2017 IEEE International Parallel and Distributed Processing Symposium

Addressing Performance Heterogeneity in MapReduce Clusters with Elastic Tasks

Wei Chen*, Jia Rao†, Xiaobo Zhou*
*University of Colorado, Colorado Springs {cwei,xzhou}@uccs.edu
†University of Texas, Arlington jia.rao@uta.edu

***Abstract*—MapReduce applications, which require access to a large number of computing nodes, are commonly deployed in heterogeneous environments. The performance discrepancy between individual nodes in a heterogeneous cluster present significant challenges to attain good performance in MapReduce jobs. MapReduce implementations designed and optimized for homogeneous environments perform poorly on heterogeneous clusters.**

We attribute suboptimal performance in heterogeneous clusters to significant load imbalance between map tasks. We identify two MapReduce designs that hinder load balancing: (1) static binding between mappers and their data makes it difficult to exploit data redundancy for load balancing; (2) uniform map sizes is not optimal for nodes with heterogeneous performance. To address these issues, we propose *FlexMap*, a user-transparent approach that dynamically provisions map tasks to match distinct machine capacity in heterogeneous environments. We implemented *FlexMap* in Hadoop-2.6.0. Experimental results show that it reduces job completion time by as much as 40% compared to stock Hadoop and 30% to SkewTune.

I. Introduction

MapReduce is a popular parallel programming model for processing large data sets on distributed clusters [1]. The power of MapReduce is a simplified interface to express computations that need to be parallelized on clusters without worrying about fault tolerance and load balancing. Users express computation as *map* and *reduce* functions and the MapReduce library automatically parallelizes the computation by dispatching map and reduce tasks on the cluster. The automation hides the messy details of parallelization to enable an easy-to-use programming interface, but makes performance optimizations difficult on distributed clusters. The automatically generated tasks have **homogeneous** configurations and run inefficiently on machines of **heterogeneous** performance.

Performance heterogeneity has become increasingly common in MapReduce clusters. Not only can continuous device upgrading in a cluster lead to distinct processing capabilities on nodes with different generations of hardware, running MapReduce on virtual resources in a multi-tenant cloud [2], [3], [4], [5] can also introduce uncontrollable performance variations due to resource contentions on the shared cloud infrastructure. For the ease of management, most MapReduce implementations, such as Hadoop [6], assume a homogeneous cluster and design task scheduling, data placement, and fault tolerance based on this assumption. In a heterogeneous environment, these designs inevitably lead to suboptimal performance and severe load imbalance in the cluster. The stragglers that caused by slow machines can run five times longer those on fast machines. The culprit is the disparate task execution speeds on heterogeneous machines which results in excessive idleness in the cluster; fast machines that finish tasks sooner are unable to help slow machines.

Existing mechanisms in MapReduce for handling node failures and load balancing mitigate performance heterogeneity to a certain extent but can be ineffective in a realistic cluster. MapReduce runs a speculative copy of a task on a different machine if the task does not make good progress on its original host. It also allows fast worker nodes to execute tasks remotely if there are no local tasks available when fast machines become idle. Ideally, if tasks are infinitely small, these mechanisms achieve perfect load balance even on heterogeneous machines [7]. However, fine-grained tasks incur considerable overhead in MapReduce computation. Popular MapReduce frameworks, such as Hadoop [6], YARN [8] and SPARK [9], create JVMs to run individual tasks. The overhead to start a JVM is prohibitively high for small tasks as the JVM startup time can outweigh the actual task runtime [10]. Moreover, these mechanisms require that there be sufficient resources in the cluster for speculative or remote task execution [11]. If slow machines account for a large portion of the cluster or the cluster is highly loaded, there is little room for load balancing.

There are studies focusing on improving MapReduce performance in a heterogeneous environment. Zaharia et al. [12], proposed the LATE scheduler in Hadoop to improve task speculation in heterogeneous clusters. Interference [13] and communication-aware [14] task scheduling and data shuffling [15] have been proved effective in mitigating performance heterogeneity. Techniques on addressing data skewness [16], [17] can also help address performance heterogeneity as the slow execution on less powerful machines can be treated as processing computationally expensive data. Unfortunately, these approaches either require domain knowledge to distribute data across nodes or introduce network overhead when data is re-partitioned between fast and slow machines.

In this paper, we attack the performance heterogeneity problem in MapReduce clusters from a different angle. We believe that perfect load balancing or uniform task execution speed can be realized if the amount of data processed at heterogeneous machines matches their respective capabilities. To this end, we propose *FlexMap*, a new map execution engine for MapReduce to create elastic map tasks with different sizes. This allows MapReduce clusters to run **heterogeneous** (big and small) tasks on **heterogeneous** (fast and slow) machines. FlexMap breaks the designs in current MapReduce to attain elasticity: 1) map tasks have uniform inputs and 2) map tasks

1530-2075/17 $31.00 © 2017 IEEE
DOI 10.1109/IPDPS.2017.28

IEEE computer society

are statically bound to their inputs. FlexMap centers on two new designs: 1) multi-block execution (MBE) and 2) late task binding (LTB) to enable elastic tasks. MBE allows map tasks to start with a input size (e.g., 8MB) and independently grow to the optimal data size that matches the host machine's capability. Once the optimal task size is determined, LTB creates tasks using local blocks on the node to maintain data locality. We also optimize reduce scheduling to adapt to the heterogeneous computation in the map phase.

FlexMap is transparent to users and does not require any change to the existing MapReduce jobs. It continuously measures the efficiency of map task execution beginning with a small size and automatically determines the minimum task size that avoids high JVM startup overhead. FlexMap assigns the minimum task size to the slowest machine(s) and tasks on faster machines are proportionally larger based on their performance relative to the slowest machine. We implemented FlexMap in Hadoop-2.6.0 (a.k.a., YARN) and evaluated its performance on three MapReduce clusters: a 12-node heterogeneous cluster, a 20-node cluster in a university cloud, and a 40-node multi-tenant cluster. We compared our approach with stack Hadoop and a recently proposed skew mitigation approach SkewTune [16]. Experimental results with the Purdue MapReduce Benchmark suite (PUMA) [18] show that FlexMap reduce job completion time by as much as 40% compared to stock Hadoop and 30% to SkewTune.

The rest of this paper is organized as follows. Section II introduces the background of YARN, discusses existing issues and presents a motivating example. Section III elaborates on FlexMap's architecture and key designs. Section IV presents evaluation results. Related work is presented in Section V. We conclude this paper in Section VI.

II. Background and Motivation

In this section, we first describe the basics of MapReduce in the context of Apache YARN and show how automatic parallelization with homogeneous tasks causes severe performance degradation in heterogeneous environments. We further demonstrate that task size has complex implications for job performance, load balancing, and resource utilization.

A. MapReduce Execution

MapReduce execution is divided into two functions: *map* and *reduce*. The map function takes the input data and produces a list of intermediate key/value pairs. The intermediate values associated with the same key are grouped together and passed to the same reduce function via *shuffle*, an all-map-to-all-reduce communication phase. MapReduce partitions the input data into even-sized splits and stores them on a distributed file system (HDFS) throughout the cluster. Each input split corresponds to a map task and the split size matches the block size in the HDFS. The default block size is 64 MB. Each split/block is replicated on multiple nodes for fault tolerance. Map tasks are statically bound to their input splits. When a worker node has computational resources, it preferably runs map tasks which have replicas of input splits stored on the node to preserve data locality. If no local splits are available, map tasks that have splits on remote nodes will be launched.

TABLE I
THE HARDWARE CONFIGURATION OF A HETEROGENEOUS CLUSTER

Machine model	CPU model	Memory	Disk	Number
PowerEdge T320	Intel Sandy Bridge 2.2GHz	24GB	1TB	2
PowerEdge T430	Intel Sandy Bridge 2.3GHz	128GB	1TB	1
PowerEdge T110	Intel Nehalem 3.2GHz	16GB	1TB	2
OPTIPLEX 990	Intel Core 2 3.4GHz	8GB	1TB	7

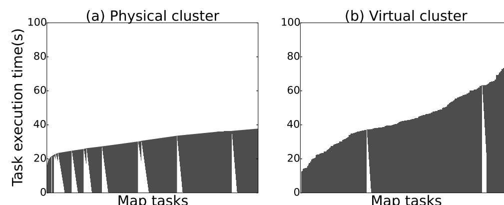

Fig. 1. Map task runtime of *wordcount* in heterogeneous clusters.

B. Degraded MapReduce Performance due to Heterogeneity

We evaluated the performance of the *wordcount* benchmark in two heterogeneous environments, a physical cluster composed of 12 machines with multiple generations of hardware and a 20-node virtual cluster in our university cloud. Machine types for physical cluster are listed in Table I.

Figure 1 (a) shows that hardware heterogeneity caused significant imbalance between individual map tasks. The slowest map task ran as much as twice longer than the fastest task. The imbalance between map tasks was exacerbated in the virtual cluster, where interference from other VMs caused large performance disparity between MapReduce nodes. As shown in Figure 1 (b), about 20% of the map tasks experienced slowdowns in the cloud and were 5x slower than the faster tasks. Although YARN implements the state-of-art LATE scheduling algorithm [12] for speculative Execution, performance heterogeneity still incurred more than 50% of runtime slowdown on the physical cluster compared to that on a same-sized homogeneous cluster containing only slow machines. While we were unable to measure *wordcount* performance on the virtual cluster in an interference-free environment, we expect the overall slowdown to be even greater than that on the physical cluster. Similar results were also observed in [14].

Analysis We attribute the root causes of load imbalance to uniform map sizes and the static binding of input splits and map tasks. Since in MapReduce, containers are granted based on their embedded locality information. Figure 2 illustrates how homogeneous map tasks can make load balancing ineffective and lead to idleness in the cluster. We assume that there are three machines, two slow and one fast nodes, in the cluster. The ratio of the machine capacities is 1:1:3. The default replication factor of 3 is used. In such a small cluster, every node stores the entire input data. Ideally, perfect load balancing guarantees that the amount of data processed at each machine is proportional to its respective capacity. However, as shown in Figure 2, the number of tasks completed (denoted by dotted rectangles) is 1:1:2. Although the fast node has access to all input data, it is unable to process data proportional to its capacity. The culprit is that map tasks have fixed sizes and static bindings to input splits. For example, because the first

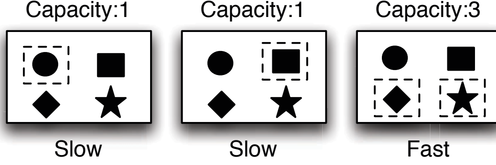

Fig. 2. Uniform map task size and static input binding limits the effectiveness of load balancing in MapReduce. The shapes denote the replicas of different splits and the dotted rectangles represent the containers that run map tasks.

two splits (denoted as the solid cycle and rectangle in Figure 2) are being processed at the two slow nodes, the fast node is unable to process their replicas even it has sufficient capacity.

Speculative execution can possibly mitigate stragglers due to a few slow machines, but it is **not a reliable** solution to load imbalance. As discussed in [12], speculation can fail in many scenarios. First, speculation is not effective if there are more slow machines than fast machines, no matter how fast these machines are. There lack sufficient containers to launch all speculative copies. Second, for MapReduce jobs with multiple waves of map tasks, speculation only occurs at the last wave. It is likely that the stragglers on slow machines have made considerable progress and are not eligible for speculation.

These findings motivated us to develop a new mechanism to addressing performance heterogeneity in MapReduce cluster. We believe that tasks running on heterogeneous machines should take different amount of input data.

C. Implications of Map Task Size

Since map tasks are statically bound to input splits, the block size in HDFS determines the size of a task. As discussed earlier, fine-grained tasks help mitigate load imbalance due to performance heterogeneity but incur high parallel overhead. It is challenging to strike a balance between *efficiency* and *load balance* in parallel computing. To quantitatively study the implications of map task sizes in heterogeneous environments, we ran *wordcount* on the 20-node virtual cluster and compared the variance of map runtimes using the default 64MB and fine-grained 8MB block sizes. The larger the variance between task execution times, the higher degree of load imbalance. Figure 3 (a) shows the probability density function (PDF) of normalized map execution time under different block sizes. As shown in the figure, small task size (i.e., 8MB) resulted in low runtime variance and most task execution time fall in the range of 0.3-0.5. In contrast, larger block size (i.e., 64MB) led to heavy tails in runtime distribution and large variance between task execution times. The results suggest that fine-grained tasks be more resilient to performance heterogeneity as more work or a large number of small tasks can be load balanced onto fast machines, which leads to uniform and short execution times.

The overall MapReduce performance is the result of complex interplays between task granularity and performance heterogeneity. Small tasks helps load balancing but incurs high overhead; large tasks enable efficient execution, though suffer load imbalance. We quantify task *productivity* by calculating the ratio of effective task runtime and total task runtime:

$$Productivity = \frac{Effective\ runtime}{Total\ runtime}, \quad (1)$$

where *effective runtime* refers to the period since a map task starts to read input from HDFS until it finishes writing intermediate results back to disk. The *total runtime* includes task execution overhead, such as YARN container allocation time and JVM startup time. *Productivity* measures the efficiency of map computation given fixed execution overhead. We further use *efficiency* of the map phase to quantify load imbalance:

$$Efficiency = \frac{Serial\ runtime}{Map\ phase\ runtime \times \#\ of\ available\ containers}. \quad (2)$$

Because the map phase does not require any synchronizations between map tasks, inefficiency is mainly due to load imbalance. We approximate jobs' serial runtime using the sum of all map tasks. We estimate the runtime of the map phase as the time between the first container starts and the last container that runs a map task stops in the cluster. The map phase runtime also includes the execution overhead due to container and JVM startup. Given the same task size, the overhead and number of containers are fixed. Thus, high *efficiency* indicates good balance.

We first ran *wordcount* on a homogeneous cluster and studied the relationship between task size, job completion time, and task productivity. Figure 3 (b) and (c) show that small task sizes incurred significant overhead with a productivity as low as 0.28 (i.e., 8MB). A low productivity indicated that most task runtime was dominated to container and JVM startup time. However, large task size is more susceptible to heterogeneity induced imbalance. Figure 3 (d) shows the job completion time and efficiency on a 6-node heterogeneous cluster. By comparing Figure 3 (b) and (d), we can see that heterogeneity inflicted significant performance slowdown at each task size and the slowdown was mainly due to dropped efficiency at each size.

Figure 3 (d) shows that, in a heterogeneous environment, job completion time initially dropped as task size increased, suggesting that improved productivity outweighed load imbalance (i.e., low efficiency). Further increasing task size led to degraded performance when load imbalance dominated. Through these experiments, we had two **key findings** on improving MapReduce performance in heterogeneous environments:

- Load balancing should be performed at fine granularity to mitigate performance heterogeneity but tasks should be run at coarse granularity to avoid execution overhead.
- The optimal task size depends on the interplay between the execution overhead, such as container and JVM startup time, the computation needed by a particular job, and the degree of performance heterogeneity.

[***Summary***] The existing homogeneous map task model in MapReduce fails to simultaneously satisfy the requirements of load balancing and execution efficiency in heterogeneous environments. Further, it is unable to exploit the data redundancy (i.e., replicas of the same HDFS block) available on MapReduce clusters to address heterogeneity.

III. FlexMap Design

The key idea of *FlexMap* is to launch heterogeneous tasks with different sizes in the map phase to match the processing capability of machines. All machines in the cluster start with

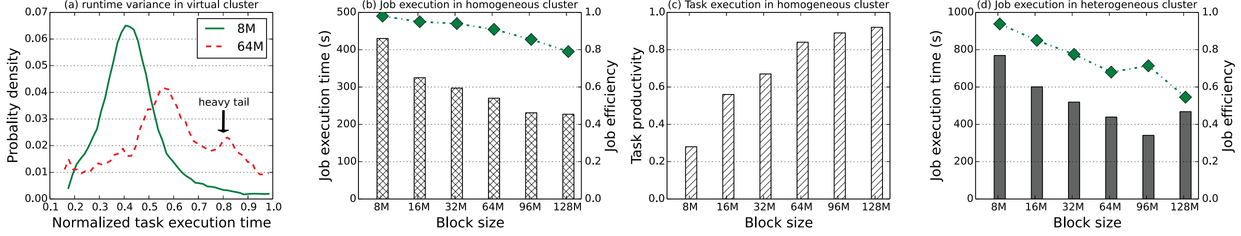

Fig. 3. Map task size has important implications for job performance and efficiency. (a) Small tasks are more resilient to performance heterogeneity; (b)-(d) small tasks incur high execution overhead and job efficiency measures the overall load balance in MapReduce clusters.

the same fine-grained map size with the basic block unit (BU), i.e., 8MB, and grow their task sizes independently. Map tasks grow based on two criteria: 1) vertical growth according to task *productivity* and 2) horizontal growth proportional to machine speed. This design achieves load balancing by assigning different task sizes to heterogeneous machines, which realizes load differentiation at steps as fine as a basic BU, and avoids high execution overhead of running small tasks.

A. Architecture Overview

FlexMap centers on the design of two new mechanisms: *multi-block execution* (MBE) and *late task binding* (LTB) to enable elastic map tasks:

- **Multi-block execution** realizes elastic tasks by dynamically changing the number of BUs in map tasks' input splits. With the new MBE engine, map tasks take an array of BUs as input.
- **Late task binding** allows map tasks to be created at the time of job submission, but delays the input-to-task binding to when tasks are dispatched to worker nodes. It maximally preserves data locality in heterogeneous environments.

Figure 4 shows the architecture of FlexMap. FlexMap augments the MapReduce Application Master(AM) with three new components: *DataProvision* (DP), *SpeedMonitor* (SM), and late task binding (LTB). To support elastic map tasks, worker nodes are equipped with *multi-block execution* (MBE). The augmented framework works as follows: upon receiving a job submission, 1) AM initializes a large number of map templates each using one basic BU (i.e., 8 MB block) as input; 2) AM requests containers for these tasks from the Resource Manager(RM). These containers embed resource demands but lack locality information; 3) RM grants containers to AM when they become available. The granted containers are bound to particular nodes; 4) given a container, AM estimates the speed of the host node of the container using SM and calculates the task size using DP. Based on the task size, LTB creates a real map task and its input split contains a sequence of basic BUs which are provisioned from the container's host node; 5) the created elastic map task is dispatched to the corresponding node; 6) worker nodes periodically update their speed to SM through heartbeat. When all BUs of a job have been provisioned, AM stops creating new map tasks.

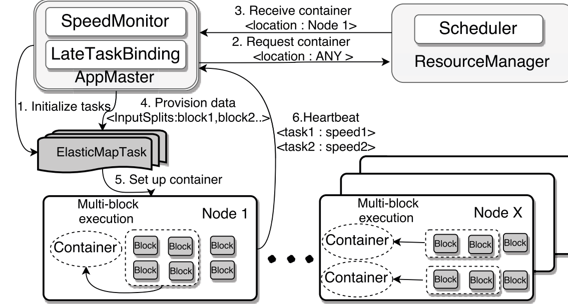

Fig. 4. The architecture of FlexMap in a YARN cluster.

B. Multi-block Execution

Multi-block execution engine is inherited from the traditional MapReduce execution engine but is able to process multiple blocks at once. The existing map execution engine has two constraints: 1) a map is bound to and can only process one block; 2) the task (block) size is defined in the cluster configuration file and cannot be changed during job execution. MBE replaces the original map execution engine and redefines the input split as an array of BUs. A BU is the smallest unit in task size changes and we empirically set it to 8 MB. The size of a map is determined by the number of elements (or BUs) in the input split array. MBE does not require any changes to user program but needs minor modifications to the existing map task interface. First, map tasks are modified to continuously read BUs until reaching the end of the input split array. Second, MBE breaks the calculation of task progress in MapReduce, which tracks how far the map has progressed through the input split. With MBE, progress calculation is based on the aggregate size of all BUs in the array.

C. Late Task Binding

Two major changes in AM are needed to support LTB. Traditional MapReduce binds individual map tasks to different HDFS blocks, one map per block, when a job is submitted. Such locality information is embedded in container requests. When a container is granted by RM, the map task that meets the locality constraint will be dispatched to the container and execute on the corresponding worker node. LTB breaks the stated map execution flow and delays the creation of map tasks to when a container is granted, from where task size can be determined based on the speed of the machine hosting the

container. The first change is to the job submission step. LTB divides a job's input file to even-sized BUs (i.e., 8MB blocks) and creates a large number of fine-grained map, each is bound to one BU. These tasks are templates from which elastic map tasks will be created. Not all the task templates will eventually turn into real map tasks. If all BUs from a job are processed, unused map templates will be discarded. Based on the map templates, AM requests a large number of containers and these containers do not have locality constraints.

The second change is to the map dispatch step. Traditional MapReduce simply dispatches map to an affiliated container. In contrast, LTB needs to create a real map task from a template before dispatching the task. As discussed earlier, the new map task engine MBE takes an array of BUs as the input split. Given a granted container, LTB is responsible for constructing the input split and forming a map task based on the speed of the machine that hosts the container. We will discuss the algorithms that determine the task size in Section III-E. The key challenge lies in preserving data locality in the newly created map task. Given a granted container and a task size of n BUs, LTB maximizes data locality by constructing the input split from BUs that have replicas on the machine hosting the container.

LTB maintains two `HashMaps` in AM to trace the locality information of unprocessed BUs of a job. The `NodeToBlock` hash map takes a node ID and outputs a list of BUs locally stored on the node. The `BlockToNode` hash map contains an inverse mapping from a BU ID to a list of nodes that store the replicas of the BU. To construct an n-BU map task, LTB obtains a list of BUs from `NodeToBlock` using the container's node ID. For each BU in the list, LTB looks up the node ID that stores a replica of the BU in `BlockToNode` and deletes the BU from the corresponding entry indexed by the node ID in `NodeToBlock`. As such, LTB guarantees that a BU will only be processed by one map task. This process is repeated for n times until an input split is formed. LTB ensures mutual exclusive access to both hash maps in case of multiple mappers being simultaneously created. If the container's hosting node has less than n BUs available, LTB chooses BUs remotely stored on other nodes to satisfy the n-BU task size requirement. LTB follows a heuristic to select remote BUs from nodes that have most unprocessed BUs.

D. Monitoring Node Speed

We use *input processing speed* (IPS) to measure the capacity of worker nodes. It is defined as:

$$IPS = \frac{HDFS_BYTES_READ}{currentTime - taskStartTime}, \quad (3)$$

where *HDFS_BYTES_READ* is the amount of data that has been processed in the task's input split and the denominator is the task's current runtime since it was started. Each container reports its IPS to AM through heartbeat, whose period is empirically set to 5 seconds. IPS alone can not accurately measure node speed as some records in the input split are more expensive to process than others, leading to varying IPS. We use the average of 5 IPSes reported by containers on the same node to measure machine speed. As such, data skewness is mitigated across multiple containers. We use a sequence number to identify heartbeat messages from different rounds and calculate the average IPS from the same round of heartbeat. The *SpeedMonitor* exposes a new interface `getSpeed` in AM that returns the average IPSes on individual machines. The relative machine speeds are used to determine the map sizes on these machines.

Algorithm 1 Dynamic map task sizing

```
1:  Variable: Node i; Current task size on the ith node m_i; Node
    list 𝕃; Size of the block unit b; Size unit on the ith node s_i.
2:  /* Initialize task size and size unit to one BU, i.e., 8MB */
3:  for each node i in list 𝕃 do
4:      m_i = s_i = b
5:  end for
6:  /* Grow task size to mitigate execution overhead */
7:  function VERTICAL_SCALING(i)
8:      if get_productivity(i) < FAST_LIMIT then
9:          s_i = s_i * 2
10:     else if get_productivity(i) < LINEAR_LIMIT then
11:         s_i = s_i + b
12:     end if
13: end function
14: /* Set task size proportional to relative machine speed */
15: function HORIZONTAL_SCALING(i)
16:     j = get_slowest_node()
17:     m_i = s_i * get_speed(i) / get_speed(j)
18: end function
```

E. Heterogeneity-aware Map Sizing

Small tasks are more resilient to performance heterogeneity but incur high overhead, while large tasks cause significant load imbalance. Multi-block execution and late task binding enable MapReduce to run map tasks with variable sizes.

The objectives are to balance load across nodes at fine granularity but run coarse-grained map tasks to avoid high overhead. To this end, FlexMap begins with fine-grained tasks and automatically grows task size according to execution overhead and relative machine speed. FlexMap devises *vertical scaling* and *horizontal scaling* to determine the optimal size for mitigating overhead and addressing heterogeneity, respectively. Algorithm 1 shows the dynamic map task sizing algorithm. Map begins with a size of one BU (i.e., 8MB) and grows in two directions. We assume that map tasks run in multiple waves, which gives us the opportunity to evaluate various map sizes.

First, the map size grows vertically and independently on each machine. We define *size unit* (SU) as the step for map size change for each wave. Vertical scaling aims to quickly determine the map size that leads to highly efficient execution. We use *productivity* defined in Section II-C to measure execution efficiency. Vertical scaling includes two phases: *fast scaling* and *linear scaling*. If map task productivity is lower than `FAST_LIMIT`, size unit is doubled at each wave (line 8-9) to quickly jump small sizes causing inefficiency. After that, task size is incremented by one BU at each wave (line 10-11). If productivity goes beyond `LINEAR_LIMIT`, tasks stop growing. We set `FAST_LIMIT` and `LINEAR_LIMIT` to 0.8 and 0.9, respectively. Second, FlexMap adjusts map sizes

horizontally across machines based on their relative speeds. FlexMap normalizes a machine's relative speed to that of the slowest machine. Speed is measured by the IPS rate reported by individual nodes. Then, the map size at particular wave equals to the size unit at the wave times the relative speed (line 17). Note that each machine can grow at different speeds, i.e., different values of size unit. This is to ensure that slow machines, whose size unit grows slowly in vertical scaling, do not prevent fast nodes from growing quickly. Through this way, for small input size with only a few waves, the straggler will not be exacerbated by slow machines, while straggler will be mitigated by dynamic map sizing for large input size.

F. Optimizing Reduce Scheduling

Reduce tasks fetch intermediate results generated by map tasks through an all-to-all shuffle phase. If data is distributed evenly among nodes, there is no locality for a reduce task as each reducer gets data from every mapper. In a heterogeneous environment, FlexMap assigns more data on fast machines, creating skewness in the distribution of intermediate results in the cluster. By default, intermediate results are partitioned evenly among reducers. As such, if reduce tasks are evenly dispatched to heterogeneous nodes, not only can slow machines delay the entire reduce phase due to one-wave execution, but also incur significant inter-machine network traffic.

To leverage the skewness created by FlexMap on intermediate data, we optimize reduce scheduling by dispatching more reducers to fast machines. Ideally, a reduce task should be scheduled on a machine that stores most of its needed data to avoid inter-machine shuffling. However, unlike mappers whose data-to-machine affinity is fixed upon job submission, reducers' input distribution is not easily known unless an expensive cluster-wide intermediate result scan is performed. We devise a simple yet effective reduce optimization. Each node has a *bias* for dispatching reducers. FlexMap normalizes machine capacity to the range of (0, 1] (denoted as c_i) with the fastest machine being speed 1. The reducer bias of node i is set to c_i^2. When deciding which node to schedule a reducer, FlexMap generates a random number in range (0, 1] and randomly picks a node i. If the random number follows in (0, c_i^2], dispatch the reducer onto this node. If not, repeat the process until a node is found to run the reducer. This design ensures that more reducers will be dispatched onto faster nodes.

G. Implementation

We have implemented FlexMap on top of YARN. The implementation was based on Hadoop-2.6.0 and consisted of about 2500 lines of Java code. *SpeedMonitor* was implemented at the AM as a standalone process. The heartbeat communications between the AM and worker nodes were implemented using RPC. Java class `ElasticMapTask` extends the default `MapTask` and Hadoop was modified to use the new map task interface when a job is submitted. The new map class provides functions to read an array of blocks in the input split. To support late task binding, we implemented a `setBlock` interface for map tasks. When the task size is determined, a mapper calls `setBlock` to expand its input split. To dynamically set mapper size when YARN containers become available, we modified `RMContainerAllocator` to signal `JobImpl` the availability of new containers. `JobImpl` then calls the dynamic map sizing algorithm (Algorithm 1) to determine the corresponding mapper size given the locality of the container.

IV. Evaluation

In this section, we evaluate FlexMap on three heterogeneous environments: 1) a 12-node physical cluster that consists of three types of machines; 2) a 20-node virtual cluster in our university cloud; and 3) a larger scale 40-node cluster running multi-tenant workloads.

A. Experimental Settings

Platform settings Table I lists the hardware configurations of the three types of machines in the physical cluster. The virtual cluster ran 8 HP BL460c G6 blade servers interconnected with 10Gbps Ethernet. Each server was equipped with 2-way Intel quad-core Xeon E5530 CPUs and 64GB memory. VMware vSphere 5.1 was used to provide the server virtualization. Each virtual node was configured with 4 vCPUs and 4GB memory. The multi-tenant cluster consisted of 40 nodes, each was equipped with two Intel Xeon E5-2640 CPUs and 128GB memory. These servers were connected with 10 Gbps Ethernet.

MapReduce settings We deployed Hadoop version 2.6.0 (a.k.a., YARN) on these clusters and each node ran Ubuntu 14.10. In each cluster, one node ran as the Resource Manager and NameNode. The remaining nodes were worker nodes for HDFS and MapReduce computation. We used two HDFS block sizes: the default 64MB and the industry recommended 128MB. If not otherwise stated, the replication factor for each HDFS block was set to the default 3. FlexMap used a basic block unit of 8MB as the starting size of all mappers.

Comparison We compare FlexMap with stock Hadoop and a recently proposed skew mitigation approach: SkewTune [16]. SkewTune parallelizes a straggler task by repartitioning and redistributing its input data across all available nodes. It assumes all slave nodes have the same processing capability. However, in physical and virtual heterogeneous clusters, the node capability varies across the cluster. Further, in virtual cluster, hotspots may change during the job execution, making it hard to identify a straggler.

Workloads We used the PUMA benchmark suite [18] as the evaluation workload. Table II shows the benchmark configurations. As the three clusters differ in size, we scaled the input sizes of the benchmarks to match the size of the clusters. We used two input sizes: small and large. The small input was used for the small scale physical and virtual clusters while the large input was for the 40-node cluster. These benchmarks had realistic input data from TeraGen, Wikipedia, and Netflix.

B. Reducing Job Completion Time

In this subsection, we study how effective FlexMap is in reducing job completion time (JCT). As we can see from

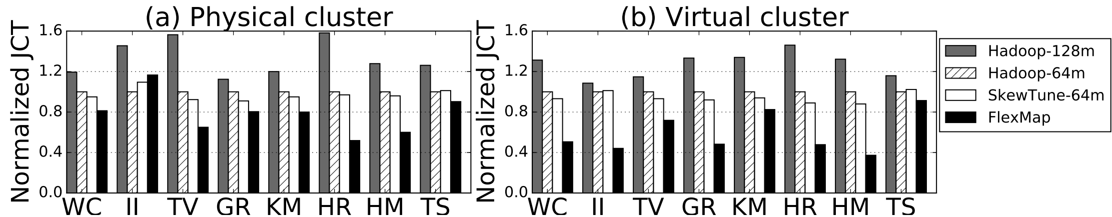

Fig. 5. FlexMap significantly outperformed stock Hadoop in job performance in two heterogeneous environments.

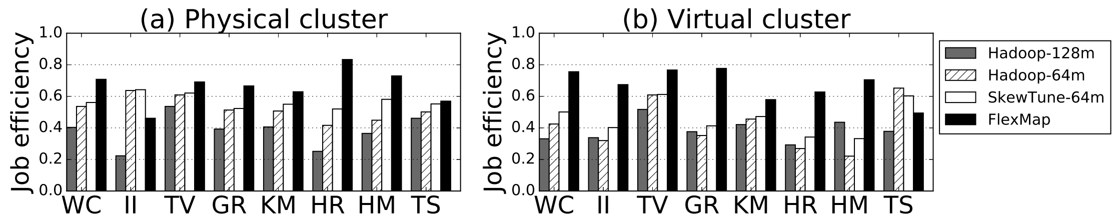

Fig. 6. FlexMap improves job efficiency in two heterogeneous environments.

TABLE II
PUMA BENCHMARK DETAILS

Benchmark	Input Size(GB)	Input Data
wordcount (WC)	20(small) 256(large)	Wikipedia
inverted-index (II)	20(small) 256(large)	Wikipedia
term-vector (TV)	10(small) 256(large)	Wikipedia
grep (GR)	20(small) 256(large)	Wikipedia
kmeans (KM)	10(small) 256(large)	Netflix data, k=6
histogram-movies (HM)	10(small) 128(large)	Netflix data
histogram-ratings (HR)	10(small) 128(large)	Netflix data
tera-sort (TS)	10(small) 128(large)	TeraGen

Figure 5, for Map-heavy benchmarks, such as *wordcount*, *grep*, and *histogram-ratings*, FlexMap outperformed the best performing settings in stock Hadoop by 18.9%, 19.5% and 48.1%, respectively in the physical cluster. FlexMap achieved even more better performance in the virtual cluster, winning over stock Hadoop in *wordcount*,*grep* and *histogram-ratings* by 49%, 51.1% and 52.2%, respectively.

The results indicate that a larger mapper size always led to worse performance in heterogeneous environments. The culprit is that stock Hadoop is unaware of performance heterogeneity and assigns considerable amount of data to slow nodes. Thus, the larger the task size, more data being processed on slow nodes, which causes more performance loss. We will show in Section IV-E, FlexMap can have much larger tasks but only ran them on fast machines.

We also observed that SkewTune only improved stock Hadoop by 5% to 10% for both physical cluster and virtual cluster. That's because SkewTune assumes each node has similar processing capability and it can only deal with data skew with a few stragglers. But in our physical cluster and virtual cluster, slow nodes may accounts for nearly 50% of total nodes.

Another observation is that, on average, FlexMap achieved a larger margin of performance gain on the virtual cluster compared with that in the physical cluster. The average performance gain due to FlexMap in the virtual cluster was more than 43% while the average gain in the physical cluster was 22.2%. The key difference between the physical and virtual clusters is that performance heterogeneity in the virtual cluster is dynamic and the capacity gap between fast and slow machines is more significant. Thus, there is more room for FlexMap to create elastic tasks and address the heterogeneity. Note that FlexMap only marginally improved the performance of *tera-sort* in both environments and even degraded the performance of *inverted-index* in the physical cluster. These applications are dominated by the reduce phase, thereby leaving little room for performance improvement at the map phase. The reduce optimization in FlexMap requires that the intermediate results generated by mappers have skewed distribution on the cluster.

Further, FlexMap inevitably incurs overhead, which is the major reason behind the degraded performance of *inverted-index* in the physical cluster. To determine the optimal task size in heterogeneous environments, FlexMap starts with small tasks and grows task sizes based on feedbacks from previous task waves. As such, MapReduce jobs inevitably spend some time running with suboptimal mapper sizes. However, experimental results in Figure 5 (a) and (b) show that the benefit of FlexMap in addressing heterogeneity outweighed its overhead in most workloads.

C. Improving Job Efficiency

As discussed in Section II-C, MapReduce jobs do not require synchronizations at the map phase. Thus, with coarse-grained tasks, the inefficiency mostly comes from load imbalance between mappers. In this subsection, we show that FlexMap not only reduces job completion time, but also significantly improves load balancing in heterogeneous environments. We use the metric of job efficiency (defined in Section II-C as equation (2)) to measure load balance.

Figure 6 (a) and (b) show the job efficiency of various benchmarks in the physical and virtual cluster, respectively. From the figures, we can see that for workloads that are dominated by the map phase, such as *wordcount*, *grep*, and

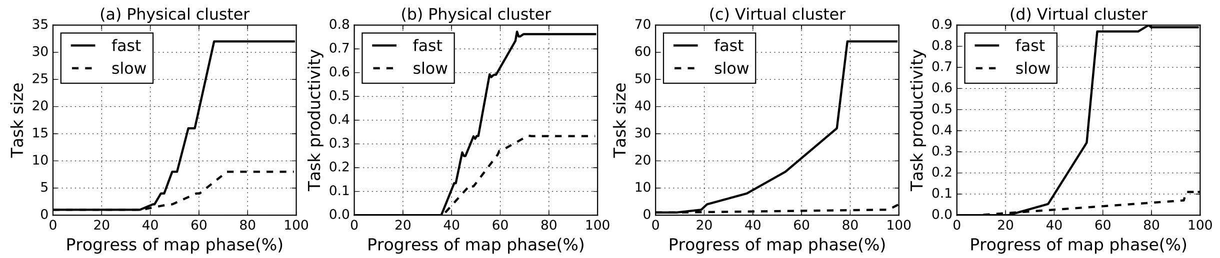

Fig. 7. The changes in map task size and productivity during the execution of *histogram-ratings* in two heterogeneous environments.

histograms-ratings, FlexMap had 17.2%, 15.2% and 41.7% higher efficiency than stock Hadoop in the physical cluster. Similarly, FlexMap outperformed stock Hadoop by 33.13%, 42.7%, and 48.4% for these workloads in the virtual cluster. The same results over SkewTune is 14.6%, 14.3% in physical cluster while 25.4%, 36.4% and 28.5% in virtual cluster. It also shows that SkewTune can not only optimize job performance but also perform low efficiency in a highly heterogeneous cluster. As discussed earlier, small tasks lead to high efficiency but inflict significant performance degradations in MapReduce jobs. In contrast, FlexMap simultaneously optimizes performance and efficiency, suggesting that improving load balance is crucial to MapReduce performance in heterogeneous environments.

As shown in Figure 5, larger task size (i.e., 128MB v.s., 64MB) always leads to worse performance. The conclusion does not extend to job efficiency, which mainly measures the skewness in mapper execution time. Figure 6 (b) shows that larger task size (i.e., 128MB) led to higher efficiency than smaller task size (i.e., 64MB) in the virtual cluster. A possible reason is that fewer larger tasks ran on fewer number of virtual nodes, thereby suffering less interference in the cloud. Similar to our observations in Figure 5, FlexMap was less effective in improving job efficiency for *inverted-index* and *tera-sort* as their computation was dominated by the reduce phase.

D. Overhead

Since individual cluster nodes grow their task sizes independently, FlexMap's overhead is mainly due to vertical scaling, during which map tasks run with suboptimal sizes. To quantify the overhead, we ran *wordcount* on a 6-node homogeneous cluster and compared the performance of FlexMap with that in stock Hadoop. On the homogeneous cluster, horizontal scaling was effectively disabled. As *wordcount* has relatively uniform record distribution in its input, each node grow task sizes at similar speeds. Results show that FlexMap incurs negligible 5% performance penalty compared to stock Hadoop. Thus, in environments with significant performance heterogeneity, the benefit of FlexMap is likely to outweigh its overhead.

E. Dynamic Mapper Sizing

It is interesting to study how FlexMap achieved the improvement and how task sizes changed during MapReduce execution. Figure 7 shows the changes in mapper size and productivity for the *histogram-ratings* benchmark in the physical and virtual clusters. Task productivity is defined in Section II-C as equation (1). It measures the portion of mapper runtime that is truly spent in the map computation. The higher the productivity, the lower the map execution overhead, including YARN container and JVM startup time. We recorded mapper size changes at the AM and calculated their respective productivities. We used a simple performance probe to identify the fastest and slowest node in the physical and virtual clusters.

Figure 7 (a) and (b) plot the task size and productivity in the physical cluster. The x-axis shows the progress of the entire map phase. Starting with the fine-grained task size, i.e., 8MB, it took almost 40% of the map phase time before the first wave of mappers finished. After the first-wave feed back was obtained, both fast and slow nodes grow their mapper sizes. Fast node grew task size at a faster speed than the slow node did. At the completion of each mapper wave, not only was the size unit on each node doubled, the fast node also increased task size in proportion to its relative speed to the slowest node. As a result, the fast node was able to quickly attain high productivity in a few waves. The slow node never grew its task size to high productivity before the map phase completed. This suggests that FlexMap was able to assign more data to fast nodes in this heterogeneous cluster. Finally, the optimal task sizes determined by FlexMap for *histogram-ratings* were 32 BUs (or $32 * 8 = 256$ MB) and 8 BUs (or $8 * 8 = 64$MB) for the fast and slow nodes, respectively. Note that the optimal task size for the fast machine was larger than the 128MB block size in stock Hadoop. It suggests that large task size does not necessarily cause performance degradation in heterogeneous environments. The key is to **match the amount of computation to machine capability**.

The performance discrepancy between fast and slow nodes was more significant on the virtual cluster. As shown in Figure 7 (c) and (d), the final task size of the slow node was 2 BUs compared to 64 BUs for the fast node. One can infer that the slow node, which had considerable interference, did not contribute much to the overall job completion. In stock Hadoop, homogeneous mappers running on such slow nodes can greatly delay the overall job completion. Speculative execution is unlikely to be efficient or even effective. Speculation can only be triggered by the lacking of task progress on slow nodes, which not only incurs repeated computation but may also miss the best timing for load balancing. In the next subsection, we show that speculation is effective if a cluster

contains a few slow machines. In contrast, FlexMap promptly identified the performance difference between machines and pro-actively assigned more work to fast machines.

F. Results on a Large Scale Multi-tenant Cluster

It is important to evaluate the effectiveness of FlexMap at a larger scale. First, on small clusters, there exists considerable data redundancy on each worker node. FlexMap can easily construct large tasks from local HDFS blocks. For example, with a replication factor of 3, each node stores 25% of the job input on a 12-node cluster [1]. Ideally, only 4 fast machines are needed to complete the job and data locality can be preserved. In such an environment, FlexMap is likely to outperform stock Hadoop. However, as cluster size increases, FlexMap may need to access remote BUs to construct a large mapper. Second, it is interesting to study how effective FlexMap is as the number of slow machines in the cluster increases.

We created performance 5%, 10%, 20% and 40% heterogeneity by co-running CPU-intensive background jobs on the 40-nodes cluster. This setting emulates a multi-tenant environment in which a varying number of co-running users create a heterogeneous environment for a foreground MapReduce job.

Stock Hadoop with speculation enabled achieved similar performance compared to FlexMap in Figure 8 (a). Speculation is effective in addressing a few faulty or slow nodes. As the portion of slow machines increased, the performance of Hadoop with and without speculation converged. It suggests that speculation alone is not an effective approach to attaining load balancing in heterogeneous environments. As shown in Figure 8 (a) - (d), FlexMap outperformed stock Hadoop when there were few slow machines and the performance gain expanded when more machines were slowed down. FlexMap was able to reduce job completion time by as much as 40% compared to Hadoop. Similar trends can also be observed for SkewTune, SkewTune can alleviate skew when there were small amount of stragglers, while its performance approached to stock Hadoop when slow machines increased. One exception is *inverted-index* in Figure 8 (a), for which FlexMap had worse performance than stock Hadoop. It again confirms that FlexMap incurs overhead and the cost of expanding task sizes can outweigh its benefit in load balance. Remote BU access seemed not to be an issue as FlexMap's gain did not plummet as the portion of slow nodes increased. 10 Gbps Ethernet could have played a role in bridging the performance gap between local and remote data access.

G. Discussion

Extensibility Since MapReduce model has been widely applied, our approach, matching task input data with machine capacities, can be also extended to other platforms. For example, Spark [9] which extends MapReduce with more flexible tasks organizations and communication patterns could possibly benefit from our proposed FlexMap. Since each Spark task(resides in executor) forms its processing data from local input and shuffled data. Similar to Hadoop, its local input data also reads from HDFS in forms of data blocks and accounts for largest part of processing data(We found less than 5% of data was shuffled from other tasks in Spark machine learning applications). We believe the stragglers will be exacerbated during iterations.

Map-heavy Workloads Previous research [19] on production traces has showed that Map-heavy workloads are dominant. 30% of jobs are map only, and thus have 0 shuffle data. For 70% of jobs, its shuffle data is only 10% of its input data, which is regarded as map-heavy jobs. So FlexMap can significantly benefit production heterogeneous Hadoop cluster.

V. Related Work

There exist studies that addressed straggler tasks caused by heterogeneity in Hadoop clusters. Zaharia et al., [12] were the first work to point out and address the shortcomings of MapReduce in heterogeneous environments. The authors observed that the built-in stragglers identification mechanism does not work correctly in heterogeneous environments. They proposed better techniques for identifying, prioritizing, and scheduling speculative tasks. Tarazu [14] found that remote map tasks on fast machines will greatly increase the network traffic. These traffic may compete with shuffle and result in performance degradation. The authors addressed this issue by performing communication-aware load balancing to avoid bursty network traffic. PIKACHU [15] extended Tarazu by implementing a new key/value partitioning scheme and further improved the performance of Hadoop in heterogeneous clusters. These studies mainly focused on optimizing communications in the reduce phase. In contrast, FlexMap identifies the inefficiencies at the map phase in heterogeneous environments and is complementary to the reduce-based optimizations.

There is also research that mitigates data skewness. SkewReudce [17] alleviated computational skew by balancing data distribution across nodes using a user-define cost function. [20] addresses reduce data skew. SkewTune [16] repartitioned the data of stragglers to take the advantage of idle slots freed by short tasks. However, unlike our approach that provision data to task before the task is dispatched, these approaches require extra I/O operations for data partitioning, which may exacerbate performance interference and increase the network traffic. There are some other work that addressed skewness from different perspectives [21] [22]. Scarlett [21] proposed a framework that replicates blocks based on their popularity. By accurately predicting file popularity, Scarlett can minimize the interference of running jobs co-hosted on the same cluster. [22] tackled the problem of performance prediction with progress indicator of MapReduce application when facing data skewness. [23] tries to optimize the start up overhead for each splits by task sampling. [24] proposes a model to optimize cost and MapReduce performance in heterogeneous cloud environments. [25] proposes a shuffle-on-write service for lower shuffle delay.

Another group of work focused on improving MapReduce performance in the cloud. AROMA [26] is a machine learning based approach to optimize resource allocation for MapReduce jobs in the cloud. FlexSlot [27] mitigated performance

[1]Assume job input size to be 1 and replicas are uniformly distributed on worker nodes. Thus, each node has access to $\frac{3}{12} = 25\%$ of the job input.

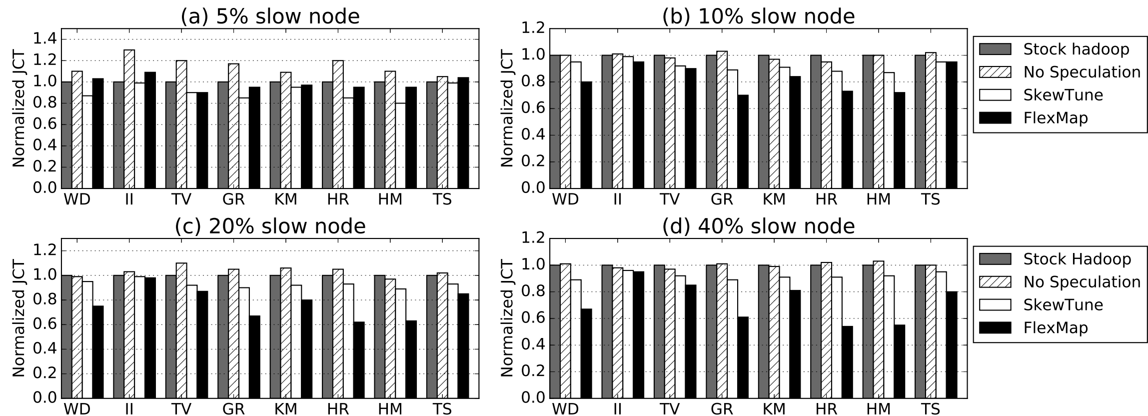

Fig. 8. Normalized job completion time (JCT) in a 40-node multi-tenant cluster with a varying number of slow machines.

interference in the cloud by dynamically changing the number and size of execution slots. Bu et al., proposed interference and locality-aware task scheduling in shared cloud environments [13]. However, these approaches focus on VM management and cannot be easily extended to Big Data frameworks.

VI. Conclusions

Optimizing MapReduce performance in heterogeneous environments has been a challenging problem. In this work, we focus on improving the task execution at the map phase by dynamically provision map tasks to match the distinct machine capacity in a heterogeneous cluster. To this end, we design *FlexMap*, to enable elastic map tasks in MapReduce. FlexMap achieves fine-grained load balancing at the granularity of block unit and avoids high execution overhead using coarse-grained map tasks. Experimental results on three heterogeneous clusters with representative workloads show its effectiveness in reducing job completion time and improving overall efficiency.

VII. Acknowledgement

This research was supported in part by U.S. NSF grants CNS-1422119, CNS-1649502 and IIS-1633753.

References

[1] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *Proc. of ACM Communications*, 2008.
[2] E. Amazon, "Amazon elastic compute cloud (amazon ec2)," 2010.
[3] "Microsofts cloud platform," *http://www.windowsazure.com/en-us/*, 2013.
[4] C. Severance, *Using Google App Engine*. "O'Reilly Media, Inc.", 2009.
[5] V. vCloud Director, "Deliver complete virtual datacenters for consumption in minutes," 2012.
[6] "Hadoop," *http://hadoop. apache. org*, 2009.
[7] K. Ousterhout, A. Panda, J. Rosen, S. Venkataraman, R. Xin, S. Ratnasamy, S. Shenker, and I. Stoica, "The case for tiny tasks in compute clusters." in *Proc. of USENIX HotOS*, 2013.
[8] 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 SoCC*, 2013.
[9] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." in *Proc. of USENIX HOTCLOUD*, 2010.
[10] I. Gog, J. Giceva, M. Schwarzkopf, K. Vaswani, D. Vytiniotis, G. Ramalingam, M. Costa, D. G. Murray, S. Hand, and M. Isard, "Broom: Sweeping out garbage collection from big data systems," in *Proc. of USENIX HotOS*, 2015.
[11] F. Dinu and T. Ng, "Understanding the effects and implications of compute node related failures in hadoop," in *Proc. of ACM HPDC*, 2012.
[12] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments." in *Proc. of USENIX OSDI*, 2008.
[13] X. Bu, J. Rao, and C.-z. Xu, "Interference and locality-aware task scheduling for MapReduce applications in virtual clusters," in *Proc. of ACM HPDC*, 2013.
[14] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. Vijaykumar, "Tarazu: optimizing MapReduce on heterogeneous clusters," in *Proc. of ACM ASPLOS*, 2012.
[15] R. Gandhi, D. Xie, and Y. C. Hu, "Pikachu: How to rebalance load in optimizing MapReduce on heterogeneous clusters." in *Proc. of USENIX ATC*, 2013.
[16] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "Skewtune: mitigating skew in MapReduce applications," in *Proc. of ACM SIGMOD*, 2012.
[17] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "Skew-resistant parallel processing of feature-extracting scientific user-defined functions," in *Proc. of ACM SoCC*, 2010.
[18] F. Ahmad, S. Lee, M. Thottethodi, and T. Vijaykumar, "Puma: Purdue MapReduce benchmarks suite," in *Proc. of Purdue ECE Tech Report*.
[19] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz, "Energy efficiency for large-scale MapReduce workloads with significant interactive analysis," in *Proc. of ACM Eurosys*, 2012.
[20] S. R. Ramakrishnan, G. Swart, and A. Urmanov, "Balancing reducer skew in MapReduce workloads using progressive sampling," in *Proc. of ACM SoCC*, 2012.
[21] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris, "Scarlett: coping with skewed content popularity in MapReduce clusters," in *Proc. of ACM Eurosys*, 2011.
[22] E. Coppa and I. Finocchi, "On data skewness, stragglers, and MapReduce polasticitorogress indicators," in *Proc. of ACM SoCC*, 2015.
[23] R. Vernica, A. Balmin, K. S. Beyer, and V. Ercegovac, "Adaptive MapReduce using situation-aware mappers," in *Proc. of International Conference on Extending Database Technology(EDBT)*, 2012.
[24] Z. Zhang, L. Cherkasova, and B. T. Loo, "Exploiting cloud heterogeneity to optimize performance and cost of MapReduce processing," in *Proc. of ACM SIGMETRICS*, 2015.
[25] Y. Guo, J. Rao, and X. Zhou, "ishuffle: Improving performance with shuffle-on-write." in *Proc. of USENIX ICAC*, 2013.
[26] P. Lama and X. Zhou, "Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud," in *Proc. of IEEE/ACM ICAC*, 2012.
[27] Y. Guo, J. Rao, C. Jiang, and X. Zhou, "Flexslot: moving hadoop into the cloud with flexible slot management," in *Proc. of IEEE SC*, 2014.