Performance Evaluation 79 (2014) 328–344

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

Profiling and evaluating hardware choices for MapReduce environments: An application-aware approach

Abhishek Verma [a,*], Ludmila Cherkasova [b], Roy H. Campbell [c]

[a] Google Inc, 1600 Amphitheatre Pkwy, Mountain View, CA 94043, United States
[b] Hewlett–Packard Laboratories, Palo Alto, CA 94304, United States
[c] Department of Computer Science, University of Illinois at Urbana–Champaign, IL 61801, United States

ARTICLE INFO

Article history:
Available online 10 July 2014

Keywords:
MapReduce
Benchmarking
Performance modeling

ABSTRACT

The core business of many companies depends on the timely analysis of large quantities of new data. MapReduce clusters that routinely process petabytes of data represent a new entity in the evolving landscape of clouds and data centers. During the lifetime of a data center, old hardware needs to be eventually replaced by new hardware. The hardware selection process needs to be driven by performance objectives of the existing production workloads. In this work, we present a general framework, called ARIEL, that automates system administrators' efforts for evaluating different hardware choices and predicting completion times of MapReduce applications for their migration to a Hadoop cluster based on the new hardware. The proposed framework consists of two key components: (i) *a set of microbenchmarks* to profile the MapReduce processing pipeline on a given platform, and (ii) *a regression-based model* that establishes a performance relationship between the source and target platforms. Benchmarking and model derivation can be done using a small test cluster based on new hardware. However, the designed model can be used for predicting the jobs' completion time on a large Hadoop cluster and be applied for its sizing to achieve desirable service level objectives (SLOs). We validate the effectiveness of the proposed approach using a set of twelve realistic MapReduce applications and three different hardware platforms. The evaluation study justifies our design choices and shows that the derived model accurately predicts performance of the test applications. The predicted completion times of eleven applications (out of twelve) are within 10% of the measured completion times on the target platforms.

1. Introduction

A meaningful analysis of large, ever growing datasets has become an increasing computing challenge. Many companies are exploiting the MapReduce paradigm and its open-source implementation Hadoop as a scalable platform for *Big Data* processing. A large number of MapReduce applications such as personalized advertising, sentiment analysis, spam and fraud detection, require completion time guarantees and often are deadline-driven. Moreover, complex business pipelines that contain these applications have to achieve different performance and service level objectives (SLOs) for producing time relevant results. This diversity creates competing requirements for performance and size of the underlying cluster.

* Corresponding author.
E-mail addresses: verma7@illinois.edu, vermaa@google.com (A. Verma), lucy.cherkasova@hp.com (L. Cherkasova), rhc@illinois.edu (R.H. Campbell).

http://dx.doi.org/10.1016/j.peva.2014.07.020
0166-5316/

While today's servers are more efficient and longer lasting, a typical data center hardware lifecycle is still around 3–5 years, and eventually, there is a need to upgrade the old data center hardware. For system administrators of Hadoop clusters, this decision point represents challenging times and inevitable efforts to analyze and compare different hardware choices as candidates for the upgrade. Currently, most system administrators' efforts for performance evaluation of future hardware and its sizing are manual and guess-based. Often, some test application, e.g., *sort*, is used to compare the performance of different hardware or cluster configurations. However, applying the outcome of such a test comparison to other production applications and predicting their performance on a new platform is difficult. There are efforts to design MapReduce benchmarks as a mix of synthetic jobs which mimic the job profiles mined from production loads, e.g., a family of GridMix benchmarks [1]. These benchmarks offer saturation tools for stressing the customer Hadoop clusters at scale. This approach pursues a different goal and does not have a *scaled down* version for evaluating alternative hardware solutions by running a *smaller* benchmark on a small cluster deployment.

In this work, we introduce a new framework, called ARIEL (Automated Resource Inference, Evaluation, and Learning), that aims to automate the system administrators' efforts on comparing different hardware choices for upgrading existing MapReduce clusters. ARIEL consists of the following key components:

- *A set of parameterizable microbenchmarks* that is used to profile different execution phases of MapReduce processing on a given platform. By executing these diverse benchmarks on a small test cluster, we create a useful training set that (implicitly) reflects the performance of these phases as a function of processed data and hardware involved in MapReduce processing. Moreover, to enhance the microbenchmark generation, we offer an automated procedure for extracting parameter values that reflect properties of a given production workload.
- *A regression-based model* that establishes a relationship between performance of considered execution phases on the source system and their executions on the destination platform. This model enables us to predict the MapReduce jobs performance on a large Hadoop cluster based on the new hardware (without directly executing these jobs), and to perform the SLO-driven sizing of this future Hadoop cluster.

While the main scenario that we consider is the upgrade of the existing MapReduce cluster and sizing the future cluster, the proposed approach and technique can be applied to solve a broader set of related problems:

1. *Choosing hardware for a new MapReduce cluster.* There are many situations when the users are initiating the adoption of *Big Data* technologies. There might be a variety of hardware offerings for the future MapReduce cluster. The proposed approach enables a better understanding of comparative performance of the available hardware options as well as the comparison of their cost/performance ratios.
2. *Comparing configuration options for a MapReduce cluster*. There is a variety of different configuration choices at the system level and for setting the Hadoop parameters. While there are many "rules of thumb" for navigating some of these choices, the proposed framework provides a general way of comprehensive comparison and understanding the impact of such choices. For example, one might evaluate the impact of additional hard disks on the performance of MapReduce processing pipeline.
3. *Evaluating the cloud offerings for a MapReduce cluster deployment.* The proposed framework offers a unified approach for a detailed performance comparison of MapReduce processing phases not only of available options of the same cloud provider but also across different cloud providers. The derived models help to predict the performance of given MapReduce applications on the target platforms and enable effective cluster sizing via simulation and replay of production workloads on the future Hadoop cluster. This helps in an accurate cost/performance comparison of different cloud options.

Our approach aims to eliminate error-prone manual processes and presents a fully automated solution. We validate our approach and its effectiveness using a set of twelve realistic applications executed on three different hardware platforms. We justify the selection of profiling phases and the benchmark generation process by presenting the impact of these choices on the accuracy of the derived model. Our evaluation shows that the automated model generation procedure effectively characterizes different execution steps of MapReduce processing on three diverse hardware platforms. The predicted completion times of eleven applications (out of twelve) are within 10% of the measured completion times of the applications executed on the new platform.

The rest of this paper is organized as follows. Section 2 provides background on MapReduce processing. Section 3 discusses the problem definition. Section 4 explains our profiling approach, microbenchmarks, and objectives for their selection. Section 5 introduces the automated model generation for performance comparison of MapReduce processing on different hardware platforms. Section 6 evaluates the effectiveness of our model and benchmarks. We discuss the approach limitations and modeling challenges in Section 7. Section 8 presents related work. Section 9 concludes with a summary and future work directions.

2. Background

In the MapReduce model [2], computation is expressed as two functions: map and reduce. The map function takes an input pair and produces a list of intermediate key/value pairs. The reduce function then merges or aggregates all the values associated with the same key. MapReduce jobs are automatically parallelized, distributed, and executed on a large cluster of commodity machines. The map and reduce stages are partitioned into map and reduce tasks respectively. The execution of

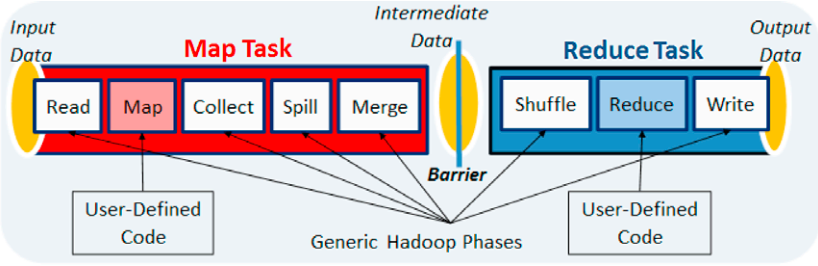

Fig. 1. MapReduce processing pipeline.

each map (reduce) task is comprised of a specific, well-defined sequence of processing phases (see Fig. 1). Only two phases with customized map and reduce functions execute the *user-defined* code. The remaining phases are *generic* (i.e., defined by the Hadoop code), and their performance mostly depends on the amount of data flowing through a phase and the underlying cluster hardware.

The map task processing consists of *read, map, collect, spill* and *merge* phases. Each map task processes a logical split of input data that generally resides on a distributed file system. Files are typically divided into uniform sized blocks (default size is 64 MB, which is a file system parameter) and distributed across the cluster nodes. The map task *read*s the data, applies the user-defined *map* function on each record (it may change the amount of the data flowing through the next phase), and *collect*s the resulting output in memory. If this intermediate data is larger than the in-memory buffer, it is *spill*ed on the local disk of the machine executing the map task and *merge*d into a single file for each reduce task.

The reduce task processing consists of *shuffle, reduce* and *write* phases. In the *shuffle* phase, the reduce tasks fetch the intermediate data files from the already completed map tasks. The intermediate files from all the map tasks are sorted, using an external merge sort if necessary. An external merge sort is used in case the intermediate data does not fit in memory as follows: the intermediate data is shuffled, merged in memory, and written to disk. After all the intermediate data is shuffled, a final pass is made to merge all these sorted files. Thus, shuffling and sorting of intermediate data is interleaved and included in the *shuffle* phase. In the *reduce* phase, the sorted intermediate data (in the form of a key and all its corresponding values) is read (from the local disk in case an external sort was used) and is passed to the user-defined reduce function (it may again change the amount of the data flowing through the next phase). The output data from the reduce function is written back to the distributed file system in the *write* phase.

Job scheduling in Hadoop is performed by a master node, which manages a number of worker nodes in the cluster. Each worker has a fixed number of *map slots* and *reduce slots*, which can run tasks. The number of map and reduce slots is statically configured (typically, one or two per core). The worker nodes periodically send heartbeats to the master to report the number of free slots and the progress of currently running through different task counters. Based on the availability of free slots and the scheduling policy, the master assigns map and reduce tasks to slots in the cluster.

An accompanying distributed file system like GFS [3] makes the input and output data management scalable and fault tolerant through replication.

3. Problem definition and our approach

Migrating an existing MapReduce cluster and its applications to a new hardware is a challenging task with many performance questions to answer prior to the event. First of all, given a set of different hardware choices for a new MapReduce cluster, the system administrators need to evaluate and compare the performance of the upgrade candidates. However, predicting performance of Hadoop clusters comprised of new hardware for SLO-driven execution of production workloads is a difficult and challenging task even for a very experienced system administrator.

In this work, we discuss a general way of benchmarking and comparing the performance of MapReduce processing pipelines of different Hadoop clusters. One of the end goals of this exercise is to be able to predict the performance of existing production jobs on the future, upgraded cluster. The performance profile of a MapReduce job can be characterized using durations (execution times) of its map and reduce tasks.[1]

Therefore, given a job execution trace collected on the original, *old* Hadoop cluster, our goal is to produce its *scaled* version that reflects the job execution on the *new*, upgraded Hadoop cluster as shown in Fig. 2.

Given such a trace transformation, we can predict the performance of production jobs on the new hardware and solve the sizing problem of a new cluster. Moreover, by having a collection of job traces that represent the production job executions on the new platform, many related workload management and job scheduling questions can be answered via pro-active simulations and informed decision making.

[1] The corresponding job trace can be extracted from the Hadoop job tracker logs using tools such as Rumen [4].

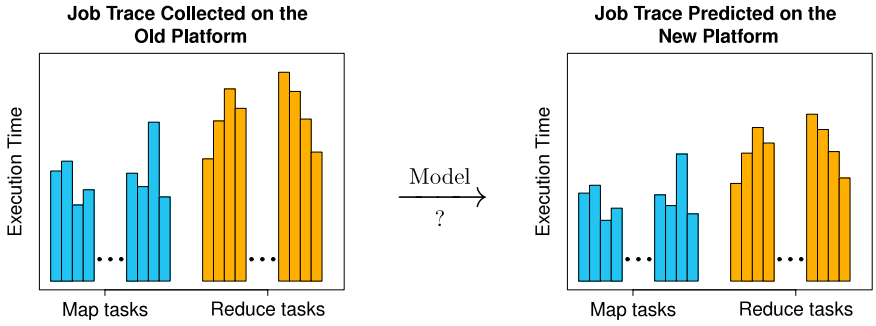

Fig. 2. Building a model between old and new platforms.

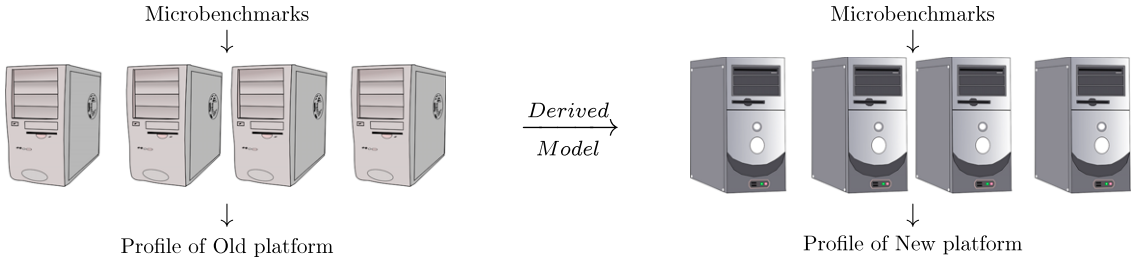

Fig. 3. Automated benchmarking-based model generation.

To accomplish this goal, we design two key components.

- *A set of parameterizable, synthetic microbenchmarks* to profile different execution phases of MapReduce processing on a given platform. These synthetic benchmarks process randomly generated data, and therefore, it is very easy to *scale up* and *down* the amount of data processed by the system as well as to define the amount of data flowing through different phases of the MapReduce processing pipeline. The generic phase performance (i.e., defined by the Hadoop code) depends on the amount of data processed by it as well as the hardware efficiency of the underlying sub-systems that are involved in this phase.
- *An automated model generation system* that establishes a relationship between execution of MapReduce processing phases on the source system and their executions on the target destination platform. This relationship is derived by running a set of microbenchmarks on each platform and building a model that relates the phase execution times of the same map and reduce tasks on both platforms as shown in Fig. 3.

Although the model is created using data from synthetic benchmarks, it offers a general model which can be applied to any MapReduce job trace collected on the old platform to produce a *transformed job trace* on the new platform. Given such a trace transformation, we can predict the performance of production jobs on the new hardware and solve the sizing problem of a new cluster by replaying the job execution traces using MapReduce simulators, e.g., Mumak [5] or SimMR [6].

For benchmarking we deploy both Hadoop clusters with similar configuration parameters. A new platform might have a different number of cores per server. We configure the Hadoop cluster on a new platform with the same number of map and reduce slots per core as on the old cluster. Platform upgrades introduce a more powerful hardware, e.g., a higher amount of memory per core. While the performed experiments in the paper are based on slots configured with the same amount of memory on both platforms, it is not a requirement of our approach. The slots (JVMs) on the new hardware might be allocated a larger amount of memory.

4. Platform profiling with benchmarks

This section introduces the collection of microbenchmarks for profiling and measuring the execution phases of MapReduce processing on a given cluster platform.

4.1. Microbenchmark suite

Our intent is to create a set of parameterizable, synthetic microbenchmarks for generating a diverse variety of data processing patterns found in the production MapReduce workloads. ARIEL supports microbenchmark generation through the following parameters:

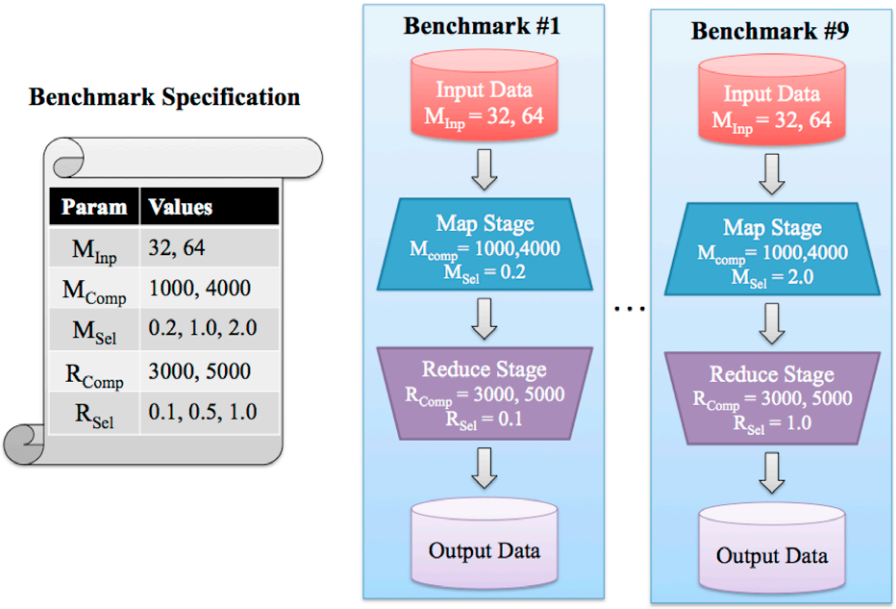

Fig. 4. Nine benchmarks constructed using the Benchmark specification.

1. *Input data size* (M_{inp}): This parameter controls the input read by each map task. We designed our microbenchmark such that each map task reads the entire file content as input. Hence, the input size is not limited to the HDFS block size (64 MB or 128 MB) and can be of arbitrary size. This parameter directly affects the *Read* phase duration.
2. *Map computation* (M_{comp}): We model the CPU computation performed by the user-defined map function by executing a loop which calculates the *n*th Fibonacci number indicated by this parameter.
3. *Map selectivity* (M_{sel}): It defines the ratio of the map output to the map input. This parameter controls the amount of data produced as the output of the map function, and therefore it directly affects the *Collect*, *Spill* and *Merge* phase durations.
4. *Reduce computation* (R_{comp}): Similar to map computation, this parameter defines the reduce function via computing the *n*th Fibonacci number.
5. *Reduce selectivity* (R_{sel}): It defines the ratio of the reduce output to the reduce input. This parameter specifies the amount of output data written back to HDFS and affects the *Write* phase duration.

Thus, the microbenchmark is parameterized as

$$B = (M_{inp}, M_{comp}, M_{sel}, R_{comp}, R_{sel}).$$

The user supplies a benchmark specification which consists of a list of values for all the parameters as shown in Fig. 4.

Each benchmark consists of a specified (fixed) number of map and reduce tasks. For example, we generate benchmarks with 20 map and 20 reduce tasks each for execution in our small cluster deployments with 5 worker nodes (see setup details in Section 6). Increasing the number of tasks improves the accuracy of a constructed model, but requires a longer benchmarking time. The benchmarking engine then generates input data consisting of 100 byte key/value pairs using *TeraGen*, a Hadoop utility for generating synthetic data. The size of the input data for each task is selected in a round robin fashion from the list of input data size values in the benchmark specification. This data is used by each microbenchmark. For every value of M_{sel} and R_{sel}, a new microbenchmark is created and executed. Thus, there are $M_{sel} \times R_{sel}$ benchmarks executed in total. Within each MapReduce microbenchmark, the map tasks round robin through the list of M_{comp} values, and the reduce tasks round robin through the list of R_{comp} values as shown in Fig. 4.

When a set of production jobs for migration is known, we offer an automated procedure for extracting *customized* parameters that reflect properties of a given workload. First, we extract the parameters of each application in the set. Then, these profiles are combined to build the CDF (Cumulative Distribution Function) of all the parameter values exercised by these applications. Fig. 5(a)–(e) show the CDFs of five parameters (*input data sizes*, *map* and *reduce selectivities*, *map* and *reduce computations*) for a set of 12 applications used for a performance evaluation in Section 6. From the CDF plots, we can determine the appropriate values for the microbenchmark parameters by using a clustering algorithm like k-means. Red rhombuses in Fig. 5(a)–(e) show the outcome of the clustering algorithm for $k = 4$. By choosing a smaller or larger number k of clusters in the k-means algorithm, we can influence the number of parameter values (and the number of benchmarks) used in the microbenchmark suite to control the overall benchmarking time.

Fig. 5(f) shows the relationship between the Fibonacci number n (X-axis) and its computation time (Y-axis). This relationship is used for defining durations of map and reduce phases whose CDFs are shown in Fig. 5(d)–(e) respectively.

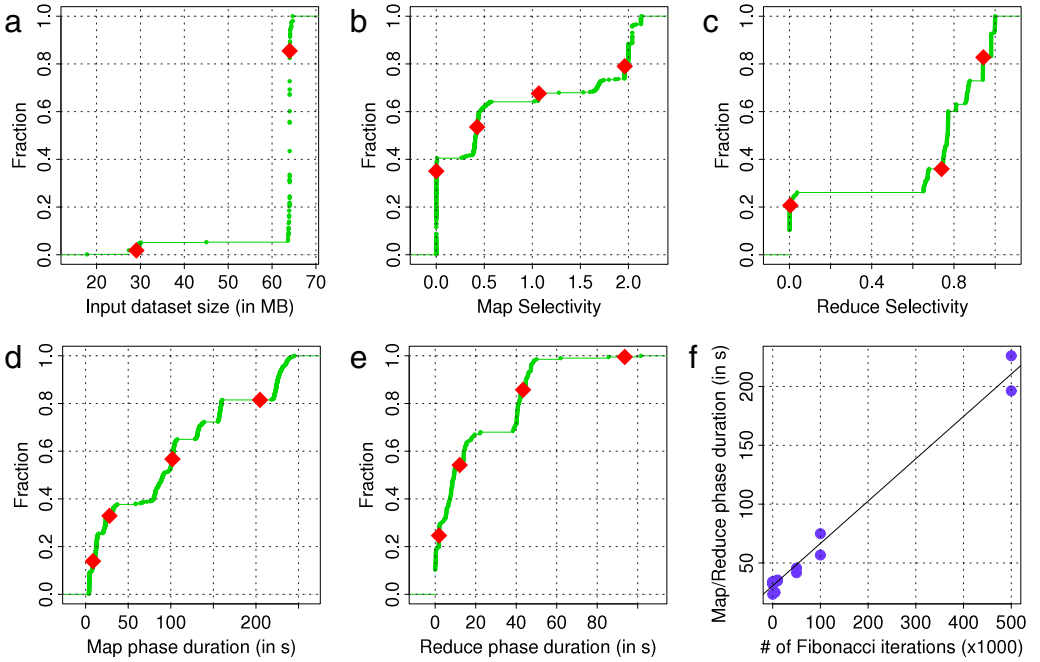

Fig. 5. Different application parameters: input dataset size, map selectivity, reduce selectivity, map phase duration, reduce phase duration, and number of Fibonacci iterations.

Clearly, the proposed approach of representing the arbitrarily complex user-defined map and reduce computation phases via Fibonacci numbers computation is simplistic, and we discuss challenges related to modeling of these phases in Section 7.

4.2. Profiling MapReduce execution phases

The execution of a map (reduce) task is comprised of a specific, well-defined sequence of processing phases (see Fig. 1). Intuitively, the execution of a particular phase depends on the amount of data flowing through this phase and the performance of different hardware sub-parts involved in this step. After the cluster upgrade, when this processing pipeline is executed on a different platform, the performance of underlying hardware might impact the performance of these execution phases in a different way, and therefore, the scaling factors of these phases with respect to the old platform might be different as well. In order to construct an accurate predictive model, we need to profile the performance and durations of these phases separately during the task execution, and then determine their individual scaling functions from the collected measurements.

We measure five phases of the map task and three phases of the reduce task execution as shown in Fig. 1. Map task processing consists of the following phases:

1. *Read*—it measures the time taken to read the map inputs from the distributed file system. A map task typically reads a block (e.g., 64 MB). However, map tasks of many applications read entire files or compressed files of varying sizes. The *read* duration is primarily a function of the disk read throughput.
2. *Map*—it measures the time taken for executing the user supplied map function on the input key–value pair. It depends on the CPU performance.
3. *Collect*—it measures the time taken to buffer map phase outputs into memory. It is a function of the memory bandwidth of the system.
4. *Spill*—it measures the time taken to (locally) sort the intermediate data and partition for the different reduce tasks, applying the combiner if available, and writing the intermediate data to local disk. Multiple system components impact this phase performance.
5. *Merge*—it measures the time taken to merge the different spill files into a single spill file for each reduce task. *Merge* phase depends on the disk read and write throughput.

Reduce task processing consists of the following phases:

1. *Shuffle*—it measures the time taken to transfer intermediate data from map tasks to the reduce tasks and merge-sorting them together. We combine the shuffle and sort phases because in the Hadoop implementation, these two sub-phases are interleaved. The shuffle duration primarily depends on the network performance and disk read/write throughput.

Table 1
Fragment of a platform profile: map and reduce task processing.

Benchmark *ID*	Map task *ID*	*Read* ms	*Map* ms	*Collect* ms	*Spill* ms	*Merge* ms
1	1	1010	220	610	5310	10710
1	2	1120	310	750	5940	11650
…	…	…	…	…	…	…

Benchmark *ID*	Reduce task *ID*	*Shuffle* ms	*Reduce* ms	*Write* ms
1	1	10110	330	2010
1	2	9020	410	1850
…	…	…	…	…

2. *Reduce*—it measures the time taken to apply the user supplied reduce function on the input key and all the values corresponding to it. Like the *map* phase, it depends on the CPU performance.
3. *Write*—it measures the amount of time taken to write the reduce output to the distributed file system. This operation depends on the disk write (and possibly network) throughput.

In the next section, we describe how these measurements are utilized for building the platform profiles.

4.3. Platform profiles

We generate *platform profiles* by running a set of our microbenchmarks on the Hadoop clusters being compared. After the execution of each microbenchmark, we gather durations of the execution phases of all processed map and reduce tasks. A set of these measurements defines the *platform profile* that is used as the training data for the model. We collect durations of *eight* execution phases that reflect essential steps in processing of map and reduce tasks on the given platform:

- *Map task:* in the collected platform profiles, we denote the phase duration measurements for *read, map, collect, spill,* and *merge* as D_1, D_2, D_3, D_4, and D_5 respectively.
- *Reduce task:* in the collected platform profiles, we denote phase duration measurements for *shuffle, reduce,* and *write* as D_6, D_7, and D_8 respectively.

Table 1 shows two fragments of a collected platform profile as a result of the executed benchmarking set.

4.4. Profiling overhead

Job traces must be gathered from the production MapReduce applications running on the source platform. Hence, it is important for our monitoring environment to be lightweight. To obtain phase durations, we design a profiling tool inspired by Starfish [7] based on *BTrace*—a dynamic instrumentation tool for Java [8]. BTrace intercepts class byte codes at run-time based on event–condition–action rules and injects byte codes for the associated actions. Dynamic instrumentation has the appealing property that it has zero overhead when monitoring is turned off. Also, no code modifications to the deployed production framework and applications are needed. We instrument selected Java classes and functions internal to Hadoop using BTrace and measure the time taken for executing different phases.

We also explore an alternative profiling implementation to reduce the profiling overhead. Currently, Hadoop includes several counters such as number of bytes read and written. These counters are sent by the worker nodes to the master periodically with each heartbeat. We modified the Hadoop code by adding counters that measure the durations of *eight phases* to the existing counter reporting mechanism. We evaluate the runtime overhead of different profiling techniques in Section 6.7.

5. Model generation

This section describes how to create a model $M^{ARIEL}_{src \to dst}$ which characterizes the relationship between MapReduce job executions on two different Hadoop clusters, denoted here as *src* and *dst* clusters. To accomplish this goal, we first, find the relationships between durations of different execution phases on the given Hadoop clusters, i.e., we build *eight* submodels $M_1, M_2, \ldots, M_7, M_8$ that define the relationships for *read, map, collect, spill, merge, shuffle, reduce,* and *write* respectively on two given Hadoop clusters. To build these submodels, we use the platform profiles gathered by executing a set of microbenchmarks on the original Hadoop cluster and the target destination platform (see Table 1).

Below, we explain how to build a submodel M_i, where $1 \leq i \leq 8$. By using values from the collected platform profiles, we form a set of equations which express duration of the specific execution phase on the destination platform *dst* as a linear function of the same execution phase on the source platform *src*. Note that both the sides of equations below relate the phase duration of the same task (map or reduce) and of the same microbenchmark on two different platforms (by using the

task and benchmark *ID*s):

$$\begin{aligned} D_{i,dst}^{1,1} &= A_i + B_i * D_{i,src}^{1,1} \\ D_{i,dst}^{1,2} &= A_i + B_i * D_{i,src}^{1,2} \\ \dots \quad &\dots \end{aligned} \tag{1}$$

where $D_{i,src}^{m,n}$ and $D_{i,dst}^{m,n}$ are the values of metric D_i collected on the source and destination platforms for the task with $ID = n$ during the execution of microbenchmark with $ID = m$ respectively.

To solve for (A_i, B_i), one can choose a regression method from a variety of known methods in the literature (a popular method for solving such a set of equations is non-negative Least Squares Regression).

Let $(\hat{A}_i, \hat{B}_i)$ denote a solution for the equation set (1). Then $M_i = (\hat{A}_i, \hat{B}_i)$ is the submodel that describes the relationship between the duration of execution phase i on the source and destination platforms. The entire model $M_{src \to dst}^{ARIEL} = (M_1, M_2, \dots, M_7, M_8)$.

Our training dataset is gathered by an automated benchmark system which runs identical benchmarks on both the source and destination platforms. Because of the non-determinism in MapReduce processing, some unexpected anomalous or background processes can skew the measurements, leading to outliers or incorrect data points. With ordinary least squares regression, even a few bad outliers can significantly impact the model accuracy, because it is based on minimizing the overall absolute error across multiple equations in the set.

To decrease the impact of occasional bad measurements and to improve the overall model accuracy, we employ iteratively re-weighted least squares [9]. This technique is from the Robust Regression family of methods designed to lessen the impact of outliers.

6. Evaluation

In this section, we justify the design choices in ARIEL: Why do we need to distinguish different *phases* for application profiling? What is the impact of varying the different *parameters* for generating the *microbenchmarks*? We evaluate the accuracy of our models using a suite of twelve applications and three hardware platforms.

6.1. Platforms

We consider the following three generations of Proliant servers for a Hadoop cluster (we name them for simplicity):

1. *old:* HP DL145 G3 machines with four AMD 2.39 GHz cores, 8 GB RAM, two 160 GB 7.2K rpm SATA drives, and interconnected with 1 GB Ethernet.
2. *new* 1: HP DL380 G4 machines with eight Intel Xeon 2.8 GHz cores, 6 GB RAM, four 300 GB 10K rpm SATA drives, and interconnected with 1 GB Ethernet.
3. *new* 2: HP DL160 G6 machines with eight Intel Xeon 2.66 GHz cores, 16 GB RAM, two 1 TB 7.2K rpm SATA drives, and interconnected with 1 GB Ethernet.

6.2. Applications

To evaluate the accuracy of ARIEL we use a set of twelve applications made available by the Tarazu project [10] and summarized in Table 2. We present results of running these applications with: (i) *small input datasets* defined by parameters shown in columns 3–4, and (ii) *large input datasets* defined by parameters shown in columns 5–6 respectively. Applications *Sort, SelfJoin*, and *AdjList* process synthetically generated data. Applications *WordCount* to *SeqCount* use the Wikipedia articles dataset as input. Applications *HistMovies* to *KMeans* use the Netflix movie ratings dataset.

6.3. Motivating scenario

Our performance study is motivated by the following typical scenario of a system administrator managing a Hadoop cluster. Our administrator has a 60-node Hadoop cluster based on the *old* hardware (see details in Section 6.1). He has a choice of two new hardware platforms: *new1* and *new2*, and needs to compare these choices. It is not obvious which platform might be a better performing one: from the high-level specifications the *new1* platform has a slightly faster server and faster disks compared to the *new2* platform. Once the new upgrade platform is chosen, the administrator needs to project the expected performance of the 12 applications (the set of critical production jobs) on the new platform for cluster sizing and workload management goals.

The *old* Hadoop cluster is configured with one map and one reduce slot per core, and 700 MB of RAM per slot. For benchmarking, we create two small Hadoop clusters with 5 worker nodes (plus one node for Hadoop management) based on *new1* and *new2* platforms. These clusters are configured similarly, i.e., one map and one reduce slots per core, and 700 MB of RAM per slot.

First, we execute the set of our microbenchmarks on the small 5-nodes clusters of *new1* and *new2* platforms. Then we create the corresponding platform profiles and build a model $M_{new1->new2}^{ARIEL}$. We use this model for comparing the

Table 2
Application characteristics.

Application	*Input data (type)*	*Input* (GB) *small*	*#Map, reduce tasks*	*Input* (GB) *large*	*#Map, Reduce tasks*
Sort	Synthetic	2.8	44, 20	31	495, 240
WordCount	Wikipedia	2.8	44, 20	50	788, 240
Grep	Wikipedia	2.8	44, 1	50	788, 1
InvIndex	Wikipedia	2.8	44, 20	50	788, 240
RankInvIndex	Wikipedia	2.5	40, 20	47	745, 240
TermVector	Wikipedia	2.8	44, 20	50	788, 240
SeqCount	Wikipedia	2.8	44, 20	50	788, 240
SelfJoin	Synthetic	2.1	32, 20	28	448, 240
AdjList	Synthetic	2.4	44, 20	28	508, 240
HistMovies	Netflix	3.5	56, 1	27	428, 1
HistRatings	Netflix	3.5	56, 1	27	428, 1
KMeans	Netflix	3.5	56, 16	27	428, 16

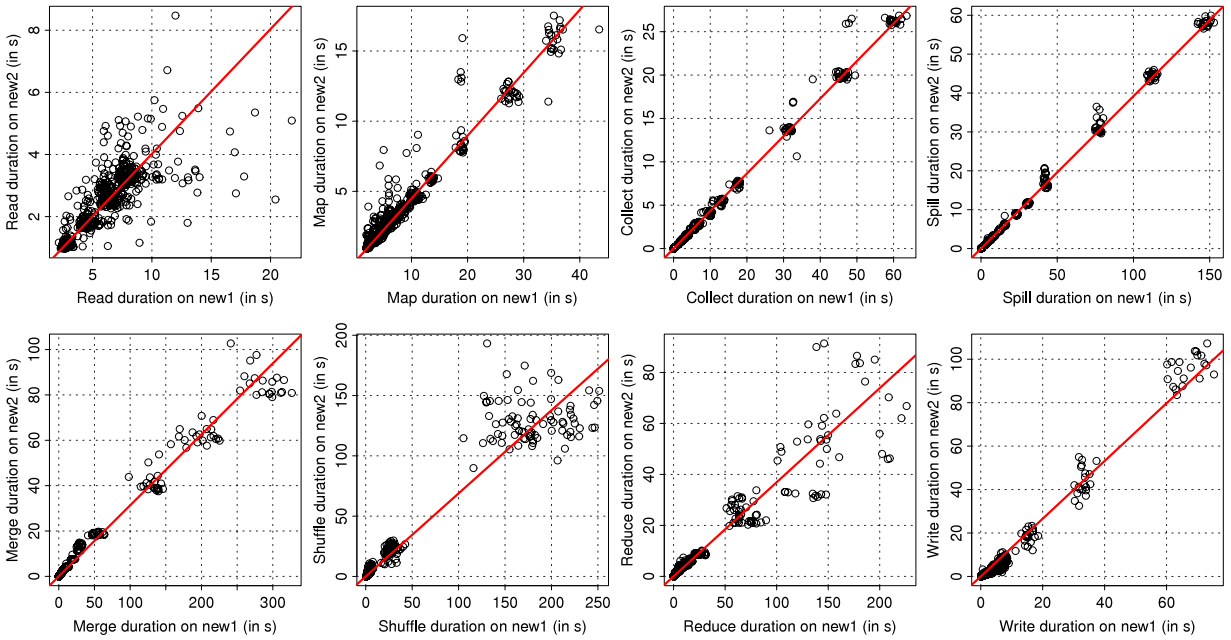

Fig. 6. Different phase durations on *new1* (X-axis) and *new2* (Y-axis) platforms.

performance of the two platforms. Fig. 6 shows the relationship for executing eight phases on the *new1* and *new2* platforms. Each graph has a collection of dots that represent phase measurements of map (reduce) tasks on two given platforms, i.e., for each dot, the X-axis value represents the phase duration on the *new1* platform while the corresponding Y-axis value represents the phase duration on the *new2* platform. The red line on the graph shows the linear regression solution that serves as a model for this phase. As we can see (visually) the linear regression provides a good solution for all the phases. Here is the derived model: $M^{ARIEL}_{new1\rightarrow new2} = (M_1, M_2, \ldots, M_7, M_8) = (0.40, 0.44, 0.43, 0.39, 0.31, 0.68, 0.37, 1.33)$. For simplicity we only show the slope values B_i and omit intercept values A_i (because the intercepts are close to 0).

The model means that if *read* takes 1000 ms on the *new1* platform then *read* execution on the *new2* platform takes on average 400 ms (according to the submodel M_1 value). However, if *write* takes 1000 ms on *new1* platform then *write* execution on the *new2* platform takes on average 1330 ms (according to the submodel M_8 value). This is an interesting observation that stresses the importance of understanding a workload of interest. In our case, *new2* platform outperforms *new1* platform in the map and reduce tasks performance in spite of slower *write*, and represents a more efficient platform.

6.4. Importance of phase modeling

While we can see that different phases on these platforms have different scaling functions, the question still remains whether one can build a simpler model, for example, based on profiling map and reduce task durations instead of creating a detailed 8-phase based profile. To answer this question we collect the platform profiles based on durations of map and reduce tasks, and use linear regression to create the model $M^{task}_{new1\rightarrow new2}$. Fig. 7(a)–(b) show the durations of map and reduce

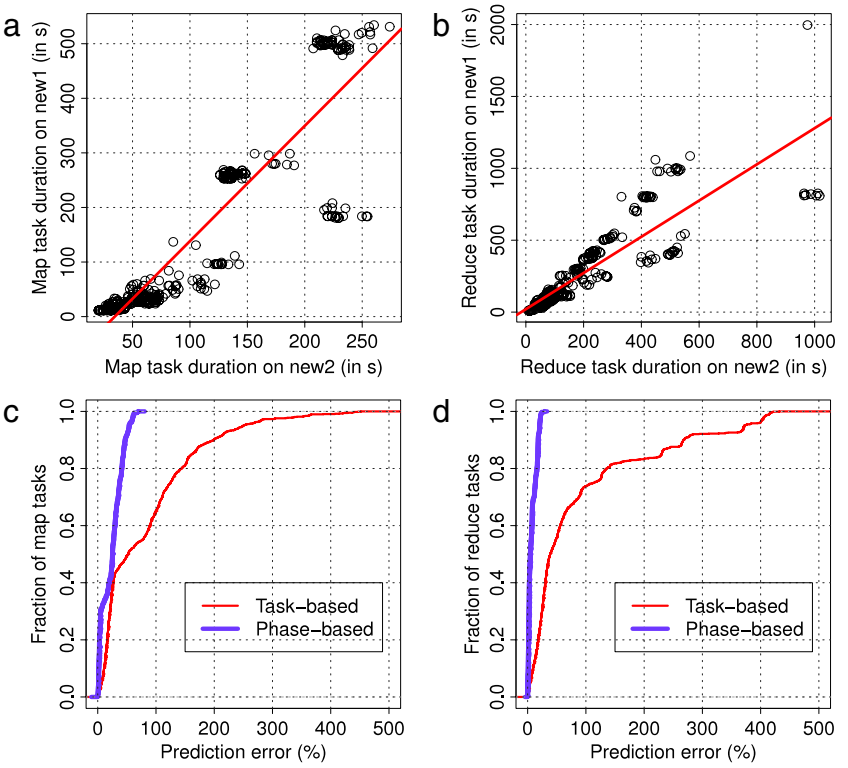

Fig. 7. Do we need phase profiling? (a)–(b) platform profiles based on map and reduce task durations; (c)–(d) CDF of relative errors for predicting the map and reduce task durations using two models: *phase-based* $M^{ARIEL}_{new1\to new2}$ and *task-based* $M^{task}_{new1\to new2}$.

tasks on *new1* and *new2* platforms after executing the set of our microbenchmarks and the model (red line) derived using the linear regression approach. From these figures we can see that the linear regression model does not provide a good fit to the underlying training data.

In order to formally compare the accuracy of generated models $M^{ARIEL}_{new1\to new2}$ and $M^{task}_{new1\to new2}$ we compute per task prediction error. For each task i and its measured duration d_i in our platform profile of the *new1* hardware we compute the task predicted duration $d_i^{predicted}$ on the *new2* platform using the derived model. Then we compare the predicted value against the measured duration $d_i^{measured}$ of the same task i on the *new2* platform. The relative error is defined as follows:

$$error_i = \frac{|d_i^{measured} - d_i^{predicted}|}{d_i^{measured}}.$$

We compute the relative error for all the tasks in the platform profile. Fig. 7(c)–(d) show the CDF of relative errors for map and reduce tasks respectively using two models: *phase-based* $M^{ARIEL}_{new1\to new2}$ (blue curves) and *task-based* $M^{task}_{new1\to new2}$ (red curves). We observe that the accuracy of the *task-based* model is much worse compared to the accuracy of the *phase-based* model. Under the *task-based* model 44% of map tasks and 26% of reduce tasks have the prediction error higher than 100%.

Therefore, deriving the individual submodels for execution phases of the MapReduce processing pipeline significantly improves the quality of the prediction model. It helps to accurately project the execution of a MapReduce application from one platform to a different one and reflect the specifics of this job execution.

Fig. 8 shows the job traces for WordCount application on two different platforms.

Fig. 8(a)–(b) show the durations of map and reduce tasks measured on the *new1* platform, and Fig. 8(c)–(d) show the predicted durations of map and reduce tasks for their execution on the *new2* platform. The "appearance" of measured and predicted job traces closely resemble each other, while the predicted durations of map and reduce tasks have almost two times difference in absolute values. The proposed approach can capture the specifics of the application execution, e.g., due to existing skews in the popularity of the reduce keys as observed in Fig. 8(b) and (d) for different reduce tasks.

By applying the ARIEL model we can scale the original MapReduce job traces from the old platform to the new one. These accurately scaled job traces on the new platform *enable a variety of different models* of the job completion time: *analytic* and *simulation* ones. Fig. 9(a) presents the predicted and measured job completion times of 12 applications introduced in Section 6.2. In these experiments we use applications with *small input datasets* shown in Table 2. First, we run these applications on 5-node Hadoop cluster based on the *new1* platform. Then we apply the derived linear regression model $M^{ARIEL}_{new1\to new2}$ to the collected job traces to transform them into the job traces on the *new2* platform. Finally, we predict

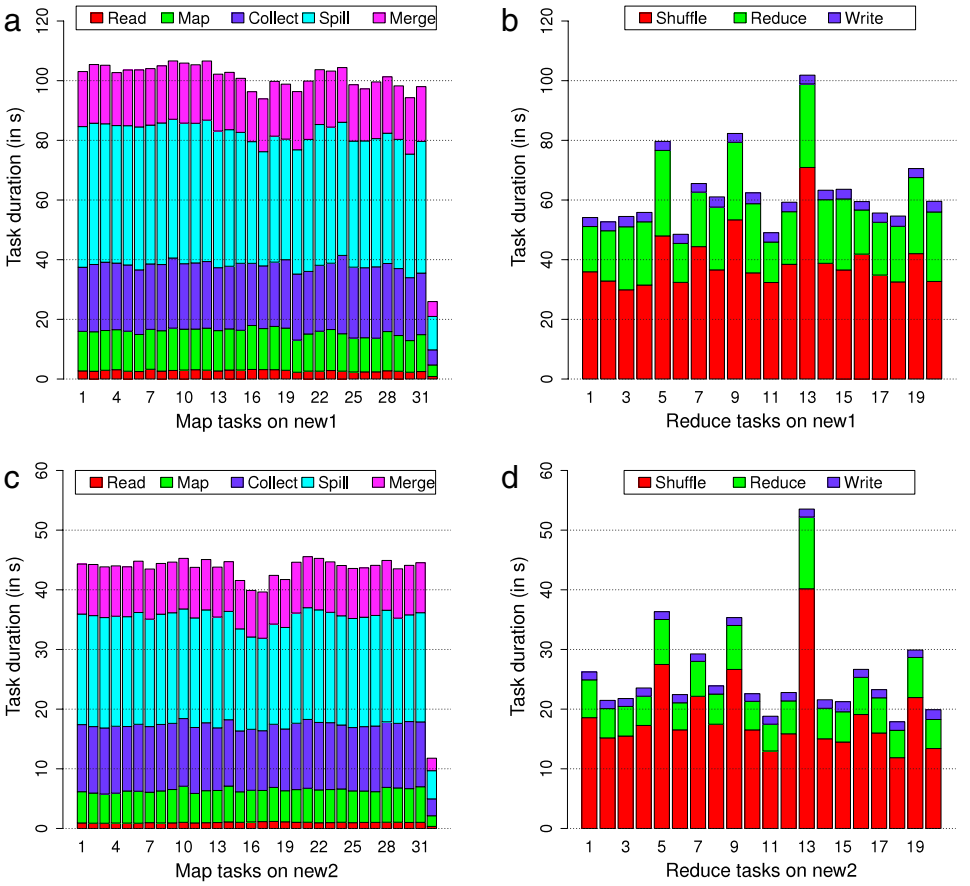

Fig. 8. Job traces (map and reduce tasks durations) of WordCount application: (a)–(b) *new1* platform; and (c)–(d) *new2* platform.

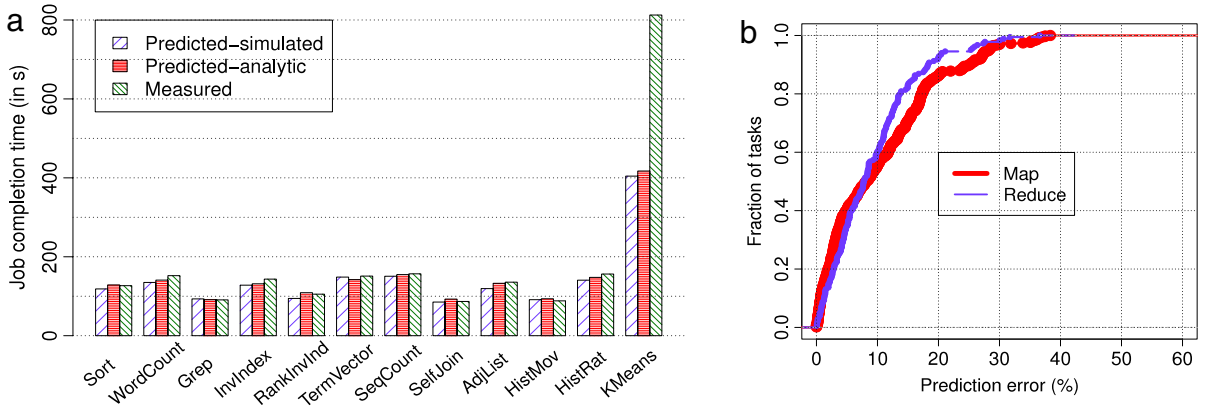

Fig. 9. (a) Predicted vs. measured completion times of 12 applications on 5-node cluster with *new2* hardware; (b) CDF of relative errors for $M^{ARIEL}_{new1 \rightarrow new2}$ model.

completion times of these applications on the 5-node Hadoop cluster with *new2* hardware. For prediction we have applied two different approaches:

- An *analytical model* ARIA [11] that utilizes the knowledge of the average and maximum durations of map and reduce tasks and provides a simple model for predicting the job completion time as a function of allocated resources. Since we have the entire tasks distribution it is easy to compute the average and maximum task durations.

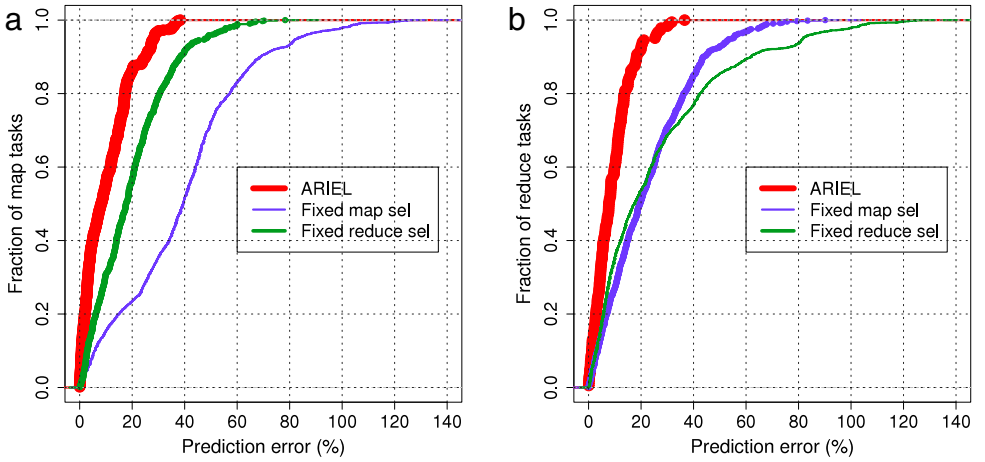

Fig. 10. CDF of relative errors for predicting (a) map and (b) reduce task durations of 12 applications with different models: $M^{ARIEL}_{new1\rightarrow new2}$, $M^{fix\_map\_sel}_{new1\rightarrow new2}$, and $M^{fix\_red\_sel}_{new1\rightarrow new2}$.

- A *MapReduce simulator* SimMR [2] [6] that supports a job trace replay on a Hadoop cluster as a way to more accurately predict the job completion time (especially under the different Hadoop schedulers).

Fig. 9(a) shows that errors between the predicted and measured job completion times are on average 6% (and within 10%) for 11 applications out of 12. The prediction error for *KMeans* is almost 50%. This application has complex, compute-intensive map and reduce functions that are difficult to predict accurately with a black-box approach (see an additional discussion in Section 7). However, the majority of MapReduce applications have relatively simple data manipulations performed by their map and reduce functions, and these applications can be accurately modeled by the "black-box" approach proposed in the paper.

Fig. 9(b) presents a different perspective on the accuracy of the designed linear-regression model. It shows the CDF of relative errors in predicting map and reduce task durations of 12 applications (combined) on the *new2* platform. For 80% of the map and reduce tasks the relative error is less than 18%. We observe that the overall job completion time prediction is quite accurate (less than 10% error for 11 applications) in spite of less accurate prediction of the task durations. During multiple waves of task executions their combined behavior is closer to the average one.

6.5. Benchmark coverage

In these experiments, we examine the importance of introduced parameters for the microbenchmark generation. We aim to answer the question whether one can choose a smaller set of parameters and execute a much smaller subset of microbenchmarks for building a good model. For example, how important is varying the map or reduce selectivities for creating the representative benchmarking set and generating an accurate model[3]? To answer this question we also fix the input data size since having a diverse input size for different map tasks partially creates the effect of different map and reduce selectivities in the MapReduce jobs. We create two models $M^{fix\_map\_sel}_{new1\rightarrow new2}$ and $M^{fix\_red\_sel}_{new1\rightarrow new2}$ using two sets of microbenchmarks generated by the following specifications respectively:

- Fixed input dataset size per map task and *fixed map selectivity* while varying other parameters;
- Fixed input dataset size per map task and *fixed reduce selectivity* while varying other parameters.

We apply these models to predict map and reduce task durations of 12 applications on the *new1* platform. In order to formally compare the accuracy of these two models and the model $M^{ARIEL}_{new1\rightarrow new2}$ that is generated using the set of all the benchmarks (i.e., while varying all the parameters) we compute the CDF of per task relative prediction errors (as defined in Section 6.4). Fig. 10 summarizes the outcome of these experiments:

Fig. 10(a)–(b) show the CDF of errors for the three models when predicting the duration of map and reduce tasks respectively. We observe that the accuracy of the models generated by the benchmarks with **fixed** map or reduce selectivity

[2] This simulator can accurately replay the job traces and reproduce the original job processing: the completion times of the simulated jobs are within 5% of the original ones as shown in [6]. Moreover, SimMR is a very fast simulator: it can process over one million events per second.

[3] It is important to accurately answer this questions because Herodotou et al. [12] are using a small set of six benchmarks in order to derive a *relative model* for Hadoop clusters comprised of different Amazon EC2 instances. Their benchmarks exercise job processing with data compression and combiner turned on and off. One explanation why this limited benchmarking set had worked well in their study is that many EC2 instances are deployed on the same hardware but with different amount of CPU and memory per VM instance. However, we are questioning whether their approach may work in a general case.

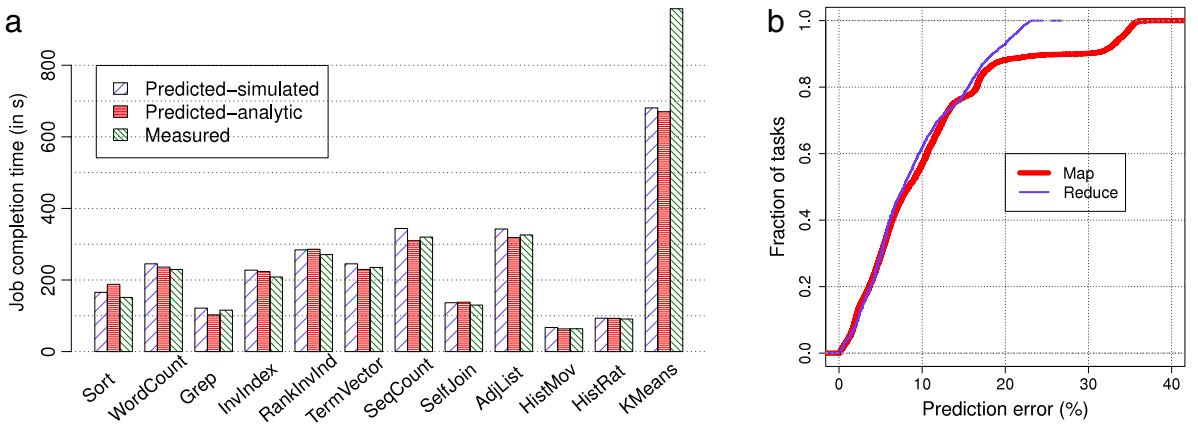

Fig. 11. (a) Predicted vs. measured completion times of 12 applications on 60-node cluster with *new2* hardware; (b) CDF of relative errors for $M^{ARIEL}_{old\rightarrow new2}$ model on the 60-node *new2*.

is significantly worse compared to the accuracy of the model built using all the benchmarks (where we used **varying** set of values for defining map and reduce selectivities). For example, the model obtained using benchmarks with *fixed* map selectivity 50% of map tasks have the prediction error higher than 40%.

6.6. Model accuracy

In this section, we follow the motivating scenario described in Section 6.3. After comparing two hardware platforms *new1* and *new2*, the system administrator selects a better performing *new2* platform. Now, the next task is to project the expected performance of 12 applications (the set of critical production jobs) on the new platform for cluster sizing and workload management goals. According to our approach we need to benchmark both platforms and then build a model using the platform profiles. Since we have already benchmarked the *new2* hardware and have built its platform profile we only need to execute the suite of microbenchmarks on the original *old* platform and extract its platform profile. Here is the derived model:$M^{ARIEL}_{old\rightarrow new2} = (0.21, 0.17, 0.37, 0.6, 0.46, 0.86, 0.219, 0.32)$. For simplicity we show the slope values B_i for each submodel and omit the intercept values A_i (since they are close to 0). For example, it means that if *read* takes 1000 ms on the *old* platform then its execution on the *new2* platform takes on average only 200 ms. The execution phases are much more efficient of the new platform. The *shuffle* phase has somewhat similar durations on both platforms because of a similar networking infrastructure in both clusters.

To evaluate the model accuracy, we perform a set of experiments with 12 applications (to process *large* datasets as shown in Table 2) on the *large* Hadoop clusters (60 worker-nodes) based on the *old* and *new2* platforms.

Fig. 11(a) shows that the error between predicted and measured job completion times are on average 6% (and within 10%) of the measured completion times for 11 applications out of 12. The prediction error for *KMeans* is still high (around 30%). In Section 7 we discuss a way to assess a possible prediction inaccuracy due to map and reduce function executions.

Fig. 11(b) shows the overall CDF of relative errors in predicting map and reduce task durations of 12 applications (combined) on the *new2* platform. For 88% of the tasks the relative error is less than 20%.

For 11 applications (out of 12), we observe that the model derived by benchmarking a *small* 5-node test cluster enables an accurate prediction of job completion times in the *large* 60-node production cluster.

6.7. Profiling overhead

In order to evaluate the profiling overhead incurred by different profiling techniques, we execute our set of twelve applications on the *new2* platform without any profiling, with counter-based profiling, and with BTrace-based profiling. Fig. 12 shows the job completion time for the different profilers. We observe that counter-based profiling adds an average of 8% (and up to 13%) overhead. BTrace-based profiling has an average of 10% (and up to 28%) overhead. The compute intensive *KMeans* application has the highest overhead, since the map and reduce functions compete with the profiler for CPU resources. However, both profilers have a modest overhead even when they are profiling each task execution.

7. Discussion

In this section, we discuss some limitations of the proposed approach and provide comments on how to improve the modeling accuracy with ARIEL.

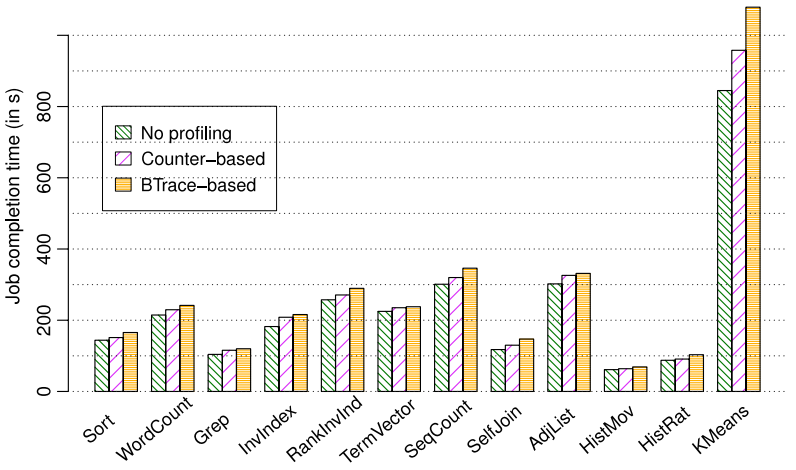

Fig. 12. Job completion time with different profilers.

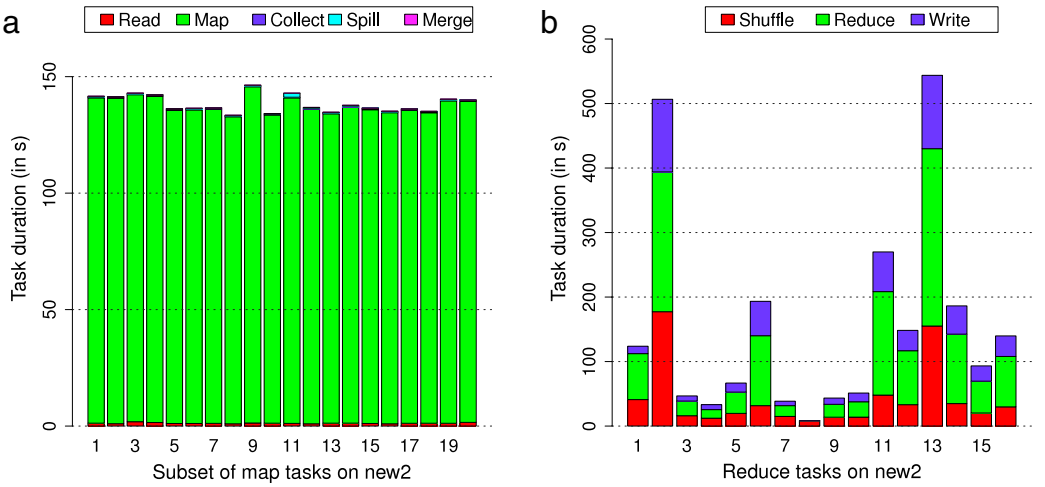

Fig. 13. Job trace of KMeans on *new2* platform.

Modeling compute-intensive map and reduce functions: Some MapReduce programs, e.g., *KMeans*, involve complex, compute-intensive map/reduce functions. Fig. 13 shows that 94% of the map task duration and 30% of the reduce task duration are due to *map* and *reduce* phase processing respectively.

Our simple approximation of computing power of different platforms with Fibonacci number iterations cannot provide an accurate scaling function for complex user-defined computations. To provide feedback to the user about possible prediction inaccuracy, we can perform a quick application analysis by inspecting the fraction of the map and reduce phase durations in the entire task duration.

We can easily determine such applications and provide a "warning" to the user by inspecting the proportion of the map and reduce phase durations as compared to the total task duration. It is difficult to predict the execution time of an arbitrary user-defined custom function on a different hardware without running some samples of this code on both hardware (possibly on a smaller random sample of input data as it is proposed in [13]). Predicting the performance of a general program on a different hardware remains an open problem over last two decades.

First shuffle phase measurements and modeling: The shuffle phase for a "first wave" of reduce tasks overlaps with the map stage execution. We model the *first shuffle* separately by including the non-overlapping portion of the execution time (similar to the approach as in ARIA [11]) and building a separate linear model for this phase.

Scaling network resources: The set of twelve applications can be executed in 115 min using the FIFO scheduler on the 60-node *old* cluster. The following table shows the number of *new2* platform machines required to satisfy a given SLO. These estimates are obtained with our MapReduce simulator [ ].[4] It shows that to achieve the same completion time on a *new2*

[4] The reference is removed due to the double blind review requirements.

platform, we need a 16-node cluster.

SLO (mins)	120	115	60	45
# machines	15	16	30	64

However, as a larger number of machines have to process the same amount of data in a shorter time, the network should be capable of supporting a higher bisection bandwidth as compared to the *old* platform. Analysis of required network resources can be performed using simulators like ns-2. In our future work, we plan to design network microbenchmarks that transfer varying amounts of intermediate data from memory of the machines processing map tasks to memory of the machines processing the reduce tasks (without any disk accesses) and thus accurately measure the shuffle performance.

8. Related work

The problem of predicting the application performance on a new or different hardware has fascinated researchers and been an open challenge for a long time. The body of work [14,15] goes back almost two decades. In 1995, Larry McVoy and Carl Staelin introduced the *lmbench* [14]—a suite of operating system microbenchmarks that provides an extensible set of portable programs for system profiling and the use in cross-platform comparisons. Each microbenchmark captures some unique performance properties and features that were present in popular and important applications of that time. Although such microbenchmarks can be useful in understanding the end-to-end behavior of a system, the results of these microbenchmarks provide little information to indicate how well a particular application will perform on a particular system.

Seltzer et al. [15] in their position paper argue for an application-specific approach to benchmarking. The authors suggest a vector-based approach for characterizing an underlying system by a set of microbenchmarks (e.g., *lmbench*) that describes the behavior of the fundamental primitives of the system. The results of these microbenchmarks constitute the *system* vector and it characterizes the system performance. They suggest to construct an *application* vector that quantifies the way the application makes use of the various primitives supported by the system. The product of these two vectors yields a relevant performance metric. Constructing the *application* vector for a general application is the most difficult and questionable part of the proposed approach, and it was only demonstrated using simple examples.

We adopt a different approach in our design. We use a set of specially generated microbenchmarks to characterize the relative performance of the processing pipelines of two underlying Hadoop clusters: *old* and *new* ones. Then we apply the derived model to the application traces and use these scaled traces for predicting the application performance on a *new* Hadoop cluster. In our work, we derive a *relative model* that predicts performance differences between a pair of given Hadoop clusters. This contrasts with an *absolute model* that is applied directly to the analyzed workload or application for predicting its performance.

Recently, several *absolute models* have been designed for predicting the performance of MapReduce applications [12,7,16,11]. Our earlier work ARIA [11] proposes a framework that automatically extracts compact job profiles from the past application run(s). These job profiles form the basis of a *MapReduce analytic performance model* that computes the lower and upper bounds on the job completion time. ARIA provides a fast and efficient capacity planning model for a MapReduce job with timing requirements.

SimMR [6] additionally offers an accurate and fast MapReduce simulator that can replay the collected or synthetic job traces under different Hadoop schedulers and workload management strategies.

Tian and Chen [16] propose an approach to predict MapReduce program performance from a set of test runs on small input datasets and small number of nodes. By executing 25–60 diverse test runs the authors create a training set for building a regression-based model of a given application. The derived model is able to predict the application performance on a larger input and a different size Hadoop cluster.

There were a few efforts on developing simulation tools for MapReduce environments [17,5,6] that can replay the collected or synthetic job traces under different Hadoop schedulers and workload management strategies.

Starfish [7] applies dynamic Java instrumentation to collect a run-time monitoring information about job execution by extracting a diverse variety of metrics. Such a detailed job profiling enables the authors to predict job execution under different Hadoop configuration parameters, and automatically derive an optimized configuration. However, collecting a large set of metrics comes at a cost, and to avoid significant overhead profiling should be applied to a small fraction of tasks. The authors introduce analytic and simulation models for predicting the job completion time and addressing a cluster sizing problem [12].

Prior examples of successfully building relative models include a *relative fitness model* for storage devices [18] using CART models, and a *relative model* between the native and virtualized systems [19] based on a linear-regression technique. The main challenge outlined in both works [18,19] is the tailored benchmark design and the benchmark coverage. Both of these challenges are non-trivial: if a benchmark collection used for system profiling is not representative or complete to reflect important workload properties then the created model might be inaccurate. Herodotou et al. [12] attempt to derive a *relative model* for Hadoop clusters comprised of different Amazon EC2 instances. They use the Starfish profiling technique and a small set of six benchmarks to exercise job processing with data compression and combiner turned on and off. The model is generated with the M5 Tree Model approach [20]. The authors report that the predicted makespan is within 15%

of the measured one for the *combined* execution of six applications. One explanation why this limited benchmarking set might worked in their study is that many EC2 instances are deployed on the same hardware but with different amount of CPU and memory per VM instance. We argue that a model derived using six benchmarks with fixed parameters cannot accurately predict performance of general MapReduce applications on diverse hardware. In our evaluation study (see results and a discussion in Section 6.5), we observe that fixed parameters of map and reduce selectivities in benchmarks lead to inaccurate results. The presented performance analysis justifies ARIEL's decision choices for building the microbenchmarks suite and our relative model. ARIEL is validated on a broad application set and diverse hardware platforms.

9. Conclusion

Our work is motivated by the challenges related to MapReduce cluster hardware upgrades. Many existing Hadoop clusters are empirically tuned for optimized performance on production workloads. Often, the jobs are partitioned into different classes of service (e.g., platinum, silver, and bronze at Facebook [21]) and then processed by different Hadoop clusters with specially created management and resource allocation strategies to guarantee performance isolation and predictable completion time.

When the time for hardware upgrades comes, system administrators have a long list of performance questions to answer. To assist system administrators and automate their efforts in solving these problems, we introduced ARIEL—a novel framework that enables performance assessment of different hardware choices as candidates for a reliable upgrade of existing MapReduce clusters. The proposed approach might be useful to hardware and service providers. "Shopping" for a Hadoop cluster will be easier if providers can offer an additional performance comparison of different hardware choices.

Acknowledgments

This material is based on research sponsored by the Air Force Research Laboratory and the Air Force Office of Scientific Research, under agreement number FA8750-11-2-0084. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. This research was performed during A. Verma's internship at HP Labs and while completing his Ph.D. dissertation at the University of Illinois. A. Verma is currently working at Google Inc.

References

[1] Apache GridMix, https://issues.apache.org/jira/browse/MAPREDUCE/component/12313086.
[2] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM.
[3] S. Ghemawat, H. Gobioff, S. Leung, The Google file system, ACM SIGOPS Oper. Syst. Rev. 37 (5) (2003) 43.
[4] Apache Rumen: a tool to extract job characterization data from job tracker logs, https://issues.apache.org/jira/browse/MAPREDUCE/component/12313617.
[5] Apache Mumak: Map-Reduce simulator, https://issues.apache.org/jira/browse/MAPREDUCE-728.
[6] A. Verma, L. Cherkasova, R. Campbell, Play It Again, SimMR!, in: IEEE Conf. on Cluster Computing, 2011.
[7] H. Herodotou, et al. Starfish: a self-tuning system for big data analytics, in: Proc. of CIDR, 2011.
[8] BTrace: A Dynamic Instrumentation Tool for Java, http://kenai.com/projects/btrace.
[9] P. Holland, R. Welsch, Robust regression using iteratively reweighted least-squares, Commun. Stat.- Theory Methods 6 (9) (1977) 813–827.
[10] F. Ahmad, et al. Tarazu: optimizing MapReduce on heterogeneous clusters, in: Proc. of ASPLOS, 2012.
[11] A. Verma, L. Cherkasova, R. Campbell, ARIA: automatic resource inference and allocation for mapreduce environments, in: Proc. of IEEE/ACM Intl. Conference on Autonomic Computing (ICAC), 2011.
[12] H. Herodotou, F. Dong, S. Babu, No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics, in: Proc. of ACM Symposium on Cloud Computing, 2011.
[13] A. Verma, L. Cherkasova, R.H. Campbell, Resource Provisioning Framework for MapReduce Jobs with Performance Goals, in: Proc. of the 12th ACM/IFIP/USENIX Middleware Conference.
[14] L. McVoy, C. Staelin, lmbench: portable tools for performance analysis, in: Proc. of USENIX ATC, 1996.
[15] M. Seltzer, et al. The case for application-specific benchmarking, in: Proc. of HotOS, 1999.
[16] F. Tian, K. Chen, Towards optimal resource provisioning for running MapReduce programs in public clouds, in: Proc. of IEEE Intl. Conference on Cloud Computing, 2011.
[17] G. Wang, A. Butt, P. Pandey, K. Gupta, A simulation approach to evaluating design decisions in mapreduce setups, in: Intl. Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, 2009.
[18] M. Mesnier, et al. Modeling the relative fitness of storage, in: Proc. of ACM SIGMETRICS, 2007.
[19] T. Wood, et al. Profiling and modeling resource usage of virtualized applications, in: Proc. of ACM/IFIP/USENIX Middleware, 2008.
[20] J. Quinlan, Learning with continuous classes, in: Proc. of Australian joint Conference on Artificial Intelligence, 1992.
[21] A. Thusoo, et al., Data warehousing and analytics infrastructure at Facebook, in: Proc. of SIGMOD, ACM, 2010.

Abhishek Verma is currently working on the scheduling infrastructure at Google. He received his Ph.D. from the Computer Science department at the University of Illinois at Urbana–Champaign in 2012 where his research focused on performance modeling of MapReduce environments. He received his MS in Computer Science from the same university in 2010. Prior to that, he earned his B.Tech. in Computer Science and Engineering from NIT Trichy, India in 2008.

Ludmila Cherkasova is a principal scientist at HP Labs, Palo Alto, USA. Her current research interests are in developing quantitative methods for the analysis, design, and management of distributed systems (such as emerging systems for Big Data processing, internet and enterprise applications, virtualized environments, and next generation data centers). She is the ACM Distinguished Scientist and is recognized by the Certificate of Appreciation from the IEEE Computer Society.

Roy H. Campbell is the Sohaib and Sara Abbasi Professor of Computer Science at the University of Illinois at Urbana–Champaign, where he leads the Systems Research Group. He received his Ph.D. in Computer Science from University of Newcastle upon Tyne in 1977. His current research projects include assured cloud computing, security assessment of SCADA networks, operating system dependability and security, and active spaces for ubiquitous computing.