2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems

MPLEX: In-situ Big Data Processing with Compute-Storage Multiplexing

Joy Rahman
Department of Computer Science
University of Texas at San Antonio
San Antonio, Texas 78258
Email: joy.rahman@utsa.edu

Palden Lama
Department of Computer Science
University of Texas at San Antonio
San Antonio, Texas 78258
Email: palden.lama@utsa.edu

***Abstract*—Cloud-based services are increasingly popular for big data analytics due to the flexibility, scalability, and cost-effectiveness of provisioning elastic resources on-demand. However, data analytics-as-a-service suffers from the overheads of data movement between compute and storage clusters, due to their decoupled architecture in existing cloud infrastructure. In this work, we propose a novel approach of in-situ big data processing on cloud storage by dynamically offloading data-intensive jobs from compute cluster to storage cluster, and improve job throughput. However, it is challenging to achieve this goal since introducing additional workload on the storage cluster can significantly impact interactive web requests that fetch cloud storage data, with strict SLA (service-level agreement) for tail latency. In this work, we present MPLEX, a system that augments data analytics-as-a-service by efficiently multiplexing compute and storage cluster to improve job throughput without violating the SLA of cloud storage service in terms of tail response time. It applies an SLA-aware opportunistic job scheduling technique supported by a machine learning based prediction model to exploit the dynamic workload conditions in the compute, and storage cluster. Performance evaluations on an OpenStack Swift cluster, and an OpenStack based virtual cluster of Hadoop VMs built atop NSFCloud's Chameleon testbed show that MPLEX improves the Hadoop job throughput by up to 1.7X, while maintaining the SLA for cloud storage service requests.**

I. Introduction

Today cloud-based services are increasingly popular for storing, and processing large-scale data, which is evident from the rise of numerous analytics-as-a-service solutions [29]. Big data analytics using parallel programming paradigms such as Hadoop, Dyrad, Spark etc. largely benefit from the cloud's resource elasticity, and pay-per-usage model [16], [21], [28]. Existing data analytics-as-a-service solutions offered by major cloud providers including Amazon Web Services, Google Cloud, and Microsoft Azure [1], [2], [4] mainly involve two decoupled service layers i.e, compute and storage. The compute layer (e.g. Amazon EC2) provide virtual machines (VMs) that run parallel frameworks like Hadoop MapReduce, Apache Spark, etc. and the storage layer (e.g. Amazon S3) hosts persistent data in object storage cluster due to its low cost and horizontal scalability. Hence, the data to be processed by the analytics engine needs to be copied from the object storage to the compute cluster, and the results are copied back for persistent storage. As a result, analytics-as-a-service suffers from large overheads of data movement both before and after data processing.

Cloud data centers consolidate physical machines in the compute cluster by using server virtualization, and achieve additional power savings by dynamically migrating VMs onto fewer hosts, and powering off idle hosts. Such agility in resource management has a significant impact on the total cost of ownership (TCO), and power efficiency of data centers [23], [26]. However, unlike compute clusters, cloud storage servers can not be scaled down easily even when the load is minimal. Cloud storage clusters use object-based storage technology (e.g OpenStack Swift [8], Ceph [34], Amazon S3, etc.) to store large-scale enterprise data in a cost-effective, and scalable manner. These storage servers hosting persistent data typically with a replication factor of 3, need to be up and running 24/7 to provide high availability and fault tolerance, irrespective of changing workload conditions. This motivates us to explore a novel approach of augmenting data analytics-as-a-service through in-situ big data processing on cloud storage. Multiplexing cloud resources by dynamically offloading data analytics jobs from compute to storage clusters can improve the job throughput by cutting down the data transfer overhead between the compute and storage clusters, and also by reducing the load on the compute cluster. We envision that such approach will enable a cloud provider to offer more cost-effective, and competitive data analytics service, which provides improved job throughput at no additional cost or, optionally at a fraction of a cost of provisioning more VMs.

Although compute-storage multiplexing approach seems intuitive, there are important challenges that need to be addressed. First, cloud storage services are already associated with high latency variance which is problematic when users are fetching and storing content for interactive applications [32], [35]. Hence, introducing additional workloads on the storage cluster can significantly impact interactive web requests that fetch cloud storage data, with strict SLA (service-level agreement) requirements for tail latency (e.g. 95^{th} percentile response time). This is verified by our motivating examples in Section II. Second, object-based cloud storage service provides lower throughput than ephemeral (non-persistent) storage, which are locally attached to the compute cluster in the cloud [31], [21]. Hence, naively offloading too many jobs to the storage cluster can be counterproductive. Hence, an important research challenge is to decide when to schedule data analytics workload on the cloud storage so that the overall job throughput can be improved without severely degrading the performance of the cloud storage service.

We present MPLEX, a system that augments data analytics-as-a-service by efficiently multiplexing compute and storage cluster with the aim to improve job throughput without violating the performance SLA of cloud storage service. To the best of our knowledge, this is the first study that provides a compute-storage multiplexing technique for in-situ big data processing. We make the following key contributions in MPLEX.

- MPLEX employs a supervised machine learning technique to train a logistic regression model that predicts the probability of violating the tail (e.g 95^{th} percentile) response time target of storage service requests, if additional data analytics jobs are offloaded to the storage cluster under current workload conditions. Such calibrated probabilities can be interpreted as confidence in making job scheduling decision.
- MPLEX continuously monitors the performance of the compute and storage clusters, and ranks them based on a scoring function to make effective job scheduling decisions that prefer the cluster with less load, and better performance.
- MPLEX applies an SLA-aware opportunistic job scheduling algorithm to exploit the dynamic workload conditions in the compute, and storage cluster.
- MPLEX is designed to interact with the compute, and storage clusters through standard REST APIs of Hadoop, and OpenStack Swift. It does not require any modification of Hadoop, or Swift, and can be easily extended to work on any data analytics-as-a-service platform.

We conduct extensive evaluations of MPLEX on NSF-Cloud's Chameleon testbed using data-analytics workload derived from the PUMA benchmark [9] along with Facebook's Hadoop job request trace profile [20], and object storage workload derived from the Intel COSBench (Cloud Object Storage Benchmark) [38] along with real traffic profile of Wikipedia. We demonstrate that MPLEX improves the Hadoop job throughput by up to 1.7X, while maintaining the 95^{th} percentile response time target of the cloud storage service.

The rest of the paper is organized as follows. Section 2 describes the motivation for in-situ big data processing in cloud storage, and the associated challenges. Section 3 presents the design, and implementation details of MPLEX. Section 4 evaluates our approach through extensive experiments. Section 5 discusses the related work. Section 6 concludes the paper.

II. Background and Motivation

To motivate our approach, we demonstrate that a simple offloading of Hadoop jobs from the compute to storage cluster can achieve significant improvement in job throughput. Then, we discuss the challenging issue of performance SLA violation for object storage workloads when Hadoop jobs are naively offloaded to the storage cluster, without a dynamic scheduling policy.

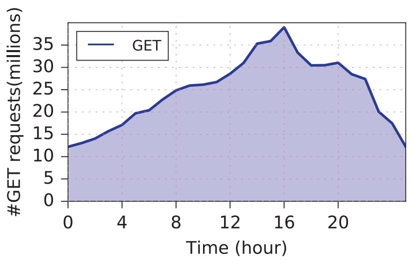


Figure 1: Wikipedia 24 hour request access pattern

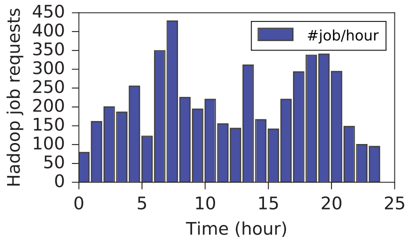


Figure 2: Facebook's 24 hour Hadoop job submission pattern

A. *Opportunity for In-situ Big Data processing*

In our case study, we setup an OpenStack-based cloud environment hosting a virtual cluster of Hadoop VMs for running data-analytics workload, and a separate OpenStack Swift cluster for object storage. Both clusters were built on the NSFCloud's Chameleon testbed. The Hadoop cluster, which was managed by the YARN [33] resource manager, ran data-analytics workload derived from the PUMA benchmark [9] along with Facebook's Hadoop job request trace profile [20] as shown in Figure 2. While the OpenStack Swift cluster ran a representative cloud storage benchmark, Intel COSBench (Cloud Object Storage Benchmark) [38], with real traffic profile of Wikipedia. As shown in Figure 1, a typical 24-hour web request pattern of Wikipedia website, generated from log files dump [14] shows significant variations at different times of the day, with a peak traffic for only 30% of the time.

In order to enable the execution of Hadoop jobs on top of OpenStack Swift nodes, we started Hadoop's daemon processes on the Swift cluster, and setup the Swift proxy server to adopt a dual role of object storage proxy, as well as YARN resource manager. The Hadoop jobs running on the Swift cluster can directly access the data objects via a Swift adapter similar to the one used in [3]. We applied a simple static multiplexing scheme to offload Hadoop jobs with a fixed 2:1 ratio between the compute, and storage clusters. Figure 3 shows that a simple static multiplexing approach can yield about 40% improvement in the average job throughput, even while ignoring the overhead of moving the input and output data of Hadoop jobs between the compute, and storage cluster.

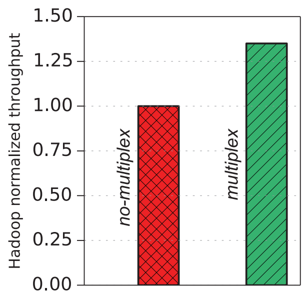


Figure 3: Throughput improvement by compute-storage multiplexing

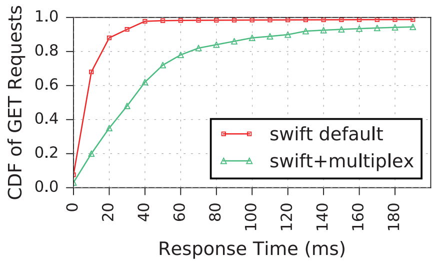


Figure 4: SLA challenge during multiplexing

B. Meeting Storage Performance SLA

Although offloading Hadoop jobs from the compute cluster to the storage cluster improves the overall job throughput, this additional workload can potentially deteriorate the object storage workload's response time, leading to SLA violations. Interactive web requests that fetch cloud storage data often have strict SLA requirements for tail latency (e.g. 95th percentile response time). Figure 4 shows that the 95th percentile response time of storage workloads derived from COSBench degrades drastically by almost 5X (from 37 ms to 180 ms) when Hadoop jobs are offloaded to the Swift cluster. Hence, in-situ big data processing requires an SLA-aware job scheduling technique that can multiplex compute, and storage clusters without violating the performance SLA of cloud storage service.

III. MPLEX Design and Implementation

We design, and implement MPLEX as a plugin module to augment data analytics-as-a-service solution. Figure 5 shows the overview of MPLEX architecture. It consists of four major components: the performance monitor, machine learning based prediction model, cluster score ranking module, and the SLA-aware opportunistic job scheduler. The performance monitor measures the current hadoop load, and job processing rate of the compute and storage clusters. In addition, it monitors the request response time, and current storage request load in the object storage cluster. The machine learning model is trained to predict the probability of violating the tail (e.g 95^{th} percentile) response time target of storage service requests,

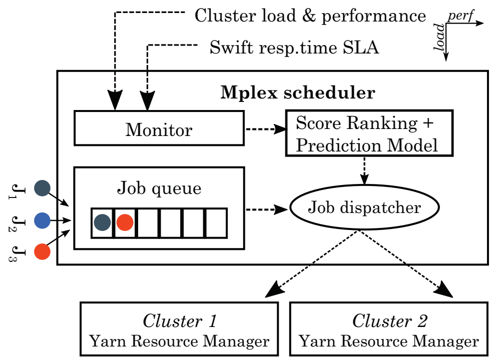


Figure 5: MPLEX Architecture

if additional data analytics jobs are offloaded to the storage cluster under current workload conditions. The cluster score ranking module calculates the performance scores of the two clusters based on the measured hadoop load, and job processing rates. The job scheduler is the central component of MPLEX that dispatches incoming jobs on one of the two clusters based on the cluster performance scores, while avoiding SLA violation.

A. Performance Monitoring

A key feature of MPLEX is its ability to react to changing workload conditions, and dynamic performance characteristics of the compute, and storage clusters. MPLEX interacts with the Hadoop YARN managers running on both clusters through REST APIs [11] in order to monitor their current load, and job performance. In particular, it uses Hadoop ResourceManager's REST API `/ws/v1/cluster/metrics` to measure the current load in terms of the number of jobs running, and the number of jobs currently waiting in the queue on each cluster. It also collects job execution statistics including the total job execution time, and the amount of data processed by each job, using the `/ws/v1/history/mapreduce/jobs` API. To keep track of the performance of object storage workloads, MPLEX periodically queries the Swift proxy server via the Recon API [5]. The time interval for statistics collection is set to be one second, which is small enough to capture changing workload conditions, and also coincides with Hadoop's default heartbeat interval. A Statsd[13] daemon is configured in a separate host to collect the following metrics:

- `proxy-server.object.GET.200.timing`
- `proxy-server.object.PUT.201.timing`
- `proxy-server.object.GET.200.xfer`
- `proxy-server.object.PUT.201.xfer`

B. Machine Learning Based Prediction Model

MPLEX uses a logistic regression (LR) model to predict the probability of performance SLA violation, if more Hadoop jobs are offloaded to the OpenStack Swift cluster, under current workload condition. Logistic regression (LR) is a widely used technique that models the probabilities of

Table I: CONFUSION MATRIX

		Predicted	
		SLA violation	No-violation
Actual	SLA violation	2454	45
	No-violation	120	784

dichotomous response variables such as 1 and 0, or true and false as a function of some explanatory variables. Unlike other classifiers, LR has the advantage of providing a quantified value for the strength of prediction, i.e. probability of an outcome, as opposed to only predicting the outcome. LR involves fitting the response variable using an equation of the form:

$$\begin{aligned} Logit(p) &= ln(p/(1-p)) \\ &= C_0 + C_1X_1 + C_2X_2 + \cdot\cdot + C_nX_n \end{aligned} \tag{1}$$

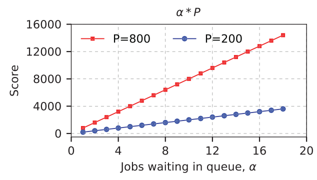
where p is the probability that the response variable (Y) is 1 (SLA violation occurs). C_0 is the intercept, and $C_1, C_2, C_3.., C_n$ are the coefficients, which measure the contribution of explanatory variables (features) to the variations in Y. In MPLEX, the features of the prediction model are chosen to be the 95^{th} percentile response time of Swift requests in the previous sampling interval, the current swift request load [1], the number of hadoop jobs running in the Swift cluster, and the number of hadoop jobs waiting in the queue. These features are selected by using a standard technique called *Stability selection* using Randomized Lasso [10].

1) Collection of Training Data: We use the implementation of logistic regression from scikit-learn, a machine learning package for python. The training process is conducted on the platform described in Section IV. Training data sets are collected by submitting several small hadoop jobs to the Swift cluster, while simultaneously running the COSBench workload at various load intensities, and workload profiles. The data corresponding to the explanatory variables is sampled at one second interval using standard REST APIs of Hadoop, and Swift. The binary response variable for the training data is derived by evaluating whether the 95^{th} percentile response time of Swift requests exceed a given SLA target or not. Our overall training data set has 3403 instances.

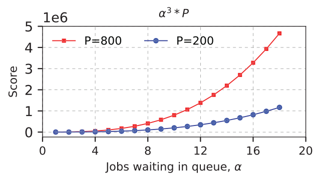
2) Evaluation of Logistic Regression Model: To verify the effectiveness of our logistic regression model, we applied a stratified 10-fold cross validation on the entire data set. It shows 3238/3403 (or 95%) overall success rate. Table I gives the confusion matrix.

C. Scoring Compute and Storage Clusters

MPLEX ranks the compute, and storage clusters based on a scoring function to make job scheduling decisions that prefer the cluster with less load, and better performance. The final score of a cluster, $Score_{cluster}$ is computed as the product



(a) linear score



(b) cubic score

Figure 6: Comparison between linear and cubic scoring functions. For differing values of P, the difference in queue-size estimates required for the scores of two clusters to be equal is smaller for the cubic function (thus penalizing longer queues)

of cluster load based score, and performance based score, as shown in Equation 2.

$$Score_{cluster} = L_{cluster} * P_{cluster} \tag{2}$$

Here, the job performance based score, $P_{cluster}$, is derived from the average normalized execution time. First, for each job_i, normalized execution time P_i is calculated by equation 3, where T_i denotes the execution time of job_i and S_i denotes the input data size processed by the job.

$$P_i = \frac{T_i}{S_i} \tag{3}$$

To filter out the noise from the job performance based score, we use the exponential moving average of P_i as shown in Equation 4. Here, P_{curr} is the current value of normalized execution time, and P_{prev} is the value of normalized execution time in the previous interval. More preference is given to the recent values based on the predefined constant σ.

$$P_{cluster} = \sigma * P_{curr} + (1-\sigma) * P_{prev} \tag{4}$$

As shown in Equation 5, the cluster load based score, $L_{cluster}$, is calculated as a cubic function of the number of Hadoop jobs waiting to be processed in the cluster, denoted by α. As a result, Equations 2 gives more weight to α, in order to punish longer job queues. This is in contrast to an alternative linear scoring function that simply calculates a product of α, and P. Similar to the approach used in [32], Figure 6, shows that under a linear scoring scheme, for a queue-size estimate of 4 at the slower server cluster, only a corresponding value of 16 at the faster cluster would cause the MPLEX job scheduler to prefer the slower cluster. Under such condition, the faster cluster may build up a very long queue of jobs. However, if the relative performance of the cluster decreases due to change in Swift workload intensity, all jobs in its queue will incur higher waiting times. MPLEX uses cubic scoring function to address this issue.

$$L_{cluster} = \alpha^3_{cluster} \tag{5}$$

[1]average system load across swift object servers

Algorithm 1 Mplex Algorithm.

```
1:  Variables: Swift SLA violation probability bound $\lambda$=0.5;

2:  /* MONITOR+SCORE runs every $\triangle$ interval in background */
3:  procedure MONITOR+SCORE
4:      Collect Hadoop job specific cluster load, and performance data from YARN Resource Managers.
5:      Collect the 95$^{th}$ percentile response time, and average swift request load from Swift Proxy.
6:      $L_s$ = Swift cluster load (Eq. 5)
7:      $L_h$ = HDFS cluster load (Eq. 5)
8:      $P_s$ = Swift performance score (Eq. 4)
9:      $P_h$ = HDFS performance score (Eq. 4)
10:     $Score_s = L_s * P_s$
11:     $Score_h = L_h * P_h$
12: end procedure

13: procedure SCHEDULE
14:     while $true$ do
15:         get job from MPLEX queue
16:         $p$ = predict probability of SLA violation if one more job is submitted to Swift cluster.
17:         if $Score_s < Score_h$ and $p < \lambda$ then
18:             submit job to Swift cluster
19:         else
20:             submit job to HDFS cluster
21:         end if
22:     end while
23: end procedure
```

Table II: Experiment Testbed

VM	Core	Mem	Disk	Net
Storage Proxy Server	2	8GB	100GB	1G
Storage Object Server	2	8GB	100GB	1G
Compute Master Node	4	8GB	30GB	1G
Compute Worker Node	4	8GB	30GB	1G

D. SLA-aware Opportunistic Job Scheduling

MPLEX uses an SLA-aware opportunistic job scheduling algorithm to efficiently multiplex the compute and storage clusters in the face of changing system dynamics. Algorithm 1 details the scheduling policy. The *monitor+score* procedure (lines 3-12) is executed in background every $\triangle$ time interval to collect Hadoop job specific cluster load, and performance data from YARN Resource Managers, and the 95^{th} percentile response time, and average swift request load from the Swift Proxy. It also computes the cluster scores given by Equation 2. In our current implementation of MPLEX, $\triangle$ is set to one second, which is small enough to reflect dynamic changes in the compute, and storage clusters. The overhead of collecting the cluster statistics is negligible since MPLEX only polls information from the Swift proxy, and the YARN resource managers, and not from individual cluster nodes. In general, we suggest setting $\triangle$ to a smaller value than the minimum inter-job arrival times.

Table III: Swift traffic profile

Swift Load	% cluster load	#workers
Low	0%-5%	8
Medium	10%-15%	16
High	>15%	20

Table IV: Swift benchmark profile

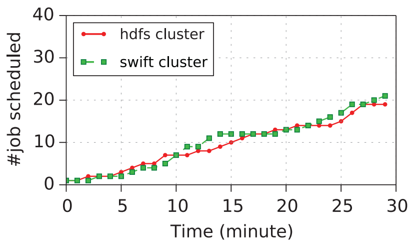
Traffic profile	%distribution	total read	total write
read-heavy	r:w=90:10	45 GB	5GB
write-heavy	r:w=50:50	25GB	25GB
r/w-balanced	r:w=70:30	38GB	12GB

The *schedule* procedure (lines 13-23) dispatches incoming Hadoop jobs on one of the two clusters based on their scores, and the predicted probability of SLA violation. In particular, the cluster with the lowest score is preferred for job execution. Furthermore, the scheduler uses the logistic regression model to predict the probability of violating the 95^{th} percentile response time target of Swift storage requests, if one more Hadoop job is submitted to the Swift cluster. A job is submitted to the storage cluster, only if the predicted probability of SLA violation is below the given threshold λ. In this paper, we apply a commonly used threshold of 0.5 (50%) to determine the outcome predicted by logistic regression. However, more conservative threshold values can be used to gain more confidence in avoiding SLA violation. Furthermore, if there is any SLA violation due to prediction errors, MPLEX is designed to kill the Hadoop jobs running in the Swift cluster, and re-submit them to the HDFS cluster.

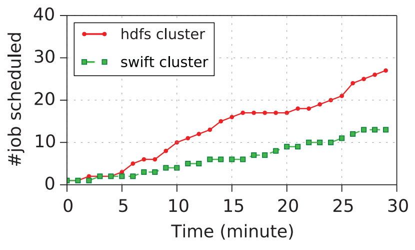
IV. Evaluation

A. Testbed Setup

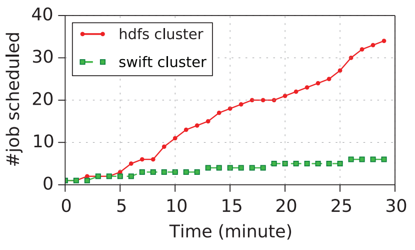
For performance evaluation, we setup an OpenStack-based cloud environment hosting a virtual cluster of Hadoop VMs for running data-analytics workload, and a separate OpenStack Swift cluster for object storage. Both clusters are built on the NSFCloud's Chameleon testbed. The virtual Hadoop cluster hosted on the compute nodes is managed by the YARN [33] resource manager, and is configured with HDFS [17] file system. The Hadoop version used is 2.7.3. There is one resource manager node and eight worker nodes. The resource manager node also hosts the Namenode for HDFS. On each worker node, a node manager daemon, and a data node daemon are running with an HDFS partition of size $30GB$. We deploy the default capacity scheduler [30] for YARN where each job demands the request based on CPU and memory capacity. The storage cluster is based on OpenStack Swift (Kilo) object storage implementation [7]. There is one proxy server to handle all the users' GET/PUT requests and eight object servers that actually host the object files. All of the



(a)



(b)



(c)

Figure 8: Multiplexing of Hadoop jobs between compute and storage cluster in (a) Low Swift traffic (b) Medium Swift traffic and (c) High Swift traffic hour

Table V: Hadoop benchmark

Benchmark	%job mix	Input Size	#Maps	#Reduces
grep	40%	1-32GB	64	1-4
wordcount	30%	1-32GB	64	1-4
histogram-ratings	30%	1-16GB	64	1-4

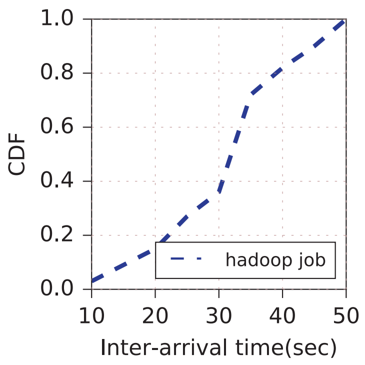


Figure 7: Hadoop job inter-arrival time distribution

object servers are configured under zone-1 with a replication factor of three. Each object server has a special software loopback partition with a capacity of $100GB$ to host all the object data. The servers are connected by $1G$ of network configuration. A separate keystone [6] server is used to handle all of the authentication requests.

In order to enable the execution of Hadoop jobs on top of OpenStack Swift nodes, we started Hadoop's daemon processes on the Swift cluster, and setup the Swift proxy server to adopt a dual role of object storage proxy, as well as YARN resource manager. The Hadoop jobs running on the Swift cluster can directly access the data objects via a Swift adapter. The MPLEX system is run on a separate virtual machine that has connectivity to both the YARN resource manager as well as to the Swift proxy daemon over REST APIs [11], [5]. The same host is used to receive Hadoop job requests form users.

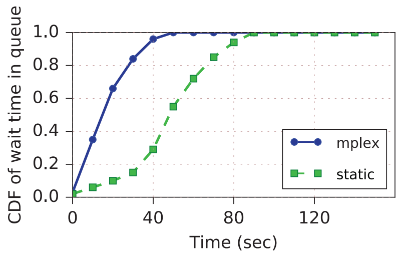
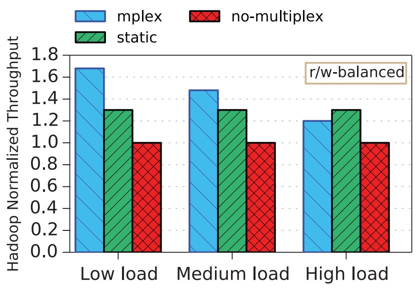


Figure 9: Hadoop job queue waiting time in the Swift cluster

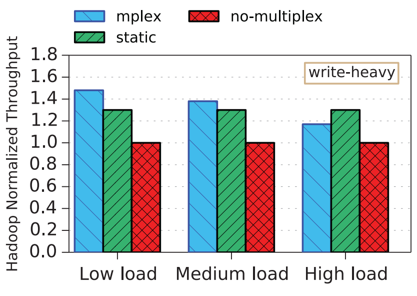
B. Workloads

To understand the effectiveness of MPLEX, we use a set of benchmarks with profiles that models the production workload. For Swift workload, we use the Intel COSBench (Cloud Object Storage Benchmark) [38]. COSBench provides several customization options, particularly allows defining the length of the run, the number of concurrent requests, workload profile, and data size. To generate synthetic workload that is scaled down to fit our cluster while still keeping key features of production load, we define three different traffic load hours. As shown in Table III, these traffic hours generate proportionally distinct load. For example, the low traffic hour is the time that has minimum incoming Swift traffic request. The medium traffic hour is the time where the cluster still sees the good amount of Swift web traffic. The high traffic hour, on the contrary, is the time segment where the cluster is overloaded with Swift traffic requests. This three-time segment can perfectly align with the observed traffic behavior as shown in Figure 1(a).

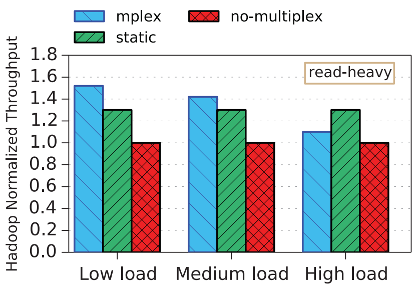
Object storage is by nature read intensive and exhibits better read performance compared to write[31]. The real web traffic nature is also GET intensive. Hence, we modeled our Swift workload to have three different profile as shown in Table IV based on the proportion of GET requests. The *ready-heavy* profile exhibits almost 90% of GET requests, the *write heavy* profile exhibits as much as 50% of PUT requests while the *read-write balanced* profile exhibits the closest web traffic nature of 70%GET and 30%PUT. Among these



(a)

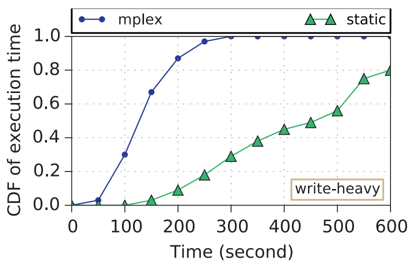


(b)

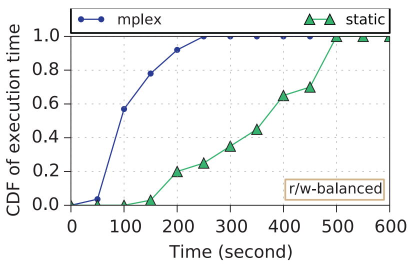


(c)

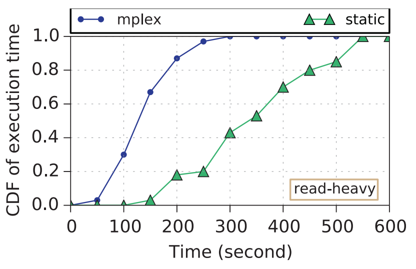
Figure 10: Throughput improvement by placing Hadoop jobs to storage cloud (a) Low Swift traffic (b) Medium Swift traffic and (c) High Swift traffic



(a)



(b)



(c)

Figure 11: Hadoop job execution time in storage cluster at (a) write-heavy GET traffic (b) r/w balanced GET traffic and (c) read-heavy GET traffic

workload profiles, we use a file size distribution where 80% of the objects are between $64KB$ to $512KB$, 10% between $1 \sim 64KB$ and the rest is between $512 \sim 4096KB$. This accurately mimics the production storage traffic as outlined in [15].

For the Hadoop workload, we use the PUMA benchmark [9]. The Table V shows the benchmark and associated configuration used. When reading from the Swift file system, the number of mappers is determined by the number of input files in the container. All of our experiments maintain a fixed number of 64 mappers with same file size for both HDFS and Swift input files so that performance can be compared against each other. The workload arrival pattern follows Facebooks Hadoop job request trace profile. Figure 7 shows the inter-arrival time distribution of Hadoop jobs.

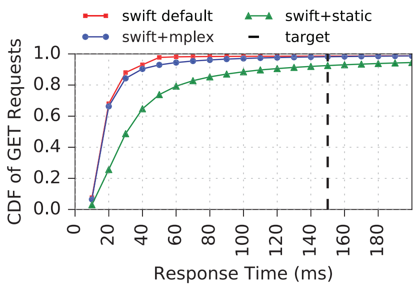
C. Scheduling in Action

In this section, we evaluate the effectiveness of MPLEX in making opportunistic job scheduling decisions in the face of changing workload conditions. We run experiments with three different clusters load scenario as described in Table III. Figure 8 shows the number of the jobs scheduled on each cluster over time in these three different traffic load hours. As shown in Figure 8(a), the storage cluster is able to accommodate a large number of Hadoop jobs, due to low Swift traffic load. For medium Swift traffic load, around 70% of the jobs are sent to HDFS cluster and the rest to the Swift cluster as shown in Figure 8(b). For a high Swift load scenario shown in Figure 8(c), only 15% of the jobs are placed on the Swift cluster.

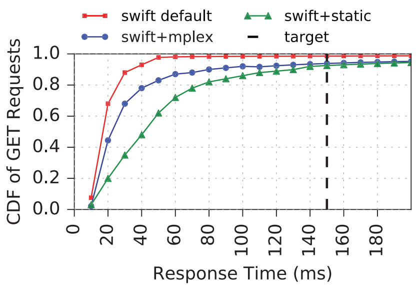
The opportunistic job scheduling of MPLEX clearly sets it apart from a static multiplexing scheme that offloads Hadoop jobs with a fixed 2:1 ratio between the compute, and storage clusters, irrespective of changing cluster load, and performance behavior. This is further supported by Figure 9, which compares the cumulative distribution function (CDF) of queue waiting times for the jobs submitted to the Swift cluster, in case of MPLEX, and the static multiplexing scheme. The worst case queue waiting time observed in case of the static multiplexing scheme is 100 seconds, which is roughly twice the worst case waiting time observed in case of MPLEX.

D. Improving Hadoop Job Performance

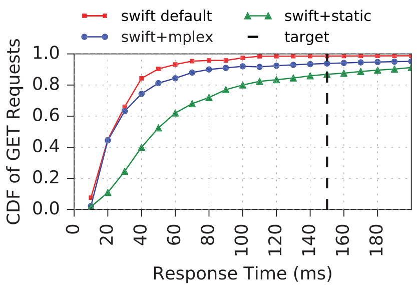
Next, we evaluate the impact of MPLEX on Hadoop job performance, and compare its performance with that of no-multiplexing scheme, and a static multiplexing scheme that offloads Hadoop jobs with a fixed 2:1 ratio between the compute, and storage clusters. Figure 10 shows that MPLEX improves the overall total job throughput significantly on various traffic load profile. For performance comparison, total Hadoop job throughput is normalized by the throughput that



(a)

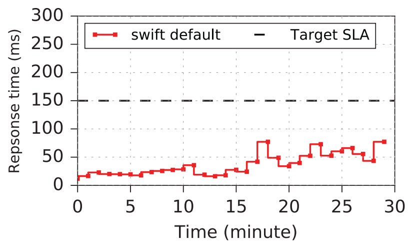


(b)

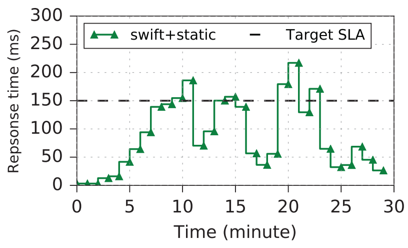


(c)

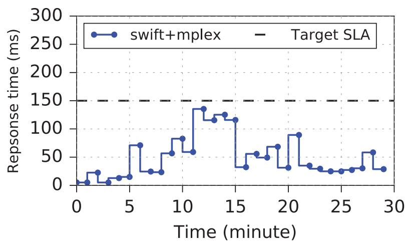
Figure 12: Swift GET response time in (a)Low (b)Medium and (c)High traffic scenario



(a)



(b)



(c)

Figure 13: SLA violation over time in (a)Swift-default (b)Swift+static multiplexing and (c)Swift+MPLEX

is observed with no-multiplexing scenario. Our results show that in case of the r/w balanced profile, MPLEX improves the job throughput by 1.7 times compared to 1.3 times by the static scheduler in low load scenario. We see significant improvement particularly in low load scenario, a moderate amount of improvement in medium load scenario, and a small improvement in high load scenario. That is due to the fact that fewer number of jobs can be placed in the Swift cluster, when the storage cluster is facing high load, as shown in Figure 8(c).

Figure 11(a),(b) and (c) show that MPLEX improves the worst case job execution time by almost twice compared to the static multiplexer in all three cases of different Swift request profile. The r/w balanced Swift workload profile exhibits the best Hadoop job performance while read-heavy profile has the worst. The superior performance of MPLEX is due to the fact that unlike the static multiplexing scheme, it avoids submitting too many jobs to the Swift cluster. Since Swift cluster's object storage based file system is much slower than the HDFS file system [31], too many jobs can impact per job execution time as well as contributing to long waiting time in the queue to be serviced.

E. Achieving Storage Performance SLA

Next, we evaluate the capability of MPLEX to improve overall Hadoop job throughput, without violating the performance SLA of the cloud storage service. We define the SLA target for Swift GET response time, $t = 150ms$ for 95^{th} percentile of the traffic. As part of the design, MPLEX schedules Hadoop jobs on the Swift cluster only if the probability of SLA violation is low.

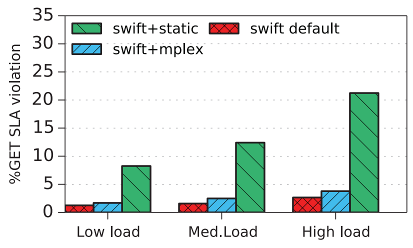
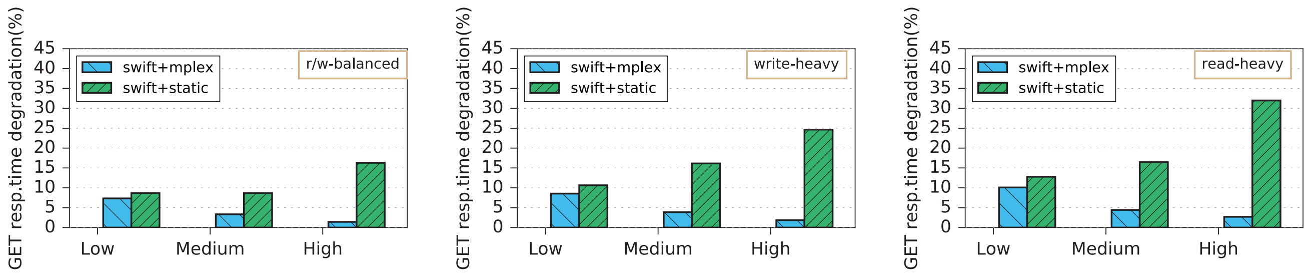


Figure 14: Percentage of SLA violation

Figure 12 compares the cumulative distribution function (CDF) of Swift request response time achieved with Swift-default (no multiplexing), static multiplexing, and MPLEX for low, medium and high traffic load. In low traffic situation, shown in Figure 12(a), MPLEX achieves Swift request response times that are very close to that of Swift-default case. The amount of performance degradation is between $1 \sim 2\%$. For medium traffic load, shown in Figure 12(b), the degradation of response time is around 5%. However, MPLEX still maintains the SLA target. Even in high traffic scenario, shown in Figure 12(c), MPLEX almost closely resembles Swift-default response time. This is due to the fact that



(a) (b) (c)

Figure 15: Average GET response time degradation(%) (a) r/w balanced traffic (b) write heavy traffic and (c) read-heavy traffic

MPLEX's SLA-aware job scheduling algorithm slows down its Hadoop job placement rate on the Swift cluster. In contrast, the static multiplexing scheme continues placing many Hadoop jobs inside the Swift cluster even in high load scenario. This results in around 15% degradation with a large amount of SLA violation.

Figure 13 shows the SLA violation over time for Swift-default, static multiplexing, and MPLEX respectively. Figure 14 summarizes the percentage of request that violates the SLA target for the three different load scenarios. In all three cases, the amount of violation is minimal for both Swift-default and MPLEX whereas static scheduling shows as much as 21% of violation. Furthermore, Figure 15 shows the average response time degradation with respect to no-multiplexing case for (a) r/w balanced traffic, (b) write-heavy traffic and read-heavy traffic. We observe that while MPLEX achieves consistently low response time degradation for all traffic loads, the static multiplexing scheme exhibits increasing worse behavior with increasing traffic load.

V. Related Work

Big-data analytics on the cloud is an active area of research. There is a large body of work [19], [24], [25], [30] devoted to Big-data analysis on the cloud and optimizing Hadoop performance. There are prior works that tried to explore running Hadoop directly inside the storage cloud [31]. However, to the best of our knowledge, none of the existing work provide a compute-storage multiplexing mechanism to optimize Hadoop job throughput, while maintaining the performance SLA of cloud storage service.

Cloud storage performance, and various optimizations have been studied in previous works [16], [18], [21], [27], [32], [28], [35]. Guoxin *at el.* [27] proposed a data reallocation algorithm to balance the load on cloud storage servers with the aim to achieve storage performance SLA. Suresh *at el.* [32] presented an adaptive replica selection algorithm to improve the tail latency in cloud data stores. Similar to the approach used in [32], MPLEX uses a score ranking based policy to schedule jobs between compute, and storage clusters. While their work used ranking for each storage server for adaptive replica selection, MPLEX uses ranking for the whole compute, and storage cluster to efficiently schedule data-analytics jobs between the two clusters. MPLEX also applies a machine learning based prediction model to make job scheduling decisions that avoid the SLA violation of cloud storage service. Furthermore, most existing works involve intrusive changes in the cloud storage implementation. In contrast, our work does not introduce any modification to the core service of cloud storage, or data-analytics framework such as Hadoop. Hence, our approach has a general applicability.

Multiplexing resources between different clusters has been exploited in previous work by [22], [36]. Goiri *at el.* [22] focused on improving energy efficiency by scheduling hadoop jobs in a way that maximizes the usage of green energy. They multiplex Hadoop job placement between multiple data centers, based on the prediction of green energy supply and demand. While their work aimed at minimizing brown energy consumption, our work differs in the approach that we aimed in maximizing throughput of Hadoop job processing by leveraging idle resources from the storage cloud.

VI. Conclusion

We present MPLEX, a system that augments data analytics-as-a-service through an SLA-aware opportunistic job scheduling technique that efficiently multiplexes the compute and storage cluster with the aim to improve job throughput, without violating the performance SLA of cloud storage service. MPLEX uses real-time performance and load information as a feedback mechanism, and a machine learning based prediction model to schedule jobs between compute and storage clusters. It is applicable to any data analytics-as-a-service platform irrespective of the underlying data processing, and cloud storage system. Extensive evaluations on NSFCloud's Chameleon testbed demonstrate that MPLEX improves the Hadoop job throughput by up to 1.7X, while maintaining the 95^{th} percentile response time target of the object storage workload.

Acknowledgment

Results presented in this paper were obtained using the Chameleon test-bed supported by the National Science Foundation.

References

[1] *Amazon EMR*, 2017. https://aws.amazon.com/documentation/elastic-mapreduce/.

[2] *Hadoop on Google Compute Engine.*, 2017. https://cloud.google.com/solutions/hadoop.

[3] *Hadoop OpenStack Support*, 2017. https://hadoop.apache.org/docs/stable2/hadoop-openstack/index.html.

[4] *Microsoft Azure HDInsight.*, 2017. http://azure.microsoft.com/en-us/services/hdinsight.

[5] *Object Storage Monitoring*, 2017. http://docs.openstack.org/admin-guide/objectstorage-monitoring.html.

[6] *OpenStack Keystone*, 2017. http://docs.openstack.org/developer/keystone/.

[7] *Openstack Kilo*, 2017. https://www.openstack.org/software/kilo/.

[8] *Openstack Swift*, 2017. http://docs.openstack.org/developer/swift.

[9] *PUMA: Purdue MapReduce Benchmark Suite*, 2017. http://web.ics.purdue.edu/~fahmad/benchmarks.htm.

[10] *Randomized Lasso*, 2017. http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RandomizedLasso.html.

[11] *Resource Manager REST API*, 2017. http://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/ResourceManagerRest.html.

[12] *S3 Benchmark*, 2017. http://twopieceset.blogspot.com/2009/06/s3-performance-benchmarks.html.

[13] *StatsD aggregator*, 2017. https://github.com/etsy/statsd.

[14] *wikimedia page counts*, 2017. https://dumps.wikimedia.org/other/pagecounts-raw/2016/2016-05/.

[15] A. Anwar, Y. Cheng, A. Gupta, and A. R. Butt. Taming the cloud object storage with mos. In *Proceedings of the 10th Parallel Data Storage Workshop*, pages 7–12. ACM, 2015.

[16] A. Anwar, Y. Cheng, A. Gupta, and A. R. Butt. Mos: Workload-aware elasticity for cloud object stores. In *Proc. of ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2016.

[17] D. Borthakur. Hdfs architecture guide. *HADOOP APACHE PROJECT http://hadoop. apache. org/common/docs/current/hdfs design. pdf*, page 39, 2008.

[18] I. Cano, S. Aiyarn, V. Aroran, M. Bhattacharyyan, A. Chagantin, C. Cheahn, B. Chunn, K. Guptan, V. Khotn, and A. Krishnamurthyw. Curator: Self-managing storage for enterprise clusters. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017.

[19] K. Chen, J. Powers, S. Guo, and F. Tian. Cresp: Towards optimal resource provisioning for mapreduce computing in public clouds. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 25(6):1403–1412, 2014.

[20] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. The case for evaluating mapreduce performance using workload suites. In *Proc. of IEEE Annual international symposium on modelling, analysis, and simulation of computer and telecommunication systems (MASCOTS)*, 2011.

[21] Y. Cheng, M. S. Iqbal, A. Gupta, and A. R. Butt. Cast: Tiering storage for data analytics in the cloud. In *Proc. of the 24th International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2015.

[22] Í. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. Greenhadoop: leveraging green energy in data-processing frameworks. In *Proc. of ACM european conference on Computer Systems (EuroSys)*, pages 57–70, 2012.

[23] J. Hamilton. Cooperative expendable micro-slice servers (cems): low cost, low power servers for internet-scale services. In *Proc. of Conference on Innovative Data Systems Research (CIDR)*, 2009.

[24] M. Ishii, J. Han, and H. Makino. Design and performance evaluation for hadoop clusters on virtualized environment. In *The International Conference on Information Networking 2013 (ICOIN)*, pages 244–249. IEEE, 2013.

[25] C. Ji, Y. Li, W. Qiu, U. Awada, and K. Li. Big data processing in cloud computing environments. In *2012 12th International Symposium on Pervasive Systems, Algorithms and Networks*, pages 17–23. IEEE, 2012.

[26] J. Leverich and C. Kozyrakis. Reconciling high server utilization and sub-millisecond quality-of-service. In *Proc. of the Ninth European Conference on Computer Systems (EuroSys)*, 2014.

[27] G. Liu and H. Shen. Deadline guaranteed service for multi-tenant cloud storage. In *Proc. of IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 1–10, 2015.

[28] K. Oh, A. Chandra, and J. Weissman. Wiera: Towards flexible multi-tiered geo-distributed cloud storage instances. In *Proc. of ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2016.

[29] F. Pace, M. Milanesio, D. Venzano, D. Carra, and P. Michiardi. Experimental performance evaluation of cloud-based analytics-as-a-service. *arXiv preprint arXiv:1602.07919*, 2016.

[30] B. T. Rao and L. Reddy. Survey on improved scheduling in hadoop mapreduce in cloud environments. *arXiv preprint arXiv:1207.0780*, 2012.

[31] L. Rupprecht, R. Zhang, and D. Hildebrand. Big data analytics on object stores: A performance study. In *Proc. of International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2014.

[32] L. Suresh, M. Canini, S. Schmid, and A. Feldmann. C3: Cutting tail latency in cloud data stores via adaptive replica selection. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015.

[33] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proc. of ACM Annual Symposium on Cloud Computing (SoCC)*, 2013.

[34] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proc. of the 7th USENIX symposium on Operating systems design and implementation (OSDI)*, 2006.

[35] Z. Wu, C. Yu, and H. V. Madhyastha. Costlo: Cost-effective redundancy for lower latency variance on cloud storage services. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 543–557, 2015.

[36] Y. Zhang, Y. Wang, and X. Wang. Greenware: Greening cloud-scale data centers to maximize the use of renewable energy. In *Proc. of ACM/IFIP/USENIX International Conference on Middleware*, 2011.

[37] L. Zhao, S. Sakr, A. Liu, and A. Bouguettaya. *Cloud Data Management*. Springer, 2014.

[38] Q. Zheng, H. Chen, Y. Wang, J. Zhang, and J. Duan. Cosbench: cloud object storage benchmark. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, pages 199–210. ACM, 2013.