S3: Joint Scheduling and Source Selection for Background Traffic in Erasure-Coded Storage

Shijing Li[1], Tian Lan[1], Moo-Ryong Ra[2], Rajesh Panta[2]
[1]ECE, George Washington University, [2]AT&T Labs Research
{shijing, tlan}@gwu.edu, {mra, rpanta}@research.att.com

***Abstract*—Erasure-coded storage systems have gained considerable adoption recently since they can provide the same level of reliability with significantly lower storage overhead compared to replicated systems. However, background traffic of such systems – e.g. repair, rebalance, backup and recovery traffic – often has large volume and consumes significant network resources. Independently scheduling such tasks and selecting their sources can easily create interference among data flows, causing severe deadline violation. We show that the well-known heuristic scheduling algorithms fail to consider important constraints, thus resulting in unsatisfactory performance. In this paper, we claim that an optimal scheduling algorithm that aims to maximize the number of background tasks completed before deadlines must simultaneously consider deadline-aware scheduling, network topology, chunk placement, and time-varying resource availability. To solve this problem, we propose a novel algorithm, called Linear Programming for Selected Tasks (LPST) to maximize the number of successful tasks and improve overall utilization of the datacenter network. It jointly schedules tasks and selects their sources based on a notion of Remaining Time Flexibility, which measures the slackness of the starting time of a task. We evaluated the efficacy of our algorithm using extensive simulations. Our results show that, under certain scenarios, LPST can perform 7x∼70x better than the heuristics which blindly treat the infrastructure as a collection of homogeneous resources, and 46.6%∼65.9% better than the algorithms that take into account the network topology.**

I. Introduction

Many commercially available large scale storage systems are increasingly adopting erasure-coding technology [1], [2], [3], [4]. Erasure-coded storage systems can provide the same level of reliability as replicated systems, but with significantly lower storage space overhead [5]. However, a major drawback of large scale erasure-coded storage systems is that they generate large amounts of background traffic. The background traffic tends to be large in volume, and consequently consumes significant network resources [6].

Existing practical systems often schedule each background task independently without considering the impact on one another. These distributed tasks share network, compute, and storage resources of the underlying infrastructure, which causes interference among competing data flows, resulting in poor resource utilization and violation of Service Level Agreements (SLAs) associated with the background tasks. To mitigate this problem, [7] proposed a technique that can allocate bandwidth between servers and there exists many related work [8], [9]. With these techniques, necessary amount of network resources can be assigned to each background task, taking into account total resource budget.

This paper proposes a novel and practical way of scheduling background jobs in a holistic manner by jointly taking into account all current background jobs together. Deadline-aware scheduling has been studied extensively in many domains [10], [11], [12]. However, scheduling background tasks in an erasure-coded storage system running in a large datacenter environment is unique and arguably more difficult, because it introduces three challenging dimensions to the problem — task scheduling over time, data source selection, and bandwidth allocation in each network segment to each background task.

Existing scheduling algorithms designed for other problem domains consider mostly the first challenge and often assume homogeneous resources, e.g., processor scheduling across CPUs, MapReduce jobs across worker processes, etc.

In order to address these problems, we develop an online algorithm to maximize the number of tasks that successfully meet deadlines,under data placement, network topology and bandwidth constraints. To optimally schedule each task, we need to jointly solve: (i) a chunk selection problem that determines the (erasure-coded) chunks used to generate background traffic, (ii) a bandwidth allocation problem that apportions bandwidth at TOR and aggregation switches among active tasks, and (iii) a scheduling problem that schedules tasks with respect to their deadlines.

We evaluate existing algorithms as well as our proposed algorithm extensively in simulation. We demonstrate that the proposed algorithm can indeed improve the number of tasks meeting deadlines significantly under various combinations of arrival patterns, system parameters and resource availability. Our results show that, under certain scenarios, our proposed algorithm can perform 7x∼70x better than the heuristics which blindly treat the infrastructure as a collection of homogeneous resources, and 46.6%∼65.9% better than the algorithms that take into account the network topology (Sec. IV).

II. System Model and Problem Formulation

We consider a datacenter storage system with an aggregator switch connecting u Top-of-Rack (TOR) switches. r storage servers ($\mathcal{R} = \{1, 2, \ldots, r\}$) are placed in u racks ($\mathcal{U} = \{1, 2, \ldots, u\}$), each of which is connected to a TOR switch. As shown in Figure 1, the traffic between servers in the same rack does not need to flow to the aggregator switch, while the traffic between servers in different racks needs to flow through two TOR switches and the aggregator switch.

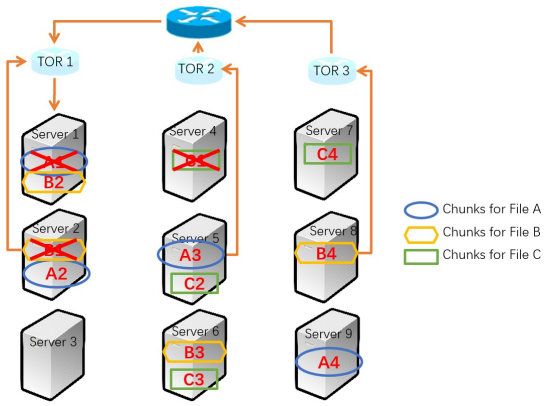


Fig. 1. An illustrative example with 3 racks and 9 servers

Each file i is stored using (n_i, k_i) erasure coding. We consider Maximum-Distance-Separable (MDS) codes, which ensures that any k_i out of n_i chunks are sufficient for reconstructing the file i.

A. Problem Formulation

Our problem formulation is described as follows. Let $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$ denote a set of background tasks, such as backup, repair, and re-balance. Each task, A_i, is associated with a number of parameters, including n_i potential sources of data chunks (denoted as $o_{i,1} \in \mathcal{U}$, $o_{i,2} \in \mathcal{U}$, ..., $o_{i,n_i} \in \mathcal{U}$), one destination (denoted as $p_i \in \mathcal{U}$), the number k_i of chunks to be retrieved, volume (denoted as v_i) for each chunk, task starting time (denoted as s_i), and task deadline (denoted as d_i). Task starting time and deadline are given in seconds, satisfying $0 \le s_i \le d_i$. To formally formulate this optimization problem, we consider a time slotted system. Suppose $y_{i,j}$ is a binary chunk selection variable, such that $y_{i,j} = 1$ if chunk j is selected to execute task A_i, and $y_{i,j} = 0$ otherwise. Since k_i data chunks must be selected, we have

$$\sum_j y_{i,j} = k_i, \ \forall i \tag{1}$$

where the selection remains fixed while task i is running.

To count the number of successfully completed tasks, we use a binary variable z_i, which is 1 if task A_i is finished before the deadline, and 0 otherwise. Let $x_{t,i,j}$ be the bandwidth assigned in time slot t to the data flow transferring chunk j of task A_i. If the task is successfully completed before a deadline d_i, all of the k flows should finish before d_i, implying a deadline constraint for successful tasks:

$$\sum_{t=s_i}^{d_i} x_{t,i,j} y_{i,j} \ge v_i, \ \text{if} \ z_i = 1, \forall i, \forall j, \tag{2}$$

Since each source-destination pair has a predetermined route, for a given set of tasks, we use RC_g to denote the set of tasks/chunk flows traversing a (TOR or aggregator) switch g, i.e., $(i,j) \in RC_g$ if flow of chunk j of task i uses switch g. Similarly, SC_h is the set of tasks/chunk flows using a server h. Further, each TOR has capacity limit CTA, and each server has capacity limit CST. Thus, we have the following capacity constraints:

m	number of tasks
(n, k)	erasure code parameters
$\mathcal{A}$	A set of m tasks $A_1, \ldots, A_m$
RC_g	A set of tasks traversing TOR/aggregator switch g
SC_h	A set of tasks using server h
r	number of storage servers
u	number of racks
CST	Link capacity from servers to each TOR
CTA	Link capacity from each TOR to the aggregator
$x_{t,i,j}$	Bandwidth assigned at t to send chunk j of task A_i
w_i	Number of candidate sources/chunks for task A_i
z_i	Whether task i is completed before deadline
$o_{i,1}, \ldots, o_{i,w_i}$	Candidate sources/chunks for task A_i
$o'_{i,s}$	Selected source/chunk for sub-task $A'_{i,s}$
p_i (p'_i)	Destination of task A_i (sub-task $A'_{i,s}$)
v_i	Volume (chunk size) of task A_i
d_i	Deadline of task A_i
s_i	Starting time of task A_i
LRB	Least required bandwidth of task A_i
f_i ($f'_{i,s}$)	RTF of task A_i (sub-task $A'_{i,s}$)

TABLE I
TABLE OF KEY NOTATIONS

$$\sum_{(i,j) \in RC_g} x_{t,i,j} y_{i,j} \le CTA, \ \forall g, t \tag{3}$$

$$\sum_{(i,j) \in SC_h} x_{t,i,j} y_{i,j} \le CST, \ \forall h, t \tag{4}$$

Our goal is to maximize the number of tasks that can be successfully completed before deadline in erasure-coded storage. This is formulated as a joint Scheduling and Source Selection (denoted as $S3$) problem, i.e.,

$$\max \quad \sum_i z_i \tag{5}$$

$$\text{s.t.} \quad \sum_j y_{i,j} = k_i, \ \forall i \tag{6}$$

$$\sum_{t=s_i}^{d_i} x_{t,i,j} y_{i,j} \ge v_i z_i, \ \forall i, \tag{7}$$

$$\sum_{(i,j) \in RC_g} x_{t,i,j} y_{i,j} \le CTA, \ \forall g, t \tag{8}$$

$$\sum_{(i,j) \in SC_h} x_{t,i,j} y_{i,j} \le CST, \ \forall h, t \tag{9}$$

$$var. \quad x_{t,i,j} \ge 0, \ y_{i,j} \in \{0,1\}, \ z_i \in \{0,1\} \tag{10}$$

Here the deadline constraint (7) is exactly (2) for successful tasks with $z_i = 1$, and is superfluous when $z_i = 0$. Note that replication can be considered as a special case of our proposed optimization with $k_i = 1$, i.e., the entire file is replicated across the network.

III. LPST Algorithm Design and Competitors

In this section, we describe our proposed algorithm, called Linear Programming for Selected Tasks (LPST), which harnesses resource-aware chunk selection, deadline-aware task

Algorithm 1: LPST Algorithm

// Phase I: Source Selection Procedure
foreach *task* i **do**
    Least required bandwidth: $LRB_i = v_i/(d_i - t)$;
    Sort w_i candidate sources by the largest congestion factor in each path from source to destination;
    Find k_i source servers with least fulfilled path;
    Create k_i new subtask $A^{'}_{i,s}$;
    Add LRB_i to congestion factor of links in each subtask's path;
end

// Phase II: Selecting Emergent Tasks
foreach *subtask* i **do**
    Calculate RTF $f_i = \min_s \left(d^{'}_i - max(t, s_i) - v^{'}_n/C_{o_{i,s},p_i}\right)$;
end
Initialize $\mathcal{T} = \{\}$, remaining bandwidth for each link;
Find task i with smallest f_i;
while *task* i *is feasible w.r.t. remaining bandwidth* **do**
    $\mathcal{T} \leftarrow \mathcal{T} \bigcup \{A_i\}$
    Assign initial bandwidth $b_i = LRB_i$;
    Update remaining bandwidth;
    Find next task i with smallest f_i;
end

// Phase III: Optimize bandwidth for admitted tasks in $\mathcal{T}$;
Solve the following optimization problem using LP;

$$\max \sum_{i:A_i \in \mathcal{T}} b_i$$
$$\text{s.t.} \sum_{(i,s) \in RC_g} b_i \leq CTA, \forall g$$
$$\sum_{(i,s) \in CS_h} b_i \leq CST, \forall h$$
$$b_i(d_i - s_i) \geq v_i \ \forall i$$
$$\text{var. } \{b_i, \ \forall i \in \mathcal{T}\}$$

prioritization, and bandwidth optimization via linear programming. As a result, LPST maximizes the number of tasks that can meet their deadlines. We compare LPST to a set of heuristic algorithms and their qualitative descriptions are presented in Section III-B. Quantitative comparison results are in Section IV.

A. Linear Programming for Selected Tasks (LPST)

LPST algorithm takes three steps to determine a scheduling strategy at any given time – selecting sources of erasure coded data, selecting emergent tasks and assigning bandwidth for each task.

Selecting Sources (Phase I): When a new task A_i arrives to the system, it is split into k_i subtasks, each of which has a distinct source. Each subtask $A^{'}_{i,s}$ (s=1..k_i) has 5 properties: a) source ($o^{'}_{i,s}$), b) destination ($p^{'}_{i,j} = p_i$), c) volume ($v^{'}_i = v_i$), d) starting time ($s^{'}_i = s_i$), and e) deadline ($d^{'}_i = d_i$). Note that while each subtask $A^{'}_{i,s}$ has its own selected source, all subtasks belonging to A_i must be completed before d_i to meet a common deadline. For each of the k_i subtasks, we calculate its *least required bandwidth (LRB)*, defined by the minimum amount of bandwidth that is necessary to finish the task before the deadline. Let t be the current system time. LRB can be calculated using the following equation.

$$LRB_i = v_i/(d_i - t). \tag{11}$$

Then, for the corresponding servers or TORs in the path, we add LRB_i to their congestion factors. Then we calculate the congestion factors for all subtasks, and we select k_i sources with least fulfilled links (smallest congestion factor).

Prioritizing Tasks (Phase II): Once the sources are chosen, we could generate a plan on how we may allocate bandwidth for the tasks to satisfy our objective, e.g., maximizing network utilization of our datacenter. However, blindly applying existing optimizing technique, such as linear programming, is likely to cause a scalability problem (Section IV). Therefore, in LPST we first sort all subtasks based on a metric, called *remaining time flexibility (RTF)*, which quantifies the flexibility in scheduling a task with respect to its deadline and resource availability, reflecting how emergent the task is. After a list of admitted tasks are identified, a linear programming problem is solved to optimize bandwidth allocation for maximizing network utilization for the admitted tasks.

In particular, for subtask $A^{'}_{i,s}$, a chunk of size $v^{'}_i$ needs to be transferred, from source server $o^{'}_{i,s}$ to destination server $p^{'}_i$, which has pre-determined route with maximum available capacity $C_{o_{i,s},p_i}$. The task starting time is s_i and deadline is $d^{'}_i$. Then RTF $f^{'}_{i,s}$ of the subtask $A^{'}_i$ can be calculated as follows.

$$f_{i,s}{}^{'} = d^{'}_i - max(t, s_i) - v^{'}_n/C_{o_{i,s},p_i} \tag{12}$$

where t is current timestamp and $C_{o_{i,s},p_i}$ is the maximum available link capacity from source server $o^{'}_{i,s}$ to destination server $p^{'}_i$. Next, the RTF of task A_i is defined as the minimum RTF of all its subtasks, i.e.,

$$f_i = \min_s f^{'}_{i,s}. \tag{13}$$

Intuitively RTF f_i measures the maximum allowed delay to begin processing task A_i, in order to meet its deadline. If f_i value is smaller, the task is more emergent and we may need to schedule it right away by delaying some other tasks that have higher RTF values.

Finally, we rank all tasks according to their RTF in ascending order, and admit tasks one-by-one until no more task with higher RTF can be added.

Assigning Bandwidth (Phase III): After we get a final list of feasible tasks, we formulate a network optimization to assign bandwidth b_i for each task by maximizing network link utilization. While this step does not directly affect the number of tasks that are completed before deadline, it maximizes resource utilization and thus reduces the overall completion time required by currently admitted tasks. This has two benefits. First, when we use the proposed LPST algorithm in an iterative fashion, optimizing bandwidth utilization allows us to accommodate more tasks by re-running the procedure in Phase I and II. Second, this is particularly important in an online setting – by completing the current, admitted tasks as fast as possible, we can make more resources available for new tasks that arrive in the future. The bandwidth assignment in Phase III is solved as a linear programming problem with network capacity and deadline constraints. The admitted tasks are guaranteed to meet their individual deadlines.

Supporting Different Network Topologies: Although in

this paper we formulate our optimization for a hierarchical datacenter network topology involving TOR and aggregator switches, the results can be readily extended to arbitrary topologies such as fat-tree or Bcube [15] [16]. In particular, source selection (Phase I) and bandwidth assignment (Phase III) need to reflect updated link capacity constraints due to new network topologies, while task prioritization (Phase II) remains the same. More complicated network topologies, such as B-cube or fat-tree, may introduce more link capacity constraints, but they are still linear constraints and can be solved by linear programming. Since LPST uses task prioritization, the complexity of linear programming will still be limited due to small number of variables.

B. Competing Algorithms

We compare LPST against several variants of well-known heuristic scheduling algorithms. The competing algorithms that we considered in this paper are three-fold – FIFO and its variants, EDF and its variants, and Linear Programming.

FIFO family: First In First Out (FIFO) schedules a task to the first available resource in a sequential manner. FIFO has an obvious inefficiency when two consecutive tasks share the same network link. In Fig. 1, consider the case in which the task A_1 is transmitting data from server 2 to server 1 and the task A_2 is sending data from server 8 to server 5. In FIFO, A_2 will need to wait until A_1 completes. To address this issue, we come up with a disjoint version of FIFO (DisFIFO). In DisFIFO, the tasks that do not share network links can be scheduled at the same time, and consequently result in better performance. Lastly, all algorithms in FIFO family choose sources randomly in erasure-coding case.

EDF family: Earliest Deadline First (EDF) algorithm is also well-studied in the scheduling literature. In our problem setting, the EDF algorithm has the same problem like FIFO. So we developed DisEDF using the similar technique. The difference between EDF and DisEDF is exactly the same as that between FIFO and DisFIFO. Moreover, we developed an enhanced version of DisEDF called DisEDF-S, which does a better job of source selection. In DisEDF-S, k chunks with the smallest task number in their transmission path are selected from n sources. This means that they have less opportunities to have conflicts with other transmission tasks.

Linear Programming: We utilize a recent advance in datacenter networking, i.e. bandwidth reservation, to devise an algorithm called Linear Programming applied on All tasks (LPAll). Whenever a new task arrives to the system or a task finishes, LPAll assigns bandwidth to a given set of tasks using the linear programming technique. The formulation is same as that of LPST bandwidth allocation scheme, i.e., the objective function is to maximize bandwidth utilization under link capacity and task deadlines constraints.

IV. Evaluation

A. Methodology, Simulator Setup

We built a custom simulator of all algorithms in a java platform. We implemented a *task generator* to feed tasks. The simulator takes the *task generator*'s output as input and simulates the behavior of various algorithms. The simulator captures essential resource constraints including network topology, bandwidth limitation, task deadlines, and erasure-code source selection. The task generator and simulator are synchronized. The task information in simulator updates whenever the generator generates a new task. The constructed topology is similar to Fig. 1, with each rack consisting of 10 servers.

All results presented in this paper in each setup are average values computed over 2000 tasks. We evaluate different algorithms using three metrics — number of tasks completed by the remaining volume, and link utilization. *Remaining volume* refers to the amount of data in GB, whose transmission to the destination server was not completed by the deadline. *Link utilization* is the ratio of the total amount of data that can be transferred through a given network link to the total amount of data that was actually transferred through that link. In all settings, erasure-coded chunks are placed uniformly following the best practices of many distributed storage system in a real world as discussed in Section III.

B. Sensitivity Analysis via Simulations

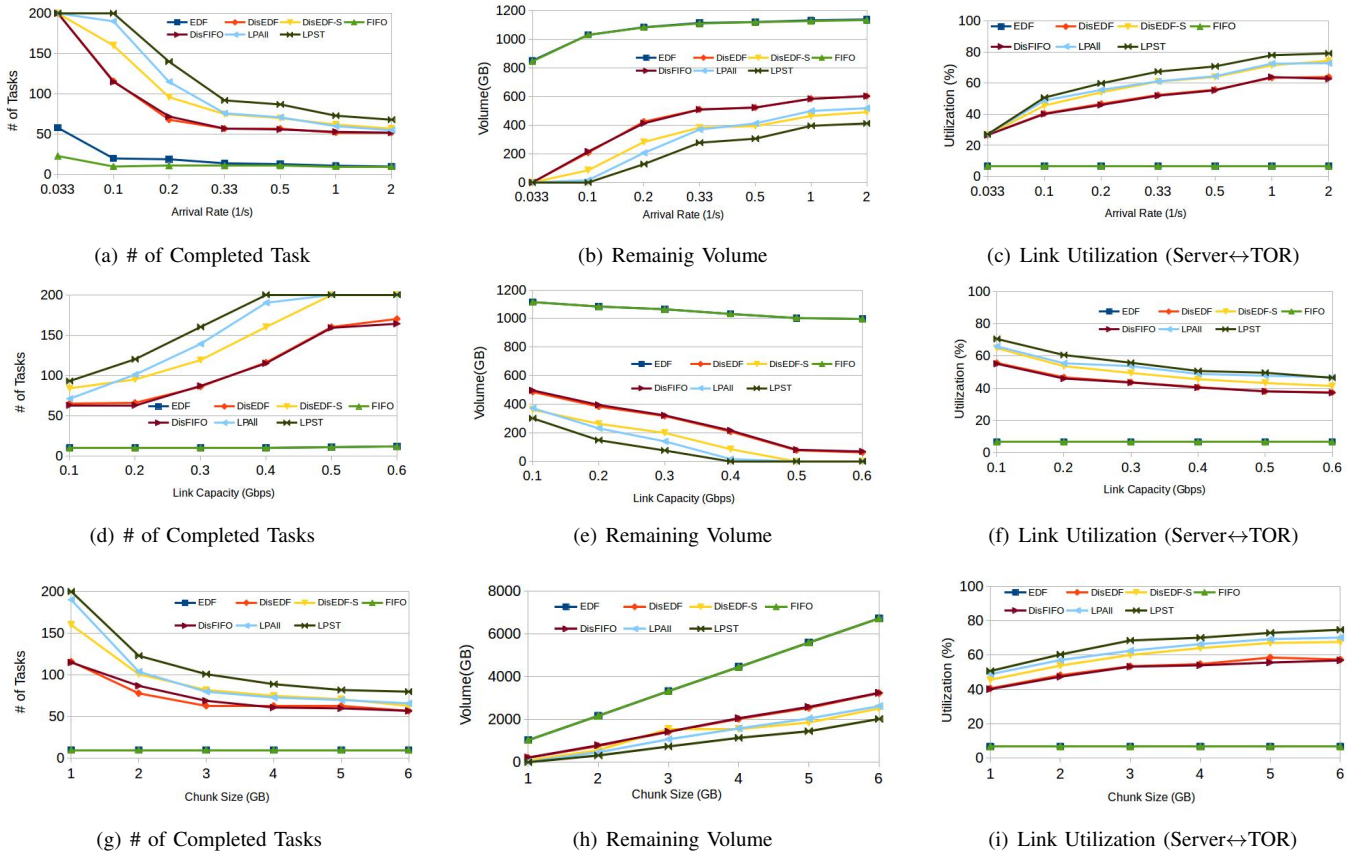
Next we thoroughly investigate the effect of several parameters on the performance of various agorithms. We chose three important parameters – arrival rate, available link capacity, and data size. Fig. 2 shows the results. Overall, for almost all parameter space we explored, we found that LPST either outperforms competing algorithms, or performs at least as good as other algorithms. The parameters used for the simulation runs are described in Table II.

Arrival Rate: The rate at which background jobs arrive in the system is an important parameter that can affect performance. We conducted a simulation study with different arrival rates while fixing other parameters. The results are shown in Fig. 2(a)$\sim$2(c). The impact of arrival rate is quite significant, e.g. the number of completed tasks can be degraded by 66% under demanding arrival rates. Not surprisingly, as the arrival pattern becomes more sparse, the performance gap between LPST and greedy alternatives gets narrower. In the most sparse arrival pattern we tried (arrival rate of 0.033 tasks per second), many algorithms perform equally well.

Link Capacity: Available link capacity is yet another important factor. We vary the link capacity from 100 Mbps to 600 Mbps. The results are presented in Fig. 2(d)$\sim$2(f). It turns out that the impact of changing link capacity is very similar to that of changing deadlines. With more available bandwidth, LPST has more room for optimization, so it outperforms other algorithms. However, if we have just enough bandwidth for a given workload, many algorithms show good performance.

	# of Tasks	Erasure Code	Arrival Rate(s^{-1})	Chunk Size	Link (Mbps)	Deadline
Baseline	200	(9,6)	Poisson, 0.1	1 GB	250/875	$^{+}$LRT * Uniform(1,15)
Arrival Rate	200	(9,6)	Poisson, 1/30∼2	1 GB	400/1400	LRT * 10
Link Capacity	200	(9,6)	Poisson, 0.1	1 GB	100/350, 200/700, . . ., 500/1750	LRT * 10
Volume	200	(9,6)	Poisson, 0.1	1∼6 GB	400/1400 Mbps	LRT * 10

TABLE II
PARAMETERS USED FOR SIMULATION & EXPERIMENTS. $^{+}$LEAST REQUIRED TIME.



(a) # of Completed Task (b) Remainig Volume (c) Link Utilization (Server↔TOR)

(d) # of Completed Tasks (e) Remaining Volume (f) Link Utilization (Server↔TOR)

(g) # of Completed Tasks (h) Remaining Volume (i) Link Utilization (Server↔TOR)

Fig. 2. Sensitivity analysis via simulation: arrival rate (2(a), 2(b), 2(c)), link capacity (2(d), 2(e), 2(f)), and data sizes (2(g), 2(h), 2(i)).

As with the case of changing deadlines, the link utilization decreases gradually for the same reason.

Data Size: Lastly, we examine the sensitivity of various algorithms with respect to erasure-coded chunk size. Fig. 2(g)∼2(i) show that LPST performs consistently better than other algorithms in the entire range of data size we used in our experiments. In other words, unlike other resources tightly coupled with the infrastructure, data size has less impact on the relative performance of these algorithms. This is mainly because data size impacts all algorithms in a similar way. Data size does not impact the factors for which these algorithms are designed. Specifically, data size does not directly affect parameters like network topology, source selection, bandwidth allocation, and deadlines.

V. RELATED WORK

Although LPST is the first algorithm to tackle S3 problem (Sec. II), it is inspired by vast amount of related work.

Our notion of remaining time flexibility is inspired by a classic scheduling algorithm called Least Slack Time First (LSTF) [17]. LSTF used a metric called *slack*, which is conceptually similar to RTF, to schedule tasks to a single or multiple processors and it can be easily applied to packet scheduling problem as well [18]. In S3 problem, similar to the reason that other simple heuristics will not work well, it is not enough to blindly apply LSTF since we need to additionally consider source selection and bandwidth allocation problems.

Aside from LSTF, many heuristic algorithms have been extensively studied in the community. The representative algorithms include Early Deadline First (EDF), First In First Out (FIFO), and Linear Programming (LP) and we discussed these

algorithms with respect to S3 problem in Sec. III-B.

Some advanced algorithms based on these concepts are as follows. Algorithms based on FIFO has been applied for multicast traffic [13] and packet scheduling [14] to maximize system throughput. In [10], authors described a Global EDF algorithm to schedule parallel real-time tasks, which has provable performance bounds and overcomes task heterogeneity noted in [11], [12]. Lastly, using the model of a time-slotted system, traffic scheduling with deadlines can be formulated as a Linear Program (LP) problem. The complexity analysis of LP can be found in [19], [20], [21]. However, traffic scheduling complexity grows quickly as network size and granularity increase [22], and it may lead to integer constraints when source selection and routing are involved [23].

Complementary to our work, substantial amount of work is proposed on reducing the amount of repair traffic in erasure coded storage systems. The list includes practical implementations that maintain local parities [24], [25] and novel codes that provide theoretical guarantees, e.g., MSR and MBR codes [26]. Since we assume MDS code in this paper and the majority of erasure codes used in practice maintain MDS property, our algorithm can be directly applicable to most work in this category.

Tangentially related to our work, there exist a set of studies that can fortify the importance of S3 problem. [6] characterized backup workloads in EMC Data Domain backup systems in production use and showed that on average, background traffic per week is equivalent to about 21% of total stored data. For repair traffic, a study of failure and repair characteristics of tasks and servers using the Google Cloud trace is provided in [27]. Within the trace spanning 29 days, there were over 144 million and 37 thousand events for tasks and servers respectively. The vast majority of repair times are relatively short but had large deviation [28], leading to undesirable impact to the infrastructure.

VI. Conclusions

In this paper, we consider the problem of optimizing background traffic in erasure-coded distributed storage systems, to maximize the number of tasks meeting deadlines under data placement, network topology and bandwidth constraints. The proposed solution makes use of *Remaining Time Flexibility* to select active tasks for each scheduling interval and linear programming to apportion bandwidth among the them. Our evaluation results based on both simulations and experiments on a real cluster showed that our proposed algorithm significantly outperforms six competing algorithms. In the future, we plan to evaluate LPST using other topologies, such as fat-tree or Bcube, and prove a performance bound for the algorithm.

References

[1] "Ceph," http://ceph.com/.

[2] "Yahoo Cloud Object Store," http://yahooeng.tumblr.com/post/116391291701/yahoo-cloud-object-store-object-storage-at.

[3] "Google Colossus File System," http://static.googleusercontent.com/media/research.google.com/en/university/relations/facultysummit2010/storage_architecture_and_challenges.pdf.

[4] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, E. Lin, W. Liu, S. Pan, S. Shankar, V. Sivakumar, L. Tang *et al.*, "F4: Facebooks warm blob storage system," in *OSDI*, 2014.

[5] Hakim Weatherspoon and John D. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Peer-to-Peer Systems*, 2002, pp. 328–337.

[6] Wallace, Grant, et al., "Characteristics of backup workloads in production systems," in *USENIX FAST*, 2012.

[7] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos, "Elasticswitch: practical work-conserving bandwidth guarantees for cloud computing," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 351–362, 2013.

[8] Ballani et al., "Chatty tenants and the cloud network sharing problem." in *NSDI*, 2013, pp. 171–184.

[9] Jeyakumar et al., "Eyeq: Practical network performance isolation for the multi-tenant cloud," in *HotCloud*. USENIX Association, 2012.

[10] Jing Li, Zheng Luo, David Ferry, Kunal Agrawal, Christopher Gill, and Chenyang Lu, "Global edf scheduling for parallel real-time tasks," in *Real-Time Syst*, 2014.

[11] Abusayeed Saifullah, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher Gill, "Multi-core real-time scheduling for generalized parallel task models," in *Real-Time Syst*, 2012.

[12] Lakshmanan K, Kato S, Rajkumar R., "Scheduling parallel real-time tasks on multi-core processors," in *Real-Time Systems Symposium (RTSS)*, 2010.

[13] Deng Pan and Yuanyuan Yang, "Fifo-based multicast scheduling algorithm for virtual output queued packet switches," in *IEEE TRANSACTIONS ON COMPUTERS*, vol. 54, no. 10, 2005.

[14] Kirill Kogan, Alejandro Lopez-Ortiz, Sergey I. Nikolenko, and Alexander V. Sirotkin, "Online scheduling fifo policies with admission and push-out," in *Theory Comput Syst*, 2016.

[15] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: a high performance, server-centric network architecture for modular data centers," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 63–74, 2009.

[16] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint vm placement and routing for data center traffic engineering," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 2876–2880.

[17] J. Y.-T. Leung, "A new algorithm for scheduling periodic, real-time tasks," *Algorithmica*, vol. 4, no. 1-4, pp. 209–219, 1989.

[18] R. Mittal, R. Agarwal, S. Ratnasamy, and S. Shenker, "Universal packet scheduling," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, 2016, pp. 501–521.

[19] L. G. Khachiyan, "Polynomial algorithms in linear programming," *USSR Computational Mathematics and Mathematical Physics*, vol. 20, no. 1, pp. 53–72, 1980.

[20] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proceedings of the sixteenth annual ACM symposium on Theory of computing*. ACM, 1984, pp. 302–311.

[21] N. Megiddo, "Linear programming in linear time when the dimension is fixed," *Journal of the ACM (JACM)*, vol. 31, no. 1, pp. 114–127, 1984.

[22] P. R. Kumar and Sean P. Meynf, "Duality and linear programs for stability and performance analysis of queueing networks and scheduling policies," in *IEEE Transactions on Automatic Control*, vol. 41, no. 1, 1996.

[23] Kamal Al-Subhi Al-Harbi, Shokri Z. Selim, and Mazen Al-Sinan, "A multiobjective linear program for scheduling repetitive projects," in *Mazen Cost Engineering*, 1996.

[24] 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 *International Conference on Very Large Data Bases (VLDB)*, 2013.

[25] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, S. Yekhanin *et al.*, "Erasure coding in windows azure storage." in *Usenix annual technical conference*. Boston, MA, 2012, pp. 15–26.

[26] A. G. Dimakis, P. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *Information Theory, IEEE Transactions on*, vol. 56, no. 9, pp. 4539–4551, 2010.

[27] Peter Druschel and Antony Rowstron, "Storage management and caching in past, a largescale, persistent peer-to-peer storage utility," in *ACM SOSP*, 2001.

[28] W. Bolosky, J. Douceur, D. Ely, and M. Theimer, "Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs," in *Sigmetrics*, 2000.