1 Good Morning，I am XX, very happy to meet all of you here. Today my topic is **Joint source selection and transfer optimization for erasure coding storage system**

2 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.

Erasure coding 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 fetches the file, any subset of k out of n chunks are needed to reconstruct it

3 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 random source selection will make some nodes have heavy load and this will magnify latency. TCP is fair transfer method, but in erasure coding storage system, a file can be reconstructed only when the client receives all the chunks.

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

4 We are the **first** to joint source selection and chunk 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 chunks. Trace-driven simulation shows that D-Target performs D- Target performs 2.5, 1.7, 1.8, 3.6 better than TCP, Barrat , Aalo , pFabric respectively.

5 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}. Firstly, we consider two requests RA and RB that arrive simultaneously at t = 0 to fetch file A and file B respectively. For RA, 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 RA chooses (S 2, S 3) and RB chooses (S 3, S 4). If the transfer policy is TCP, then the file access time forRA is tA =2and for RB is tB =2. The AFAT(average file access time) is 2.

Just think the simple case, there are two request for file A, RA1 arrives at t = 0 and RA2 arrives at t = 0.1. RA1 chooses(S1,S2) as the sources and RA2 chooses (S2,S3) as the sources. For TCP transfer, FAT( file accesstime)forRA1 ist=1.9andFATforRA2 ist=2.0,so that AFAT( average file access time) is (1.9 + 2.0)/2 = 1.95. However, if we let RA1 have higher priority than RA2 . Then FATforRA1 ist=1andFATforRA2 ist=2,sothat AFAT is (1 + 2)/2 = 1.5.

6 Previous work mainly focus on **efficient encoding and decoding**, Overhead of CPU will significantly reduce.

When accessing file or one blocks that are sufficient to recover all the data. This process can add **heavy burden** to network.

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**.

7 DCTCP, D2TCP,L2DCT,PDQ,pFarbic,LPD and D3 are **flow level methods**.DCTCP – fair sharing D2TCP, LPD and D3 are deadline-aware methods.Barrat, Varys,Aalo, Sunflow regard flows of applications **as a whole**.Barrat schedules task in FIFO order but avoids head-of-line blocking by dynamic changing the level of multiplexing in the network

Varys and Aalo, sunflows try to minimize average coflow completion time.

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

8 -11Recent studies 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. In this paper, we use this assumption and only take the contention of ingress and egress ports into consideration.

12 We consider a non-blocking data center that consists of n storage nodes, denoted by S.Then r files, denoted by F.For each file fi, we partition it into ki fixed-size trunks and then encode it using an (ni,ki) MDS erasure code to generate ni distinct trunks of the same size for fi. The ni distinct shares are stored at ni machines. The m clients, denoted by C.Assume all the L requests arrive at time (k) 0 and the k − th request denoted by Tf k means requesting file fk(fk ∈ F). The k-th request T(k)consists of parallel fk subtasks, each subtask is a flow from a server to the client

As the non-blocking model’s ingress and egress ports have unit capacity, so the transfer time for sub-task t(k,fk) is t(k,fk).

The problem of non-preemptive Idealized File Access Time Minimization (IFATM) Problem can be defined as

13 we firstly consider a simple case, in which ni = ki and ni 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

14 The best solution to the problem of minimizing completion time of concurrent open shop problem is a 2-approximate solution. 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 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. In this case, the scheduler does not need to take the load diversity of ports into

account 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 fb’s compress ratio as αfb = trunk size of fb and file size of fb compute chunk size according to this. Then we get the online schedule Algorithm 2.

15 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

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

16 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.

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 , pFabric and TCP perform about 30%, 30%, 27%, 33%, 20% better than random source selection respectively.

17 Fig.4(a) shows that with the increase of n, factor of im- provement 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 per- forms 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.

18 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.

19 Thanks. Any problems?