Exploiting Cloud Heterogeneity to Optimize Performance and Cost of MapReduce Processing

Zhuoyao Zhang
Google Inc.
Mountain View, CA 94043, USA
zhuoyao@google.com

Ludmila Cherkasova
Hewlett-Packard Labs
Palo Alto, CA 94303, USA
lucy.cherkasova@hp.com

Boon Thau Loo
University of Pennsylvania
Philadelphia, PA 19104, USA
boonloo@cis.upenn.edu

ABSTRACT

Cloud computing offers a new, attractive option to customers for quickly provisioning any size Hadoop cluster, consuming resources as a service, executing their MapReduce workload, and then paying for the time these resources were used. One of the open questions in such environments is the right choice of resources (and their amount) a user should lease from the service provider. Typically, there is a variety of different types of VM instances in the Cloud (e.g., *small, medium,* or *large* EC2 instances). The capacity differences of the offered VMs are reflected in VM's pricing. Therefore, for the same price a user can get a variety of Hadoop clusters based on different VM instance types. We observe that the performance of MapReduce applications may vary significantly on different platforms. This makes a selection of the best cost/performance platform for a given workload a non-trivial problem, especially when it contains multiple jobs with different platform preferences. We aim to solve the following problem: given a completion time target for a set of MapReduce jobs, determine a *homogeneous* or *heterogeneous* Hadoop cluster configuration (i.e., the number, types of VMs, and the job schedule) for processing these jobs within a given deadline while minimizing the rented infrastructure cost. In this work,[1] we design an efficient and fast simulation-based framework for evaluating and selecting the right underlying platform for achieving the desirable Service Level Objectives (SLOs). Our evaluation study with Amazon EC2 platform reveals that for different workload mixes, an *optimized* platform choice may result in 45-68% cost savings for achieving the same performance objectives when using different (but seemingly equivalent) choices. Moreover, depending on a workload the heterogeneous solution may outperform the homogeneous cluster solution by 26-42%. We provide additional insights explaining the obtained results by profiling the performance characteristics of used applications and underlying EC2 platforms. The results of our simulation study are validated through experiments with Hadoop clusters deployed on different Amazon EC2 instances.

1. INTRODUCTION

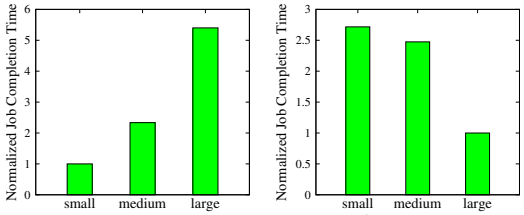
Cloud computing offers a new delivery model with virtually unlimited computing and storage resources. This is an attractive option for many users because acquiring, setting up, and maintaining a complex, large-scale infrastructure such as a Hadoop cluster requires a significant up-front investment in the new infrastructure, training new personnel, and then a continuous maintenance and management support, that can be difficult to justify. Cloud computing offers a compelling, cost-efficient approach that allows users to rent resources in a "pay-per-use" manner. For many users this creates an attractive and affordable alternative compared to acquiring and maintaining their own infrastructure.

A typical cloud environment provides a selection of different capacity Virtual Machines for deployment at different prices per time unit. For example, the Amazon EC2 platform offers a choice of *small, medium,* and *large* VM instances (among the other choices), where the CPU and RAM capacity of a *medium* VM instance is two times larger than the capacity of a *small* VM instance, and the CPU and RAM capacity of a *large* VM instance is two times larger than the capacity of a *medium* VM instance. This resource difference is also reflected in the price: the *large* instance is twice (four times) more expensive compared with the *medium (small)* VM instance. Therefore, a user is facing a variety of platform and configuration choices that can be obtained for the same cost.

To demonstrate the challenges in making an optimized platform choice we performed a set of experiments with two popular applications *TeraSort* and *KMeans*[2] on three Hadoop clusters[3]. deployed with different type VM instances:

- 40 *small* VMs, each configured with 1 map and 1 reduce slot;
- 20 *medium* VMs, each configured with 2 map and 2 reduce slots, and
- 10 *large* VMs, each configured with 4 map and 4 reduce slots.

Therefore, the three Hadoop clusters can be obtained for the same price per time unit, and they have the same number of map and reduce slots for processing (where each slot is provisioned with the same CPU and RAM capacities). Figure 2 shows the summary of our experiments with *TeraSort* and *KMeans*.



(a) *TeraSort* (b) *KMeans*

Figure 1: Normalized completion time of two applications executed on different types of EC2 instances.

Apparently, the Hadoop cluster with 40 *small* VMs provides the best completion time for a *TeraSort* application as shown in Fig-

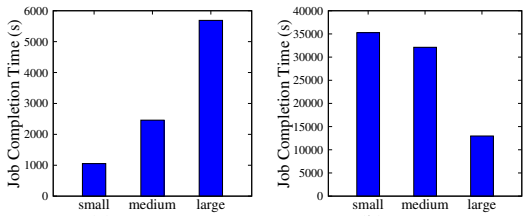
[1]This paper is an extended version of our ealier workshop paper [20]. This work was originated during Z. Zhang's internship at HP Labs. Prof. B. T. Loo is supported in part by NSF grants CNS-1117185 and CNS-0845552.

[2]In this work, we use a set of 13 applications released by the Tarazu project [2] with *TeraSort* and *KMeans* among them. Table 2 in Section 6 provides application details and corresponding job settings (the number of map/reduce tasks, datasets sizes, etc.)

[3]We use Hadoop 1.0.0 in all experiments in the paper.

ure 1 (a). The completion time of *TeraSort* on the cluster with *small* VMs is 5.5 (2.3) times better, i.e.,smaller, than on the cluster with *large* (*medium*) VMs. Since the cost of all three clusters per time unit is the same, the *shortest completion time results in the lowest monetary cost* the customer should pay. Therefore, the Hadoop cluster with 40 *small* VMs offers the best solution for *TeraSort*. By contrast, the Hadoop cluster with 10 *large* VMs is the best option for *KMeans* as shown in Figure 1 (b). It outperforms the Hadoop cluster with *small* VMs by 2.6 times when processing *KMeans*. This experiment demonstrates that seemingly equivalent platform choices for a Hadoop cluster in the Cloud might result in a different application performance that could lead to a different provisioning cost.

The problem of optimized platform choice becomes even more complex when a given workload contains multiple jobs with different performance preferences. Intuitively, if performance of jobs in the set would benefit from the *small* VMs (or *large* VMs) then the platform choice for a corresponding Hadoop cluster is relatively straightforward. However, if a given set of jobs has the applications with different performance preferences, then a platform choice becomes non-trivial. Figure 2 shows completion times (absolute, not normalized) of *TeraSort* and *KMeans* on three Hadoop clusters deployed with different type VM instances (these graphs resemble the normalized results shown in Figure 1). Apparently, when making a decision on the best platform for a Hadoop cluster to execute both of these applications (as a set) in the most cost effective way, one needs to look at the reduction of absolute execution times due to the choice of a common underlying platform.



(a) *TeraSort* (b) *KMeans*

Figure 2: Completion time of two applications executed on different types of EC2 instances.

Apparently, the absolute time benefits from processing *KMeans* on the *large* VMs significantly overweight the benefits of processing *TeraSort* on the *small* VMs.

In this work, we aim to solve the **problem of the platform choice** to provide the best cost/performance trade-offs for a given MapReduce workload and the Hadoop cluster. As shown in Figures 1, 2 this choice is non-trivial and depends on the application characteristics. The problem is even more difficult when the performance objective is to minimize the *makespan* (the overall completion time) of a given job set. In this work, we first offer a framework for solving the following two problems. Given a workload, select the type and size of the underlying platform for a *homogeneous* Hadoop cluster that provides best cost/performance trade-offs: *i)* minimizing the cost (budget) while achieving a given makespan target, or *ii)* minimizing the achievable jobs makespan for a given budget.

We also observe that a user might have additional considerations for a **case with node failure(s)**. Hadoop is designed to support fault-tolerance, i.e., it will finish job processing even in the case of a node failure by using the remaining resources and restarting/recomputing failed tasks. However, if the cluster is based on 40 *small* VM instances then a single node failure leads to a loss of 2.5% of the overall resources, and it impacts only a limited number of map and reduce tasks. While in the cluster based on 10 *large* VM instances, a single node failure leads to a loss of 10% of the overall resources and a much higher number of impacted map and reduce tasks. We provide an extension of the proposed framework for selecting and sizing a Hadoop cluster to support the job performance objectives in case of node failure(s) in the cluster.

In our earlier work [22], we discussed a framework for the optimized platform selection of a single homogeneous Hadoop cluster. However, a **homogeneous** cluster might not always present the best solution. Intuitively, if a given set of jobs has the applications with different platform preferences then a **heterogeneous** solution (that combines Hadoop clusters deployed with different instance types) might be a better choice. To support the choice of the heterogeneous solution, we introduce an application *preference ranking* to reflect the "strength" of application preference between different VM types and the possible impact on the provisioning cost (see our discussion related to the absolute completion times of *KMeans* and *TeraSort* shown in Figure 2). This *preference ranking* guides the construction of a heterogeneous solution. In the designed simulation-based framework, we collect jobs' profiles from a given set, create an optimized job schedule that minimizes jobs' makespan (as a function of job profiles and a cluster size), and then obtain the accurate estimates of the achievable makespan by replaying jobs' traces in the simulator. Based on the cost of the best homogeneous Hadoop cluster, we provide a quick walk through a set of heterogeneous solutions (and corresponding jobs' partitioning into different pools) to see whether there is a heterogeneous solution that can process given jobs within a deadline but at a smaller cost.

In our performance study, we use a set of 13 diverse MapReduce applications for creating three different workloads. Our experiments with Amazon EC2 platform reveal that for different workloads, an *optimized* platform choice may result in up to 45%-68% cost savings for achieving the same performance objectives when using different (but seemingly equivalent) choices. Moreover, depending on a workload the heterogeneous solution may outperform the homogeneous one by 26-42%. The results of our simulation study are validated through experiments with Hadoop clusters deployed on different Amazon EC2 instances.

The rest of the paper is organized as follows. Section 2 outlines our approach and explain details of the building blocks used in our solution. Section 3 described the general problem definition (two separate cases) for the homogeneous cluster case and outlines both solutions. Section 4 outlines the extension of the proposed framework for a case with node failure(s). Section 5 motivates the heterogeneous clusters solution and provides the corresponding provisioning algorithm. Section 6 presents the evaluation study by comparing the effectiveness of the proposed algorithms and their outcomes for different workloads. Section 7 outlines related work. Section 8 summarizes our contribution and gives directions for future work.

2. BUILDING BLOCKS

In this section, we outline our approach and explain details of the following building blocks used in our solution: *i)* collected job traces and job profiles; *ii)* an optimized job schedule to minimize the jobs' execution makespan; *iii)* the Map-Reduce simulator to replay the job traces according to the generated job schedule for obtaining the accurate estimates of jobs performance and cost values.

1) Job Traces and Profiles: In summary, the MapReduce job execution is comprised of two stages: map stage and reduce stage. The map stage is partitioned into *map tasks* and the reduce stage is partitioned into *reduce tasks*, and they are distributed and executed across multiple machines.

Each map task processes a logical split of the input data that generally resides on a distributed file system. The map task applies the user-defined map function on each record and buffers the resulting output. This intermediate data is hash-partitioned for the different

reduce tasks and written to the local hard disk of the worker executing the map task.

We use the past job run(s) for creating the job traces that contain recorded durations of all processed map and reduce tasks[4]. A similar job trace can be extracted from the Hadoop job tracker logs using tools such as Rumen [1]. The obtained map/reduce task distributions can be used for extracting the distribution parameters and generating scaled traces, i.e., generating the replayable traces of the job execution on the large dataset from the sample job execution on the smaller dataset as described in [13]. These job traces can be replayed using a MapReduce simulator [12] and used for creating the *compact job profile* for analytic models.

For predicting the job completion time we use a *compact job profile* that characterize the job execution during map, shuffle, and reduce phases via *average* and *maximum* task durations. The proposed MapReduce performance model [14] evaluates lower bounds T_J^{low} and upper bounds T_J^{up} on the job completion time. It is based the Makespan Theorem [13] 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 completion time of the entire n tasks 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 was shown in [14], the average of lower and upper bounds (T_J^{avg}) is a good approximation of the job completion time (typically, it is within 10%). Using this approach, we can estimate the duration of map and reduce stages of a given job as a function of allocated resources (i.e., on different size Hadoop clusters). In particular, we apply this analytic model in the process of building an optimized job schedule to minimize the overall jobs' execution time.

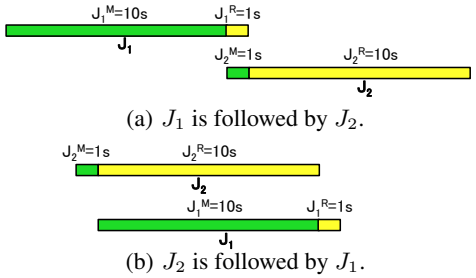
2) An Optimized Job Schedule: It was observed [15, 21] that for a set of MapReduce jobs (with no data dependencies between them) the order in which jobs are executed might have a significant impact on the jobs *makespan*, i.e., jobs overall completion time, and therefore, on the cost of the rented Hadoop cluster. For data-independent jobs, once the first job completes its map stage and begins the reduce stage, the next job can start executing its map stage with the released map resources in a pipelined fashion. There is an "overlap" in executions of map stage of the next job and the reduce stage of the previous one. As an illustration, let us consider two MapReduce jobs that have the following map and reduce stage durations:

- Job J_1 has a map stage duration of $J_1^M = 10s$ and the reduce stage duration of $J_1^R = 1s$.
- Job J_2 has a map stage duration of $J_2^M = 1s$ and the reduce stage duration of $J_2^R = 10s$.

There are two possible executions shown in Figure 3:

- J_1 is followed by J_2 shown in Figure 3(a). The reduce stage of J_1 overlaps with the map stage of J_2 leading to overlap of only $1s$. The total completion time of processing two jobs is $10s + 1s + 10s = 21s$.
- J_2 is followed by J_1 shown in Figure 3(b). The reduce stage of J_2 overlaps with the map stage of J_1 leading to a much better pipelined execution and a larger overlap of 10s. The total makespan is $1s + 10s + 1s = 12s$.

[4]The shuffle stage is included in the reduce task. For a first shuffle phase that overlaps with the entire map phase, only a complementary (non-overlapping) portion is included in the reduce task.



(a) J_1 is followed by J_2.

(b) J_2 is followed by J_1.

Figure 3: Impact of the job schedule on their completion time.

There is a significant difference in the jobs makespan (75% in the example above) depending on the execution order of the jobs.

Since in this work we consider a problem of minimizing the cost of rented Hadoop cluster and the jobs completion time directly impacts this cost, we aim to generate the job executions order that minimizes the jobs' makespan. Thus, let $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ be a set of n MapReduce jobs with no data dependencies between them. For minimizing the makespan of a given set of MapReduce jobs, we apply the classic Johnson algorithm [6] that was proposed for building an optimal job schedule in two-stage production systems. Johnson's schedule can be efficiently applied to minimizing the makespan of MapReduce jobs as it was shown in [15].

Let us consider a collection $\mathcal{J}$ of n jobs, where each job J_i is represented by the pair (m_i, r_i) of map and reduce stage durations respectively. Note, we can estimates m_i and r_i by using bounds-based model. Let us augment each job $J_i = (m_i, r_i)$ with an attribute D_i that is defined as follows:

$$D_i = \begin{cases} (m_i, \mathtt{m}) & \text{if } \min(m_i, r_i) = m_i, \\ (r_i, \mathtt{r}) & \text{otherwise.} \end{cases}$$

The first argument in D_i is called the *stage duration* and denoted as D_i^1. The second argument is called the *stage type* (map or reduce) and denoted as D_i^2.

Algorithm 1 shows how an optimal schedule can be constructed using Johnson's algorithm.

Algorithm 1 Johnson's Algorithm

Input: A set $\mathcal{J}$ of n MapReduce jobs. D_i is the attribute of job J_i as defined above.
Output: Schedule σ (order of jobs execution.)

```
1: Sort the original set J of jobs into the ordered list L using their stage
   duration attribute D_i^1
2: head ← 1, tail ← n
3: for each job J_i in L do
4:     if D_i^2 = m then
5:         // Put job J_i from the front
6:         σ_head ← J_i, head ← head + 1
7:     else
8:         // Put job J_i from the end
9:         σ_tail ← J_i, tail ← tail - 1
10:    end if
11: end for
```

First, we sort all the n jobs from the original set $\mathcal{J}$ in the ordered list L in such a way that job J_i precedes job J_{i+1} if and only if $min(m_i, r_i) \leq min(m_{i+1}, r_{i+1})$. In other words, we sort the jobs using the stage duration attribute D_i^1 in D_i (it represents the smallest duration of the two stages). Then the algorithm works by taking jobs from list L and placing them into the schedule σ from the both ends (head and tail) and proceeding towards the middle. If the stage type in D_i is $\mathtt{m}$, i.e., represents the map stage, then the job J_i is placed from the head of the schedule, otherwise from the tail. The complex-

ity of Johnson's Algorithm is dominated by the sorting operation and thus is $\mathcal{O}(n \log n)$.

3) MapReduce Simulator: Since the users rent Cloud resources in a "pay-per-use" fashion, it is important to accurately estimate the execution time of a given set of jobs according to a generated Johnson schedule on a Hadoop cluster of a given size. In this work, we use the enhanced version of MapReduce simulator SimMR [12]. 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 [12]. Moreover, SimMR is a very fast simulator: it can process over one million events per second. Therefore, we can quickly explore the entire solution space (in brute-force search manner).

The main structure of SimMR is shown in Figure 4.

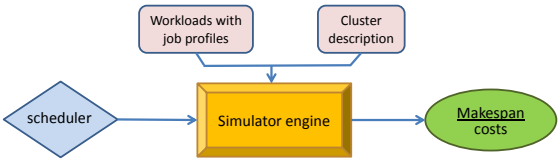


Figure 4: MapReduce Simulator SimMR.

The basic blocks of the simulator are the following:

1. *Trace Generator* – a module that generates a replayable workload trace. This trace is generated either from the detailed job profile (provided by the Job Profiler) or by feeding the distribution parameters for generating the synthetic trace (this path is taken when we need to generate the job execution traces from the sampled executions on the smaller datasets).

2. *Simulator Engine* – a discrete event simulator that takes the cluster configuration information and accurately emulates the Hadoop job master decisions for map/reduce slot allocations across multiple jobs.

3. *Scheduling policy* – a scheduling module that dictates the jobs' ordering and the amount of allocated resources to different jobs over time.

Thus, for a given Hadoop cluster size, given set of jobs, and generated Johnson's schedule, the simulator can accurately estimate the jobs' completion time (makespan) by replaying the job traces accordingly to the generated schedule.

3. HOMOGENEOUS CLUSTER SOLUTION

In this work, we consider *the following two problems* for the homogeneous cluster case.

- For a given workload defined as a set of jobs $\mathcal{W} = \{J_1, J_2, \ldots, J_n\}$ to be processed within deadline $\mathcal{D}$, determine a Hadoop cluster configuration (i.e., the number and types of VM instances, and the job schedule) for processing these jobs within a given deadline while minimizing the monetary cost for rented infrastructure.

- For a given workload $\mathcal{W} = \{J_1, J_2, \ldots, J_n\}$ and a given a customer budget $\mathcal{B}$, determine a Hadoop cluster configuration (i.e., the number and types of VM instances, and the job schedule) for processing these jobs within an allocated monetary cost for rented infrastructure while minimizing the jobs' processing time.

Our solution is based on a simulation framework: in a brute-force manner, it searches through the entire solution space by exhaustively enumerating all possible candidates for the solution and checking whether each candidate satisfies the required problem's statement. Figure 5 shows the diagram for the framework execution in decision making process per selected platform type. For example, if the platforms of interest are *small, medium,* and *large* EC2 VM instances then the framework will generate three trade-off curves. For each platform and a given Hadoop cluster size, the *Job Scheduler* component generates the optimized MapReduce job schedule. Then the jobs' makespan is obtained by replaying the job traces in the simulator according to the generated schedule. After that the size of the cluster is increased by one instance (in the cloud environment, it is equivalent to adding a node to a Hadoop cluster) and the iteration is repeated: a new job schedule is generated and its makespan is evaluated with the simulator, etc. We have a choice of *stop* conditions for iterations: either a user can set a range of values for the cluster size N_{max}^{type} (driven by budget $\mathcal{B}$, which a customer intends to spend), or at some point, the increased cluster size does not improve the achievable makespan. The latter condition typically happens when the Hadoop cluster is large enough to accommodate all the jobs to be executed concurrently, and therefore, the increased cluster size cannot improve the jobs makespan.

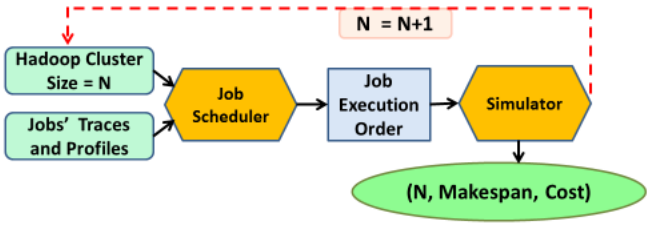


Figure 5: Outline of the homogeneous cluster solution.

Assume that a set of given jobs should be processed within deadline $\mathcal{D}$, and let $Price^{type}$ be the price of a *type* VM instance per time unit. Then a customer with budget $\mathcal{B}$ can rent N_{max}^{type} of VMs instances of a given *type*:

$$N_{max}^{type} = \mathcal{B}/(\mathcal{D} \cdot Price^{type}) \quad (1)$$

Algorithm 2 shows the pseudo code to determine the size of a cluster which is based on the *type* VM instances for processing $\mathcal{W}$ with **deadline** $\mathcal{D}$ and which results in the minimal monetary cost.

The algorithm iterates through the increasing number of instances for a Hadoop cluster. It simulates the completion time of workload $\mathcal{W}$ processed with Johnson's schedule on a given size cluster and computes the corresponding cost (lines 2-6). Note, that k defines the number of *worker nodes* in the cluster. The overall Hadoop cluster size is $k + 1$ nodes (we add a dedicated node for Job Tracker and Name Node, which is included in the *cost*). The min_cost^{type} keeps track of a minimal cost so far (lines 7-8) for a Hadoop cluster which can process $\mathcal{W}$ within deadline $\mathcal{D}$.

One of the reader questions might be why do we continue iterating through the increasing number of instances once we found a solution which can process $\mathcal{W}$ within deadline $\mathcal{D}$? At a glance, when we keep increasing the Hadoop cluster – the solution becomes more expensive cost-wise. In reality, it is not always true: where could be a situation when a few additional nodes and a different Johnson schedule might significantly improve the makespan of a given workload, and as a result the cost of this larger cluster is smaller (due

Algorithm 2 Provisioning solution for a **homogeneous** cluster to process $\mathcal{W}$ with **deadline** $\mathcal{D}$ while minimizing the cluster cost

Input:
$\mathcal{W} = \{J_1, J_2, ...J_n\} \leftarrow$ workload with traces and profiles for each job;
$type \leftarrow$ VM instance type, e.g., *type*$\in$ {*small, medium, large*};
$N_{max}^{type} \leftarrow$ the maximum number of instances to rent;
$Price^{type} \leftarrow$ unite price of a *type* VM instance;
$\mathcal{D} \leftarrow$ a given time deadline for processing $\mathcal{W}$.
Output:
$N^{type} \leftarrow$ an optimized number of VM *type* instances for a cluster;
$min\_cost^{type} \leftarrow$ the minimal monetary cost for processing $\mathcal{W}$.

$min\_cost^{type} \leftarrow \infty$
for $k \leftarrow 1$ **to** N_{max}^{type} **do**
  // Simulate completion time for processing workload $\mathcal{W}$ with k VMs
  $Cur\_CT = Simulate(type, k, \mathcal{W})$
  // Calculate the corresponding monetary cost
  $cost = Price^{type} \times (k+1) \times Cur\_CT$
  if $Cur\_CT \leq \mathcal{D}$ & $cost < min\_cost^{type}$ **then**
    $min\_cost^{type} \leftarrow cost,\ N^{type} \leftarrow k$
  end if
end for

to significantly improved workload completion time). Later, in the evaluation section, we will show examples of such situations.

We apply Algorithm 2 to different types of VM instances, e.g., *small, medium,* and *large* respectively. After that we compare the produced outcomes and make a final provisioning decision.

In a similar way, we can solve a related problem, when for a given a customer budget $\mathcal{B}$, we need to determine a Hadoop cluster configuration for processing a given workload $\mathcal{W}$ within an allocated monetary cost for rented infrastructure while minimizing the jobs' processing time.

Algorithm 3 shows the pseudo code to determine the size of a cluster which is based on the *type* VM instances for processing $\mathcal{W}$ with a monetary **budget** $\mathcal{B}$ and which results in the minimal workload processing.

Algorithm 3 Provisioning solution for a **homogeneous** cluster to process $\mathcal{W}$ with a **budget** $\mathcal{B}$ while minimizing the processing time

Input:
$\mathcal{W} = \{J_1, J_2, ...J_n\} \leftarrow$ workload with traces and profiles for each job;
$type \leftarrow$ VM instance type, e.g., *type*$\in$ {*small, medium, large*};
$N_{max}^{type} \leftarrow$ the maximum number of instances to rent;
$Price^{type} \leftarrow$ unite price of a *type* VM instance;
$\mathcal{B} \leftarrow$ a given monetary budget for processing $\mathcal{W}$.
Output:
$N^{type} \leftarrow$ an optimized number of VM *type* instances for a cluster;
$min\_CT^{type} \leftarrow$ the minimal completion time for processing $\mathcal{W}$.

$min\_CT^{type} \leftarrow \infty$
for $k \leftarrow 1$ **to** N_{max}^{type} **do**
  // Simulate completion time for processing workload $\mathcal{W}$ with k VMs
  $Cur\_CT = Simulate(type, k, \mathcal{W})$
  // Calculate the corresponding monetary cost
  $cost = Price^{type} \times (k+1) \times Cur\_CT$
  if $Cur\_CT < min\_CT^{type}$ & $cost \leq \mathcal{B}$ **then**
    $min\_CT^{type} \leftarrow Cur\_CT,\ N^{type} \leftarrow k$
  end if
end for

Algorithms 2 and 3 follow a similar structure: in a brute-force manner, they search through the entire solution space by exhaustively enumerating all possible candidates for the solution and checking whether each candidate satisfies the required problem's statement.

4. GENERAL CASE WITH NODE FAILURES

The application performance of a customer workload may vary significantly on different platforms. Seemingly equivalent platform choices for a Hadoop cluster in the Cloud might result in a different application performance and a different provisioning cost, which leads to the problem of an optimized platform choice that can be obtained for the same budget. Moreover, there could be an **additional issue** for the user to consider: the impact of node failures on the choice of the underlying platform for a Hadoop cluster. Hadoop is designed to support fault-tolerance, i.e., it finishes job processing even in case with node failures. It uses the remaining resources for restarting and recomputing failed tasks. Let us consider a motivating example described in Section 1, where a user may deploy three different clusters with different types of VM instances for the same budget:

- 40 *small* VMs, each configured with 1 map and 1 reduce slot;
- 20 *medium* VMs, each configured with 2 map and 2 reduce slots, and
- 10 *large* VMs, each configured with 4 map and 4 reduce slots.

Now, let us see how a node failure may impact cluster performance when Hadoop nodes are based on different VM types. If a Hadoop cluster is based on 40 small instances then a single node failure leads to a loss of 2.5% of the overall resources and only limited number of map and reduce tasks might be impacted. While in the cluster based on 10 large instances a single node failure leads to a loss of 10% of the overall resources and a much higher number of map and reduce tasks might be impacted.

For a business-critical, production workload $\mathcal{W}$, a user may consider the generalized service level objectives (SLOs) that include two separate conditions:

- a desirable completion time $\mathcal{D}$ for the entire set of jobs in the workload $\mathcal{W}$ under normal conditions;
- an acceptable degraded completion time $\mathcal{D}_{deg}$ for processing $\mathcal{W}$ in case of 1-node failure.

So, the **problem** is to determine a Hadoop cluster configuration (i.e., the number and types of VM instances, and the job schedule) for processing workload $\mathcal{W}$ with the makespan target $\mathcal{D}$ while minimizing the cost, such that the chosen solution also supports a degraded makespan $\mathcal{D}_{deg}$ in case of 1-node failure during the jobs processing.

The approach, proposed in the previous Section 3, can be generalized for the case with node failures. Figure 6 shows the extended diagram for the framework execution and decision making process per selected platform type with node failures. For example, if the platforms of interest are *small, medium,* and *large* EC2 VM instances then the framework will generate three different trade-off sets. For each platform and a given Hadoop cluster size N, the *Job Scheduler* component generates an optimized MapReduce job schedule, based on Johnson's algorithm. Then the jobs' makespan (in the normal mode) is obtained by replaying the job traces in the simulator according to the generated schedule. In parallel (see, the lower branch in Figure 6, that represents a case of 1-node failure), the jobs' makespan is obtained by replaying the job traces according to the same generated job schedule in the decreased cluster size $N-1$. After both branches are finished, the size of the cluster is increased by one instance (in the cloud environment, it is equivalent to adding a node to a Hadoop cluster) and the iteration is repeated: a new job schedule is generated and its makespan is evaluated with the simulator for both modes: normal and 1-node failure, etc.

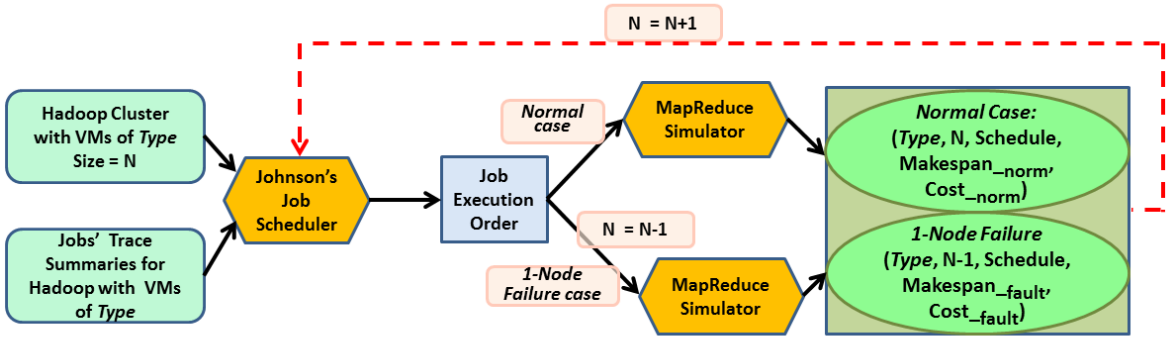


Figure 6: Solution Outline.

The cluster size (for each instance type) is selected based on validity of both SLOs conditions: for normal execution to satisfy $\mathcal{D}$ and in case of 1-node failure to support a degraded makespan $\mathcal{D}_{deg}$. Algorithm 4 shows the pseudo-code to determine the size of a cluster which is based on the *type* VM instances for processing $\mathcal{W}$ with generalized SLOs and that results in the minimal monetary cost[5].

Algorithm 4 Provisioning solution for a **homogeneous** cluster to process $\mathcal{W}$ with a **deadline** $\mathcal{D}$ and with a degraded **deadline** $\mathcal{D}_{deg}$ in case of 1-node failure while minimizing the cluster cost

Input:
$\mathcal{W} = \{J_1, J_2, ...J_n\} \leftarrow$ workload with traces and profiles for each job;
$type \leftarrow$ VM instance type, e.g., $type \in$ {*small, medium, large*};
$N_{max}^{type} \leftarrow$ the maximum number of instances to rent;
$Price^{type} \leftarrow$ unite price of a *type* VM instance;
$\mathcal{D} \leftarrow$ a given time deadline for processing $\mathcal{W}$,
$\mathcal{D}_{deg} \leftarrow$ a given degraded deadline for processing $\mathcal{W}$ with 1-node failure.
Output:
$N^{type} \leftarrow$ an optimized number of VM *type* instances for a cluster;
$min\_cost^{type} \leftarrow$ the minimal monetary cost for processing $\mathcal{W}$.

1: $min\_cost^{type} \leftarrow \infty$
2: **for** $k \leftarrow 1$ **to** N_{max}^{type} **do**
3: // Simulate completion time for processing workload $\mathcal{W}$ with k VMs
4: $Cur\_CT = Simulate(type, k, \mathcal{W})$
5: // Simulate processing $\mathcal{W}$ in a degraded mode with $(k-1)$ VMs
6: $Cur\_CT_{deg} = Simulate_{deg}(type, k-1, \mathcal{W})$
7: // Calculate the corresponding monetary cost
8: $cost = Price^{type} \times (k+1) \times Cur\_CT$
9: **if** $cur\_CT \leq \mathcal{D}$ & $cur\_CT_{deg} \leq \mathcal{D}_{deg}$ & $cost < min\_cost^{type}$ **then**
10: $min\_cost^{type} \leftarrow cost$, $N^{type} \leftarrow k$
11: **end if**
12: **end for**

The algorithm iterates through the increasing number of instances for a Hadoop cluster. It simulates the completion time of workload $\mathcal{W}$ processed with Johnson's schedule on a cluster of a given size k. Note, that the same job schedule is used for processing $\mathcal{W}$ in case of a node failure, i.e., on a cluster of size $k-1$.

The overall Hadoop cluster size is $k+1$ nodes (k defines the number of *worker nodes* in the cluster, and we add a dedicated node for Job Tracker and Name Node, which is included in the *cost*). The min_cost^{type} keeps track of a minimal cost so far (lines 7-8) for a Hadoop cluster which can process $\mathcal{W}$ within deadline $\mathcal{D}$ under normal conditions and within deadline $\mathcal{D}_{deg}$ in case of 1-node failure.

[5]The proposed solution can be generalized for a case with multiple node failures.

5. HETEROGENEOUS SOLUTION

In Section 1, we discussed a motivating example by analyzing *TeraSort* and *KMeans* performance on Hadoop clusters formed with different EC2 instances, and observing that these applications benefit from different types of VMs as their preferred choice. Therefore, a single homogeneous cluster might not always be the best choice for a workload mix with different applications, and a heterogeneous solution might offer a better cost/performance outcome.

However, a single (individual) application preference choice often depends on the size of a Hadoop cluster and given performance goals. Continuing the motivating example from Section 1, Figure 7 shows the trade-off curves for three representative applications *TeraSort, Kmeans*, and *AdjList*[6] obtained as a result of exhaustive simulation of application completion times on different size Hadoop clusters. The Y-axis represents the job completion time while the X-axis shows the corresponding monetary cost. Each figure shows three curves for application processing by a homogeneous Hadoop cluster based on *small, medium*, and *large* VM instances respectively.

First of all, the same application can result in different completion times when being processed on the same platform at the same cost. This reflects an interesting phenomenon of "pay-per-use" model. There are situations when a cluster of size N processes a job in T time units, while a cluster of size $2 \cdot N$ may process the same job in $T/2$ time units. Interestingly, these two different size clusters have the same cost, and if the purpose is meeting deadline $\mathcal{D}$ where $T \leq \mathcal{D}$ then both clusters meet the performance objective.

Second, we can see an orthogonal observation: in many cases, the same completion time can be achieved at a different cost (on the same platform type). Typically, this corresponds to the case when an increased size Hadoop cluster does not further improve the job processing time.

Finally, according to Figure 7, we can see that for *TeraSort*, the *small* instances results in the best choice, while for *Kmeans* the *large* instances represent the most cost-efficient platform. However, the optimal choice for *AdjList* is not very clear, it depends on the deadline requirements, and the trade-off curves are much closer to each other than for *TeraSort* and *Kmeans*.

Another important point is that the cost savings vary across different applications, e.g., the execution of *Kmeans* on *large* VM instances leads to higher cost savings than the execution of *TeraSort* on *small* VMs. Thus, if we would like to partition a given workload $\mathcal{W} = \{J_1, J_2, ...J_n\}$ into two groups of applications each to be executed by a Hadoop cluster based on different type VM instances,

[6]Table 2 in Section 6 provides details about these applications and their job settings.

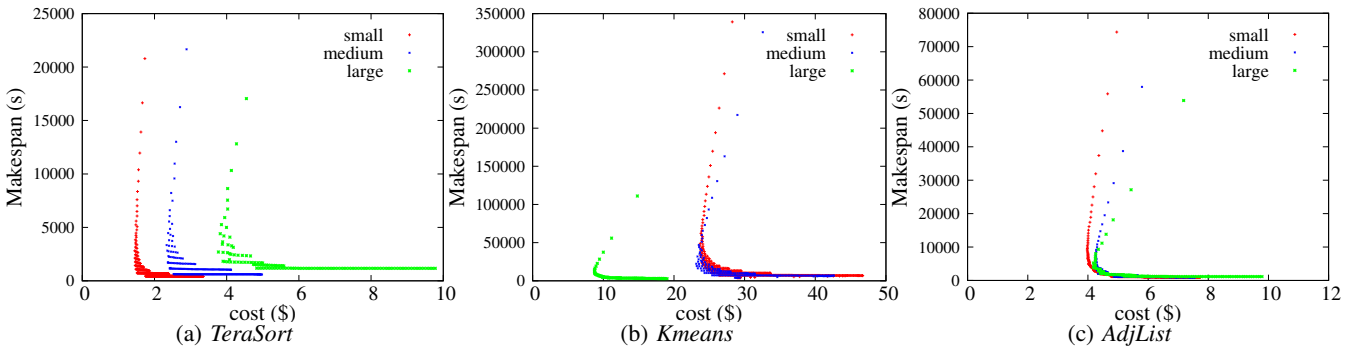


Figure 7: Performance versus cost trade-offs for different applications.

we need to be able to **rank** (order) these application with respect to their preference "strength" between two considered platforms.

In this work, we consider a heterogeneous solution that consists of two homogeneous Hadoop sub-clusters deployed with different type VM instances[7]. As an example, we consider a heterogeneous solution formed by *small* (S) and *large* (L) VM instances. To measure the "strength" of application preference between two different VM types we introduce an application **preference score** $PScore^{S,L}$ defined as a difference between the normalized costs of simulated cost-performance curves (such as shown in Figure 7):

$$PScore^{S,L} = \frac{\sum_{1 \le i \le N^S_{max}} Cost^S_i}{N^S_{max}} - \frac{\sum_{1 \le i \le N^L_{max}} Cost^L_i}{N^L_{max}} \quad (2)$$

where N^S_{max} and N^L_{max} are defined by Eq. 1 for Hadoop clusters with *small* and *large* VM type instances respectively.

The value of $PScore^{S,L}$ indicates the possible impact on the provisioning cost, i.e, a large negative (positive) value indicates a stronger preference of *small* (*large*) VM instances, while values closer to 0 reflect less sensitivity to the platform choice.

For optimized heterogeneous solution, we need to determine the following parameters:

- The number of instances for each sub-cluster (i.e., the number of worker nodes plus a dedicated node to host JobTracker and Name Node for each sub-cluster).
- The subset of applications to be executed on each cluster.

Algorithm 5 shows the pseudo code of our heterogeneous solution. For a presentation simplicity, we show the code for a heterogeneous solution with *small* and *large* VM instances.

First, we sort the jobs in the ascending order according to their preference ranking $PScore^{S,L}$. Thus the jobs in the beginning of the list have a performance preference for executing on the *small* instances. Then we split the ordered job list into two subsets: first one to be executed on the cluster with *small* instances and the other one to be executed on the cluster with *large* instances (lines 4-5). For each group, we use **Algorithm 2** for homogeneous cluster provisioning to determine the optimized size of each sub-cluster for processing the assigned workload with a deadline $\mathcal{D}$) that leads to the minimal monetary cost (lines 6-7). We consider all possible splits by iterating through the split point from 1 to the total number of jobs N and use a variable min_cost^{S+L} to keep track of the found minimal total cost, i.e, the sum of costs from both sub-clusters (lines 9-12).

[7]The designed framework can be generalized for a larger number of clusters. However, this might significantly increase the algorithm complexity without adding new performance benefits.

Algorithm 5 Provisioning solution for **heterogeneous** clusters to process $\mathcal{W}$ with a **deadline** $\mathcal{D}$ while minimizing the clusters cost

Input:
$\mathcal{W} = \{J_1, J_2, ...J_n\} \leftarrow$ workload with traces and profiles, where jobs are sorted in ascending order by their preference score $PScore^{S,L}$;
$\mathcal{D} \leftarrow$ a given time deadline for processing $\mathcal{W}$.
Output:
$N^S \leftarrow$ number of *small* instances;
$N^L \leftarrow$ number of *large* instances;
$\mathcal{W}^S \leftarrow$ List of jobs to be executed on *small* instance-based cluster;
$\mathcal{W}^L \leftarrow$ List of jobs to be executed on *large* instance-based cluster;
$min\_cost^{S+L} \leftarrow$ the minimal monetary cost of heterogeneous clusters.

1: $min\_cost^{S+L} \leftarrow \infty$
2: **for** $split \leftarrow 1$ **to** $n-1$ **do**
3: // Partition workload $\mathcal{W}$ into 2 groups
4: $Jobs^S \leftarrow J_1, ..., J_{split}$
5: $Jobs^L \leftarrow J_{split+1}, ..., J_n$
6: $(\tilde{N^S}, min\_cost^S) =$ **Algorithm 2**$(Jobs^S, small, \mathcal{D})$
7: $(\tilde{N^L}, min\_cost_L) =$ **Algorithm 2**$(Jobs^L, large, \mathcal{D})$
8: $total\_cost \leftarrow min\_cost^S + min\_cost^L$
9: **if** $total\_cost < min\_cost^{S+L}$ **then**
10: $min\_cost^{S+L} \leftarrow total\_cost$
11: $\mathcal{W}^S \leftarrow Jobs^S$, $\mathcal{W}^L \leftarrow Jobs^L$
12: $N^S \leftarrow \tilde{N^S}$, $N^L \leftarrow \tilde{N^L}$
13: **end if**
14: **end for**

6. EVALUATION

In this section, we describe the experimental testbeds and MapReduce workloads used in our study. We analyze the application performance and the job profiles when these applications are executed on different platforms of choice, e.g., *small, medium,* and *large* Amazon EC2 instances. The study aims to evaluate the effectiveness of the proposed algorithms for selecting the optimized platform for a Hadoop cluster and compare the outcomes for different workloads.

6.1 Experimental testbeds and workloads

In our experiments, we use the Amazon EC2 platform. It offers different capacity Virtual Machines (VMs) for deployment at different price. Table 1 provides descriptions of VM instance types used in our experiments. As it shows, the compute and memory capacity of a *medium* VM instance (*m1.medium*) is doubled compared to a *small* VM instance (*m1.small*) and similarly, a *large* VM instance (*m1.large*) has a doubled capacity compared to the *medium* VM. These differences are similarly reflected in pricing. We deployed Hadoop clusters that are configured with different number of map and reduce slots per different type VM instances (according to the

capacity) as shown in Table 1. Each VM instance is deployed with 100GB of Elastic Block Storage (EBS). 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.

Instance type	price	*CPU capacity (relative)*	RAM (GB)	#m,r slots
Small	\$0.06 ph	1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit)	1.7	1, 1
Medium	\$0.12 ph	2 EC2 Compute Unit (1 virtual core with 2 EC2 Compute Units)	3.75	2, 2
Large	\$0.24 ph	4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units)	7.5	4, 4

Table 1: EC2 Testbed description.

In the performance study, we use a set of 13 applications released by the Tarazu project [2]. Table 2 provides a high-level summary of the applications with the corresponding job settings (e.g., the number of map/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, which result in different resource requirements. 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).

Application	*Input data (type)*	*Input data (GB)*	*Interm data (GB)*	*Output data (GB)*	*#map,red tasks*
1. TeraSort	Synthetic	31	31	31	495, 240
2. WordCount	Wikipedia	50	9.8	5.6	788, 240
3. Grep	Wikipedia	50	1	$1x10^{-8}$	788, 1
4. InvIndex	Wikipedia	50	10.5	8.6	788, 240
5. RankInvIndex	Wikipedia	46	48	45	745, 240
6. TermVector	Wikipedia	50	4.1	0.002	788, 240
7. SeqCount	Wikipedia	50	45	39	788, 240
8. SelfJoin	Synthetic	28	25	0.014	448, 240
9. AdjList	Synthetic	28	29	29	508, 240
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, 50
13. KMeans	Netflix	27	27	27	428, 50

Table 2: Application characteristics.

6.2 Application performance analysis

We execute the set of 13 applications shown in Table 2 on three Hadoop clusters[8] deployed with different types of EC2 VM instances (they can be obtained for the same price per time unit): *i)* 40 *small* VMs, *ii)* 20 *medium* VMs, and *iii)* 10 *large* VM instances. We configure these Hadoop clusters according to their nodes capacity as shown in Table 1, with 1 additional instance deployed as the NameNode and JobTracker.

These experiments pursue the following goals: *i)* to demonstrate the performance impact of executing these applications on the Hadoop clusters deployed with different EC2 instances; and 2) to collect the detailed job profiles for creating the job traces used for replay by the simulator and trade-off analysis in determining the optimal platform choice.

[8]All the experiments are performed five times, and the measurement results are averaged. This comment applies to all the results.

Figure 8 (a) presents the completion times (CT) of 13 applications executed on the three different EC2-based clusters. The results show that the platform choice may significantly impact the application processing time. Note, we break the Y-axis as *KMeans* and *Classification* executions take much longer time to finish compared to other applications. Figure 8 (b) shows the normalized results with respect to the execution time of the same job on the Hadoop cluster formed with *small* VM instances. For 7 out 13 applications, the Hadoop cluster formed with *small* instances leads to the best completion time (and the smallest cost). However, for the CPU-intensive applications such as *Classification* and *KMeans*, the Hadoop cluster formed with *large* instances shows better performance.

Tables 3-5 summarize the job profiles collected for these applications. They show the average and maximum durations for the map, shuffle and reduce phase processing as well as the standard deviation for these phases. The analysis of the job profiles show that the shuffle phase durations of the Hadoop cluster formed with *large* instances are much longer compared to the clusters formed with *small* instances. The reason is that the Amazon EC2 instance scaling is done with respect to the CPU and RAM capacity, while the storage and network bandwidth is only fractionally improved. As we configure a higher number of slots on *large* instances, it increases the I/O and network contention among the tasks running on the same instance, and it leads to significantly increased durations of the shuffle phase. At the same time, the map task durations of most applications executed on the Hadoop cluster with *large* instances are significantly improved, e.g., the map task durations of *Classification* and *KMeans* applications improved almost three times.

The presented analysis of job profiles show that a platform choice for a Hadoop cluster may have a significant impact on the application performance. This analysis further demonstrates the importance of an effective mechanism and algorithms for helping to make the right provisioning decisions based on the workload characteristics.

6.3 Comparison of homogeneous and heterogeneous solutions

In this section, we use workloads created from the applications shown in Table 2 for comparing the results of both homogeneous and heterogeneous provisioning solutions. The following Table 6 provides an additional application characterization by reflecting the application preference score $PScore^{S,L}$. A positive value (e.g, *Kmeans, Classification*) indicates that the application is more cost-efficient on *large* VMs, while a negative value (e.g., *TeraSort, Wordcount*) means that the application favors *small* VM instances. The absolute score value is indicative of the preference "strength". When the preference score is close to 0 (e.g., *Adjlist*), it means that the application does not have a clear preference between the instance types.

Application	$PScore^{S,L}$
1. TeraSort	-3.74
2. WordCount	-5.96
3. Grep	-3.30
4. InvIndex	-7.90
5. RankInvIndex	-5.13
6. TermVector	3.11
7. SeqCount	-4.23
8. SelfJoin	-5.41
9. AdjList	-0.7
10. HistMovies	-1.64
11. HistRatings	-2.53
12. Classification	19.59
13. KMeans	18.6

Table 6: Application Preference Score.

We perform our case studies with three workloads $\mathcal{W}1$, $\mathcal{W}2$ and $\mathcal{W}3$ described as follows:

- $\mathcal{W}1$ – it contains all 13 applications shown in Table 2.

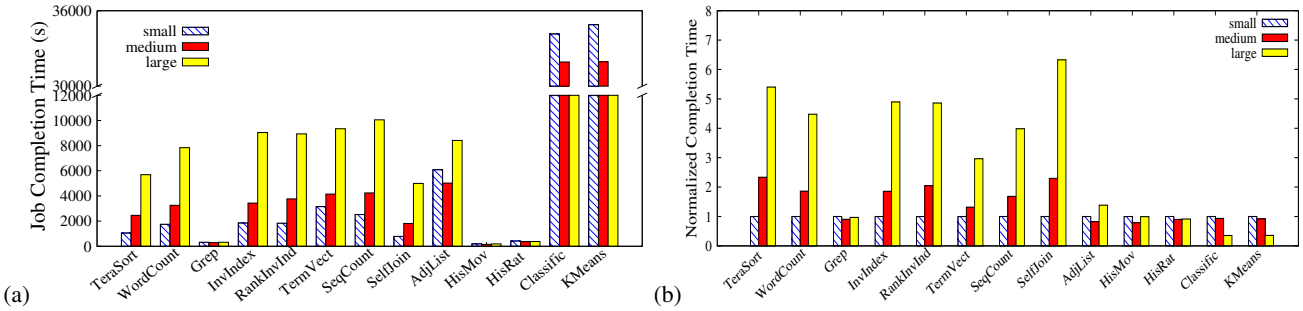


Figure 8: Job completion times (CT) on different EC2-based clusters: (a) absolute CT, (b) normalized CT.

Application	*avgMap*	*maxMap*	*avgShuffle*	*maxShuffle*	*avgReduce*	*maxReduce*	*map* STDEV	*shuffle* STDEV	*reduce* STDEV
TeraSort	29.1	46.7	248.5	317.5	31.2	41.3	0.82%	4.51%	0.97%
WordCount	71.5	147.0	218.7	272.0	12.1	22.4	1.16%	5.83%	3.68%
Grep	19.0	51.4	125.7	125.7	4.5	4.5	1.19%	26.43%	10.53%
InvIndex	83.9	170.0	196.8	265.2	18.2	27.6	1.33%	8.03%	3.96%
RankInvIndex	35.4	68.8	376.0	479.0	81.9	102.0	1.05%	3.79%	0.81%
TermVector	98.9	160.3	360.0	1239.5	137.2	1110.7	0.78%	2.45%	2.45%
SeqCount	101.2	230.0	256.8	454.2	54.1	82.3	1.01%	3.63%	6.62%
SelfJoin	11.9	20.4	217.9	246.5	12.3	21.4	0.70%	4.87%	3.12%
AdjList	265.9	415.9	72.7	121.8	291.1	398.1	1.53%	6.57%	0.84%
HistMovies	17.9	49.2	138.9	138.9	3.4	3.4	1.49%	40.85%	34.84%
HistRating	58.9	109.8	111.8	111.8	4.8	4.8	2.10%	35.58%	22.41%
Classif	3147.3	4047.2	58.5	61.5	4.0	6.9	1.21%	12.76%	3.13%
Kmeans	3155.9	3618.5	80.4	201.9	87.5	480.9	0.32%	30.09%	11.43%

Table 3: Job profiles on the EC2 cluster with small instances (time in sec)

Application	*avgMap*	*maxMap*	*avgShuffle*	*maxShuffle*	*avgReduce*	*maxReduce*	*map* STDEV	*shuffle* STDEV	*reduce* STDEV
TeraSort	36.9	46.1	466.3	553.1	26.5	34.3	1.06%	14.07%	1.21%
WordCount	83.0	127.4	562.4	771.6	11.6	23.0	0.48%	7.01%	9.09%
Grep	23.8	56.7	256.6	256.6	3.2	3.2	4.95%	24.13%	9.48%
InvIndex	101.0	150.4	449.5	536.3	13.6	20.2	0.52%	8.65%	1.62%
RankInvIndex	45.7	81.1	741.6	876.5	64.0	77.2	0.63%	9.40%	2.77%
TermVector	128.1	189.3	432.4	1451.4	71.9	576.4	0.23%	7.08%	2.81%
SeqCount	126.8	251.0	482.1	557.1	35.0	43.2	0.52%	21.70%	14.98%
SelfJoin	11.1	18.2	408.1	475.1	11.2	19.9	0.92%	13.86%	1.65%
AdjList	270.1	420.0	163.2	221.6	206.4	281.8	2.74%	8.70%	1.16%
HistMovies	20.1	47.7	246.7	246.7	3.7	3.7	3.14%	26.39%	17.04%
HistRating	71.7	103.7	240.4	240.4	5.0	5.0	0.23%	31.39%	14.22%
Classif	3013.8	4074.3	177.2	211.8	3.9	6.1	0.82%	44.03%	4.33%
Kmeans	2994.0	3681.2	189.7	392.1	51.7	280.4	3.93%	80.84%	6.96%

Table 4: Job profiles on the EC2 cluster with medium instances (time in sec)

Application	*avgMap*	*maxMap*	*avgShuffle*	*maxShuffle*	*avgReduce*	*maxReduce*	*map* STDEV	*shuffle* STDEV	*reduce* STDEV
TeraSort	27.3	55.7	806.4	1128.4	20.0	70.6	0.66%	7.78%	16.14%
WordCount	54.7	126.3	1028.6	1163.9	12.9	59.2	4.33%	10.24%	9.15%
Grep	18.3	59.7	791.8	791.8	4.3	4.3	3.50%	16.48%	22.81%
InvIndex	61.8	180.4	1152.6	1374.5	14.9	61.7	6.47%	5.10%	8.68%
RankInvIndex	28.3	71.5	1155.8	1308.6	40.5	88.5	1.49%	9.20%	8.19%
TermVector	85.3	194.7	1007.6	1573.9	30.2	259.2	3.88%	5.98%	10.04%
SeqCount	62.0	117.5	1046.1	1283.2	37.6	90.9	1.51%	6.70%	2.10%
SelfJoin	16.4	32.4	1015.7	1235.9	18.5	88.7	1.93%	4.86%	19.11%
AdjList	149.0	311.3	436.9	531.5	149.1	348.1	0.56%	13.34%	2.78%
HistMovies	22.3	80.2	724.2	724.2	5.2	5.2	6.97%	22.46%	17.25%
HistRating	51.4	187.8	628.6	628.6	3.6	3.6	10.59%	21.01%	40.83%
Classif	1004.6	1946.5	711.2	1113.3	3.9	9.4	0.87%	37.15%	27.74%
Kmeans	1024.6	2044.7	716.9	866.9	58.5	364.3	1.31%	10.75%	5.25%

Table 5: Job profiles on the EC2 cluster with large instances (time in sec)

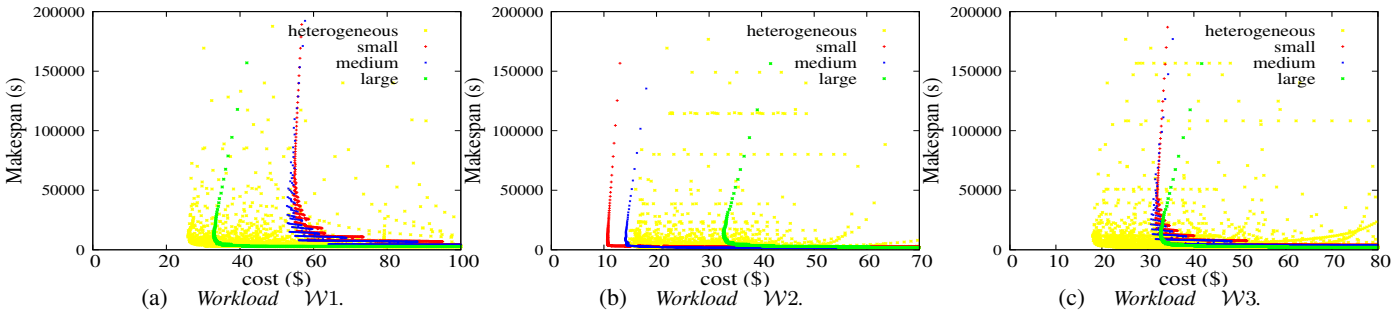


Figure 9: Performance versus cost trade-offs for different workloads.

- $\mathcal{W}2$ – it contains 11 applications: 1-11, i.e., excluding *KMeans* and *Classification* from the application set.
- $\mathcal{W}3$ – it contains 12 applications: 1-12, i.e., excluding *KMeans* from the application set.

Intuitively, there is a different number of applications that **strongly** favor *large* VM instances in each workload: $\mathcal{W}1$ has both *KMeans* and *Classification*, workload $\mathcal{W}2$ does not have any of them, and workload $\mathcal{W}3$ has only *Classification*.

Figure 9 shows the simulated cost/performance trade-off curves for three workloads executed on both homogeneous and heterogeneous Hadoop cluster(s). These trade-off curves are results of the brute-force algorithm design, it searches through the entire solution space by exhaustively enumerating all possible candidates for the solution. So, these trade-off curves do show all the solutions that our algorithms iterate through. For *homogeneous provisioning*, we show the three trade-off curves of Algorithm 2 for Hadoop clusters based on *small, medium* and *large* VM instances respectively.

Figure 9 (a) shows that workload $\mathcal{W}1$ is more cost-efficient when executed on the Hadoop cluster with *large* VMs (among the homogeneous clusters). Such results can be expected because $\mathcal{W}1$ contains both *KMeans* and *Classification* that have very strong preference towards *large* VM instances (see their high positive $PScore^{S,L}$). In comparison, $\mathcal{W}2$ cointains applications that mostly favor the *small* VM instances, and as a result, the most efficient trade-off curve belongs to a Hadoop cluster based on the *small* VM instances. Finally, $\mathcal{W}3$ represents a mixed case: it has *Classification* application that strongly favors *large* VM instances while most of the remaining applications prefer *small* VM instances. Figure 9(c) shows that a choice of the best homogeneous platform depends on the workload performance objectives (i.e., deadline $\mathcal{D}$).

The yellow dots in Figure 9 represent the completion time and monetary cost when we exploit a *heterogeneous provisioning* case with Algorithm 5. Each point corresponds to a workload split into two subsets that are executed on the Hadoop cluster formed with *small* and *large* VM instances respectively. This is why instead of the explicit trade-off curves as in the homogeneous cluster case, the simulation results for the heterogeneous case look much more scattered across the space.

To evaluate the efficiency of our provisioning algorithms, we consider different performance objectives for each workload:

- $\mathcal{D}$= 20000 seconds for workload $\mathcal{W}1$;
- $\mathcal{D}$= 10000 seconds for workload $\mathcal{W}2$;
- $\mathcal{D}$= 15000 seconds for workload $\mathcal{W}3$.

Tables 7-9 present the provisioning results for each workload with homogeneous and heterogeneous Hadoop clusters that have minimal monetary costs while meeting the given workload deadlines.

Among the homogeneous Hadoop clusters for $\mathcal{W}1$, the cluster with *large* VM instances has the lowest monetary cost of $32.86, that provides **41%** cost saving compared to a cluster with *small* VMs.

Cluster type	Number of Instances	Completion Time (sec)	Monetary Cost ($)
small (homogeneous)	210	15763	55.43
medium (homogeneous)	105	15137	53.48
large (homogeneous)	39	12323	32.86
small+large heterogeneous	48 *small* + 20 *large*	14988	**24.21**

Table 7: Cluster provisioning results for workload $\mathcal{W}1$.

Cluster type	Number of Instances	Completion Time (sec)	Monetary Cost ($)
small (homogeneous)	87	7283	**10.68**
medium (homogeneous)	43	9603	14.08
large (homogeneous)	49	9893	32.98
small+large heterogeneous	76 *small* + 21 *large*	6763	14.71

Table 8: Cluster provisioning results for workload $\mathcal{W}2$.

Cluster type	Number of Instances	Completion Time (sec)	Monetary Cost ($)
small (homogeneous)	140	13775	32.37
medium (homogeneous)	70	13118	31.05
large (homogeneous)	36	13265	32.72
small+large heterogeneous	74 *small* + 15 *large*	10130	**18.0**

Table 9: Cluster provisioning results for workload $\mathcal{W}3$.

By contrast, for workload $\mathcal{W}2$, the homogeneous Hadoop cluster with *small* VMs provides the lowest cost of $10.68, that provides **68%** cost saving compared to a cluster with *large* VM instances.

For $\mathcal{W}3$, all the three homogeneous solutions lead to a similar minimal cost, and the Hadoop cluster based on *medium* VMs has a slightly better cost than the other two alternatives.

Intuitively, these performance results are expected from the trade-off curves for three workloads shown in Figure 9.

The best *heterogeneous solution* for each workload is shown in the last row in Tables 7-9. For $\mathcal{W}1$, the minimal cost of the heterogeneous solution is $24.21 which is **26%** improvement compared to the minimal cost of the homogeneous solution based on the *large* VM instances. In this heterogeneous solution, the applications *SelfJoin, WordCount, InvIndex* are executed on the cluster with *small* VMs and applications *Classif, Kmeans, TermVector, Adjlist, HistMovies, HistRating, Grep, TeraSort, SeqCount, RankInvInd* are executed on the cluster with *large* VM instances.

The cost benefits of the heterogeneous solution is even more significant for $\mathcal{W}3$ as shown in Table 9. The minimal cost for heterogeneous cluster is $18.0 compared with the minimal cost for a homogeneous provision of $31.05, it leads to cost savings of **42%** compared to the minimal cost of the homogeneous solution. In this heterogeneous solution, the applications *HistMovies, HistRating, Grep, TeraSort, SeqCount, RankInvInd, SelfJoin, WordCount, InvIndex* are executed on the cluster with *small* VMs and applications *Classif,*

TermVector, Adjlist are executed on the cluster with *large* VMs.

However, for workload $\mathcal{W}2$, the heterogeneous solution does not provide additional cost benefits as shown in Table 8. One important reason is that for a heterogeneous solution, we need to maintain additional nodes deployed as JobTracker and NameNode for each sub-cluster. This increases the total provisioning cost compared to the homogeneous solution which only requires a single additional node for the entire cluster. The workload properties also play an important role here. As $\mathcal{W}2$ workload does not have any applications that have "strong" preference for *large* VM instances, the introduction of a special sub-cluster with *large* VM instances is not justified.

6.4 Impact of node failures on the cluster platform's selection

In this section, we show how the cluster platform's selection may be impacted when a user additionally considers a possibility of a node failure(s), and he/she is interested in achieving the generalized service level objectives (SLOs) which include two different performance goals for workload execution under a normal scenario and a case with 1-node failure:

- a desirable completion time $\mathcal{D}$ for the entire set of jobs in the workload $\mathcal{W}$ under normal conditions;
- an acceptable degraded completion time $\mathcal{D}_{deg}$ for processing $\mathcal{W}$ in case of 1-node failure.

Intuitively, a node failure in the Hadoop cluster formed with *small* EC2 instances may have smaller impact than a node failure in the cluster formed with *large* EC2 instances.

Let us demonstrate the decision making process for two applications from our set (see Table 2): *TermVector* and *AdjList*.

The *completion time* versus *cost* curves for applicatiions *TermVector* and *AdjList* are shown in Figures 10 and 7 (c) respectively.

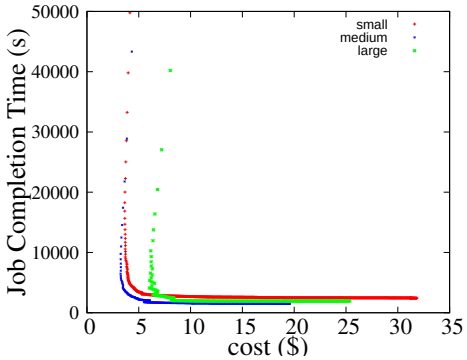


Figure 10: Performance versus cost trade-offs for *TermVector*.

From these figures and the preference score $PScore^{S,L}$ shown in Table 6, we can see that *TermVector* slightly favors *large* VM instances, while *AdjList* is practically neutral to the choice of *small*, *medium*, or *large* EC2 instances.

For these two applications, we apply our approach for selecting the underlying platform (a choice between *small*, *medium*, and *large* EC2 instances) to achieve the following performance objectives:

- *TermVector*:
  - $\mathcal{D}$ = 2900 seconds (regular case, no node failures);
  - $\mathcal{D}_{deg}$ = 2930 seconds (in case of 1-node failure).
- *AdjList*:
  - $\mathcal{D}$ = 1940 seconds (regular case, no node failures);
  - $\mathcal{D}_{deg}$ = 1945 seconds (in case of 1-node failure).

Table 10 summarizes the cluster provisioning results for a regular case and a scenario with 1-node failure for *TermVector*. We use abbreviations CT_{reg} and CT_{fail} to denote the completion time in regular and 1-node failure cases respectively. Table 10 shows that in a regular case scenario, a Hadoop cluster with *large* EC2 instances offers the best solution. However, if a user has concerns about a possible node failure, and aims to meet stringent performance objectives then the platform choice based on *small* VM instances is a better choice.

VM type	Regular, No-Failure Case $\mathcal{D}$ = 2900 sec			1-Node Failure Scenario $\mathcal{D}$= 2900 sec and $\mathcal{D}_{deg}$= 2930 sec			
	CT_{reg} *sec*	#VMs	Cost $	CT_{reg} *sec*	CT_{fail} *sec*	#VMs	Cost $
Small	2898	139	6.76	2898	2903	139	**6.76**
Large	2877	34	**6.71**	2842	2877	35	6.83

Table 10: *TermVector*: cluster provisioning results for a regular case and a scenario with 1-node failure.

Table 11 summarizes the cluster provisioning results for a regular case and a scenario with 1-node failure for *AdjList* application.

VM type	Regular, No-Failure Case $\mathcal{D}$ = 1940 sec			1-Node Failure Scenario $\mathcal{D}$ = 1940 sec and $\mathcal{D}_{deg}$= 1945 sec			
	CT_{reg} *sec*	#VMs	Cost $	CT_{reg} *sec*	CT_{fail} *sec*	#VMs	Cost $
Small	1939	139	4.52	1924	1939	140	**4.52**
Medium	1935	69	**4.51**	1931	1935	70	4.57

Table 11: *AdjList*: cluster provisioning results for a regular case and a scenario with 1-node failure.

In a regular case scenario, a Hadoop cluster with *medium* EC2 instances offers the best solution for *AdjList*. However, in the scenario with 1-node failure, the platform choice based on *small* VM instances is a better choice.

The achievable cost and performance advantages are more significant for workloads that require small-size Hadoop clusters for achieving their performance objectives. In large Hadoop clusters, a loss of 1-node results in a less pronounced performance impact.

6.5 Validation of the simulation results

To validate the accuracy of the simulation results, we chose workload $\mathcal{W}2$ and select the makespan target of 20000 seconds. We use our simulation results (shown in Figure 9 (b)) and identify four closest points that represent the corresponding four solutions. The selected points correspond to simulated homogeneous Hadoop clusters with 28, 20, 24 nodes formed by *small, medium,* and *large* EC2 instances respectively, and to a heterogeneous solution with two Hadoop sub-clusters based on 26 *small* nodes and 20 *large* nodes. We deployed the Hadoop clusters with the required number of instances and have executed workload $\mathcal{W}2$ (with the corresponding Johnson job schedule) on the deployed clusters. Figure 11 shows the comparison between the simulated and the actual measured makespan (we repeated measurements 5 times).

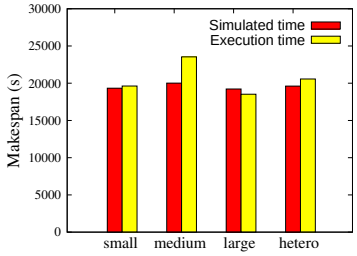


Figure 11: Validation of the simulation results.

Table 12 summarizes validation results shown in Figure 11.

The simulated results with *small* and *large* EC2 instances, as well as the heterogeneous solution show 2-8% error compared to measured results.

	small	*medium*	*large*	*heterogeneous*
Simulated time (sec)	19327	20013	19224	19612
Measured time (sec)	19625	23537	18521	21368

Table 12: Summary of the validation results.

We can see a higher prediction error (17%) for *medium* instances. Partially, it is due to a higher variance in the job profile measurements collected on *medium* instances.

6.6 Discussion

Towards a better understanding of what causes the application performance to be so different when executed by Hadoop clusters based on different VM instances, Figure 12 shows a detailed analysis of the execution time breakdown for *Terasort* and *Kmeans* on the *small, medium,* and *large* EC2 instances (we use the same Hadoop cluster configurations as described in our motivating example in Section 1).

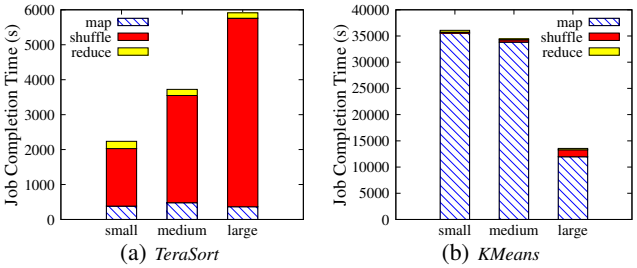


Figure 12: Analysis of *TeraSort* and *KMeans* on different EC2 instances.

Terasort performance is dominated by the shuffle phase. Moreover, the shuffle duration is increasing when executed by Hadoop based on on *medium* and *large* instances compared to the Hadoop execution based on the *small* EC2 instances. The significantly longer shuffle time leads to an increased overall job completion time as shown in Figure 12 (a). One explanation is that the increased size EC2 instances are provided with a scaled capacities of CPU and RAM, but not of the network bandwidth. As we configure more slots on the *large* EC2 instances, it increases amount of the I/O and network traffic (as well as the contention) per each VM, and this leads to the increased duration of the shuffle phase. On the contrary, for *Kmeans* shown in Figure 12 (b), the map stage duration dominates the application execution time, and the map phase execution is significantly improved when executed on *large* EC2 instances. This can be explained by checking the CPU models of the underlying server hardware used to host different types of EC2 instances.

Over a month, every day we have reserved 20 instances of *small, medium,* and *large* EC2 instances to gather their CPU information from the servers used for hosting these instances. Table 13 below summarizes the CPU models' statistics accumulated during these sampling experiments.

Majority of *large* EC2 instances (75%) are hosted on a later generation, more powerful, and faster CPU model compared to the *small* and *medium* EC2 instances. Also, practically the same CPU models are used for hosting the *small* and *medium* EC2 instances, which explains why the performance difference between *small* and *medium* EC2 instances were significantly smaller compared to the *large* ones, e.g., see *Kmeans* performance shown in Figure 12 (b).

Instance type	*CPU type*
Small	90% Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz 9% Intel(R) Xeon(R) CPU E5645 @ 2.40GHz 1% Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Medium	83% Intel(R) Xeon(R) CPU E5-2650 @ 2.00GHz 8% Intel(R) Xeon(R) CPU E5507 @ 2.27GHz 7% Intel(R) Xeon(R) CPU E5645 @ 2.40GHz 2% Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
Large	75% Intel(R) Xeon(R) CPU E5-2651 v2 @ 1.80GHz 12% Intel(R) Xeon(R) CPU E5507 @ 2.27GHz 8% Intel(R) Xeon(R) CPU E5430 @ 2.66GHz 3% Intel(R) Xeon(R) CPU E5645 @ 2.40GHz 2% Intel(R) Xeon(R) CPU E5-2650 @ 2.00GHz

Table 13: CPU types used by different EC2 instances.

7. RELATED WORK

In the past few years, performance modeling and simulation of MapReduce environments has received much attention, and different approaches [5, 4, 14, 13] were offered for predicting performance of MapReduce applications, as well as optimizing resource provisioning in the Cloud [7, 11]. A few MapReduce simulators were introduced for the analysis and exploration of Hadoop cluster configuration and optimized job scheduling decisions. The designers of *MRPerf* [16] aim to provide a fine-grained simulation of MapReduce setups. To accurately model inter- and intra rack task communications over network *MRPerf* uses the well-known ns-2 network simulator. The authors are interested in modeling different cluster topologies and in their impact on the MapReduce job performance. In our work, we follow the directions of *SimMR* simulator [12] and focus on simulating the job master decisions and the task/slot allocations across multiple jobs. We do not simulate details of the TaskTrackers (their hard disks or network packet transfers) as done by *MRPerf*. In spite of this, our approach accurately reflects the job processing because of our profiling technique to represent job latencies during different phases of MapReduce processing in the cluster. *SimMR* is very fast compared to *MRPerf* which deals with network-packet level simulations. Mumak [3] is an open source Apache's MapReduce simulator. It replays traces collected with a log processing tool, called Rumen [1]. The main difference between Mumak and *SimMR* is that Mumak omits modeling the shuffle/sort phase that could significantly affect the accuracy.

There is a body of work focusing on performance optimization of MapReduce executions in heterogeneous environments. Zaharia et al. [19], focus on eliminating the negative effect of stragglers on job completion time by improving the scheduling strategy with speculative tasks. The Tarazu project [2] 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. [18] 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. [9] 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.

Another group of related work is based on resource management that considers monetary cost and budget constraints. In [10], the authors provide a heuristic to optimize the number of machines for a bag of jobs while minimizing the overall completion time under a given budget. This work assumes the user does not have any knowledge about the job completion time. It starts with a single machine

and gradually adds more nodes to the cluster based on the average job completion time updated every time when a job is finished. In our approach, we use job profiles for optimizing the job schedule and provisioning the cluster.

In [17], the authors design a budget-driven scheduling algorithm for MapReduce applications in the heterogeneous cloud. They consider iterative MapReduce jobs that take multiple stages to complete, each stage contains a set of map or reduce tasks. The optimization goal is to select a machine from a fixed pool of heterogeneous machines for each task to minimize the job completion time or monetary cost. The proposed approach relies on a prior knowledge of the completion time and cost for a task i executed on a machine j in the candidate set. In our paper, we aim at minimizing the makespan of the set of jobs and design an ensemble of methods and tools to evaluate the job completion times as well as their makespan as a function of allocated resources. In [8], Kllapi et al. propose scheduling strategies to optimize performance/cost trade-offs for general data processing workflows in the Cloud. Different machines are modelled as containers with different CPU, memory, and network capacities. The computation workflow contains a set of nodes as operators and edges as data flows. The authors provide both greedy and local search algorithms to schedule operators on different containers so that the optimal performance (cost) is achieved without violating budget or deadline constraints. Compared to our profiling approach, they estimate the operator execution time using the CPU container requirements. This approach does not apply for estimating the durations of map/reduce tasks – their performance depends on multiple additional factors, e.g., the amount of RAM allocated to JVM, the I/O performance of the executing node, etc. The authors present only simulation results without validating the simulator accuracy.

8. CONCLUSION

In this work, we designed a novel simulation-based framework for evaluating both homogeneous and heterogeneous Hadoop solutions to enhance private and public cloud offerings with a cost-efficient, SLO-driven resource provisioning. We demonstrated that seemingly equivalent platform choices for a Hadoop cluster might result in a very different application performance, and thus lead to a different cost. Our case study with Amazon EC2 platform reveals that for different workloads an *optimized* platform choice may result in 45-68% cost savings for achieving the same performance objectives. In our future work, we plan to use a set of additional microbenchmarks to profile and compare generic phases of the MapReduce processing pipeline across Cloud offerings, e.g., comparing performance of the shuffle phase across different EC2 instances to predict the general performance impact of different platforms on the user workloads.

9. REFERENCES

[1] Apache Rumen: a tool to extract job characterization data from job tracker logs. `https://issues.apache.org/jira/browse/MAPREDUCE-728`.

[2] F. Ahmad et al. Tarazu: Optimizing MapReduce on Heterogeneous Clusters. In *Proc. of ASPLOS*, 2012.

[3] Apache. Mumak: Map-Reduce Simulator. `https://issues.apache.org/jira/browse/MAPREDUCE-751`.

[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] S. Johnson. Optimal Two- and Three-Stage Production Schedules with Setup Times Included. *Naval Res. Log. Quart.,1954*.

[7] K. Kambatla, A. Pathak, and H. Pucha. Towards optimizing hadoop provisioning in the cloud. In *Proc. of the First Workshop on Hot Topics in Cloud Computing*, 2009.

[8] H. Kllapi et al. Schedule Optimization for Data Processing Flows on the Cloud. In *Proc. of the ACM SIGMOD'2011*.

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

[10] J. N. Silva et al. Heuristic for Resources Allocation on Utility Computing Infrastructures. In *Proc. of MGC'2008 wokshop*.

[11] 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)*.

[12] A. Verma, L. Cherkasova, and R. H. Campbell. Play It Again, SimMR! In *Proc. of Intl. IEEE Cluster*, 2011.

[13] A. Verma, L. Cherkasova, and R. H. Campbell. Resource Provisioning Framework for MapReduce Jobs with Performance Goals. *Proc. of the 12th Middleware Conf.*, 2011.

[14] A. Verma et al. ARIA: Automatic Resource Inference and Allocation for MapReduce Environments. *Proc. ICAC'2011*.

[15] A. Verma et al. Two Sides of a Coin: Optimizing the Schedule of MapReduce Jobs to Minimize Their Makespan and Improve Cluster Performance. *Proc. of MASCOTS*, 2012.

[16] G. Wang, A. Butt, P. Pandey, and 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 (MASCOTS)*, 2009.

[17] Y. Wang and W. Shi. On Optimal Budget-Driven Scheduling Algorithms for MapReduce Jobs in the Hetereogeneous Cloud. *Technical Report TR-13-02, Carleton Univ.*, 2013.

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

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

[20] Z. Zhang, L. Cherkasova, and B. T. Loo. Exploiting Cloud Heterogeneity for Optimized Cost/Performance MapReduce Processing. In *Proc. of the 4th Intl. Workshop on Cloud Data and Platforms (CloudDP'2014)*, 2014.

[21] Z. Zhang et al. Automated Profiling and Resource Management of Pig Programs for Meeting Service Level Objectives. In *Proc. of IEEE/ACM ICAC'2012*.

[22] Z. Zhang et al. Optimizing Cost and Performance Trade-Offs for MapReduce Job Processing in the Cloud. In *Proc. of IEEE/IFIP NOMS*, May, 2014.