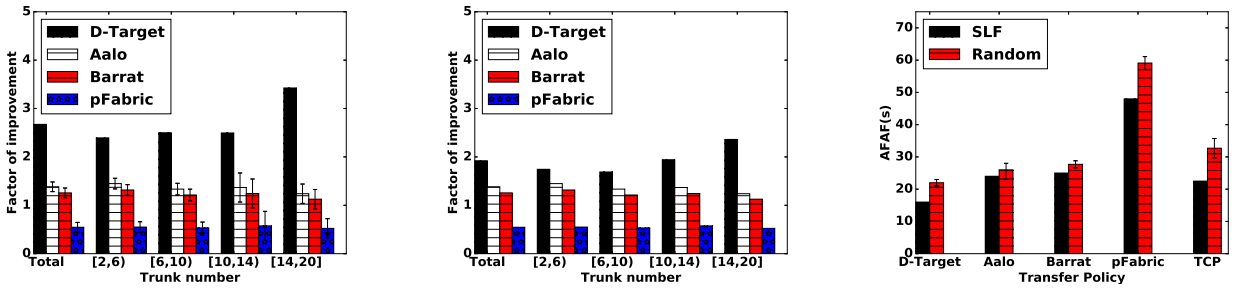
Fig. 3(a) shows the result of average FAT. We can see that factor of improvement for D-Target is 2.5, while for Aalo, Barrat, pFarbric are 1.5, 1.2, 0.8, respectively. For the distributed erasure coding storage system, the response always contains parallel chunks and according to the width of the response, we divide them into four groups. We can see that factor of improvement for D-Target is 2.2, 2.4, 2.8, 3, when parallel chunk number is [2,6), [6,10), [10,14) and [14,20] respectively. Factor of improvement becomes larger with the increasing of parallel of chunks. The reason for this is that, for bigger number of parallel chunk, collision between requests become higher, so that the necessity of doing source selection and traffic control are larger.

In modern erasure coding storage system, when requests arrive, scheduler always uses random source selection to choose sources. In this paper, we purpose the *smallest load first* heuristic to choose sources. In this case, less collides will occur. Fig. 3(b) shows the results that all the transfer methods use *smallest load first* heuristic to select sources. We can see that gap between D-Target and other methods become small. D-Target performs about 1.9×, 1.7×, 2× better than TCP, Aalo, Barrat and pFabric, respectively.

Fig. 3(c) shows average file access time (FAT) comparison for different methods . We can see that with *smallest load first* heuristic source selection, D-Target, Aalo [8], Barrat [14], pFabric [5] and TCP perform about 30%, 30%, 27%, 33%, 20% better than random source selection respectively.

C. Performance under different settings

In this part, we explore D-Target's performance under different settings. We know that FAT is influenced by source



(a) Average FAT comparison (b) Average FAT with SLF comparison (c) With and without SLF comparison

Fig. 3: Results with AT&T trace, TCP is selected as the baseline. Note for random selection, source results are different every time, to eliminate accidental fluctuation, we run the trace 100 times for each group.

selection and trunk transfer time. For distributed erasure coding storage system, a file is encode by (n, k) MDS code, where n denotes the number of chunks that a file is encoded and k denotes the number of chunks are needed to reconstruct the file. k and n decides the width of the response task.

TABLE I: Default parameters in each group of experiment

Number	File size	n	k	α	Capacity	Arrival
512	[100KB,10GB]	8	4	1.5	1GB	[0s,1000s]

As we want to explore the influence of each parameter, so in each group of experiment, we should fix the value of other parameters and only change the factor we want to study. TABLE I shows the default settings of each experiment. There are 512 files and size of file is between 100KB and 10GB. Default parameter for MDS is value is(8,4) and the arrival time of each request is between 0 and 1000s. Ingress and egress port capacity in our experiment are 1GB. For each group of experiment, we generate parameters 100 times and then draw maximum value, minimum value, average value in the picture.

Fig.4(a) shows that with the increase of n, factor of improvement for D-Target becomes larger. When n is 4, factor of improvement is 1.8 and when n is 18, factor of improvement is 2.5. D-Target performs better for larger value of n. The reason for this is that for larger n, conflicts between nodes become higher, so that it is more necessary to optimize source selections and transfer.

Fig.4(b) shows that with the increase of k, D-Target performs better. When k is 3, factor of improvement is 2.2 and when k is 7, factor of improvement is 2.5, which is about 20% larger. For larger value of k, D-Target improves higher due to less collides.

In real word, requests can arrive at any time. Arrival time of requests can impact the traffic load in data center network. In this part, we explore the impact of transfer concurrency. Changing the number of requests that start at t = 0. Fig.4(c) shows the result. We can see that for larger concurrent number of requests, improvement of D-Target becomes bigger. This is because, the heavier the network load is, the larger optimization space there is for preemptive scheduling.

D. Distance of the online method

Comparing with the offline 2-approximate algorithm, algorithm 2 ignores load diversity and this may lead to performance loss. In this section we study the performance gap between the online and the 2-approximate offline algorithm. Use the trace of AT&T and set all the request arrive at t = 0, then Fig. 5 shows the result.

From Fig. 5(a), we can see that factor of improvement for the 2-approximate method is 2.4, while for the online method is 2.1. The online method has about 15% performance loss. Fig. 5(b) demonstrates the distribution of FAT, we can see that more than 80% of the FAT is less than 300s for the 2-approximate method, while that for the online method is about 65%. Fig. 5(c) shows the performance of the two methods that are without *smallest load first* heuristic, we can see that performance gap between the online and 2-approximate is about 18%. Fig. 5(d) shows the distribution of FAT. We can see that about 80% of the FAT is within 350s, which is about 30% worse than the method with SLF.

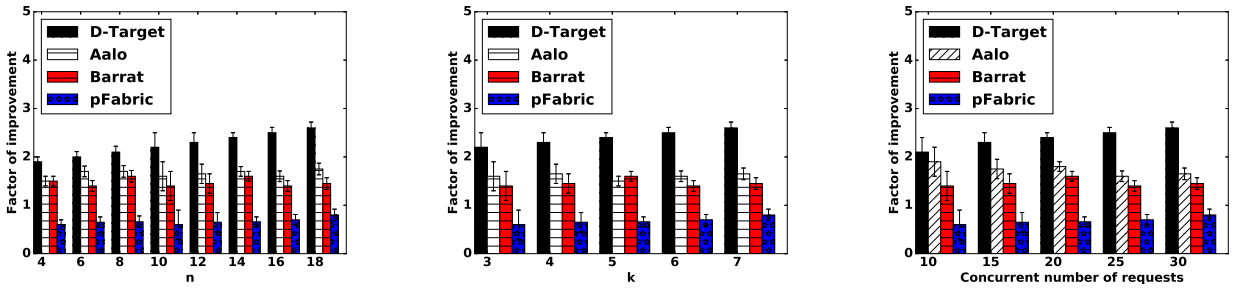
VII. CONCLUSION

In this paper, we joint the two problems together to optimize. Our optimization goal is to minimize average file access time (FAT). To achieve this, we propose smallest load first heuristic to do source selection and design an online algorithm to reduce trunk transfer latency. Based on this, we design and implement D-Target, a centralized scheduler that tries to minimize average FAT in distributed erasure coding storage system. We then test D-Target's performance by trace-driven simulation. Results show that, for the trace of AT&T distribute erasure coding storage system, D-Target performs 2.5×, 1.7×, 1.8×, 3.6× better than TCP, Aalo, Barrat and pFabric respectively.

Now, we just test the performance of D-Target by trace-driven simulation. In the future, we will evaluate D-Target in the industry erasure coding storage system such as ceph [1]and deploy it in real world.

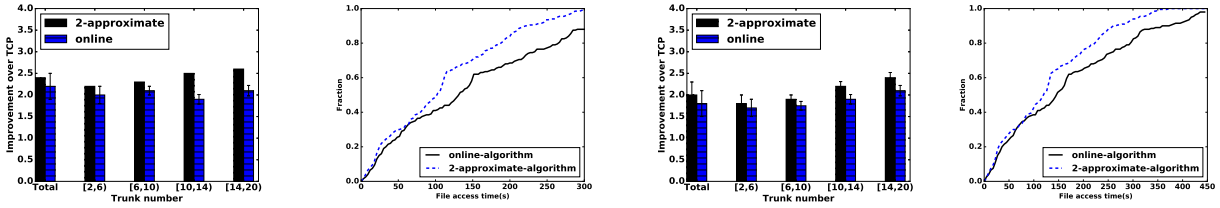
REFERENCES

[1] ceph. https://github.com/ceph/ceph/.
[2] D-target. https://github.com/zhanghan1990/D-Target/.
[3] Tahoe-lafs. https://github.com/tahoe-lafs/tahoe-lafs/.



(a) Average FAT comparison (b) Average FAT comparison (c) Average FAT comparison

Fig. 4: Performance comparison under different settings, TCP is selected as the baseline



(a) Average FAT comparison (b) FAT distribution (c) Average FAT comparison (without SLF) (d) FAT distribution (without SLF)

Fig. 5: Performance comparison between 2-approximate and online algorithm. TCP is selected as the baseline.

[4] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). SIGCOMM '10, pages 63–74, 2010.

[5] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pfabric: Minimal near-optimal datacenter transport. SIGCOMM '13, pages 435–446, 2013.

[6] D. Beaver, S. Kumar, H. C. Li, J. Sobel, P. Vajgel, et al. Finding a needle in haystack: Facebook's photo storage. In *OSDI*, volume 10, pages 1–8, 2010.

[7] Z.-L. Chen and N. G. Hall. Supply chain scheduling: Assembly systems. Technical report, Working Paper, Department of Systems Engineering, University of Pennsylvania, 2000.

[8] M. Chowdhury and I. Stoica. Efficient coflow scheduling without prior knowledge. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 393–406. ACM, 2015.

[9] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient coflow scheduling with varys. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 443–454. ACM, 2014.

[10] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[11] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. *IEEE Transactions on Information Theory*, 56(9):4539–4551, 2010.

[12] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran. Decentralized erasure codes for distributed networked storage. *arXiv preprint cs/0606049*.

[13] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran. Decentralized erasure codes for distributed networked storage. *IEEE/ACM Transactions on Networking (TON)*, 14(SI):2809–2816, 2006.

[14] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron. Decentralized task-aware scheduling for data center networks. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 431–442. ACM, 2014.

[15] C.-Y. Hong, M. Caesar, and P. Godfrey. Finishing flows quickly with preemptive scheduling. *ACM SIGCOMM Computer Communication Review*, 42(4):127–138, 2012.

[16] X. S. Huang, X. S. Sun, and T. E. Ng. Sunflow: Efficient optical circuit scheduling for coflows. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 297–311. ACM, 2016.

[17] J. Li and B. Li. Erasure coding for cloud storage systems: a survey. *Tsinghua Science and Technology*, 18(3):259–272, 2013.

[18] H.-Y. Lin and W.-G. Tzeng. A secure erasure code-based cloud storage system with secure data forwarding. *IEEE transactions on parallel and distributed systems*, 23(6):995–1003, 2012.

[19] S. Luo, H. Yu, Y. Zhao, S. Wang, S. Yu, and L. Li. Towards practical and near-optimal coflow scheduling for data center networks. *IEEE Transactions on Parallel and Distributed Systems*, 27(11):3366–3380, 2016.

[20] M. Mastrolilli, M. Queyranne, A. S. Schulz, O. Svensson, and N. A. Uhan. Minimizing the sum of weighted completion times in a concurrent open shop. *Operations Research Letters*, 38(5):390–395, 2010.

[21] A. Munir, I. Qazi, Z. Uzmi, A. Mushtaq, S. Ismail, M. Iqbal, and B. Khan. Minimizing flow completion times in data centers. In *INFOCOM, 2013 Proceedings IEEE*, pages 2157–2165, April 2013.

[22] T. A. Roemer. A note on the complexity of the concurrent open shop problem. *Journal of scheduling*, 9(4):389–396, 2006.

[23] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur. Xoring elephants: Novel erasure codes for big data. In *Proceedings of the VLDB Endowment*, volume 6, pages 325–336. VLDB Endowment, 2013.

[24] E. Schurman and J. Brutlag. The user and business impact of server delays, additional bytes, and http chunking in web search. In *Velocity Web Performance and Operations Conference*, 2009.

[25] B. Vamanan, J. Hasan, and T. Vijaykumar. Deadline-aware datacenter tcp (d2tcp). SIGCOMM '12, pages 115–126, 2012.

[26] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. Better never than late: Meeting deadlines in datacenter networks. SIGCOMM '11, pages 50–61, 2011.

[27] J. Wu, L. Ping, X. Ge, Y. Wang, and J. Fu. Cloud storage as the infrastructure of cloud computing. In *Intelligent Computing and Cognitive Informatics (ICICCI), 2010 International Conference on*, pages 380–383. IEEE, 2010.

[28] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng. Coda: Toward automatically identifying and scheduling coflows in the dark. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 160–173. ACM, 2016.

[29] H. Zhang, X. Shi, X. Yin, F. Ren, and Z. Wang. More load, more differentiationa design principle for deadline-aware congestion control. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 127–135. IEEE, 2015.

[30] H. Zhang, X. Shi, X. Yin, Z. Wang, and Y. Guo. Fdrc-flow duration time based rate control in data center networks. In *Quality of Service (IWQoS), 2016 IEEE/ACM 24th International Symposium on*, pages 1–10. IEEE, 2016.