Performance Modeling of MapReduce Jobs in Heterogeneous Cloud Environments

Zhuoyao Zhang
University of Pennsylvania
zhuoyao@seas.upenn.edu

Ludmila Cherkasova
Hewlett-Packard Labs
lucy.cherkasova@hp.com

Boon Thau Loo
University of Pennsylvania
boonloo@seas.upenn.edu

***Abstract*—Many companies start using Hadoop for advanced data analytics over large datasets. While a traditional Hadoop cluster deployment assumes a homogeneous cluster, many enterprise clusters are grown incrementally over time, and might have a variety of different servers in the cluster. The nodes' heterogeneity represents an additional challenge for efficient cluster and job management. Due to resource heterogeneity, it is often unclear which resources introduce inefficiency and bottlenecks, and how such a Hadoop cluster should be configured and optimized. In this work[1], we explore the efficiency and performance accuracy of the *bounds-based* performance model for predicting the MapReduce job completion times in heterogeneous Hadoop clusters. We validate the accuracy of the proposed performance model using a diverse set of 13 realistic applications and two different heterogeneous clusters. Since one of the Hadoop clusters is formed by different capacity VM instances in Amazon EC2 environment, we additionally explore and discuss factors that impact the MapReduce job performance in the Cloud.**

***Keywords*-MapReduce, heterogeneous clusters, performance modeling, efficiency.**

I. Introduction

In recent years, there has been an emergence of several data-parallel middleware platforms primarily targeted towards large-scale data processing in the cloud environment, for example, MapReduce [3] and its popular open-source Hadoop implementation. These middleware platforms provide simple yet powerful and extremely versatile programming models, and can be applied to a wide spectrum of domains where large-scale data analytics is performed.

To better manage cloud resources, there has been a series of efforts aimed at developing performance models [4], [5], [6], [7], [9], [10], [11], [16] for MapReduce applications. These performance models are used for predicting the job completion times resources. They are also used as a basis for the resource provisioning problem: and estimating the amount of resources required to complete applications with given deadlines. One *major shortcoming* of the existing models is that they do not consider the *resource heterogeneity* which causes inaccurate prediction on heterogeneous clusters. We argue that cluster deployments are grown incrementally over time. It is not unusual for companies to start off with an initial cluster, and then gradually add more compute and I/O resources as the number of users increases. More often than not, it is economical to add newer more powerful servers rather than discard the old hardware. So, *the problem is* to design and offer the efficient performance models that work well in heterogeneous environments.

Performance models that rely on extensive fine-granularity profiling techniques [4], [5] or complex analysis using machine learning [9], are unlikely to work well for clusters with high degrees of heterogeneity due to the complexity and execution variability in the heterogenous environment. In this paper, we explore the applicability and performance accuracy of an alternative *bounds-based* approach introduced in the ARIA project [10] for *homogeneous* Hadoop clusters. The model utilizes a few essential job characteristics such as the *average* and *maximum* phase execution times (e.g., map, shuffle, and reduce phases) during the task processing. We argue that this simple model does work well for heterogeneous environments. Intuitively, in a heterogeneous Hadoop cluster, slower nodes result in longer task executions. These measurements then are reflected in the calculated average and maximum task durations that comprise the job profile. While the bounds-based performance model does not explicitly consider different types of nodes, their performance is implicitly reflected in the job profile and therefore is effectively incorporated in *predicting the job completion times*.

We validate the accuracy of the proposed bounds-based performance model using a diverse set of realistic applications and two different testbeds: *i)* an Amazon EC2 heterogeneous cluster formed by different capacity VM instances, and *ii)* the UPenn Hadoop cluster with three different types of nodes. The predicted completion times are within 10% of the measured ones for 12 (out of 13) applications in the workload set.

While evaluating the bounds-based performance model, we make a few interesting observations on the behavior of heterogeneous Hadoop cluster where machines vary in capabilities:

- The number of map and reduce slots per node should be configured dynamically according to the nodes' capacity. In a cluster with a high degrees of heterogeneity (e.g., Amazon EC2's *small, medium*, and *large* VM instances), an efficient use of the cluster requires us to vary the number of map and reduce slots per node based on its compute capacity. However, in the UPenn cluster the nodes differ slightly in processor speeds. Therefore, we use a fixed number of map and reduce slots on each node.
- A simple "rule-of-thumb" [12] states that the completion time of a MapReduce job is decreased twice when it is processed by two times larger Hadoop cluster. We compare the completion times of 13 applications in our experimental set when they are processed on the EC2-based Hadoop clusters formed by *small* versus *large* VM instances respectively. The CPU and RAM capacity of a *large* VM instance in EC2 is four times larger compared to a capacity of a *small* VM instance. However, for most applications in our set, the measured speedup is much lower than expected under the traditional "rule-of-thumb" (i.e., 4 times). The reason is that Cloud environments

[1]This work was originated during Z. Zhang's internship at HP Labs. Prof. B. T. Loo a nd Z. Zhang are supported in part by NSF grants CNS-1117185 and CNS-0845552.

use VM instance scaling with respect to CPU and RAM capacity, while the storage and network bandwidth (that is critical for MapReduce job performance) is only fractionally improved for different capacity VM instances.

This further stresses the importance of MapReduce job profiling and performance modeling for making the right business decisions on the choice of the underlying Hadoop clusters.

The rest of the paper is organized as follows. Section II provides background information on MapReduce. Section III describes the bounds-based performance model and argues why it works well in heterogeneous environment. Section IV evaluates the accuracy and effectiveness of the proposed approach. Section V outlines related work. Section VI summarizes our contribution and gives directions for future work.

Acknowledgments: Our sincere thanks to one of the anonymous reviewers for a detailed list of useful comments, edits, and suggestions.

II. Background

This section provides an overview of the MapReduce [3] abstraction, execution, and scheduling.

In the MapReduce model, computation is expressed as two functions: map and reduce. The map function takes an input pair (k_1, v_1) and produces a list of intermediate key/value pairs. The intermediate values associated with the same key k_2 are grouped together and then passed to the reduce function. The reduce function takes intermediate key k_2 with a list of values and processes them to form a new list of values.

$$\begin{aligned} map(k_1, v_1) &\rightarrow list(k_2, v_2) \\ reduce(k_2, list(v_2)) &\rightarrow list(v_3) \end{aligned}$$

MapReduce jobs are executed across multiple machines: the map stage is partitioned into *map tasks* and the reduce stage is partitioned into *reduce tasks*. The map and reduce tasks are executed by *map slots* and *reduce slots*.

In the *map stage*, each map task reads a split of the input data, applies the user-defined map function, and generates the intermediate set of key/value pairs. The map task then sorts and partitions these data for different reduce tasks according to a partition function.

In the *reduce stage*, each reduce task fetches its partition of intermediate key/value pairs from all the map tasks and sorts/merges the data with the same key (it is called the shuffle/sort phase). After that, it applies the user-defined reduce function to the merged value list to produce the aggregate results (it is called the reduce phase). Then the reduce outputs are written back to a distributed file system.

Job scheduling in Hadoop is performed by a master node called the JobTracker, which manages a number of worker nodes in the cluster. Each worker node in the cluster is configured with a fixed number of map and reduce slots. The worker nodes periodically connect to the JobTracker to report their current status and the available slots. The JobTracker decides the next job to execute based on the reported information and according to a scheduling policy. The popular job schedulers include FIFO (First In First Out), Fair [14], and Capacity scheduler [2]. The assignment of tasks to slots is done in a greedy way: assign a task from the selected job immediately whenever a worker reports to have a free slot. If the number of tasks belonging to a MapReduce job is greater than the total number of slots available for processing the job, the task assignment will take multiple rounds, which we call *waves*.

The Hadoop implementation includes *counters* for recording timing information such as start and finish timestamps of the tasks, or the number of bytes read and written by each task. These counters are written to logs after the job is completed.

III. Bounds-Based Performance Model

In this section, we provide an outline of the *bounds-based* performance model proposed in ARIA [10] for predicting the MapReduce job completion time, and then explain why this model does work well in heterogeneous environments.

The proposed MapReduce performance model [10] evaluates lower and upper bounds on the job completion time. It is based the Makespan Theorem [10] for computing performance bounds on the completion time of a given set of n tasks that are processed by k servers, (e.g., n map tasks are processed by k map slots in MapReduce environment). The assignment of tasks to slots is done using an online, *greedy* algorithm: assign each task to the slot which finished its running task the earliest. Let avg and max be the *average* and *maximum* duration of the n tasks respectively. Then the completion time of a greedy task assignment is proven to be at least:

$$T^{low} = avg \cdot \frac{n}{k}$$

and at most

$$T^{up} = avg \cdot \frac{(n-1)}{k} + max$$

The difference between lower and upper bounds represents the range of possible completion times due to task scheduling non-determinism.

As motivated by the above model, in order to approximate the overall completion time of a MapReduce job J, we need to estimate the *average* and *maximum* task durations during different execution phases of the job, i.e., *map, shuffle/sort,* and *reduce* phases. These measurements can be obtained from the job execution logs. By applying the outlined bounds model, we can estimate the completion times of different processing phases of the job. For example, let job J be partitioned into N_M^J map tasks. Then the lower and upper bounds on the duration of the entire map stage in the *future* execution with S_M^J map slots (denoted as T_M^{low} and T_M^{up} respectively) are estimated as follows:

$$T_M^{low} = M_{avg}^J \cdot N_M^J / S_M^J \quad (1)$$

$$T_M^{up} = M_{avg}^J \cdot (N_M^J - 1)/S_M^J + M_{max}^J \quad (2)$$

where M_{avg} and M_{max} are the average and maximum of the map task durations of the past run respectively. Similarly, we can compute bounds of the execution time of other processing phases of the job.

The **reduce stage** consists of the *shuffle/sort* and *reduce* phases. The shuffle phase begins only after the first map task has completed. The shuffle phase completes when the entire map stage is complete and all the intermediate data generated by the map tasks has been shuffled to the reduce tasks and has been sorted.

The **shuffle phase** of the *first* reduce wave may be significantly different from the shuffle phase of subsequent reduce

waves. This happens because the shuffle phase of the first reduce wave overlaps with the entire map stage, and hence depends on the number of map waves and their durations. Therefore, from the past execution, we extract two sets of measurements: (Sh^1_{avg}, Sh^1_{max}) for the shuffle phase of the first reduce wave (called *first shuffle*) and $(Sh^{typ}_{avg}, Sh^{typ}_{max})$ for shuffle phase of the other waves (called, *typical shuffle*). Moreover, we characterize the first shuffle in a special way and include only the non-overlapping portion (with map stage) in our metrics: Sh^1_{avg} and Sh^1_{max}. This way, we carefully estimate the latency portion that contributes explicitly to the job completion time. The *typical shuffle* phase is computed as follows:

$$T^{low}_{Sh} = \left(N^J_R / S^J_R - 1\right) \cdot Sh^{typ}_{avg} \quad (3)$$

$$T^{up}_{Sh} = \left((N^J_R - 1)/S^J_R - 1\right) \cdot Sh^{typ}_{avg} + Sh^{typ}_{max} \quad (4)$$

The **reduce phase** begins only after the shuffle phase is complete. From the distribution of the reduce task durations of the past run, we compute the *average* and *maximum* metrics: R_{avg} and R_{max}. Similarly, the Makespan Theorem [10] can be directly applied to compute the lower and upper bounds of completion times of the reduce phase (T^{low}_R, T^{up}_R) when a different number of allocated reduce slots S^J_R is given.

Finally, we can put together the formulae for the lower and upper bounds of job completion time:

$$T^{low}_J = T^{low}_M + Sh^1_{avg} + T^{low}_{Sh} + T^{low}_R \quad (5)$$

$$T^{up}_J = T^{up}_M + Sh^1_{max} + T^{up}_{Sh} + T^{up}_R \quad (6)$$

This simple model should work well in heterogeneous environments. Intuitively, in a heterogeneous Hadoop cluster, slower nodes result in longer task executions. These measurements then are reflected in the calculated average and maximum task durations that comprise the job profile. While the bounds-based performance model does not explicitly consider different types of nodes, their performance is implicitly reflected in the job profile and used in the future prediction.

IV. Performance Study

In this section, we evaluate the accuracy of the bounds-based performance model using a set of 13 diverse applications and two different heterogeneous Hadoop clusters, and also discuss its limitations for predicting the job completion time. Then, by performing a detailed analysis of job profiles and application completion times, we reveal factors that impact the MapReduce job performance in the Cloud.

A. Experimental Testbeds and Workloads

We use the following two environments in our experiments.

a) **The UPenn heterogeneous cluster:** It contains 36 worker nodes of 3 different types as shown in Table I.

TABLE I
UPenn cluster description.

Node type	#nodes	*CPU type*	RAM (GB)	#m,r slots
Type1	16	Xeon X3220 (quad-core) compute nodes, Quad Core Intel Xeon X3220, 2.40GHz	4.0	2, 1
Type2	12	Xeon X3363 (quad-core) compute nodes, Quad Core Intel Xeon X3363, 2.83GHz	4.0	2, 1
Type3	8	Xeon X3450 (quad-core) compute nodes, Quad Core Intel X3450 Xeon 2.66GHz	4.0	2, 1

The heterogeneity is caused by the extension of the cluster over time. In 2007, the cluster had 16 nodes with the same hardware (*Type1* nodes). Then, two years later, 12 more nodes were added to the cluster with more powerful CPUs (*Type2* nodes). Finally, in 2010, 8 more nodes (*Type3*) were added to the cluster to satisfy the growing workloads and computing demands. Each node has a Dual SATA 250GB drive. All the nodes are connected to the same rack: each node has 1 Gbit/s network connection to a 10 Gbit/s switch. An additional server node (*Type3*) runs the NameNode and JobTracker of the deployed Hadoop cluster. While the nodes in the UPenn cluster represent different server generations, thay all have the same number of CPU cores and the same amount of RAM. This explains why we configure these nodes in a similar way, with the same number of map and reduce slots.

b) **The Amazon EC2 platform:** The EC2 environment offers a choice of different capacity Virtual Machines (VMs) for deployment. These VMs can be deployed on a variety of hardware and be allocated different amounts of system resources. We build a *heterogeneous* Hadoop cluster that consists of different VM types:

- **10** VMs based on *small instances (m1.small)*,
- **10** VMs based on *medium instances (m1.medium)*, and
- **10** VMs based on *large instances (m1.large)*.

The description of each VM instance type is shown in Table II. Since the compute and memory capacity of a medium VM instance is doubled compared to a small VM instance (similarly, large VM instances have a doubled capacity compared to the medium ones), we configured different numbers of map and reduce slots for different VM instances as shown in Table II. Each VM instance is deployed with 100GB of Elastic Block Storage (EBS).

TABLE II
EC2 Testbed description.

Instance type	#VMs	*CPU capacity (relative)*	RAM (GB)	#m,r slots
Small	10	1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit)	1.7	1, 1
Medium	10	2 EC2 Compute Unit (1 virtual core with 2 EC2 Compute Unit)	3.75	2, 1
Large	10	4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each)	7.5	4, 2

We use an additional *high-CPU VM instance* for running the NameNode and JobTracker of the Hadoop cluster.

In our performance study, we also use two EC2-based *homogeneous* Hadoop clusters, which are deployed with 30 nodes, each by using only *small* or *large* VM instances.

We use Hadoop 1.0.0 in all the experiments. The file system blocksize is set to 64MB and the replication level is set to 3. We disabled speculative computations in all our experiments as they did not lead to any significant improvements.

In the performance study, we use a set of 13 applications made available by the Tarazu project [1]. Table III provides a high-level summary of these 13 applications with the corresponding job settings (e.g, numbers of map and reduce tasks). Applications 1, 8, and 9 process synthetically generated data. Applications 2 to 7 use the Wikipedia articles dataset as input. Applications 10 to 13 use the Netflix movie ratings dataset. These applications perform very different data manipulations. To provide some additional insights in the amounts of data flowing through the MapReduce processing pipeline, we also

show the overall size of the input data, intermediate data (i.e., data generated between map and reduce stages), and the output data (i.e., the data written by the reduce stage).

TABLE III
APPLICATION CHARACTERISTICS.

Application	*Input data (type)*	*Input data (GB)*	*Interm data (GB)*	*Output data (GB)*	*#map,red tasks*
1. TeraSort	Synthetic	31	31	31	495, 60
2. WordCount	Wikipedia	50	9.8	5.6	788, 60
3. Grep	Wikipedia	50	1	$1x10^{-8}$	788, 1
4. InvIndex	Wikipedia	50	10.5	8.6	788, 60
5. RankInvIndex	Wikipedia	46	48	45	745, 60
6. TermVector	Wikipedia	50	4.1	0.002	788, 60
7. SeqCount	Wikipedia	50	45	39	788, 60
8. SelfJoin	Synthetic	28	25	22	448, 60
9. AdjList	Synthetic	28	29	29	508, 60
10. HistMovies	Netflix	27	$3x10^{-5}$	$7x10^{-8}$	428, 1
11. HistRatings	Netflix	27	$2x10^{-5}$	$6x10^{-8}$	428, 1
12. Classification	Netflix	27	0.008	0.006	428, 16
13. KMeans	Netflix	27	27	27	428, 16

B. Prediction Accuracy of Bounds-Based Model in Heterogeneous Environments

In this section, we evaluate the accuracy of the bounds-based performance model for predicting the job completion time in heterogeneous environments. We extract the job profiles while executing these jobs on a given heterogeneous Hadoop cluster. Many production jobs and applications are executed periodically for processing the new datasets. The extracted job profiles and the bounds-based performance model can be used for predicting the completion times of these future executions. Figure 1 shows the predicted vs measured results for 13 applications processed on the UPenn heterogeneous Hadoop cluster.[2] Given that the completion times of different programs range between 80 seconds (for *HistMovies*) and 48 minutes (for *KMeans*), we **normalize** the predicted completion times with respect to the measured ones for the sake of readability and comparison.

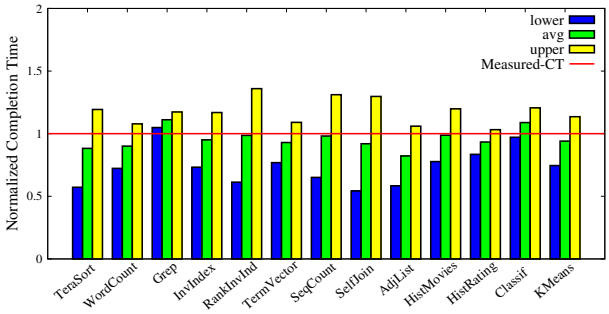


Fig. 1. Predicting the job completion time in the UPenn cluster.

The three bars in Figure 1 represent the normalized predicted completion times based on the lower (T^{low}) and upper (T^{up}) bounds, and the average of them (T^{avg}). We observe that the actual completion times (shown as the straight *Measured-CT* line) of 12 programs fall between the lower and upper bound estimates (except for the *Grep* application). Moreover, the predicted completion times based on the average of the upper and lower bounds are within 10% of the measured results for 11 out of the 13 applications. The worst prediction is around 18% error for the *AdjList* application.

[2]All the experiments are performed five times, and the measurement results are averaged. This comment also applies to the results in Figure 2.

UPenn cluster contains servers of three different CPU generations, but each node has a similar number of CPU cores, memory, disk storage and network bandwidth. The bounds-based model does work well in this environment. Now, we perform a similar comparison for EC2-based heterogeneous cluster, where the cluster nodes are formed by VM instances with very different computing capacities.

Figure 2 shows the normalized predicted completion times (based on the lower, upper, and average bounds) compared to the measured ones for executing 13 applications on the EC2-based heterogeneous cluster. The results validate the accuracy of the proposed model: the measured completion times of all 13 programs fall between the lower and upper bound estimates. The average of the lower and upper bounds are within 10% of the measured value for 9 out of 13 applications with a worst case of 13% error for the *WordCount* application.

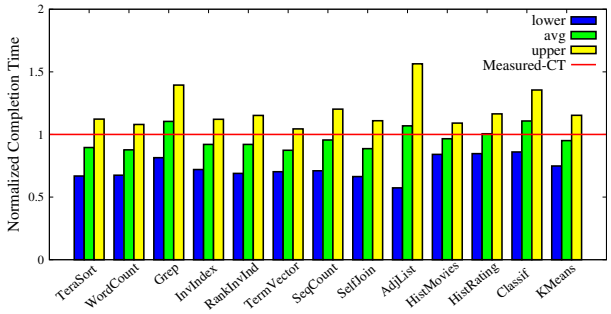


Fig. 2. Predicting job completion time in heterogeneous EC2 cluster.

Among the 13 applications, there are 3 special ones: *Grep, HistMovies,* and *HistRatings,* which have a single reduce task (defined by the special semantics of these applications). The shuffle/reduce stage durations of such an application depend on the Hadoop node type that is allocated to execute this single reduce task. If there is a significant difference in the Hadoop nodes, it may impact the completion times of the shuffle/reduce stages across the different runs, and therefore, make the prediction inaccurate.

Figure 3 analyzes the executions of *Grep, HistMovies,* and *HistRatings,* on the heterogeneous EC2-based cluster. The

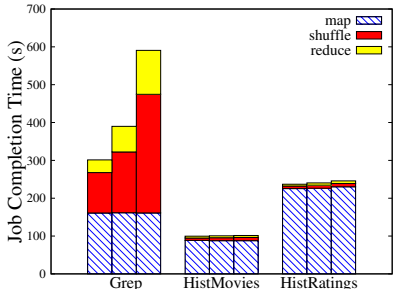


Fig. 3. A special case of jobs with a single reduce task: their possible executions on the heterogeneous EC2 cluster.

three bars within each group represent (from left to right) the job completion times when the reduce task is executed by the reduce slot of a *large, medium,* or *small* VM instance. Each bar is further split to represent the relative durations of the map, shuffle, and reduce stages in the job execution. We observe that the completion time of the *Grep* application is significantly impacted by the type of VM instance allocated to the reduce task execution. The reduce task execution time

on the *small* VM instance is almost twice as long as that on a large VM instance In comparison, the execution times of *HistMovies* and *HistRatings* on different capacity VMs are not significantly different, because for these applications the reduce task duration constitutes a very small fraction of the overall completion time (see Figure 3). In summary, if the execution time of a reduce stage constitutes a significant part of the total completion time, the accuracy of the bounds-based model may be adversely impacted by the node type allocated to the reduce task execution.

C. Analysis of Job Execution Times on Different Hadoop Clusters

Heterogeneity in a Hadoop cluster may significantly impact the application performance and prediction accuracy, as shown in the previous section. There is an additional difference between the two heterogeneous clusters used in our experiments. The UPenn Hadoop cluster runs on a dedicated physical infrastructure, while the EC2-based Hadoop cluster is deployed using VMs in the Cloud. Does this difference matter for the application performance, jobs execution stability, and performance predictions?

Traditionally, a simple rule of thumb states [12] that if T is the completion time of a MapReduce job in a Hadoop cluster with N nodes then the same job can be completed in $T/2$ on a larger Hadoop cluster with $2 \times N$ nodes. Note that the CPU and RAM capacity of a *large* VM instance in EC2 is four times larger than the capacity of a *small* VM instance. This difference in capacity is reflected in VM instance pricing: the price per use of a large VM instance is four times higher than the price per use of a small VM instance. The question is whether by using the large VM instances instead of the small VM ones, we can see a similar scaling factor in application performance improvements.

To answer these questions, we perform a detailed performance analysis of the job executions (13 applications from Table III) on the following four Hadoop clusters:

- the heterogeneous 36-node UPenn Hadoop cluster;
- the heterogeneous 30-node EC2-based Hadoop cluster;
- the homogeneous 30-node EC2-based Hadoop cluster, where each node is formed by a *small* VM instance with one map and one reduce slot per node;
- the homogeneous 30-node EC2-based Hadoop cluster, where each node is formed by a *large* VM instance with four map slots and two reduce slot per node.

We execute the set of 13 applications on each cluster 5 times, and collect the job profiles for each run.

Tables IV-VII summarize the results of these experiments. Each table reflects multiple interesting metrics that provide useful insight in the MapReduce job executions of the 13 applications under study. The first eight columns show the average and maximum durations of *i)* processed map tasks, *ii)* shuffle phases (both first and typical shuffle), and *iii)* reduce phase processing. Note that not all the jobs have measurements of the typical shuffle, i.e., the shuffle phase of the second and greater waves of reduce task executions (see Section III for formal definitions). For some applications such as *Grep, HistMovies and HistRatings* there is a single reduce task in the job, and this reduce task is always executed during the first reduce wave. For some cluster experiments, such as the homogeneous EC2-based cluster formed with large VM instances, there are 60 reduce slots in the cluster, and all the applications in our workload set have 60 or less reduce tasks configured. So all these reduce tasks can be processed during the execution of the first reduce wave.

Among the highlights of the presented job profiles there are the following interesting observations:

- While the map tasks of all the 13 applications process equal size splits of 64 MB, the average map task duration varies significantly, even for the jobs executed on the same cluster. Apparently, the semantics of the custom user-defined map function has a significant impact on the execution time of the map tasks. For example, the *KMeans* application performs a complex CPU-intensive computation during its map task processing that takes on average 365 sec. on the UPenn cluster, and 1500 sec. on the heterogeneous EC2-based cluster, compared with the average map task durations of 9.4 sec and 11.8 sec of the *Grep* application on the same clusters.
- We also observe that the shuffle phase performance on the UPenn cluster is significantly better than the shuffle performance on the EC2 Hadoop cluster (for the same applications). One explanation is that the networking infrastructure in the Cloud is shared between multiple users, while UPenn cluster has a dedicated networking used by the Hadoop jobs only. This may be an essential difference factor impacting the Hadoop performance in the Cloud.

The 9th column in each table reflects the job completion time. While all 13 applications process the input datasets ranging between 27GB and 50GB, the completion times of these applications vary by two orders of magnitude. For example, in the heterogeneous EC2-based cluster (see Table V), the completion time of *HistMovies* is 104 sec, while *KMeans* processing takes 11551sec (3.2 hours).

Finally, the columns 10-13 show the standard deviation for the measured map stage, shuffle stage, reduce stage, and the total completion time across five runs. Map stage durations are very stable across different applications and clusters (*standard deviation* is less than 10% for all cases). While shuffle and reduce stages for some applications exhibit a higher standard deviation, the overall job completion times are quite stable (*stdev* is less than 8%) for 12 out 13 applications executed on heterogeneous Hadoop clusters (see Tables VI,VII). The *Grep* execution shows the highest completion time variance (up to 30%) due to a reason discussed in the previous section. The execution of all 13 applications on homogeneous Hadoop clusters show a stable completion time across multiple runs (*stdev* is less than 7%, see Tables VI, VII).

To further understand different stage contributions to the overall job completion time, we present the normalized completion time break-down with respect to the map/shuffle/reduce stage durations. Figures 4 (a)-(d) present a detailed analysis of job completion times across the four Hadoop clusters and 13 applications used in our experiments. Figure 4 (b) reflects the completion time structure for the

TABLE IV
JOB PROFILES ON THE UPENN HETEROGENEOUS CLUSTER (TIME IN SEC)

Application	*avg map task*	*max map task*	*avg first shuffle*	*max first shuffle*	*avg typical shuffle*	*max typical shuffle*	*avg reduce task*	*max reduce task*	*Application Completion Time*	*map stage STDEV*	*shuffle stage STDEV*	*reduce stage STDEV*	*total time STDEV*
TeraSort	13.3	29.1	6.6	45.2	44.6	53.5	20.4	64.3	238.4	4.34%	6.63%	12.19%	6.88%
WordCount	17.5	37.5	4.5	47.5	42.0	49.1	12.3	26.3	321.2	0.96%	11.40%	12.99%	1.96%
Grep	9.4	25.1	33.9	65.7	-	-	24.8	30.6	225.1	6.63%	56.34%	19.50%	12.76%
InvIndex	18.7	46.8	7.4	43.3	39.9	46.9	18.2	56.8	347.7	3.84%	15.07%	2.72%	2.65%
RankInvInd	13.7	33.7	7.0	29.4	56.5	83.0	88.7	207.2	435.5	1.67%	4.78%	7.22%	2.56%
TermVector	18.5	43.2	3.1	33.5	35.3	38.4	6.2	15.8	307.1	5.86%	8.19%	8.22%	4.81%
SeqCount	23.5	61.6	4.6	36.8	67.4	91.0	80.8	217.7	562.4	0.73%	3.16%	13.09%	2.23%
SelfJoin	10.2	21.4	6.5	46.0	38.2	46.1	8.0	13.4	176.4	8.32%	4.35%	3.49%	1.68%
AdjList	83.0	158.6	183.4	305.9	200.3	310.8	286.6	404.9	1837.0	1.91%	6.59%	0.47%	0.78%
HistMovies	6.9	32.4	21.4	25.3	-	-	3.4	4.9	80.3	9.51%	15.17%	19.74%	5.14%
HistRatings	16.0	26.3	21.5	25.7	-	-	4.0	5.2	139.9	0.98%	16.11%	20.22%	2.61%
Classification	8.5	19.8	20.2	23.9	-	-	3.6	5.9	93.6	1.30%	10.29%	5.98%	2.20%
KMeans	336.5	878.1	13.8	100.5	-	-	113.8	361.4	2876.0	2.75%	26.68%	3.33%	2.49%

TABLE V
JOB PROFILES ON THE EC2-BASED HETEROGENEOUS CLUSTER (TIME IN SEC)

Application	*avg map task*	*max map task*	*avg first shuffle*	*max first shuffle*	*avg typical shuffle*	*max typical shuffle*	*avg reduce task*	*max reduce task*	*Application Completion Time*	*map stage STDEV*	*shuffle stage STDEV*	*reduce stage STDEV*	*total time STDEV*
TeraSort	20.7	44.4	208.0	218.2	363.2	442.4	35.8	91.4	718.0	1.42%	12.51%	1.05%	7.90%
WordCount	40.5	123.0	229.7	295.9	454.1	504.4	21.7	55.3	1170.7	4.12%	4.64%	5.71%	3.47%
Grep	11.8	23.8	159.8	311.7	-	-	46.2	114.8	337.6	1.33%	23.88%	39.52%	30.22%
InvIndex	45.7	122.7	273.0	302.0	458.6	538.2	32.9	83.9	1262.6	4.17%	6.89%	4.33%	2.57%
RankInvInd	23.1	72.2	239.6	299.2	626.7	716.9	139.3	279.2	1341.7	1.57%	3.81%	2.82%	2.43%
TermVector	46.1	116.4	258.8	283.6	437.7	531.2	9.1	40.0	1194.0	1.98%	10.78%	4.27%	4.75%
SeqCount	59.4	225.2	276.8	339.8	498.0	649.0	127.9	244.3	1672.2	3.74%	13.19%	3.15%	4.39%
SelfJoin	15.4	41.4	139.1	179.0	293.5	381.7	10.2	26.3	524.2	1.03%	10.18%	3.95%	7.24%
AdjList	204.5	656.2	407.6	495.8	472.8	1275.5	528.8	1370.6	3703.7	2.57%	6.48%	3.42%	2.85%
HistMovies	11.1	26.5	7.8	11.8	-	-	3.6	5.3	104.0	0.93%	47.37%	23.57%	4.90%
HistRatings	31.6	52.4	3.9	10.5	-	-	3.9	5.1	237.6	1.20%	91.75%	18.39%	2.61%
Classification	14.1	37.5	89.3	103.5	-	-	3.5	6.9	207.5	3.02%	9.89%	8.67%	4.22%
KMeans	1500.4	4326.5	12.4	96.8	-	-	166.9	581.3	11551.8	1.28%	27.32%	13.42%	1.17%

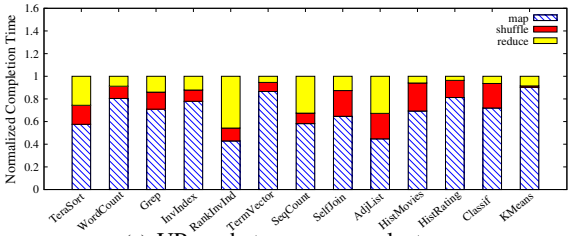
TABLE VI
JOB PROFILES ON THE EC2-BASED HOMOGENEOUS CLUSTER WITH *small* VM INSTANCES (TIME IN SEC)

Application	*avg map task*	*max map task*	*avg first shuffle*	*max first shuffle*	*avg typical shuffle*	*max typical shuffle*	*avg reduce task*	*max reduce task*	*Application Completion Time*	*map stage STDEV*	*shuffle stage STDEV*	*reduce stage STDEV*	*total time STDEV*
TeraSort	23.0	53.0	46.4	70.4	197.1	224.8	57.0	85.4	814.7	0.17%	6.04%	1.48%	1.85%
WordCount	53.0	124.5	32.6	47.1	195.0	236.0	30.6	49.7	1797.9	2.79%	8.00%	3.90%	2.96%
Grep	10.2	19.8	174.2	184.4	-	-	120.6	128.3	616.2	0.38%	6.07%	4.52%	2.14%
InvIndex	63.2	130.9	34.5	46.7	200.9	231.9	49.9	72.8	2118.5	1.19%	4.83%	4.55%	1.48%
RankInvIndex	27.7	90.0	62.7	109.2	280.1	320.6	214.5	313.6	1614.7	1.32%	5.40%	2.96%	2.19%
TermVector	60.5	121.4	18.7	49.8	144.4	193.0	12.5	46.1	1905.6	2.93%	4.79%	5.53%	2.92%
SeqCount	84.4	244.3	25.9	42.0	312.4	360.0	190.5	262.9	3149.4	1.19%	1.47%	2.45%	1.58%
SelfJoin	17.4	39.9	62.0	99.9	151.7	170.7	19.0	39.9	536.9	1.82%	2.48%	8.51%	1.73%
AdjList	278.5	686.5	14.0	41.9	1446.4	1906.2	999.2	1398.8	8929.9	0.48%	1.20%	0.37%	0.47%
HistMovies	12.4	24.7	5.4	9.9	-	-	4.9	5.8	181.2	0.76%	72.21%	14.43%	1.16%
HistRatings	30.8	44.0	4.4	11.5	-	-	3.8	5.2	491.3	0.84%	87.71%	31.20%	0.94%
Classification	14.7	28.1	3.6	12.3	-	-	3.7	7.3	256.4	1.36%	30.07%	5.76%	0.34%
KMeans	3265.3	4669.3	27.7	138.2	-	-	196.6	501.1	48877.6	0.69%	6.89%	3.52%	0.74%

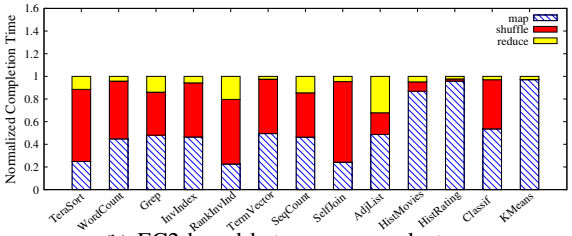
TABLE VII
JOB PROFILES ON THE EC2-BASED HOMOGENEOUS CLUSTER WITH *large* VM INSTANCES (TIME IN SEC)

Application	*avg map task*	*max map task*	*avg first shuffle*	*max first shuffle*	*avg typical shuffle*	*max typical shuffle*	*avg reduce task*	*max reduce task*	*Application Completion Time*	*map stage STDEV*	*shuffle stage STDEV*	*reduce stage STDEV*	*total time STDEV*
TeraSort	19.7	38.6	251.3	315.2	-	-	25.7	55.3	406.4	1.05%	8.62%	3.49%	4.91%
WordCount	38.9	69.9	354.5	444.0	-	-	15.6	29.7	692.7	0.49%	2.29%	8.53%	1.56%
Grep	10.1	14.3	126.3	130.6	-	-	30.7	34.0	233.0	0.55%	8.58%	9.46%	4.45%
InvIndex	41.4	79.3	347.8	432.7	-	-	24.1	43.5	708.3	0.46%	3.23%	3.45%	2.68%
RankInvIndex	22.8	46.5	432.1	512.9	-	-	137.5	212.1	780.8	0.75%	4.03%	3.87%	2.13%
TermVector	38.7	68.7	351.2	473.2	-	-	6.8	16.9	676.2	0.47%	7.17%	2.87%	4.04%
SeqCount	51.5	130.2	409.2	493.8	-	-	133.4	264.4	998.8	0.52%	4.54%	4.12%	1.21%
SelfJoin	15.4	30.3	219.3	279.1	-	-	8.1	27.6	326.8	1.84%	5.33%	5.38%	7.02%
AdjList	145.9	308.0	402.3	638.3	-	-	353.4	615.2	1666.2	0.63%	3.65%	0.78%	1.75%
HistMovies	9.8	16.5	22.0	27.0	-	-	3.9	6.3	91.2	0.61%	16.38%	28.10%	2.67%
HistRatings	30.4	42.3	5.2	10.5	-	-	3.6	6.0	150.0	0.90%	45.74%	24.66%	3.25%
Classification	14.2	32.9	48.9	67.6	-	-	3.7	6.5	126.7	0.89%	3.5%	3.49%	5.46%
KMeans	1028.7	2173.5	16.7	99.8	-	-	129.1	343.4	4582.5	0.55%	7.7%	3.69%	2.06%

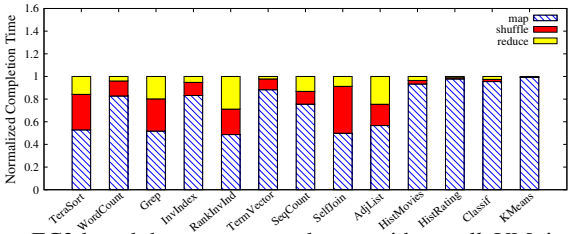
jobs executed on the heterogeneous EC2-based cluster. We see that the shuffle stage contributes a significant fraction in the overall execution time, while the shuffle fraction is much smaller in the job execution time on the UPenn cluster shown in Figure 4 (a). We made similar observations by analyzing the the job profiles shown in Tables IV-V. We can see that for some applications (e.g., *KMeans*), the overall execution time is completely dominated by the map stage processing.



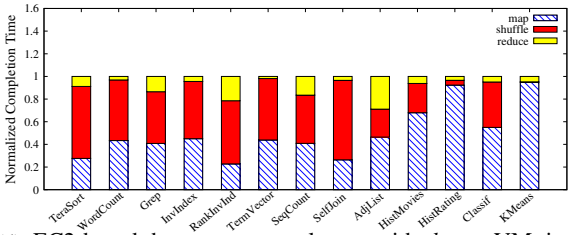
(a) UPenn heterogeneous cluster.



(b) EC2-based heterogeneous cluster.



(c) EC2-based homogeneous cluster with *small* VM instances.



(d) EC2-based homogeneous cluster with *large* VM instances.

Fig. 4. Breakdown analysis of the job completion times on different Hadoop clusters.

It is interesting to see how the profile of the completion time breakdown changes when the same applications are executed on two different EC2-based homogeneous Hadoop clusters: one formed by the *small* VM instances and the second cluster formed by the *large* VM instances. These results are shown in Figures 4 (c)-(d). In the Hadoop cluster with *small* VM instances, the completion time of most jobs is dominated by the map stage execution. However, when the same applications are executed on the Hadoop cluster with *large* VM instances, this changes the balance. Now the map stage is executed faster (the cluster has a larger number of map slots), and the shuffle phase (that needs to shuffle a larger amount of data produced per time unit) dominates the job execution times of most applications in our set.

Figure 5 presents the normalized completion times of 13 applications executed on three different EC2-based clusters. We normalize the measured job completion times with respect to the execution time of the same job on the Hadoop cluster formed with *small* VM instances. Note that the CPU and RAM capacity of the *large* VM instance in EC2 is four times larger compared to a capacity of the *small* VM instance. We try to answer two questions here. First, whether the completion time reduction is similar to the allocated resources increase? Second, what are performance scaling factors of the job completion times when the 13 applications are executed on a more powerful platform?

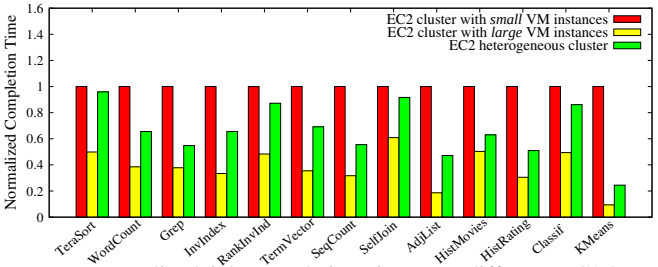


Fig. 5. Normalized job completion times on different EC2-based Hadoop clusters.

As Figure 5 shows, the completion time *speedup* significantly depends on the nature (semantics) of the applications. If we compare the job completion times on two EC2-based clusters formed by *small* and *large* VM instances, the obtained speedup can be summarize as follows:

- speedup of 1.6-2.1 times (5 jobs: *TeraSort, RankInvIndex, SelfJoin, HistMovies, Classification*);
- speedup of 2.6-3.3 times (6 jobs: *WordCoundt Grep, InvIndex, TermVector, SeqCount, HistRatings*);
- speedup of 5.4 times (1 job: *Adjlist*);
- speedup of 10.7 times (1 job: *KMeans*).

As we can see, CPU-intensive applications (such as *KMeans*) could get significant benefits of increased CPU and RAM capacity. However, for most applications in our set, the measured speedup is much lower than expected under the traditional "rule-of-thumb" (i.e., 4 times). The reason is that in the Cloud, the VM instance scaling is done with respect to CPU and RAM capacity, while the storage and network bandwidth (that is critical for MapReduce job performance) is only fractionally improved.

If we compare the job completion times on the EC2-based homogeneous cluster formed by *small* VM instances and the heterogeneous cluster, then the obtained speedup can be summarize as follows:

- speedup of 1.1-1.2 times (4 jobs: *TeraSort, RankInvIndex, SelfJoin, Classification*);
- speedup of 1.5-1.9 times (6 jobs: *WordCount, InvIndex, TermVector, HistMovies,Grep, SeqCount*)
- speedup of 2.1-2.4 times (2 jobs: *Adjlist, HistRatings*);
- speedup of 4.2 times (1 job: *KMeans*).

The presented analysis further stresses the importance of MapReduce job profiling and performance modeling for making the right decisions when choosing Hadoop clusters.

V. Related Work

In the past few years, performance modeling in MapReduce environments has received much attention, and different approaches [4], [5], [9], [10], [11] were offered for predicting performance of MapReduce applications.

Morton et al. [7] propose an estimator for evaluating the MapReduce job progress. It uses debug runs of the same query on a data sample to estimate the relative processing speeds of map and reduce stages. Their later work [6] propose ParaTimer which extends the work for estimating the progress of parallel queries expressed as Pig scripts that can translate into directed acyclic graphs (DAGs) of MapReduce jobs.

Tian and Chen [9] propose an approach to predict MapReduce program performance from a set of test runs on a 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. However, for each application, a different set of test runs is required to derive the specific model for the application, which limits its applicability in general cases.

In *Starfish* [5] Herodotou et al. apply dynamic Java instrumentation to collect run-time monitoring information about job execution at a fine granularity and extract diverse metrics. Such a detailed job profiling enables the authors to predict job execution under different Hadoop configuration parameters, derive an optimized the cluster configuration, and solve cluster sizing problem [4]. However, collecting a large set of metrics comes at a cost. To avoid significant overhead profiling should be only applied to a small fraction of tasks.

ARIA [10] proposes a framework that automatically extracts compact job profiles from past application run(s). These job profiles form a 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. In later work [11], this model is extended with additional framework for deriving the scaling factors that are used for predicting the job completion times of MapReduce applications for processing larger datasets.

There is a body of work focusing on performance optimization of MapReduce executions in heterogeneous environments. Zaharia et al. [15], focus on eliminating the negative effect of stragglers on job completion time by improving the scheduling strategy with speculative tasks. The Tarazu project [1] provides a communication-aware scheduling of map computation which aims at decreasing the communication overload when faster nodes process map tasks with input data stored on slow nodes. It also proposes a load-balancing approach for reduce computation by assigning different amounts of reduce work according to the node capacity. Xie et al. [13] try improving the MapReduce performance through a heterogeneity-aware data placement strategy: a faster nodes store larger amount of input data. In this way, more tasks can be executed by faster nodes without a data transfer for the map execution. Polo et al. [8] show that some MapReduce applications can be accelerated by using special hardware. The authors design an adaptive Hadoop scheduler that assigns such jobs to the nodes with corresponding hardware.

One major shortcoming of these existing performance models is that they are valid only for homogeneous Hadoop clsuters and do not take into account *resource heterogeneity*.

VI. Conclusion and future work

Hadoop is increasingly being deployed in enterprise private clouds and offered as a service by public cloud providers. We observe that in many practical deployments today, clusters are grown incrementally over time, and therefore combine heterogeneous resources. In this work, we have focused on the design and evaluation of a performance model for predicting job completion times in heterogeneous MapReduce environments. We have assessed the efficiency and accuracy of the bounds-based performance model using a diverse set of 13 realistic applications and two different testbeds. The predicted completion times are within 10% of the measured ones for 12 (out of 13) applications in the workload set.

In our future work, we plan to explore the properties of the job profiles as well as the analysis of the job completion time breakdown to utilize potential performance benefits of the job execution on different types of heterogeneous nodes in the cluster, (e.g., executing the CPU-intensive jobs on the nodes with a higher CPU capacity). We are also going to investigate the strategy that determines the optimal cluster deployment for applications with completion time requirement.

References

[1] F. Ahmad et al. Tarazu: Optimizing MapReduce on heterogeneous clusters. In *Proc. of ASPLOS*, 2012.

[2] Apache. Capacity Scheduler Guide, 2010.

[3] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1), 2008.

[4] H. Herodotou, F. Dong, and S. Babu. No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics. In *Proc. of ACM Symposium on Cloud Computing*, 2011.

[5] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. Cetin, and S. Babu. Starfish: A Self-tuning System for Big Data Analytics. In *Proc. of 5th Conf. on Innovative Data Systems Research (CIDR)*, 2011.

[6] K. Morton, M. Balazinska, and D. Grossman. ParaTimer: a progress indicator for MapReduce DAGs. In *Proc. of SIGMOD*. ACM, 2010.

[7] K. Morton, A. Friesen, M. Balazinska, and D. Grossman. Estimating the progress of MapReduce pipelines. In *Proc. of ICDE*, 2010.

[8] J. Polo et al. Performance management of accelerated mapreduce workloads in heterogeneous clusters. In *Proc. of the 41st Intl. Conf. on Parallel Processing*, 2010.

[9] F. Tian and K. Chen. Towards Optimal Resource Provisioning for Running MapReduce Programs in Public Clouds. In *Proc. of IEEE Conference on Cloud Computing (CLOUD 2011)*.

[10] A. Verma, L. Cherkasova, and R. H. Campbell. ARIA: Automatic Resource Inference and Allocation for MapReduce Environments. *Proc. of the 8th ACM Intl Conf. on Autonomic Computing (ICAC)*, 2011.

[11] A. Verma, L. Cherkasova, and R. H. Campbell. Resource Provisioning Framework for MapReduce Jobs with Performance Goals. *Proc. of the 12th ACM/IFIP/USENIX Middleware Conference*, 2011.

[12] T. White. *Hadoop:The Definitive Guide*. Page 6,Yahoo Press.

[13] J. Xie et al. Improving mapreduce performance through data placement in heterogeneous hadoop clusters. In *Proc. of the IPDPS Workshops: Heterogeneity in Computing*, 2010.

[14] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In *Proc. of EuroSys*. ACM, 2010.

[15] M. Zaharia et al. Improving mapreduce performance in heterogeneous environments. In *Proc. of OSDI*, 2008.

[16] Z. Zhang, L. Cherkasova, A. Verma, and B. T. Loo. Automated Profiling and Resource Management of Pig Programs for Meeting Service Level Objectives. In *Proc. of IEEE/ACM Intl. Conference on Autonomic Computing (ICAC)*, 2012.