Accelerating Big Data Analytics Using Scale-up/out Heterogeneous Clusters

Zhuozhao Li
Department of Computer Science
University of Chicago
Email: zhuozhao@uchicago.edu

Haiying Shen
Department of Computer Science
University of Virginia
Email: hs6ms@virginia.edu

Lee Ward
Sandia National Laboratories
Email: lee@sandia.gov

***Abstract*— Production data analytic workloads typically consist of a majority of jobs with small input data sizes and a small number of jobs with large input data sizes. Recent works advocate scale-up/scale-out heterogeneous clusters (in short Hybrid clusters) to handle these heterogeneous workloads, since scale-up machines (i.e., adding more resources to a single machine) can process small jobs faster than simply scaling out the cluster with cheap machines. However, there are several challenges for job placement and data placement to implement such a Hybrid cluster. In this paper, we propose a job placement strategy and a data placement strategy to solve the challenges. The job placement strategy places a job to either scale-up or scale-out machines based on the job's characteristics, and migrates jobs from scale-up machines to under-utilized scale-out machines to achieve load balance. The data placement strategy allocates data replicas in the two types of machines accordingly to increase the data locality in Hybrid cluster. We implemented a Hybrid cluster on Apache YARN, and evaluated its performance using a Facebook production workload. With our proposed strategies, a Hybrid cluster can reduce the makespan of the workload up to 37% and the median job completion time up to 60%, compared to traditional scale-out clusters with state-of-the-art schedulers.**

I. Introduction

Many big data analytic clusters like MapReduce [1] process thousands of jobs and a large amount of data every day [2]. Conventionally, these clusters consist of many homogeneous scale-out machines. Recent studies [3–5] advocate to explore hybrid scale-up and scale-out heterogeneous clusters (in short Hybrid clusters) to handle these workloads, since previous studies [6, 7] show that a large number of jobs (e.g., more than 80%) in these workloads only process small data size and have diverse job characteristics (e.g., shuffle data size). Here, scale-up is vertical scaling, which means adding more resources to the nodes of a system, typically the processors and RAM, and scale-out is horizontal scaling, which refers to adding more nodes with few processors and RAM to a system. Appuswamy *et al.* [4] evaluated the jobs with different characteristics on scale-up and scale-out machines, and found that scale-up is significantly better in some cases, than scale-out. Hence, we are motivated to design Hybrid cluster to handle the diverse workloads for high performance.

There have been studies [3, 8–10] focusing on improving the performance in heterogeneous clusters. However, since these proposals do not consider the diverse job characteristics and do not take advantage of this feature, they are not suitable for Hybrid cluster, which is designed to process such workloads. For example, since we *intentionally* use scale-up machines to deal with job diversity, we expect the jobs that favor scale-up to run on scale-up machines, which is not considered in previous work [3, 8–10]. Li *et al.* [5, 11] identified job and data placement challenges to design Hybrid cluster and configured it with a remote file system to solve the challenges. However, the proposed solution with remote file system causes a large amount of remote data transfer.

In this paper, we focus on designing Hybrid cluster with distributed file system (e.g., HDFS), where some scale-out machines are replaced by scale-up machines that have the same cost as the scale-out machines. The goal of Hybrid cluster is to improve the performance of big data analytics with the same monetary cost, namely a more *cost-effective* cluster. First, we identify the key challenges in designing Hybrid clusters to improve the performance of big data analytic clusters. There are two main challenges – job placement challenges (J.1, J.2, and J.3) and data placement challenge (D.1).

- **J.1** The jobs benefit differently from scale-up and scale-out machines. Thus, we need to adaptively place the jobs to scale-up or scale-out machines based on their job characteristics to achieve the most benefits for the jobs.
- **J.2** The job placement should consider the load balancing. After we schedule the jobs to scale-up or scale-out machines based on their job characteristics, load imbalance may occur on different types of machines. For example, suppose a large amount of small jobs are submitted to Hybrid cluster simultaneously, while there are not many large jobs. If we still run the jobs on different machines based on their job characteristics, it leads to overload on the scale-up machines, while under-utilizing on the scale-out machines.
- **J.3** The different computing speed results in significant imbalance progress of tasks within a job, that is, fast machines complete the tasks faster and need to wait for the slow machines to complete the tasks of the same job. This leads to a non-negligible delay and significantly degrades the performance of the job [8, 10].
- **D.1** Since we adaptively place a job to scale-up or scale-out machines based on its job characteristics, in order to maintain data locality [12], we need to accordingly place the data of every job to the machines that the job is supposed to run on.

Second, we propose job placement and data placement strategies to handle the challenges.

Job placement strategy. In order to achieve the best performance on Hybrid cluster, we use a Support Vector Machine (SVM) model to classify the jobs into two groups, scale-up jobs (i.e., small jobs) and scale-out jobs (i.e., large jobs), based on the job characteristics. We use the term *scale-up jobs* (*scale-out jobs*) to refer to the jobs that are better to run on scale-up machines (scale-out machines).

Further, to balance the loads between scale-up and scale-out machines, we propose a job stealing strategy, which adaptively steals scale-up jobs to run on scale-out machines when the scale-out machines are under-utilized.

Data placement strategy. To solve the data placement challenge, we leverage the replicas in the clusters and place the data replicas to scale-up or scale-out machines, considering both job characteristics and job stealing.

Finally, we implement a Hybrid cluster with the above two strategies, and evaluate its performance through real cluster run and large-scale trace-driven simulation. Using the workload derived from Facebook [6], we show that with our proposed strategies, the Hybrid cluster can reduce the makespan of the workload up to 37% and the median job completion time up to 60%, compared to traditional scale-out clusters with state-of-the-art schedulers.

The rest of the paper is organized as follows. In Section II, we present the background and motivations. We describe the main design of Hybrid cluster with corresponding job placement and data placement strategies in Section III and present our experiment evaluation in Section IV. Section VI concludes this paper with remarks on our future work.

II. Background

We use MapReduce as the example framework in this paper. However, the ideas of exploiting Hybrid cluster and differentiating small and large jobs to accelerate big data analytics can be applied to other frameworks, such as Spark [13].

A. Hadoop MapReduce

A MapReduce job consists of map and reduce stages, which contain multiple map and reduce tasks respectively. Each map task processes one input data block and generates intermediate data (called shuffle data). Each reduce task has two steps – shuffle and reduce. In the shuffle, all shuffle data with the same key is transferred to the same reduce task.

Hadoop [14], a popular implementation of MapReduce, has three main components - Resource Manager (RM), Application Master (AM) and Node Manager (NM). The scheduler (default is Fair [15] or Capacity [16]) is an essential component of RM, which determines how much and where to allocate the resources to each application. Each application (i.e., job) has an associated AM, which is responsible for requesting resources from RM. RM responds to a resource request by granting a resource container managed by a specific NM.

Hadoop uses Hadoop Distributed File System (HDFS) as its primary distributed data storage system. Data is broken down into smaller blocks and stored in HDFS. To ensure fault tolerance, HDFS uses replication strategy. By default, the number of replicas is three for each block in HDFS.

B. Opportunities, Objectives and Benefits

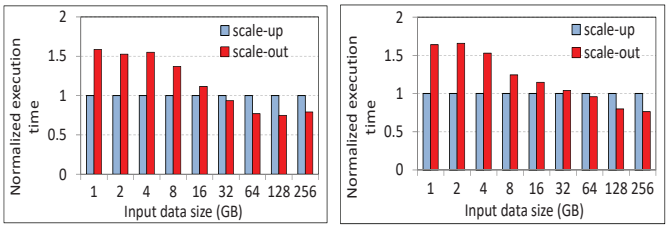
The large memory of scale-up machines provides benefits for the jobs with large shuffle data size [4]. First, we can set a higher heap size with the large memory. The heap is used to buffer the in-memory data (e.g., shuffle data). Data is spilled to the storage when buffers are full, which leads to overhead. A higher heap size can reduce the times to spill. Second, the excess memory can be used as RAMdisk to store shuffle data to accelerate its read/write. Hence, the scale-up machines can provide more benefits to the jobs with large shuffle data size, as the shuffle is efficient in scale-up.

Previous studies show that scale-up machines can process small jobs faster than scale-out machines [4, 5]. A big data analytic cluster traditionally consists of many cheap scale-out machines. Scale-up machines differ from scale-out machines in that scale-up machines have more powerful CPU and more RAM in one machine. Therefore, scale-up machines may process the small jobs faster but large jobs slower [4, 5] due to the fact that large jobs are generally data-intensive and can be processed faster with higher parallelism on scale-out machines. In this paper, *we use scale-up (scale-out) job and small (large) job interchangeably.*

We have conducted a measurement study to compare the performance between scale-up machine and scale-out machines. The scale-up machine is equipped with 24 cores E5-2680V3 CPU, 128GB RAM size, while each scale-out machine is equipped with 8 cores AMD Opteron 2356 CPU, 16GB RAM size. After some market investigations [17, 18], we find that the price of each selected scale-up machine is similar to 5 selected scale-out machines. We deployed several configuration optimizations on the scale-up machine as in [4, 5]. For example, we used half of the RAM (i.e., 64 GB) as RAMdisk to store the shuffle data, and also changed the heap size from default 200MB to 2.5GB.

We ran TeraSort and WordCount [14] with different input data sizes on 1 scale-up machine and 5 scale-out machines, respectively. Figure 1 shows the execution time of TeraSort and WordCount (normalized by the results of scale-up machine) versus different input data sizes. When the input data size is smaller than certain threshold (called *crosspoint threshold*), the scale-up machine outperforms the scale-out machines by up to 60%; otherwise, the scale-out machines outperform the scale-up machine by up to 35%. Similar conclusions are also observed in [4]. This is because (1) when the input data size is small, scale-up machine benefits the job with more powerful CPU and RAMdisk; (2) as the input data size of the job increases, the total number of CPU cores and memory on scale-up machine limit the performance of the job, while scale-out machines can benefit the job with more CPU cores and higher aggregate memory bandwidth. Also, we observe that since the job characteristics of TeraSort and WordCount are different, they have different benefits from scale-up machines. Thus, the crosspoint thresholds for the two jobs are different (32GB for TeraSort and 64GB for WordCount).

The job characteristics can be predicted with small error



(a) TeraSort (b) WordCount

Fig. 1: Measurement results of TeraSort and WordCount.

[2, 19, 20]. Previous studies [2, 19, 20] show that a large number of jobs in the clusters are recurring and their job characteristics (e.g., shuffle data size) can be predicted with a small error (e.g., 6.5% [2]).

Objective. Due to the diversity of current big data workloads, our objective in this paper is to accelerate big data analytics by designing Hybrid cluster at the same cost as traditional scale-out cluster. We replace some cheap scale-out machines with scale-up machines that have the same monetary cost. **The benefits of Hybrid cluster can be summarized as follows.**

- For the small jobs, they are executed on the scale-up machines, which process the small jobs faster. Therefore, the performance of the small jobs is improved because they benefit from the scale-up machines on Hybrid cluster.
- For the large jobs, although Hybrid cluster does not have any direct improvement on them, they can also run faster. As the small jobs are executed on scale-up machines, the large jobs can run on scale-out machines with much less resource contention (e.g., CPU and network) from the large number of small jobs, and hence, the large jobs can also finish earlier.

III. Design of Hybrid Cluster

In this section, we present the design of *Hybrid* cluster, including the architecture of the cluster, job placement strategy and data placement strategy for this cluster.

A. Hybrid Cluster Architecture

The traditional Hadoop cluster generally consists of all scale-out machines organized into multiple racks [2], as shown in Figure 2(a). In Hybrid cluster, we replace some scale-out machines with scale-up machines that have the same cost as the scale-out machines. There are two questions in the design of Hybrid cluster architecture.

Where to place the scale-up machines in Hybrid? We propose the architecture of Hybrid cluster as shown in Figure 2(b) – placing the scale-up machines and the scale-out machines on separate racks so that no scale-up and scale-out machines are on the same rack, due to the reasons below.

- We can reduce cross-rack network traffic by using this architecture. Jalaparti *et al.* [2] demonstrated that most small jobs in current production workloads can be placed in only one rack without sacrificing the parallelism and compromising the performance. Using this property, we can place the input data of the small jobs in a single rack. As a result, the map tasks of a job can run on this rack for map input data locality and the map output data is also generated in the rack. Subsequently, we can schedule the reduce tasks of the job on the same rack, so that the shuffle data transfers of this job are all within one rack, which reduces the cross-rack network resource. In our designed Hybrid architecture, scale-up machines are in one rack, so that the small jobs placed on scale-up machines can run within one rack, resulting in less cross-rack network traffic from small jobs. Furthermore, this architecture can benefit the large jobs that cannot be run in a single rack, as there is less contention of cross-rack network resource from the small jobs.
- This architecture plays an important role in solving the data placement challenges. For more details, we refer to the data placement strategy in Section III-D.
- This architecture is easy to implement on Apache YARN [14]. We will explain this in details in Section IV-A.

How many scale-out machines should be replaced by scale-up machines? Empirical studies in [6, 12, 20] show that the relative proportion of small and large jobs in a cluster remains *stable* over time. For instance, the scale-out jobs are considerably fewer in typical workloads, but dominate the cluster resource usage (e.g., 80% to 99%) [6, 12, 20]. Thus, the cluster operators only need to replace a few scale-out machines (e.g., 1% to 20%) with scale-up machines to accelerate the small jobs.

B. Differentiating Small and Large Jobs

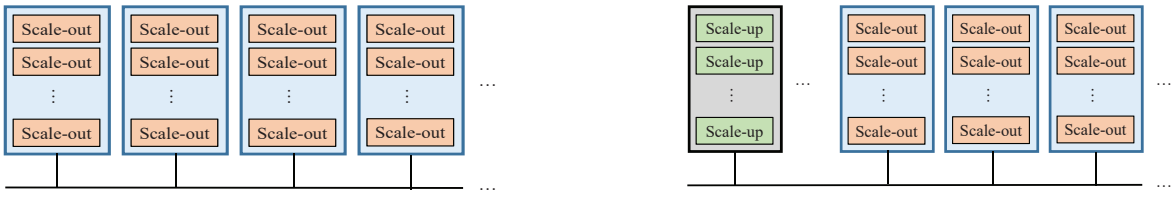
In this section, we introduce how to differentiate small and large jobs for both recurring and non-recurring jobs.

As we mentioned in Section II-B, different jobs may benefit differently from scale-up and scale-out machines. To differentiate the jobs based on their job characteristics, the natural thinking is to use machine learning technique, which takes the job characteristics as inputs and predicts each job's type. One question is what job characteristics we should use as inputs. We decide to use *the number of map/reduce tasks and shuffle data size* as the inputs, due to the reasons below: (1) As shown in Figure 1, the input data size of a job is one characteristic that affects the performance of the job on different machines. Since input data size of a job is linear to the number of map tasks of the job, we use the number of map tasks; (2) As mentioned in Section II-B, since scale-up machines can benefit the jobs with large shuffle data size because of the large memory size on scale-up machines, shuffle data size is a non-negligible characteristic; and (3) The works [4, 5] show that the number of reduce tasks of a job is a factor that affects the performance of the job on different machines.

For the machine learning, we use the Support Vector Machine (SVM) [21]. SVM is a classifier that maps the feature data (i.e., job characteristics) as points in high-dimensional space, so that the different categories are clearly separated by a gap. More formally, SVM constructs a *hyperplane* to separate the data into two categories, so that the distance from the *hyperplane* to the nearest data point on each side is maximized.

We use SVM classifiers because of the reasons below.

- The job characteristics (the number of map/reduce tasks, shuffle data size) are all continuous features.
- SVM is widely used for binary classification, which matches our case that divides jobs into two types.



(a) Traditional scale-out cluster. (b) Hybrid scale-up/out cluster.

Fig. 2: Traditional scale-out cluster versus hybrid scale-up/out heterogeneous cluster.

• The points mapped by the characteristics of jobs may not be linearly separable in space. SVM provides kernel function to create a nonlinear classifier.

• SVM constructs a clear hyperplane to separate the jobs, so that we can calculate the distance between a job to the hyperplane. This property is useful for the job stealing strategy in Section III-C.

While we can consider more job characteristics as inputs and use other machine learning models to classify the jobs, our results in Section IV show that using these three factors (the number of map/reduce tasks, shuffle data size) can already determine the types of the jobs with 96.2% accuracy.

For each Hybrid cluster, it requires us to train a corresponding SVM classifier using sufficient training data first. Then, how we classify recurring and non-recurring jobs is summarized as follows.

Recurring jobs When a recurring job is submitted to the cluster, we can predict its job type using the trained SVM classifier based on the job characteristics (i.e., the number of map/reduce tasks and shuffle data size).

Non-recurring jobs As to the non-recurring job, *only* two job characteristics (the number of map/reduce tasks) are known a priori. So, the question is what shuffle data size we should use for non-recurring job.

When determining the type of a non-recurring job, we argue that the occurrence of classifying a large job as a small job would be much worse than the occurrence of opposite. This is because classifying a large job as a small job allows the large jobs to run on the scale-up machines, which causes two issues. First, in Hybrid cluster, we leverage the scale-up machines to accelerate the small jobs, but not large jobs. The performance of large jobs may be greatly degraded, compared with the performance on scale-out machines (see Figure 1). Second, the large jobs consume a large amount of resources [20] and run for a long time. If we place the large jobs on scale-up machines when the scale-up machines are under-utilized, the large jobs may occupy all the scale-up machines for a long time. In this case, the small jobs have to wait for the large job and hence the performance of the small jobs are severely degraded. Consider an extreme case that multiple large jobs are submitted to the cluster while there are temporally no small jobs. If we run the large jobs on the scale-up machines, the whole cluster ends up running the large jobs. When some small jobs are submitted to the cluster afterwards, they have to wait for the large jobs for a long time.

On the contrary, classifying a small job as a large job allows the small jobs to run on the scale-out machines. In this case, the only impact is that this small job cannot leverage the scale-up machines to accelerate its execution, which causes minimal impact and is much less severe than the case above.

Hence, when determining the type of a non-recurring job, we aim to avoid the occurrence of classifying a large job as a small job. To achieve this, we treat the shuffle data size of the non-recurring job as 0. This is because scale-up machines can benefit the job with large shuffle data size as mentioned in Section II-B. By treating the shuffle data size of a job as 0, if the job is predicted as "small", its actual job type will definitely be "small", since with the job's actual shuffle data size, the job will benefit more on scale-up machines.

Training To train the SVM classifier, we replayed a 25428-job Facebook workload trace [6] on a real cluster using the tools provided in [6]. We first ran the 25428 jobs one by one in the workload on scale-up machines and scale-out machines, respectively, and then parsed the logs. Based on execution time of each job on different machines, we labeled the jobs and used the labeled dataset as the training dataset for SVM classifier. We applied the radial basis function (RBF) kernel [21] to train the SVM classifier. In order to avoid over-fitting and get the best parameters for SVM with RBF kernel, we used the most common method – cross validation [22].

Summary We leverage a SVM classifier to differentiate the small and large jobs based on the job characteristics. The input characteristics for a non-recurring job are the number of map/reduce tasks and a constant shuffle data size (i.e., zero), while the input characteristics for a recurring job are the number of map/reduce tasks and its shuffle data size.

C. Job Placement Strategy

In this section, we present the job placement strategy to address the job placement challenges in Section I.

Placing the jobs accordingly to scale-up or scale-out job queue (for J.1 challenge). First, when the jobs are submitted to the cluster, the job placement strategy divides the jobs into scale-up job queue and scale-out job queue, using the machine learning technique introduced in Section III-B. The scale-up job queue of jobs are scheduled on scale-up machines, while the scale-out job queue of jobs are scheduled on scale-out machines. Next, Hybrid further sorts the jobs in each queue based on the pre-defined cluster scheduler, such as Fairness [15] and Capacity [16] and put each queue into a queue. When new jobs are submitted, we repeat the previous steps to put the new jobs into corresponding queues.

Job stealing to balance the job loads for scale-up and scale-out machines (for J.2 and J.3 challenges). In order to solve

the J.2 challenge, we propose a job stealing policy that actively selects jobs from the scale-up job queue and moves them to the scale-out job queue for load balancing.

The job stealing steals the entire jobs instead of individual tasks because of challenge J.3. If the job stealing steals tasks between scale-up machines and scale-out machines, the tasks of the same job may be run on both scale-up and scale-out machines. As aforementioned, running the tasks of a job on different kinds of machines may lead to extremely poor performance for the job [3, 8–10]. Therefore, we utilize a job-level stealing to handle challenge J.3. *Notice that the job-level stealing policy does not incur any overhead since the stolen jobs are still in the queue and not started yet, and no data movement is needed using the data placement strategy in Section III-D.*

During job stealing, we propose to *restrict* the large jobs to run only on the scale-out machines, but not scale-up machines, due to the reasons mentioned in Section III-B that large jobs suffer poor performance on scale-up machines and may occupy the resources of scale-up machines for a long time, which also degrades the performance of small jobs. Besides, in a production workload, most of the jobs are small and the average arrival time between two small jobs is short [4, 6, 20]. In this case, as the small jobs are submitted very frequent, the scale-up machines are expected to be under-utilized for only a short time and will soon become fully-utilized again.

Hence, large jobs are *restricted* to run on scale-out machines, while small jobs are *not restricted* to run on scale-up machines and are allowed to run on both machines. The details of job stealing can be described in details as follows:

(i) If a scale-up machine requests for a task but there are not any scale-up jobs, RM delays to schedule any jobs to the scale-up machines until the next scale-up job is submitted.

(ii) If a scale-out machine requests for a task but there are not any scale-out jobs awaiting to schedule, RM actively "steals" a job from the scale-up job queue and moves it to the scale-out job queue. Once a job is stolen, all the tasks of this job are restricted to run on scale-out machines.

Another important issue is which job to steal. In order not to degrade the performance of the stolen job, it is better to find a scale-up job that is as close to the *hyperplane* as possible, which means that the stolen job is the most similar to scale-out jobs in the queue of scale-up jobs and hence the stealing generates minimal impact to the stolen job. Specifically, RM uses the *decision_function* in Python Scikit Learn library [23] to compute the distance between a job (represented as a sample point) and the hyperplane. Then, the job with the smallest distance is selected, and it becomes a scale-out job and is restricted to run all its tasks on scale-out machines.

D. Data Placement Strategy

In this section, we introduce the data placement strategy accompanied with the Hybrid cluster to address data placement challenges in Section I. By default, there are three replicas for each data block in the cluster [14]. HDFS's replication placement policy is to put one replica in one node in one rack, another replica in a node in a different (remote) rack, and the third replica in a different node in the same remote rack. In other words, the three replicas are placed in two racks; one replica in a rack and two replicas in another rack. In this paper, we assume that the default replication factor (i.e., 3) is used. To solve the challenges of data placement, our data placement strategy takes advantage of the replication placement strategy.

Placing data on both scale-up and scale-out machines (for D.1 challenge). To solve D.1 challenge, we can place the *first and second replicas* of scale-up/scale-out jobs on scale-up/scale-out machines, accordingly. However, since jobs may be stolen between scale-up queue and scale-out queue by the job stealing policy, the data needs to be placed in the corresponding machines in order to maintain data locality.

Thus, for a scale-up job, we place its third replica on scale-out machines, so that even when the job is stolen to the scale-out queue, RM can still locate its data on scale-out machines, which improves the performance. As scale-out jobs *never* run on scale-up machines, we place the third replica of scale-out jobs on the scale-out machines on the racks that are different from the locations of the first and second replicas.

Notice that with the help of Hybrid architecture in Section III-A, the proposed data placement strategy also satisfies the default replication placement rule in HDFS that places the replicas for each data block in two different racks.

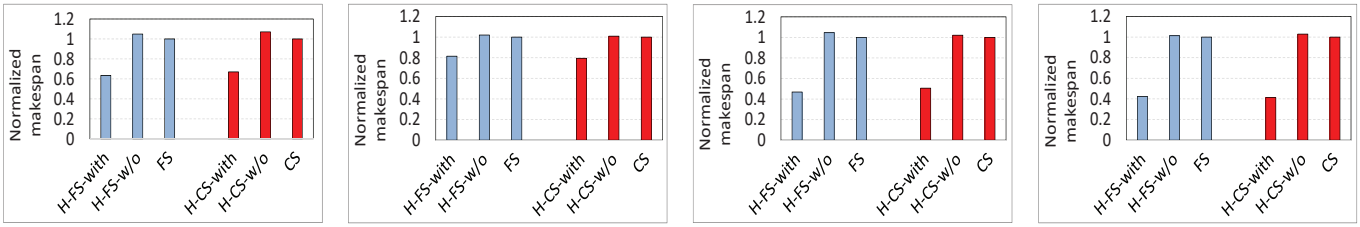
IV. Performance Evaluation

We evaluate how the Hybrid cluster performs by real cluster run and large-scale trace-driven simulation. We consider two job submission scenarios, batch and online scenarios [2]. The batch scenario means that the jobs are submitted to the cluster at the same time. The performance metric in this scenario is the makespan (i.e., the time to complete all the jobs in the batch). The online scenario means that each job is submitted to the cluster at a specific job arrival time. The performance metric in this scenario is the average job completion time (i.e., the end time of a job minus the job arrival time).

A. Experiment Setup and Workload

Real cluster run We configured a Hybrid cluster, and the configurations of scale-up and scale-out machines are the same as those mentioned in Section II-B. For Hybrid cluster, we used 2 scale-up machines from the same rack, and used 40 scale-out machines from 4 different racks, each of which contains 10 scale-out machines. Hence, Hybrid cluster consists of 4 scale-out racks and 1 scale-up rack. As to the traditional clusters, we used 50 scale-out machines from 5 different racks, each of which contains 10 scale-out machines. Thus, Hybrid cluster has a similar cost as the traditional cluster.

We implemented the job placement strategy on Hadoop 2.7.1. As introduced in Section II, each job in YARN has an Application Master (AM) to request resources from the centralized Resource Manager (RM). Through AMs, different jobs can specify the locations of their resources, such as specific machines or racks. Recall that our job placement strategy



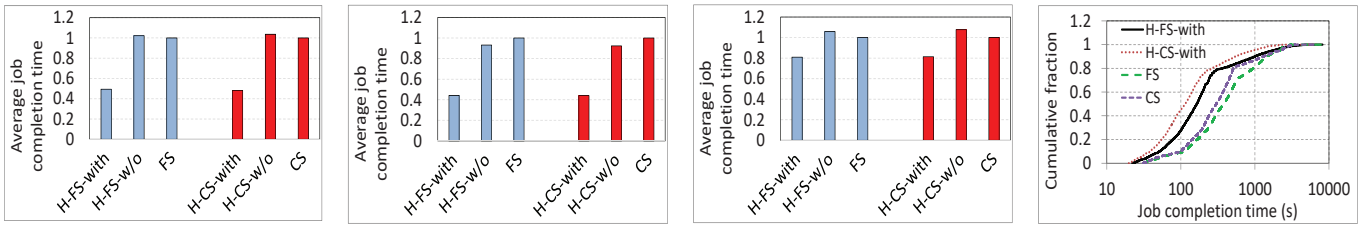
(a) Makespan for all the jobs in the real cluster run.

(b) Makespan for all the jobs in the simulation.

(c) Makespan for scale-up jobs in the real cluster run.

(d) Makespan for scale-up jobs in the simulation.

Fig. 3: Makespan results in the batch scenario.



(a) Average job completion time for entire workload.

(b) Average job completion time for scale-up jobs.

(c) Average job completion time for scale-out jobs.

(d) Cumulative fraction of job completion time.

Fig. 4: Job completion time results in the real cluster run in the online scenario.

determines whether a job should run on a scale-up or scale-out machine. The jobs use the *ResourceRequest* function in AMs to request resources in the scale-up or scale-out machine accordingly by specifying the *resource-name* parameter.

We emulated the data placement strategy on our Hybrid cluster. We placed the data blocks of each job to the racks based on our data placement strategy before the jobs were submitted to the cluster. For the traditional cluster, we use the default data placement strategy.

Large-scale simulation In order to show the performance of Hybrid cluster in a large scale, we built an event-based simulator to simulate the real cluster. In the simulation, the traditional cluster consists of 600 scale-out machines, which is organized to 20 racks with 30 scale-out machines each. The Hybrid cluster consists of 19 racks of scale-out machines and 1 rack of scale-up machines. In each rack of scale-out machine, there are 30 scale-out machines. In the rack of scale-up machine, it contains 6 scale-up machines. In the simulation, each scale-out machine can run 8 tasks simultaneously, while each scale-up machine can run 24 tasks simultaneously.

Workload We used a 24442-job Facebook workload (FB-2010) [6] to evaluate the accuracy of SVM and performance of Hybrid. For the real cluster run, we randomly selected 1000 jobs from the workload. In the online scenario, the jobs arrived uniformly in a range of $[0, 60] minutes$. For the simulation, we ran the whole workload. In the online scenario in simulation, the jobs were submitted to the cluster based on the job arrival time in the trace, which lasts for 24 hours.

Baselines. We compared our Hybrid cluster (Hybrid in short) against the baselines below.

(1) Fair scheduler [15] assigns resources to different jobs in a fair manner, so that each job receives the same resources over time. In order to evaluate the performance of our Hybrid cluster and proposed job placement and data placement strategies, we compare Hybrid with both Fair scheduling and proposed strategies (**H-FS-with**) to Hybrid cluster with Fair Scheduling but without proposed strategies (**H-FS-w/o**), and traditional scale-out cluster with Fair scheduling (**FS**).

(2) Capacity scheduler [16] shares a large cluster among different job queues, and aims to provide resource capacity guarantee for each queue. In this case, we compare Hybrid with Capacity scheduling and proposed strategies (**H-CS-with**) to Hybrid cluster with Capacity Scheduling but without proposed strategies (**H-CS-w/o**), and traditional scale-out cluster with Capacity scheduling (**CS**).

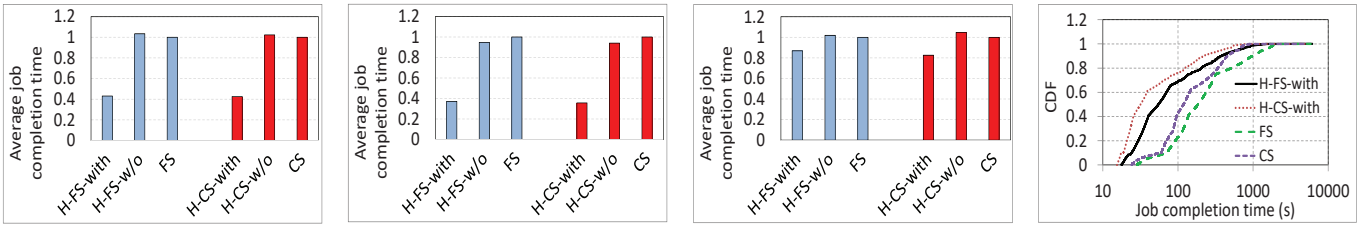
Note that Hybrid cluster is not only compatible with Fair and Capacity, but also other schedulers such as Corral [2], NAS [24], and Co-scheduler [25] by sticking to the proposed job placement and data placement strategies in the paper.

B. Results

Accuracy of SVM We used the trained SVM to predict each job in FB-2010. First, we assume all the jobs in FB-2010 are recurring and have predictable job characteristics. In this case, the SVM determines the types of all the jobs in FB-2010 with 96.2% accuracy. Then, we assume 50% of jobs are recurring – a typical portion of recurring jobs in the cluster [2, 19, 20], while the remaining jobs are non-recurring. In this case, the SVM achieves 86.7% accuracy, with no large jobs being classified as small jobs.

Next, we present the experimental results in the real cluster run and large-scale simulation. In the experiments below, as in [2, 19, 20], 50% of the jobs are recurring and the remaining jobs are non-recurring, unless otherwise specified.

Batch scenario Figure 3(a) shows the makespan of the entire workload in the real cluster run. We see that H-FS-with and H-CS-with achieve 16% and 14% reduction of the makespan over FS and CS, respectively, which demonstrates the effectiveness of Hybrid cluster over traditional cluster. The figure also shows that H-FS-with and H-CS-with significantly outperform H-FS-w/o and H-CS-w/o, respectively. Without our pro-



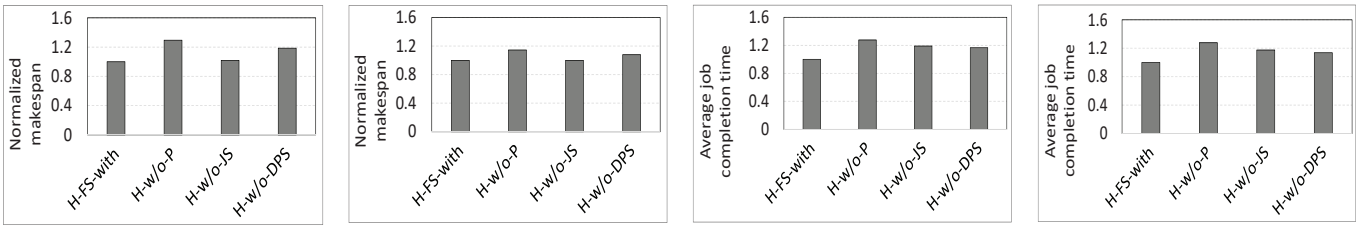
(a) Average job completion time for entire workload.

(b) Average job completion time for scale-up jobs.

(c) Average job completion time for scale-out jobs.

(d) Cumulative fraction of job completion time.

Fig. 5: Job completion time results in the simulation in the online scenario.



(a) Makespan without strategies in the real cluster run.

(b) Makespan without strategies in the simulation.

(c) Average job completion time without strategies in the real cluster run.

(d) Average job completion time without strategies in the simulation.

Fig. 6: Measurement results of each strategy in Hybrid.

posed strategies, the scale-out jobs can be placed on the scale-up machines, which severely degrades the performance of both scale-up and scale-out jobs. Therefore, H-FS-w/o and H-CS-w/o are even worse than FS and CS, respectively. The results demonstrate the effectiveness of our proposed job placement and data placement strategies. Figure 3(b) shows the makespan of the entire workload in the large-scale simulation. We see that Hybrid cluster in a large scale is consistent with the results in the small-scale real cluster run due to the same reasons.

We further measure the makespan reduction of the scale-up jobs in the real cluster run, as shown in Figure 3(c). We see that H-FS-with and H-CS-with achieve a large reduction of makespan on the scale-up jobs – 53% and 49% reduction of the makespan over FS and CS, respectively. Hybrid cluster significantly reduces the makespan of scale-up jobs due to two reasons: (i) the scale-up machines in Hybrid cluster process the scale-up jobs much faster; and (ii) the scale-up jobs has less contention of resources from the scale-out jobs. The figure also shows that H-FS-with and H-CS-with significantly outperform H-FS-w/o and H-CS-w/o, respectively, which indicates the effectiveness of our proposed job placement and data placement strategies. Figure 3(d) shows the makespan of the scale-up jobs in the large-scale simulation, which are consistent with the results in the real-cluster run due to the same reasons.

However, although the makespan of scale-up jobs is reduced significantly, we observe that the makespan of the entire workload is reduce by only 16%. This is because the FB-2010 workload is highly skewed. Most jobs (more than 85%) in the workload have input data sizes less than 100MB, while some jobs in the workload have input data sizes extremely large (more than 5TB). These large jobs are all characterized as scale-out jobs and dominate the makespan of the entire workload. This means that after all the scale-up jobs are completed, there are still many scale-out jobs running in the cluster. Therefore, although Hybrid cluster reduces the makespan of scale-up jobs significantly, it reduces the makespan of this workload by only 16%.

Online scenario In this scenario, the results of H-FS-with and H-CS-with are normalized to FS and CS, respectively. Figure 4(a) shows the average job completion time for the entire workload in the real cluster run. We see that H-FS-with and H-CS-with achieve 51% and 52% reduction of the average job completion time over FS and CS, respectively, which demonstrates the effectiveness of Hybrid cluster. The figures also show that H-FS-with and H-CS-with significantly outperform H-FS-w/o and H-CS-w/o, respectively, which indicates the effectiveness of our proposed job placement and data placement strategies. Figure 5(a) shows the average job completion time for the entire workload in the simulation, which is consistent with the result in the real cluster run.

Figures 4(b) and 4(c) show the average job completion time for scale-up jobs and scale-out jobs in the real cluster run, respectively. Figures 5(b) and 5(c) shows the average job completion time for scale-up jobs and scale-out jobs in the simulation, respectively. Comparing with FS and CS, H-FS-with and H-CS-with reduce the average job completion time of scale-up jobs significantly (more than 55%), while the average job completion time of scale-out jobs is only reduced mildly (around 12%). The figures also show that H-FS-with and H-CS-with significantly outperform H-FS-w/o and H-CS-w/o, respectively, which indicates the effectiveness of our proposed job placement and data placement strategies.

Figure 4(d) shows the cumulative distributed fraction (CDF) of job completion times in the real cluster run. We see that H-FS-with and H-CS-with outperform FS and CS, respectively, with around 60% improvement at the median for the job completion time. Especially, Hybrid has more significant effect on the jobs with job completion time less than 100s. This is because the scale-up jobs run on scale-up machines, which process small jobs much faster. Figure 5(d) shows the CDF

of job completion times in the simulation. It confirms our observations in the real cluster run due to the same reasons.

Summary: In the batch scenario, Hybrid cluster achieves a mild reduction on the makespan of the entire workload, while Hybrid cluster reduces the makespan of the scale-up jobs significantly. In the online scenario, Hybrid cluster reduces the average job completion time of all the jobs significantly.

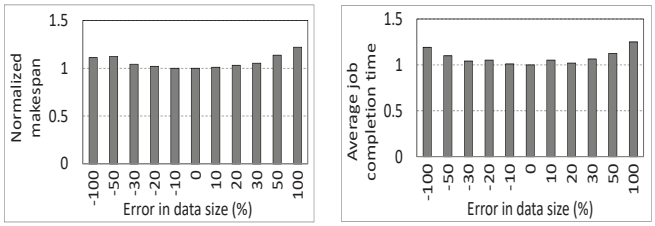
C. Effectiveness of Each Strategy

In this section, we aim to investigate the effectiveness of each strategy in Hybrid. We measure the performance of Hybrid without our job placement strategy that places jobs accordingly to scale-up or scale-out machines (*H-w/o-P*), Hybrid without job stealing policy (*H-w/o-JS*), and Hybrid without our data placement strategy (*H-w/o-DPS*). Due to space limit, we only measure them on Hybrid cluster with Fair scheduling and normalize the results to H-FS-with. Specifically, H-w/o-P uses the Fair scheduling to schedule the jobs, which does not take into account the job characteristics. H-w/o-JS does not adopt the job stealing policy, compared with H-FS-with. H-w/o-DPS does not use our replication placement technique, and only uses the default random replication placement of HDFS.

Batch scenario Figure 6(a) shows the makespan of H-FS-with, H-w/o-P, H-w/o-JS, and H-w/o-DPS in the real cluster run. We see that the makespan of H-w/o-P is increased by 20%, when comparing with H-FS-with. It indicates that the performance of H-w/o-P is even worse than the traditional cluster with Fair scheduling (shown in Figure 3(a)) because of the following reasons. Without our job placement strategy to place the jobs accordingly to scale-up or scale-out machines, (i) the scale-up jobs may be assigned to the scale-out machines and hence they cannot take advantage of the scale-up machines; and (ii) some tasks of scale-out jobs run in the scale-up machines, which is very slow and hence results in poor performance. We also observe from Figure 6(a) that H-w/o-JS actually provides the same performance as H-FS-with in the batch scenario. This is because in the batch scenario, the large jobs are all submitted to the cluster, which makes the scale-out machines fully utilized all the time. Thus, even though H-FS-with is adopted with job stealing policy, the job stealing actually does not have any effect on this FB-2010 workload as the scale-out machines are never under-utilized.

On the other hand, the makespan of H-w/o-DPS is increased by 9%, when comparing with H-FS-with. This is because without our data placement strategy, some tasks may fail to maintain data locality, which degrades the performance. Figure 6(b) shows the makespan breakdown in the large-scale simulation, which is consistent with the results in the real cluster run due to the same reasons.

Online scenario Figures 6(c) and 6(d) show the average job completion times of H-FS-with, H-w/o-P, H-w/o-JS, and H-w/o-DPS in the real cluster run and simulation, respectively. We see that compared with H-FS-with, the average job completion time of H-w/o-P is increased by 27%, while the average job completion time of H-w/o-DPS is increased by 16%. This is because of the same reasons in Figures 6(a) and 6(b). For H-w/o-JS (without job stealing), we see from the figures that it increases the average job completion time on H-FS-with in the online scenario by around 19%. This is because the job stealing policy helps to balance the load among scale-up and scale-out machines, which improves the performance.



(a) Makespan versus data size prediction error rate.

(b) Average job completion time versus data size prediction error rate.

Fig. 7: Sensitivity analysis.

D. Sensitivity Analysis

The benefits of Hybrid cluster depend on the prediction of job characteristics. In this section, we evaluate the robustness of Hybrid cluster to the prediction error. We only show the results for H-FS-with in the simulation due to space limit. In this experiment, the error rate is defined as $\frac{prediction\_value - real\_value}{real\_value}$. A negative error rate means $prediction\_value < real\_value$. For the error rate of "-100", it means $prediction\_value << real\_value$. For example, suppose the $prediction\_value$ is 0.5 GB and $real\_value$ is 16GB. Thus, the error rate is -96.875%≈-100%.

Error in predicted shuffle data size Although previous studies show that the prediction error of job characteristics can be as low as 6.5% [2], we varied the error rate of predicted job shuffle data size by up to 100%. The results below are normalized to the result of H-FS-with without error.

Figures 7(a) shows the makespan versus error in data size in the batch scenario. The figure indicates that Hybrid cluster can maintain very similar makespan when the error is less than 30%. However, as the error increases, the makespan is also increased. This is because as the error increases, the job type of more jobs may be wrongly decided and some scale-out jobs may be run on scale-up machines, reducing the benefits of Hybrid cluster. Figures 7(b) shows the average job completion time versus error in data size in the online scenario. Similar results are observed: the Hybrid cluster maintains similar performance when the error is low; however, as the error increases, the performance gets worse. The results demonstrate that the performance of Hybrid cluster is not quite sensitive to small error in predicted job data size.

V. Related Work

Li *et al.* [26–28] focused on the performance of Hadoop on HPC environments. They investigated the feasibility of replacing HDFS with the remote file system on HPCs, and proposed to dynamically select the appropriate file systems based on the job characteristics. Our paper focuses on a different perspective to accelerate the big data analytics.

Due to the diversity of job characteristics, several studies [3–5] advocate the use of scale-up machines instead of the traditional scale-out machines to handle such diversity. Appuswamy *et al.* [4] conducted a comprehensive measurement study and found that scale-up is sometimes better in some cases than scale-out for specific workloads. Motivated by these works, we aim to design the hybrid scale-up/out cluster to improve the performance of current big data analytics.

The scheduling problem in heterogeneous cluster has attracted much attention [3, 8–10]. They identify the causes of poor performance on heterogeneous cluster and improve the performance using techniques such as scheduling backup copies and estimating the job progress and prioritizing different jobs based on their progress. However, none of these proposals consider the diversity of jobs in their solutions. In this paper, we leverage the observation that different machines may result in different performance for various jobs and design a Hybrid cluster to accelerate big data analytics.

Recently, there have been many studies [2, 12, 15, 29–31] focusing on designing cluster schedulers to improve the performance of the clusters (e.g., throughput and SLOs). In this paper, we do not aim to replace these proposed schedulers, but aim to explore a novel concept for better performance. Our work is orthogonal to these studies and can be combined with them for performance improvement. For example, we can first utilize our work to place the jobs and data onto different machines, and then apply the cluster schedulers to further schedule jobs on scale-up or scale-out machines.

VI. Conclusion and Future Work

In this paper, we aim to design a Hybrid scale-up/out cluster to improve the performance of workloads that have diverse job characteristics. To solve the job and data placement challenges, we propose a Hybrid architecture with corresponding job placement strategy and data placement strategy to address the challenges. We implement Hybrid on top of YARN. We evaluate Hybrid by running a production workload (FB-2010) with both real cluster run and large-scale trace-driven simulation. The results show that accompanying with our strategies, Hybrid cluster can reduce the makespan by 16% and the median job completion time by 60%, compared to tradition scale-out cluster with state-of-the-art schedulers. In the future, we plan to further explore a more complex Hybrid cluster with more kinds of machines to fit more diverse workloads.

Acknowledgment

This research was supported in part by U.S. NSF grants NSF-1827674, CCF-1822965, OAC-1724845, ACI-1661378 and CNS-1733596, and Microsoft Research Faculty Fellowship 8300751.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proc. of OSDI*, 2004.

[2] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar, "Network-aware scheduling for data-parallel jobs: Plan when you can," in *Proc. of SIGCOMM*, 2015, pp. 407–420.

[3] G. Lee, B.-G. Chun, and H. Katz, "Heterogeneity-aware resource allocation and scheduling in the cloud," in *Proc. of HotCloud*, 2011.

[4] R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron, "Scale-up vs scale-out for hadoop: Time to rethink?" in *Proc. of SOCC*, 2013.

[5] Z. Li and H. Shen, "Designing a hybrid scale-up/out hadoop architecture based on performance measurements for high application performance," in *Proc. of ICPP*, 2015.

[6] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The Case for Evaluating MapReduce Performance Using Workload Suites." in *Proc. of MASCOTS*, 2011.

[7] K. Elmeleegy, "Piranha: Optimizing Short Jobs in Hadoop," in *Proc. of VLDB Endow*, 2013.

[8] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *Proc. of OSDI*, 2008, pp. 29–42.

[9] R. Gandhi, D. Xie, and Y. C. Hu, "Pikachu: How to rebalance load in optimizing mapreduce on heterogeneous clusters," in *Proc. of ATC*, 2013, pp. 61–66.

[10] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. N. Vijaykumar, "Tarazu: Optimizing mapreduce on heterogeneous clusters," in *Proc. of ASPLOS*, 2012, pp. 61–74.

[11] Z. Li, H. Shen, W. Ligon, and J. Denton, "An exploration of designing a hybrid scale-up/out hadoop architecture based on performance measurements," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 2, pp. 386–400, 2017.

[12] M. Zaharia, D. Borthakur, S. Sen, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *Proc. of EuroSys*, 2010.

[13] "Apache Spark," https://spark.apache.org/.

[14] "Apache Hadoop," http://hadoop.apache.org/.

[15] "Fair Scheduler," https://hadoop.apache.org/docs/r2.6.0/hadoop-yarn/hadoop-yarn-site/FairScheduler.html.

[16] "Capacity Scheduler," https://hadoop.apache.org/docs/r2.6.0/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html.

[17] "Newegg," http://www.newegg.com/.

[18] "CPU-world," http://www.cpu-world.com/.

[19] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca, "Jockey: guaranteed job latency in data parallel clusters," in *Proc. of EuroSys*, 2012, pp. 99–112.

[20] P. Delgado, F. Dinu, A.-M. Kermarrec, and W. Zwaenepoel, "Hawk: hybrid datacenter scheduling," in *Proc. of ATC*, 2015, pp. 499–510.

[21] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[22] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. of IJCAI*, 1995.

[23] "Python Scikit Learn SVM Library," http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#.

[24] Z. Li, H. Shen, and A. Sarker, "A network-aware scheduler in data-parallel clusters for high performance," in *Proc. of CCGrid*, 2018.

[25] Z. Li and H. Shen, "Co-scheduler: Accelerating data-parallel jobs in datacenter networks with optical circuit switching," in *Proc. of ICDCS*, 2019.

[26] Z. Li, H. Shen, J. Denton, and W. Ligon, "Comparing application performance on hpc-based hadoop platforms with local storage and dedicated storage," in *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 2016, pp. 233–242.

[27] Z. Li and H. Shen, "Performance measurement on scale-up and scale-out hadoop with remote and local file systems," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, 2016, pp. 456–463.

[28] ——, "Measuring scale-up and scale-out hadoop with remote and local file systems and selecting the best platform," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 11, pp. 3201–3214, 2017.

[29] H. Shen, L. Yu, L. Chen, and Z. Li, "Goodbye to fixed bandwidth reservation: Job scheduling with elastic bandwidth reservation in clouds," in *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2016, pp. 1–8.

[30] R. Grandl, M. Chowdhury, A. Akella, and G. Ananthanarayanan, "Altruistic scheduling in multi-resource clusters," in *Proc. of OSDI*, 2016.

[31] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni, "GRAPHENE: Packing and dependency-aware scheduling for data-parallel clusters," in *Proc. of OSDI*, 2016.