Background Traffic Optimization for Meeting Deadlines in Data Center 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}@gwmail.gwu.edu, {mra, rpanta}@research.att.com

***Abstract*—Background traffic, such as repair, rebalance, backup and recovery traffic, often has large volume and consumes significant network resources in cloud storage systems. While having each application independently schedule its own background traffic can easily generate interference among data flows, causing violation of desired QoS requirements (e.g., latency and deadline), heuristic scheduling algorithms like Earliest-Deadline-First and First-In-First-Out are not able to take into account data center constraints such network topology or data chunk placement, thus resulting in unsatisfactory performance. In this paper, we propose a new algorithm, Linear Programming for Selected Tasks (LPST), which coordinate background traffic of different jobs to meet traffic deadline and optimize system throughput. In particular, our goal is to maximize the number of background traffic flows that meet their target deadlines under bandwidth constraints in data center storage systems. Using realistic traffic trace, our simulation results show that the proposed algorithm significantly improves task processing time and the probability of meeting deadlines.**

I. Introduction

Background traffic constitutes a significant portion of overall traffic in a typical data center, e.g., repair traffic, rebalance traffic, backup and recovery traffic, ingestion of data in online storage and Big Data applications. For example, as for data backups, they might typically consist of a full copy of the primary data once per week (i.e., a weekly full), plus a daily backup of the files modified since the previous backup (i.e., a daily incremental) [1]. In addition, a large percentage of the space is used by files hundreds of gigabytes in size with typically 6 week data retention [1]. As a result, background traffic often has large volume and consumes significant network resources. A scheme that make each application independently schedule its own background traffic can easily generate interference among data flows, causing violation of desired application deadlines.

The goal of this paper is to develop an online optimization algorithm for scheduling background traffic of all applications in data centers to maximize the number of tasks that meet application deadlines, under data placement, network topology and bandwidth constraints. Existing heuristics such as EDF (Early Deadline First), FIFO (First In First Out) and LP (Linear Programming) are not able to take into account constraints such as data center network topology or data chunk placement, thus resulting in unsatisfactory performance. In particular, FIFO is easy to be applied in real system but has relatively low performance as observed in [11], [12]. EDF works well in networks with simple topology, but for data center networks that often employ a tiered-structure consisting of Top-of-Rack (ToR) and exaggerator switches [13], [14], [15], EDF fails to maximize the completion time of all tasks with respect to their deadlines.

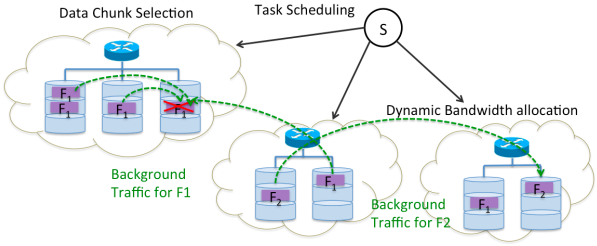


Fig. 1. A erasure-coded storage system with background traffic for 2 files, F_1 and F_2. The three key optimization degrees of freedoms are - task scheduling, data chunk selection, and bandwidth allocation.

We consider data center storage systems that use replication or erasure coding for data reliability. In a replication based storage system, copies of data are made and stored in different servers to avoid data loss caused by broken-down servers. Examples for replication based system include PAST [2] and Farsite [3]. Compared to replication, erasure codes are more complicated but have longer mean time to failure (MTTF) and occupy less bandwidth and storage to provide similar system durability [4]. Using a (n, k) erasure code, we split a file into k pieces and encode them into n chunks, each stored on a different node. The file can be retrieved by querying any k-out-of-n storage nodes. FreeHaven system used an information dispersal algorithm similar to erasure codes [7]. There is also a hybrid system, OceanStore, where replicas are used for read benefit and erasure codes are used for durability [8]. Besides storage systems, erasure coding can also be used in routing to reduce delays [9], [10].

The background traffic optimization problem aims to maximize the number of tasks that meet their deadlines in an online setting. To schedule each task, we need to jointly solve: (i) a chunk selection problem that determines the chunks used to generate background traffic, (ii) a bandwidth allocation problem that apportions bandwidth at ToR and exaggerator switches among active tasks, and (iii) a scheduling problem that schedules tasks with respect to their deadlines. This

978-1-4673-9457-4/16/$31.00 ©2016 IEEE

background traffic optimization problem can be formulated as a mixed-integer optimization that is hard to compute. The proposed algorithm uses a two-tier approach. First, we define Remaining Time Flexibility (RTF) to measure the slackness of tasks' starting time. RTF is the maximum time remaining before a task becomes infeasible given its deadline as well as current network topology, data placement, bandwidth constraints. Thus, a task with higher RTF is less urgent and can be postponed in the scheduling algorithm without increasing the risk of missing deadline. Second, we use RTF to select a subset of (relatively urgent) tasks and schedule their traffic through linear programming to determine the optimal bandwidth allocation for these tasks. The steps are then repeated for every task arrival and departure in an online setting. Using extensive simulations, we show that the proposed algorithm is able to significantly improve the number of tasks meeting deadlines for various network topology and job sizes.

II. Related Work

Scheduling problems widely exist in many fileds including manufacturing systems, computer systems and communication networks [16]. In past years, many algorithms were proposed and studied. Among them, there were three dominative algorithms, EDF (Early Deadline First), FIFO (First In First Out) and LP (Linear Programming).

In FIFO, the task with the earliest starting time will be processed. In [11], authors proposed a FIFO algorithm to manage multicast traffic, and provided hardware implementation performance. In [12], authors applied FIFO algorithm to solve online scheduling system. Their object is to maximize the total number of successfully transmitted packages. FIFO algorithms are widely used in package transmission and buffer management due to their simplicity. But it may not work well in our scenario with complicated topology.

EDF means that the task with the earliest deadline will be assigned to execute first. In [13], authors described a Global EDF algorithm to solve parallel real-time tasks. They proved that the Global EDF provided a capacity augmentation bound of 4 - 2m and a resource augmentation bound of 2 - 1m, where m means the number of tasks. They also overcame a EDF's shortage in previous publications - the unsuitability for real-time applications that employed different numbers of threads in different segments of computation [14], which was caused by decomposing tasks to subsets [15]. Althought this Global EDF seems great, it does not have constraints in its topology, which is one of the most difficult point in our scenario.

Linear programming technology is to calculate the optimal solution with constraints, which fits our goals well. Theoretically, linear programming can get the best solution. However, the calculation complexity grows dramatically when elements and constraints increase [16]. Many precursors who applied linear programming technology to solve practical problems simplified objective functions and constraints by making use of special characteristics of their scenarios [17].

In our work, we modify EDF algorithm to simplify the linear programming problem to get better solution with less time and resource.

$\mathcal{T}$	Set of time slots
$\mathcal{A}$	Given set of tasks
n	Number of tasks
$x_{t,i}$	Bandwidth of job i during time t
z_i	Whether task i is finished
$u_{i,l}$	Whether task i occupies link l
o_n	Souce of task n
i_n	Destination of task n
v_n	File volume of task n
d_n	Deadline of task n
s_n	Start time of task n
b_n	Assigned bandwidth for task n
f_n	Remaining time flexibility of task n
$\mathcal{R}$	Given set of racks
j	Number of racks
$\mathcal{S}$	Given set of servers
k	Number of servers in each rack
$\mathcal{C}_l$	Link capacity

TABLE I
Notations

III. System Model and Problem Statement

In this section, we describe the system model and problem formulation. The topology of servers is given. There is one aggregator switch, j racks and k servers in each rack. Set $\mathcal{R} = \{R_1, R_2, \ldots, R_{j-1}\}$ denotes the racks. Set $\mathcal{S} = \{S_{00}, S_{01}, \ldots, S_{0k}, S_{10}, \ldots, S_{jk}\}$ denotes the servers, where j means rack number and k means server number in this rack. If servers within the same rack want to communicate with each other, the source server sends data flow to the TOR(Top of Rack), and then the TOR sends data flow to the destination server. If servers from different racks want to transmit data, the source server first sends data to its TOR, then this TOR sends data to the aggregator. The aggregator sends data to the TOR of destionation server, and finally the destination server's TOR sends the data to the destination. Cl is link capacity. It depends on the topology. θ is the usage percentage of link capacity. Notations used in the paper is given in Table 1.

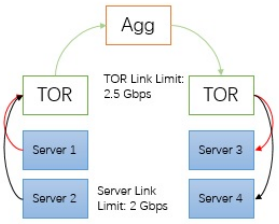
We consider both replication and erasure code based systems. In a replication based system, background traffic will be transmitted from one source to one destination. In this scenario, we are given tasks $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ for management. Suppose $\mathcal{T} = \{T_0, T_1, \ldots, T_m\}$ is the set of time slots, during which all the tasks execute. If the time slot is small enough, then it can represent the real situation without loss. Suppose the system allocates bandwidth $x_{t,i}$ to job i in time slot t ϵ $\mathcal{T}$. Variable z_i is one when the job i is finished before its deadline, and zero otherwise. Matrix U represents whether jobs occupy links. Suppose its element $u_{i,l}$ is one when job i uses the link l, and zero otherwise. v_i is the volume of the task i. The goal of our algorithm is to complete as many tasks before their deadlines as possible. We formally model this as the following task number and link usage optimal resource-allocation problem (TLORA).

In a popular (9, 3) erasure code based system, sources are not determined. If one server breaks down, 6 sources need

to be selected from 9-1 original copies. Then, background traffic will be transmitted from 6 sources to one destination. Therefore, if sources are chosen out, erasure code based scenario is similar to replication based scenario.

TLORA
max $\sum_i z_i$
s.t.
$\forall$ i, $z_i v_i \leq \sum_m x_{t,i} T_m$
$\forall\ T_m$, $\sum_i x_{t,i} u_{i,l} \leq \mathcal{C}_l \theta T_m$
$x_{t,i} \geq 0$,
$z_i\ \epsilon$ 0,1

An illustrative example. We use a simple example shown in figure 1 to illustrate that EDF and FIFO can not get the best solution. If EDF or FIFO is used, A1 will be assigned 2Gbps bandwidth until it finishes at 0.6 second. A2 will assigned only 0.5 Gbps until 0.6 second because of the TOR link bandwidth limit. And then A2 cannot meet its deadline. However, if A1 is assigned 0.95 Gbps when A2 appears. And A2 is assigned 1.55 Gbps. A2 can finish at 1.04 second. A1 can finish at 0.94 second. Both of them can meet their deadlines. LP is theoretically supposed to obtain best solution for determinate elements, but it consumes large resource and time and hard to be applied in real system [16], [17]. So, in this paper, we propose an algorithm which combines EDF and LP and therefore have advantages of both EDF and LP.



Task	Route	Start (s)	Deadline (s)	Volume (bits)
A1	Server1 to Server3	0.1	1	1 G
A2	Server2 to Server4	0.2	1.1	1.3 G

Fig. 2. Example

IV. Our Proposed Solution

In this section, we describe our proposed solution - LPST (Linear Programming for Selected Tasks) in details.

As mentioned before, there are 3 key features in our design. The first one is our seemingly greedy multi-goals. To increase the completed task number, we modify EDF algorithm to select relativly "emergent" tasks. Then we apply LP to the selected tasks. The object function of linear programming is the utilization of bandwidth, which will result in the reduction of resource usage and latency. The second difficulty is the topology. We turn the constraints in topogy to the constraint functions in LP. Therefore, the solution will suit the complicated three-layer topology. The third point is the uncertainty of source selection in erasure code based scenario. We use greedy algorithm to select sources.

In replication based scenario, we first try to select m most emergent tasks from all n tasks, and use linear programming to assign bandwidth for them. For task A_n, a file of volume v_n needs to be transmitted with assigned bandwidth b_n, from source server to destination server in before deadline d_n. f_n = deadline - present time - v_n/link capacity, which means the task is more emergent if its remaining time flexibility(RTF) is smaller. Here, we summarize the steps as follows.

Step 1: Sort n tasks by their remaining time flexibility. The smaller the remaining time flexibility is, the more emergent this task is.

Step 2: Select m most emergent tasks. Suppose current time is t_0. Suppose set $\mathcal{ST}$ include all the selected tasks. The selecting procedure is formulated as follows. This step is to increase compeleted task number.

Selection Procedures
for i = 1 to n {
$b_i = v_i/(d_i - t_0)$;
if links can support A_i, $\mathcal{ST} \leftarrow \mathcal{ST} \bigcap A_i$;
return $\mathcal{ST}$

Step 3: For m tasks in $\mathcal{ST}$, do linear programming to assign their bandwidth. The objective function is the bandwidth utilization. The constraints are the bandwidth capacity limits of the servers, TORs and aggregators. This step is to make full use of the bandwidth.

Assign Bandwidth by using LP
max $\sum_{z=1}^{m} b_z$
s.t.
$\forall$ link l,
$\sum_z b_z \leq \mathcal{C}_l$
$\forall A_z \epsilon \mathcal{ST}$,
$b_z \geq v_n/(d_n - t_0)$

Step 4: Use this bandwidth arrangement until one task is finished. If tasks cannot complete transmission before their deadlines, record how many bits they have transmitted and stop their transmission processes. Go back to step 1.

A simple example was shown in table 2 and table 3 to illustrate the algorithm. There are 10 servers and 1 rack. The link capacity is 2 Gbps for each server and 8 Gbps for the TOR.

The processing procedures are shown in table 3. (Notations: E = executing, W = waiting, F = finished). At 0.1 second time point, there are two tasks A1 and A2. f1, the RTF of A1, is smaller than f2. So A1 is more emergent than A2. The least required bandwidth of A1 is 1 $\div$ (0.6-0.1) = 2 Gbps. The server link capacity is 2 Gbps, so A1 occupies all bandwidth of server 1 link. Therefore, A2 needs to wait. At 0.5 second, A3 and A4 appear. If A3 and A4 are assigned their least required bandwidth, links can support them. So

Tasks	Starting Time(s)	Deadline(s)	Volumn(Gb)	Source	Destination
A1	0.1	0.6	1	1	2
A2	0.1	9	2.2	1	3
A3	0.5	1	0.75	3	4
A4	0.5	2	1.25	3	5
A5	2	15	3	3	9

TABLE II
AN EXAMPLE OF SCHEDULING PROBLEM WITH 5 TASKS

Time	RTF	Assigned Bandwidth(Gbps)	Task Status
0.1	f1<f2	A1(2)	A1(E),A2(W)
0.5	f1<f3<f4<f2	A1(2)A3(1.5),A4(0.5)	A1(E),A2(W),A3(E),A4(E)
0.6	f3<f4<f2	A2(0.5),A3(1.5),A4(0.5)	A1(F),A2(E),A3(E),A4(E)
1	f4<f2	A2(1),A4(1)	A2(E),A3(F),A4(E)
2	f2<f5	A2(1),A5(1)	A2(E),A4(F),A5(E)
3	f5	A5(2)	A2(F),A5(E)
4	f5	All tasks finish	A5(F)

TABLE III
ILLUSTRATION OF OUR ALGORITHM FOR THE 5-TASK EXAMPLE.

they are selected. A2 is still waiting for A1 occupies all of the link bandwidth. Apply linear programming to A1, A3, A4. We get the bandwidth assignment for the 3 tasks. Following procedures are similar.

In erasure code based scenario, we first try to select 6 sources from 8 sources. For existed tasks, calculate their least required bandwidth (volume/remaining time) and add them up. Select 6 sources from 6 least filled servers and racks. Then the input becomes 6n tasks. After that, we apply similar algorithm as replication based scenario.

V. SIMULATION RESULTS

We compared the performance of LPST with other 5 algorithms, including EDF, Disjointed EDF, FIFOFB, FIFOLB, and LPAll. EDF is early deadline first algorithm. The task with the earliest deadline will be assigned full bandwidth of the links. Disjointed EDF is an improvement of EDF. If tasks are not sharing same links, they will be assigned full bandwidth at the same time. FIFOFB is first in first out with full bandwidth arrangement. The task with the earliest start time will be assigned full bandwidth of the links. FIFOLB is first in first out with the least required bandwidth. Divide the volume of the task by its remaining time results in the least required bandwidth. Fill the links by earlier tasks with the least required bandwidth. LPAll uses linear programming to arrange all tasks. The objective function is total utilized bandwidth. The constraints are link capacity limits.

We implemented simulations in 2 topologies, small network and large network. Small network had one aggregator, one rack and 10 servers. Large network had 1 aggregator, 3 racks and 10 servers in each rack. We set completed task number, total processing time and average task completion time as three metrics of simulation. The simulation generated tasks in 1 week with two different arrival type - constant arrival and possion arrival. There were dense arrival (arrival rate was 1 task/10 min) and sparse arrival (arrival rate was 1 task/hour).

In replication based scenario, we simulated the performance of different algorithms with sparse constant task arrival and dense poisson task arrival. LPST completed more tasks with relatively less time. FIFOFB had worst performance.

Figure 1 shows how many tasks were completed in each algorithm when tasks came in sparse and constant rate. As we can see, EDF, Disjointed EDF, FIFOLB and LPST completed all of the tasks. The fact that FIFOFB and LPAll performed worse in the aspect of task completion might be due to they did not consider task deadline in task assignment. Figure 2 displays the total processing time. The less, the better. Although LPAll did not complete all of the tasks, it performed well in total processing time, which was because it made the optimal solution for bandwidth assignment regardless deadlines. In contrary, FIFOLB worked well in task completion but terrible in processing time, which was because it did not pay enough attention to bandwidth assignment. Average task completion time is shown in figure 3. The total processing time of completed tasks divided by the task completion number resulted in this metric. In this aspect, EDF and FIFOFB performed worse than others. Our solution, LPST, completed the largest number of tasks with the least total processing time and average task completion time. It was the best in this scenario.

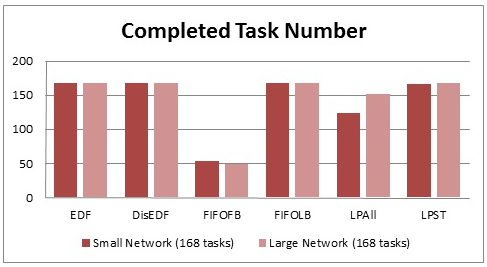


Fig. 3. Completed Task Number with Sparse Constant Task Arrival

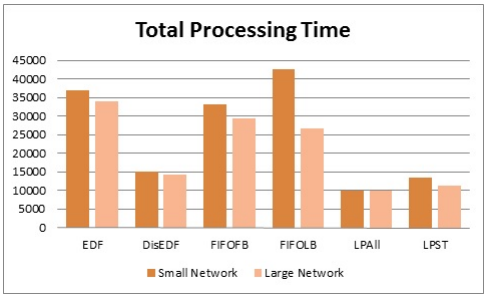


Fig. 4. Total Processing Time with Sparse Constant Task Arrival

Figure 4 displays the completed task number for dense Poisson arrival tasks. Similarly, FIFOFB and LPAll performed much worse than others. EDF is the third worst. Figure 5 describes the total processing time. LPAll performed well in

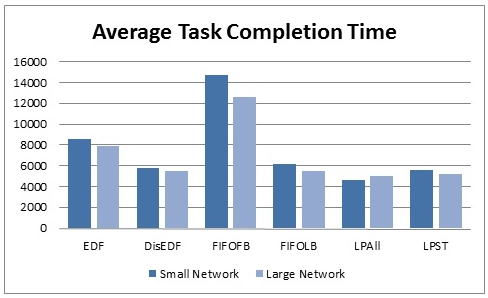


Fig. 5. Average Task Completion Time with Sparse Constant Task Arrival

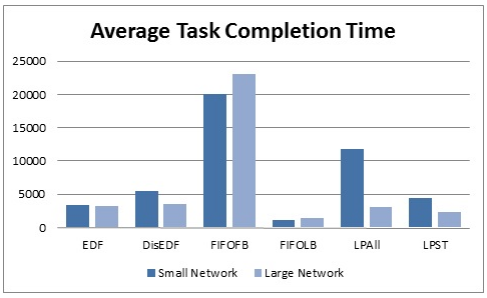


Fig. 8. Average Task Completion Time with Dense Poisson Task Arrival

this metric but it should not be considered because it did not complete nearly half of the tasks. FIFOLB and LPST were the best. In small network, FIFOLB worked even a little better. In figure 6, average task completion time is shown. FIFOLB was the best, but LPST performed well too. All in all, LPST won in replication scenario but FIFOLB also worked well.

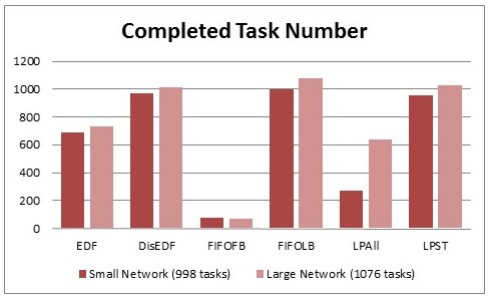


Fig. 6. Completed Task Number with Dense Poisson Task Arrival

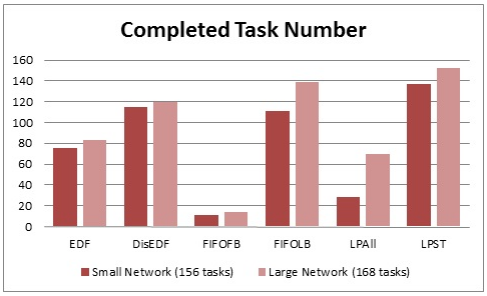


Fig. 9. Completed Task Number with Sparse Constant Task Arrival

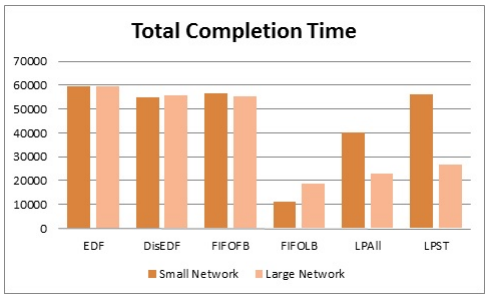


Fig. 7. Total Processing Time with Dense Poisson Task Arrival

In erasure code based scenario, we simulated the performance of different algorithms with sparse Poisson task arrival and dense constant task arrival. LPST completed much more tasks with less time. FIFOFB also had the worst performance. Figure 7 to figure 12 show their performance.

As shown in Figure 7, LPST was the best in the aspect of task completion. Although FIFOLB and Disjointed EDF also worked well, they were not as good as LPST. In Figure 8, LPST also defeated others as for total processing time. Similarly, LPAll had small total processing time because of a low task completion ratio. Figure 9 shows the average task completion time. FIFOLB performed a little better than LPST, which might be caused by each task did not wait for others to share the link. But LPST was still better than FIFOLB, because LPST completed much more tasks and consumed less time in total. In this scenario, LPST beat other algorithms more obviously.

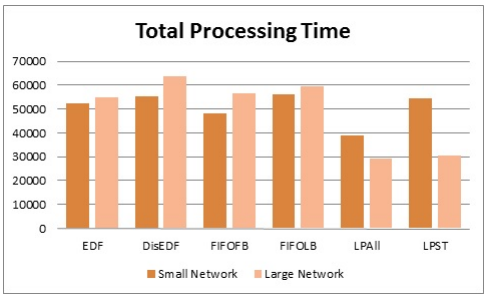


Fig. 10. Total Processing Time with Sparse Constant Task Arrival

For dense Poisson arrival tasks simulation results, shown in figure 10 to figure 12, LPST obviously defeated other algorithms. In summary, LPST was the best in erasure code based storage system, and FIFOLB was the second one. However, LPST performed much better than FIFOLB.

Therefore, our proposed algorithm, LPST, had the best performance in both replication based and erasure code based storage systems.

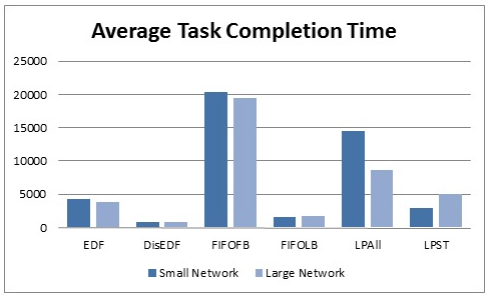


Fig. 11. Average Task Completion Time with Sparse Constant Task Arrival

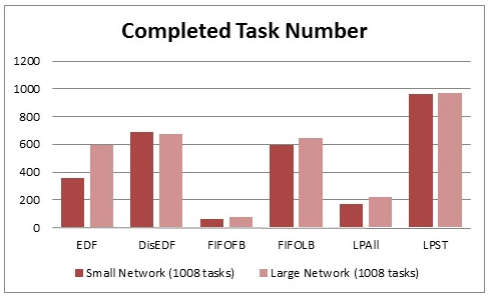


Fig. 12. Completed Task Number with Dense Poisson Task Arrival

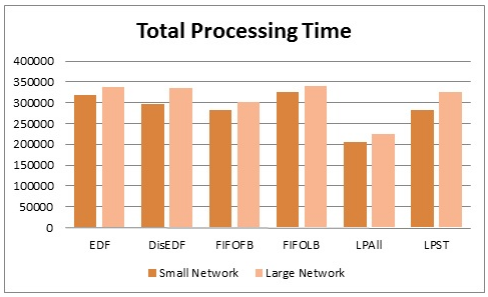


Fig. 13. Total Processing Time with Dense Poisson Task Arrival

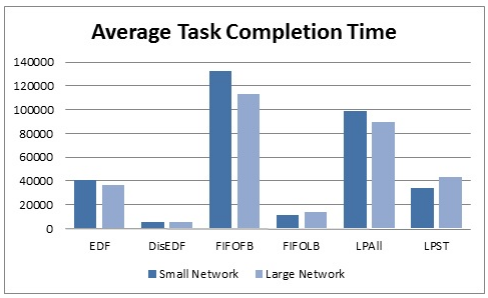


Fig. 14. Average Task Completion Time with Dense Poisson Task Arrival

VI. Conclusions

In this paper, we consider the problem of optimizing background traffic in both replication-based and erasure-coded data center storage systems. Our goals is to maximize the number of jobs 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 simulation results showed our proposed algorithm outperforms five existing heuristics. In the future we wish to implement our algorithm in a real data center storage and evaluate its performance.

References

[1] Wallace, Grant, et al. Characteristics of backup workloads in production systems. In FAST. 2012.

[2] Peter Druschel and Antony Rowstron. Storage management and caching in PAST, a largescale, persistent peer-to-peer storage utility. In Proc. of ACM SOSP. 2001.

[3] 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 Proc. of Sigmetrics. Jun. 2000.

[4] Hakim Weatherspoon and John D. Kubiatowicz. Erasure Coding Vs. Replication: A Quantitative Comparison. Peer-to-Peer Systems: pp 328-337. 2002.

[5] Dimitris S. Papailiopoulos, Alexandros G. Dimakis, and Viveck R. Cadambe. Repair Optimal Erasure Codes Through Hadamard Designs. IEEE Transactions on Information Theory, VOL. 59, NO. 5. May 2013.

[6] Alexandros G Dimakis, Vinod M Prabhakaran, and Kannan Ramchandran. Decentralized Erasure Codes for Distributed Networked Storage. IEEE Transactions on Information Theory. 2006.

[7] R. Dingledine, M. Freedman, and D. Molnar. The freehaven project: Distributed anonymous storage service. In Proc. of the Workshop on Design Issues in Anonymity and Unobservability. Jul. 2000.

[8] J. Kubiatowicz, Al. ET. Oceanstore: An architecture for global-scale persistent storage. In Proc. of ASPLOS (Nov. 2000), ACM.

[9] Luigi Rizzo. Effective Erasure Codes for Reliable Computer Communication Protocols. Computer Communication Review. 1997.

[10] Yong Wang, Sushant Jain, Margaret Martonosi, Kevin Fall. Erasure-Coding Based Routing for Opportunistic Networks. ACM Special Interest Group on Data Communication. 2005.

[11] Deng Pan and Yuanyuan Yang. FIFO-Based Multicast Scheduling Algorithm for Virtual Output Queued Packet Switches. IEEE TRANSACTIONS ON COMPUTERS, VOL. 54, NO. 10. Oct. 2005.

[12] Kirill Kogan, Alejandro Lopez-Ortiz, Sergey I. Nikolenko, and Alexander V. Sirotkin. Online Scheduling FIFO Policies with Admission and Push-Out. Theory Comput Syst (2016) 58:322344. Apr. 2015.

[13] Jing Li, Zheng Luo, David Ferry, Kunal Agrawal, Christopher Gill, and Chenyang Lu. Global EDF scheduling for parallel real-time tasks. Real-Time Syst (2015) 51:395439. Oct. 2014

[14] Abusayeed Saifullah, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher Gill. Multi-core real-time scheduling for generalized parallel task models. Real-Time Syst (2013) 49:404435. Oct. 2012

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

[16] P. R. Kumar and Sean P. Meynf. Duality and Linear Programs for Stability and Performance Analysis of Queueing networks and Scheduling Policies, IEEE Transactions on Automatic Control, Vol. 41, NO. 1, Jan. 1996

[17] Kamal Al-Subhi Al-Harbi, Shokri Z. Selim, and Mazen Al-Sinan. A Multiobjective Linear Program for Scheduling Repetitive Projects. Mazen Cost Engineering. Dec. 1996