2013 IEEE 27th International Symposium on Parallel & Distributed Processing

Cura: A Cost-optimized Model for MapReduce in a Cloud

Balaji Palanisamy Aameek Singh† Ling Liu Bryan Langston†

College of Computing, Georgia Tech †*IBM Research - Almaden*

{balaji, lingliu}@cc.gatech.edu, †{aameek.singh, bryanlan}@us.ibm.com

***Abstract*—We propose a new MapReduce cloud service model, Cura, for data analytics in the cloud. We argue that performing MapReduce analytics in existing cloud service models – either using a generic compute cloud or a dedicated MapReduce cloud – is inadequate and inefficient for production workloads. Existing services require users to select a number of complex cluster and job parameters while simultaneously forcing the cloud provider to use those potentially sub-optimal configurations resulting in poor resource utilization and higher cost. In contrast Cura leverages MapReduce profiling to automatically create the best cluster configuration for the jobs so as to obtain a *global* resource optimization from the provider perspective. Secondly, to better serve modern MapReduce workloads which constitute a large proportion of interactive real-time jobs, Cura uses a unique instant VM allocation technique that reduces response times by up to 65%. Thirdly, our system introduces deadline-awareness which, by delaying execution of certain jobs, allows the cloud provider to optimize its global resource allocation and reduce costs further. Cura also benefits from a number of additional performance enhancements including cost-aware resource provisioning, VM-aware scheduling and online virtual machine reconfiguration. Our experimental results using Facebook-like workload traces show that along with response time improvements, our techniques lead to more than 80% reduction in the compute infrastructure cost of the cloud data center.**

I. Introduction

One of the major IT trends impacting modern enterprises is *big data* and *big data analytics*. As enterprises generate more and more data, deriving business value from it using analytics becomes a differentiating capability – whether it is understanding customer buying behavior or detecting fraud in online transactions. The most popular approach towards such big data analytics is using MapReduce [2] and its open-source implementation called Hadoop [16]. With the ability to automatically parallelize the application on a scale-out cluster of machines, MapReduce can allow analysis of terabytes and petabytes of data in a single analytics job.

This MapReduce analytics capability, when paired with another major IT trend – cloud computing, offers a unique opportunity for enterprises interested in big data analytics. A recent Gartner survey shows increasing cloud computing spending with 39% of enterprises having allotted IT budgets for it [1]. Offered in the cloud, a MapReduce service allows enterprises to analyze their data without dealing with the complexity of building and managing large installations of MapReduce platforms. Using virtual machines (VMs) and storage hosted by the cloud, enterprises can simply create virtual MapReduce clusters to analyze their data.

In general, there are two approaches in use today for MapReduce in a cloud. In the first approach, customers use a dedicated MapReduce cloud service (e.g. Amazon Elastic MapReduce [14]) and buy on-demand clusters of VMs for each job or a workflow. Once the MapReduce job (or workflow) is submitted, the cloud provider creates VMs that execute that job and after job completion the VMs are deprovisioned. In the second approach customers lease dedicated clusters from a generic cloud service like Amazon Elastic Compute Cloud [15] and operate MapReduce on them as if they were using a private MapReduce infrastructure. In this case, based on the type of analytics workload they may use different number of VMs for each submitted job, however the entire cluster needs to be maintained by the client enterprise.

In this paper we argue that these MapReduce cloud models suffer from the following drawbacks:

1) **Interactive workloads:** Many modern MapReduce workloads constitute a large fraction of interactive short jobs [8], [9], [35] that require short response times. A recent study on the Facebook and Yahoo production workload traces [30], [35] show that more than 95% of their production MapReduce jobs are short running jobs with an average running time of 30 sec. These jobs typically process a smaller amount of data (less than 200 MB in the Facebook trace [30])that are part of bigger data sets, for example, a friend recommendation query that is issued interactively when a Facebook user browses his/her profile. These jobs process a small amount of data corresponding to a small subset of the social network graph to recommend the most likely known friends to the user. As these jobs are highly interactive, providing high quality of service in terms of job response time is infeasible in a dedicated MapReduce cloud model (the first approach) since it requires virtual clusters to be created afresh for each submitted job. On the other hand an owned cluster in a generic compute cloud (the second approach) has high costs due to low utilization since the cluster needs to be continuously up waiting for jobs and serving them when submitted.

2) **Lack of global optimization:** Secondly, both of these models require users to figure out the complex job configuration parameters (e.g. type of VMs, number of VMs and MapReduce configuration like number of mappers per VM etc.) that have an impact on the performance and thus cost of the job. With growing popularity of MapReduce and associated

1530-2075/13 $26.00 © 2013 IEEE
DOI 10.1109/IPDPS.2013.20

eco-system like Pig [6], Hive [7], many MapReduce jobs nowadays are actually fired by non-engineer data analysts and putting such a burden on those users is impractical. Additionally, even if MapReduce performance prediction tools like [26] are used in the current cloud models, the per-job optimized configurations created by them from a user perspective are often suboptimal from a cloud provider perspective and hence lead to requiring more cloud resources than that required by a globally optimized schema with resource management performed at the cloud provider-end. A good example of such a cloud managed system is the recent Google BigQuery system [34] which allows to run SQL-like queries against very large datasets with potentially billions of rows. In BigQuery service, customers only submit the queries to be processed on the large datasets and the Cloud service provider intelligently manages the resources for the SQL-like queries.

3) **Low service differentiation:** Thirdly, both existing models fail to incorporate significant other optimization opportunities available for the cloud provider to improve its resource utilization. MapReduce workloads often have a large number of jobs that do not require immediate execution, rather feed into a scheduled flow - e.g. MapReduce job analyzing system logs for a daily/weekly status report. By delaying the execution of such jobs, cloud provider can multiplex its resources better for significant cost savings. For instance, the batch query model in Google BigQuery service [34] has 43% lower cost than the interactive query model in which case the queries are instantaneously executed.

To alleviate these drawbacks, we propose a MapReduce cloud service model called Cura. Cura uses a secure instant VM allocation scheme that helps reduce the response time for short jobs by up to 65%. To reduce user complexity, Cura automatically creates the best cluster configuration for the customers jobs with the goal of optimizing the overall resource usage of the cloud. Finally, Cura includes a suite of resource management techniques which leverage deadline awareness for cost-aware resource provisioning. Overall, the use of these techniques including intelligent VM-aware scheduling and online VM reconfiguration techniques lead to more than 80% savings in the cloud infrastructure cost. While Cura focuses on cloud provider's resource costs, we believe that any cost savings of the cloud provider in terms of infrastructure cost and energy cost based on resource usage would in turn reflect positively in the price of the services for the customers.

The rest of the paper is organized as follows: Section II motivates the need for the global optimization model and presents the Cura model and a description of its various components. In Section III, we present the technical details of the resource management techniques in Cura namely VM-aware scheduling and Reconfiguration-based VM management. We present our experimental results in Section IV and conclude in Section VI.

II. Cura: Model and Architecture

In this section, we present the cloud service model and system architecture for Cura.

A. Cloud Operational Model

Table I shows possible cloud service models for providing MapReduce as a cloud service. The first operational model (immediate execution) is a completely customer managed model where each job and its resources are specified by the customer on a per-job basis and the cloud provider only ensures that the requested resources are provisioned upon job arrival. Many existing cloud services such as Amazon Elastic Compute Cloud [15], Amazon Elastic MapReduce [14] use this model. This model has the lowest rewards since there is lack of global optimization across jobs as well as other drawbacks discussed earlier. The second possible model (delayed start) [44] is partly customer-managed and partly cloud-managed model where customers specify which resources to use for their jobs and the cloud provider has the flexibility to schedule the jobs as long as they **begin** execution within a specified deadline. Here, the cloud provider takes slightly greater risk to make sure that all jobs begin execution within their deadlines and as a reward can potentially do better multiplexing of its resources. However, specifically with MapReduce, this model still provides low cost benefits since jobs are being optimized on a per-job basis by disparate users. In fact customers in this model always tend to greedily choose low-cost small cluster configurations involving fewer VMs that would require the job to begin execution almost immediately. For example, consider a job that takes 180 minutes to complete in a cluster of 2 small instances but takes 20 minutes to complete using a cluster of 6 large instances[1]. Here if the job needs to be completed in more than 180 minutes, the per-job optimization by the customer will tend to choose the cluster of 2 small instances as it has lower resource usage cost compared to the 6 large instance cluster. This cluster configuration, however, expects the job to be started immediately and does not provide opportunity for delayed start. This observation leads us to the next model. The third model – which is the subject of this paper – is a completely cloud managed model where the customers only submit jobs and specify job completion deadlines. Here, the cloud provider takes greater risk and performs a globally optimized resource management to meet the job SLAs for the customers. Similar high-risk high-reward model is the database-as-a-service model [10], [11], [12] where the cloud provider estimates the execution time of the customer queries and performs resource provisioning and scheduling to ensure that the queries meet their response time requirements. As MapReduce also lends itself well to prediction of execution time [24], [5], [27], [28], [4], we have designed Cura on a similar model. Another recent example of this model is the Batch query model in Google's Big Query cloud service [34] where the Cloud provider manages the resources required for the SQL-like queries so as to provide a service level agreement of executing the query within 3 hours.

[1]Example adapted from the measurements in Herodotou et. al. paper[25]

TABLE I: Cloud Operational Models

Model	Optimization	Provider risk	Potential benefits
Immediate execution	Per-job	Limited	Low
Delayed start	Per-job	Moderate	Low – Moderate
Cloud managed	Global	High	High

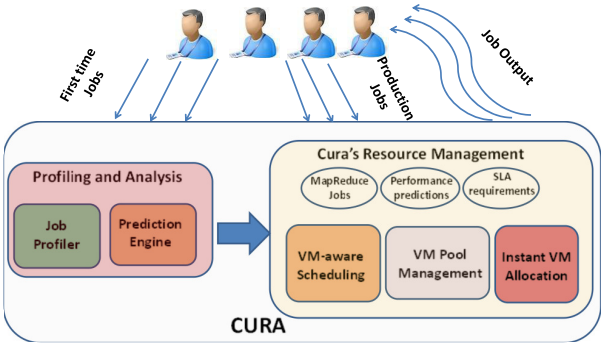


Fig. 1: Cura: System Architecture

B. System Model: User Interaction

Cura's system model significantly simplifies the way users deal with the cloud service. With Cura, users simply submit their jobs (or composite job workflows) and specify the required service quality in terms of response time requirements. After that, the cloud provider has complete control on the type and schedule of resources to be devoted to that job. From the user perspective, the deadline will typically be driven by their quality of service requirements for the job. As MapReduce jobs perform repeated analytics tasks, deadlines could simply be chosen based on those tasks (e.g. 8 AM for a daily log analysis job). For ad-hoc jobs that are not run per a set schedule, the cloud provider can try to incentivize longer deadlines by offering to lower costs if users are willing to wait for their jobs to be executed[2]. However, this model does not preclude an *immediate execution* mode in which case the job is scheduled to be executed at the time of submission, similar to existing MapReduce cloud service models.

C. System Architecture

Once a job is submitted to Cura, it may take one of the two paths (Figure 1). If a job is submitted for the very first time, Cura processes it to be *profiled* prior to execution as part of its *profile and analyze* service. This develops a performance model for the job in order to be able to generate predictions for its performance for different VM types, cluster sizes and job parameters. This model is used by Cura in optimizing the global resource allocation. MapReduce profiling has been an active area of research [24], [5], [4] and open-source tools such as Starfish [24] are available to create such profiles. Recent work had leveraged MapReduce profiling for Cloud resource management and showed that such profiles can be obtained with very high accuracy with less than 12% error rate for the predicted running time [27].

The *profile and analyze* service is used only once when a customer's job first goes from development-and-testing into production in its software life cycle. For subsequent instances of the production job, Cura directly sends the job for scheduling. Since typically production jobs including interactive or long running jobs do not change frequently (only their input data may differ for each instance of their execution), profiling will most often be a one-time cost. Further, from an architectural standpoint, Cura users may even choose to skip profiling and instead provide VM type, cluster size and job parameters to the cloud service similar to existing dedicated MapReduce cloud service models like [14]. Jobs that skip the one-time *profile and analyze* step will still benefit from the response time optimizations in Cura described below, however, they will fail to leverage the benefits provided by Cura's global resource optimization strategies. Jobs that are already profiled are directly submitted to the Cura resource management system.

Cura's resource management system is composed of the following components:

1) Secure instant VM allocation: In contrast to existing MapReduce services that create VMs on demand, Cura employs a secure instant VM allocation scheme that reduces response times for jobs, especially significant for short running jobs. Upon completion of a job's execution, Cura only destroys the Hadoop instance used by the job (including all local data) but retains the VM to be used for other jobs that need the same VM configuration. For the new job, only a quick Hadoop initialization phase is required which prevents having to recreate and boot up VMs[3]. Operationally, Cura creates pools of VMs of different instance types as shown in Figure 2 and dynamically creates Hadoop clusters on them.

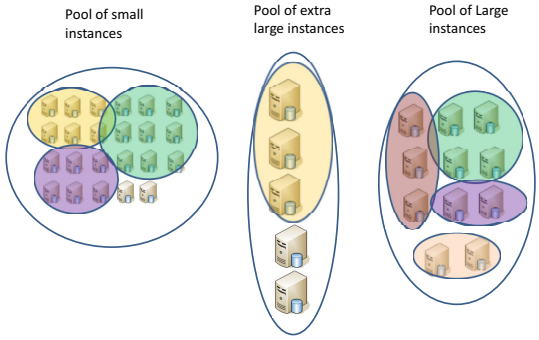


Fig. 2: VM Pool

When time sharing a VM across jobs it is important to ensure that an untrusted MapReduce program is not able to gain control over the data or applications of other customers. Cura's security management is based on SELinux [20] and is similar to that of the Airavat system proposed in [19] that showed that enforcing SELinux access policies in a MapReduce cloud does not lead to performance overheads. While Airavat shares multiple customer jobs across the same HDFS, Cura runs only one Hadoop instance at a time and the HDFS and MapReduce framework is used by only one customer before

[2]Design of a complete costing mechanism is beyond the scope of this work.

[3]Even with our secure instant VM allocation technique data still needs to be loaded for each job into its HDFS, but it is very fast for small jobs as they each process small amount of data, typically less than 200 MB in the Facebook and Yahoo workloads [30].

it is destroyed. Therefore, enforcing Cura's SELinux policies does not require modifications to the Hadoop framework and requires creation of only two SELinux domains, one trusted and the other untrusted. The Hadoop framework including the HDFS runs in the trusted domain and the untrusted customer programs run in the untrusted domain. While the trusted domain has regular access privileges including access to the network for network communication, the untrusted domain has very limited permissions and has no access to any trusted files and other system resources. An alternate solution for Cura's secure instant VM allocation is to take VM snapshots upon VM creation and once a customer job finishes, the VM can revert to the old snapshot. This approach is also significantly faster than destroying and recreating VMs, but it can however incur noticeable delays in starting a new job before the VM gets reverted to a secure earlier snapshot.

Overall this ability of Cura to serve short jobs better is a key distinguishing feature. However as discussed next, Cura has many other optimizations that benefit any type of job including long running batch jobs.

2) Job Scheduler: The job scheduler at the cloud provider forms an integral component of the Cura system. Where existing MapReduce services simply provision customer-specified VMs to execute the job, Cura's VM-aware scheduler (Section III-A) is faced with the challenge of scheduling jobs among available VM pools while minimizing global cloud resource usage. Therefore, carefully executing jobs in the best VM type and cluster size among the available VM pools becomes a crucial factor for performance. The scheduler has knowledge of the relative performance of the jobs across different cluster configurations from the predictions obtained from the *profile and analyze* service and uses it to obtain global resource optimization.

3) VM Pool Manager: The third main component in Cura is the VM Pool Manager that deals with the challenge of dynamically managing the VM pools to help the job scheduler effectively obtain efficient resource allocations. For instance, if more number of jobs in the current workload require small VM instances and the cloud infrastructure has fewer small instances, the scheduler will be forced to schedule them in other instance types leading to higher resource usage cost. The VM pool manager understands the current workload characteristics of the jobs and is responsible for online reconfiguration of VMs for adapting to changes in workload patterns (Section III-B). In addition, this component may perform further optimization such as power management by suitably shutting down VMs at low load conditions.

III. Cura: Resource Management

In this section, we describe Cura's core resource management techniques. We first present Cura's VM-aware job scheduler that intelligently schedules jobs within the available set of VM pools. We then present our reconfiguration-based VM pool manager that dynamically manages the VM instance pools by adaptively reconfiguring VMs based on current workload requirements.

A. Online VM-aware Scheduling

The goal of the cloud provider is to minimize the infrastructure cost by minimizing the number of servers required to handle the data center workload. Typically the peak workload decides the infrastructure cost for the data center. The goal of Cura VM-aware scheduling is to schedule all jobs within available VM pools to meet their deadlines while minimizing the overall resource usage in the data center reducing this total infrastructure cost.

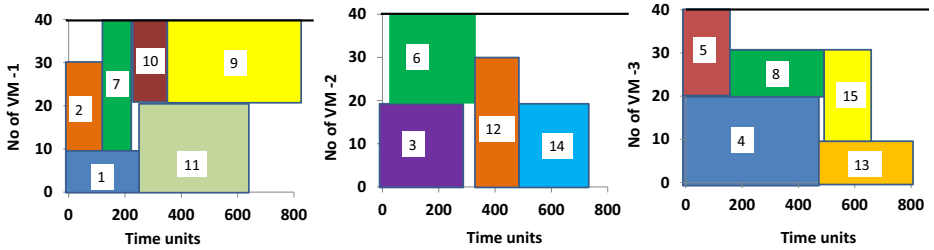
Given VM pools for each VM instance type and continually incoming jobs, the online VM-aware scheduler decides (a) when to schedule each job in the job queue, (b) which VM instance pool to use and (c) how many VMs to use for the jobs. The scheduler also decides best Hadoop configuration settings to be used for the job by consulting the *profile and analyze* service.

Depending upon deadlines for the submitted jobs, the VM-aware scheduler typically needs to make future reservations on VM pool resources (e.g. reserving 100 small instances from time instance 100 to 150). In order to maintain the most agility in dealing with incrementally incoming jobs and minimizing the number of reservation cancellations, Cura uses a strategy of trying to create minimum number of future reservations without under-utilizing any resources. For implementing this strategy, the scheduler operates by identifying the *highest priority* job to schedule at any given time and creates a tentative reservation for resources for that job. It then uses the end time of that job's reservation as the bound for limiting the number of reservations i.e. jobs in the job queue that are not schedulable (in terms of start time) within that *reservation time window* are not considered for reservation. This ensures that we are not unnecessarily creating a large number of reservations which may need cancellation and rescheduling after another job with more stringent deadline enters the queue.

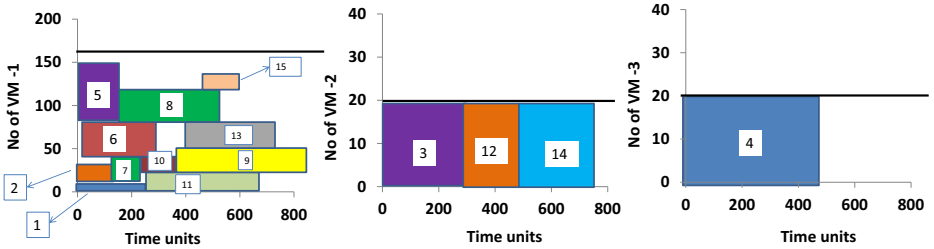
A job J_i is said to have higher priority over job J_j if the schedule obtained by reserving job J_i after reserving job J_j would incur higher resource cost compared to the schedule obtained by reserving job J_j after reserving J_i. The *highest priority* job is picked by performing pairwise cost comparisons. It is chosen such that it will incur higher overall resource usage cost if the highest priority job is deferred as compared to deferring any other job.

For each VM pool, the algorithm picks the highest priority job, J_{prior} in the job queue and makes a reservation for it using the cluster configuration with the lowest possible resource cost at the earliest possible time based on the performance predictions obtained from the *profile and analyze service*. Note that the lowest resource cost cluster configuration need not be the job's optimal cluster configuration (that has lowest per-job cost). For instance, if using the job's optimal cluster configuration at the current time cannot meet the deadline, the lowest resource cost cluster will represent the one that has the minimal resource usage cost among all the cluster configurations that can meet the job's deadline.

Once the highest priority job, J_{prior} is reserved for all VM pools, the reservation time windows for the corresponding VM



(a) VM-aware schedule



(b) VM-aware schedule with Reconf-based VM pool management

Fig. 3: Scheduling in Cura

pools are fixed. Subsequently, the scheduler picks the next highest priority job in the job queue by considering priority only with respect to the reservations that are possible within the current reservation time windows of the VM pools. The scheduler keeps on picking the highest priority job one by one in this manner and tries to make reservations to them on the VM pools within the reservation time window. Either when all jobs are considered in the queue and no more jobs are schedulable within the reservation time window or when the reservations have filled all the resources until the reservation time windows, the scheduler stops reserving.

Then at each time instance, the scheduler picks the reservations for the current time and schedules them on the VM pools by creating Hadoop clusters of the required sizes in the reservation. After scheduling the set of jobs that have reservation starting at the current time, the scheduler waits for one unit of time and considers scheduling for the next time unit. If no new jobs arrived within this one unit of time, the scheduler can simply look at the reservations made earlier and schedule the jobs that are reserved for the current time, however, if some new jobs arrived within the last one unit of time, then the scheduler needs to check if some of the newly arrived jobs have higher priority over the reserved jobs and in that case, the scheduler may require to cancel some existing reservations to reserve some newly arrived jobs that have higher priority over the ones in the reserved list.

If the scheduler finds that some newly arrived jobs take priority over some jobs in the current reservation list, it first tries to check if the reservation time window of the VM pools need to be changed. It needs to be changed only when some newly arrived jobs take priority over the current highest priority job of the VM pools that decides the reservation time window. If there exists such newly arrived jobs, the algorithm cancels all reserved jobs and moves them back to the job queue and adds all the newly arrived jobs to the job queue. It then picks the highest priority job, J_{prior} for each VM pool from the job queue that decides the reservation time window for each VM pool. Once the new reservation time window of the VM pools are updated, the scheduler considers the other jobs in the queue for reservation within the reservation time window of the VM pools until when either all jobs are considered or when no more resources are left for reservation. In case, the newly arrived jobs do not have higher priority over the time window deciding jobs but have higher priority over some other reserved jobs, the scheduler will not cancel the time window deciding reservations. However, it will cancel the other reservations and move the jobs back to the job queue along with the new jobs and repeat the process of reserving jobs within the reservation time windows from the job queue in the decreasing order of priority. For a data center of a given size, assuming constant number of *profile and analyze* predictions for each job, it can be shown that the algorithm runs in polynomial time with $O(n^2)$ complexity. We present a complete pseudo-code for this VM-aware scheduler in Algorithm 1.

While even a centralized VM-aware scheduler scales well for several thousands of servers with tens of thousands of jobs, it is also straight forward to obtain a distributed implementation to scale further. As seen from the pseudocode, the main operation of the VM-aware scheduler is finding the highest priority job among the n jobs in the queue based on pairwise cost comparisons. In a distributed implementation, this operation can be distributed and parallelized so that if there are n jobs in the queue, the algorithm would achieve a speed of x with x parallel machines, each of them performing $\frac{n}{x}$ pairwise cost comparisons.

Figure 3(a) shows an example VM-aware schedule obtained for 15 jobs using 40 VMs in each VM type, VM-1, VM-2 and VM-3. Here we assume that jobs 1, 2, 5, 6, 7, 8, 9, 10, 11, 13, 15 have their optimal cluster configuration using VM-1 and

Algorithm 1 VM-aware Scheduling

W_{list}: jobs that are waiting to be reserved or scheduled
N_{list}: jobs that arrived since the last time tick
R_{list}: jobs that have a tentative reservation
$t_{window}(V)$: reservation time window of VM type V
t_{window} : is the set of time windows of all the VM types
$Cost_{VM}(J_i, J_j, V)$: lowest possible resource usage cost of scheduling jobs J_i and J_j by reserving J_i before job J_j in VM type V
$Cost(J_i, J_j)$: lowest possible cost of scheduling jobs J_i and J_j on any VM type
$Cost_{twindow}(J_i, J_j)$: lowest possible cost of scheduling J_i and J_j by reserving J_i before J_j such that they both start within the time window of the VM pools
$Sched(J_i, t_{window})$: determines if the Job J_i is schedulable within the current time window of the VM pools
All cost calculations consider only cluster configurations that can meet the job's deadline
procedure VMAWARESCHEDULE(W_{list}, N_{list}, R_{list})
    Assign $redo\_reserve = true$ if $\exists J_n \in N_{list}$, $\exists J_r \in R_{list}$ such that $Cost(J_n, J_r) \leq Cost(J_r, J_n)$
    Assign $redo\_timewindow = true$ if $\exists J_n \in N_{list}$, $\exists J_r \in R_{list}$ such that $Cost(J_n, J_r) < Cost(J_r, J_n)$ and J_r is a time window deciding job
    if ($redo\_reserve == false$) **then**
        return
    end if
    if ($redo\_timewindow == true$) **then**
        $CJ_{list} = R_{list} \cup N_{list} \cup W_{list}$
        Cancel all reservations
        for all $V \in VMtypes$ **do**
            Pick and reserve job J_i that maximizes
            $\sum_{J_j \in CJ_{list}} Cost(J_j, J_i) - Cost_{VM}(J_i, J_j, V))$
            $t_{window}(V) = min(t_{end}(J_i), t_{bound})$
        end for
    else
        $CJ_{list} = R_{list} \cup N_{list}$
        Cancel all reservations except t_{window} deciding ones
    end if
    while ($\exists J_i \in CJ_{list} | sched(J_i, t_{window}) == true$) **do**
        Pick and reserve job J_i that maximizes
        $\sum_{J_j \in CJ_{list}} Cost_{twindow}(J_j, J_i) - Cost_{t_{window}}(J_i, J_j)$
    end while
    Run jobs having reservations start at the current time
end procedure

jobs 3, 12, and 14 are optimal with VM-2 and job 4 is optimal with VM-3. The VM-aware scheduler tries its best effort to minimize the overall resource usage cost by provisioning the right jobs in the right VM types and using the minimal cluster size required to meet the deadline requirements. However, when the optimal choice of the resource is not available for some jobs, the scheduler considers the next best cluster configuration and schedules them in a cost-aware manner. A detailed illustration of this example with cost comparisons is presented in Appendix A.

B. Reconfiguration-based VM Management

Although the VM-aware scheduler tries to effectively minimize the global resource usage by scheduling jobs based on resource usage cost, it may not be efficient if the underlying VM pools are not optimal for the current workload characteristics. Cura's reconfiguration-based VM manager understands the workload characteristics of the jobs as an online process and performs online reconfiguration of the underlying VM pools to better suit the current workload. For example, the VM pool allocation shown in Figure 2 can be reconfigured as shown in Figure 4 to have more small instances by shutting down some large and extra large instances if the current workload pattern requires more small instances.

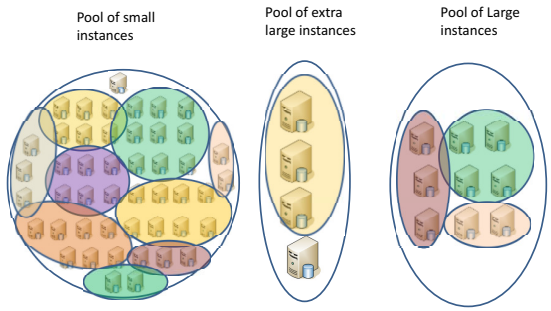


Fig. 4: Reconfiguration-based VM Management

The reconfiguration-based VM manager considers the recent history of job executions by observing the jobs that arrived within a period of time referred to as the reconfiguration time window. For each job, J_i arriving within the reconfiguration time window, the reconfiguration algorithm understands the optimal cluster configuration, $C_{opt}(J_i)$ that incurs the lowest resource usage cost among all cluster configurations that can meet the job's deadline requirements. At the end of the reconfiguration time window period, the algorithm decides on the reconfiguration plan by making a suitable tradeoff between the performance enhancement obtained after reconfiguration and the cost of the reconfiguration process. Such reconfiguration has to be balanced against the cost of reconfiguration operations (shutting down some instances and starting others). For this, we compute the benefit of doing such reconfiguration.

The algorithm triggers the reconfiguration process only if it finds that the estimated cost benefit exceeds the reconfiguration cost by a factor of β. When the reconfiguration process starts to execute, it shuts down some VMs whose instance types needs to be decreased in number and creates new VMs of the instance types that needs to created. The rest of the process is similar to any VM reconfiguration process that focuses on the *bin-packing* aspect of placing VMs within the set of physical servers during reconfiguration [13], [36].

Continuing the example of Figure 3, we find that the basic VM-aware scheduler in Figure 3(a) without reconfiguration support schedules jobs 5, 6, 8, 13, 15 using VM-2 and VM-3 types even though they are optimal with VM-1, The reconfiguration based VM-aware schedule in Figure 3(b) provisions more VM-1 instances (notice changed Y-axis scale) by understanding the workload characteristics and hence in addition to the other jobs, jobs 5, 6, 8 13 and 15 also get scheduled with their optimal choice of VM-type namely VM-1, thereby minimizing the overall resource usage cost in the cloud data center. The detailed schedule for this case is explained in Appendix A for interested readers.

IV. Experimental Evaluation

We present our experimental evaluation in terms of both performance and cost-effectiveness of Cura compared to conventional MapReduce services. We first start with our experimental setup.

A. Experimental setup

Metrics: We evaluate our techniques on four key metrics with the goal of measuring their cost effectiveness and performance– (1) *number of servers*: techniques that require more number of physical servers to successfully meet the service quality requirements are less cost-effective; this metric measures the capital expense on the provisioning of physical infrastructure in the data center, (2) *response time*: it is the total time between job submission and job completion and therefore techniques that have higher response time provide poor service quality; this metric captures the service quality of the jobs and (3) *per-job infrastructure cost* - this metric represents the average per-job fraction of the infrastructure cost; techniques that require fewer servers will have lower per-job cost and (4) *effective utilization*: techniques that result in poor utilization lead to higher cost; this metric captures both the cost-effectiveness and the performance of the techniques. It should be noted that the effective utilization captures only the useful utilization that represents job execution and does not include the time taken for creating and destroying VMs.

Cluster Setup: Our cluster consists of 20 CentOS 5.5 physical machines (KVM as the hypervisor) with 16 core 2.53GHz Intel processors and 16 GB RAM. The machines are organized in two racks, each rack containing 10 physical machines. The network is 1 Gbps and the nodes within a rack are connected through a single switch. We considered 6 VM instance types with the lowest configuration starting from 2 2 GHz VCPUs and 2 GB RAM to the highest configuration having 12 2GHz VCPUs and 12 GB RAM with each VM configuration differing by 2 2 GHz VCPUs and 2 GB RAM with the next higher configuration.

Workload: We created 50 jobs using the Swim MapReduce workload generator [30] that richly represent the characteristics of the production MapReduce workload in the Facebook MapReduce cluster. The workload generator uses a real MapReduce trace from the Facebook production cluster and generates jobs with similar characteristics as observed in the Facebook cluster. Using the *Starfish* profiling tool [24], each job is profiled on our cluster setup using clusters of VMs of all 6 VM types. Each profile is then analyzed using *Starfish* to develop predictions across various hypothetical cluster configurations and input data sizes.

Before discussing the experimental results, we briefly discuss the set of techniques compared in the evaluation.

Per-job cluster services: Per job services are similar to dedicated MapReduce services such as Amazon Elastic MapReduce [14] that create clusters per job or per workflow. While this model does not automatically pick VM and Hadoop parameters, for a fair comparison we use *Starfish* to create the optimal VM and Hadoop configuration even in this model.

Dedicated cluster services: Dedicated clusters are similar to private cloud infrastructures where all VMs are managed by the customer enterprises and Hadoop clusters are formed on demand when jobs arrive. Here again the VM and job parameters are chosen via Starfish.

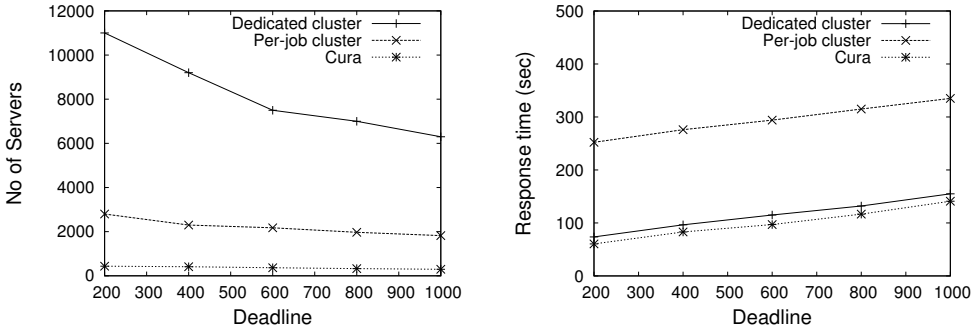
Cura: Cura incorporates both the VM-aware scheduler described in Section III-A and the Reconfiguration based VM Management (Section III-B).

B. Experimental Results

We begin by presenting the comparison of Cura with the existing techniques for various experimental conditions determined by distribution of the job deadlines, size of the MapReduce jobs and the number of servers in the system. By default, we use a composite workload consisting of equal proportion of jobs of three different categories: small jobs, medium jobs and large jobs. Small jobs read 100 MB of data, whereas medium jobs and large jobs read 1 GB and 10 GB of input data respectively. We model Poisson job arrivals with rate parameter, $\lambda = 0.5$ and the jobs are uniformly distributed among 50 customers. The evaluation uses 11,500 jobs arriving within a period of 100 minutes. Each of the arrived job represents one of the 50 profiled jobs with input data size ranging from 100 MB to 10 GB based on the job size category. Note that a job's complete execution includes both the data loading time from the storage infrastructure to the compute infrastructure and the Hadoop startup time for setting up the Hadoop cluster in the cluster of VMs. The data loading time is computed by assuming a network throughput of 50 MBps per VM [4] from the storage server and the Hadoop startup time is taken as 10 sec.

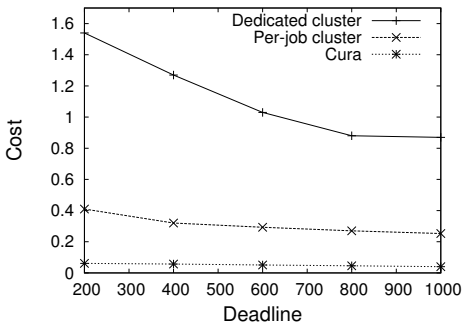
1) Effect of job deadlines: In this set of experiments, we first study the effect of job deadlines on the performance of Cura with other techniques. Figure 5(a) shows the performance of the techniques for different maximum deadlines with respect to number of servers required for the cloud provider to satisfy the workload. Here, the deadlines are uniformly distributed within the maximum deadline value shown on the X-axis. We find that provisioning dedicated clusters for each customer results in a lot of resources as dedicated clusters are based on the peak requirements of each customer and therefore the resources are under-utilized. On the other hand, per-job cluster services require lower number of servers (Figure 5(a)) as these resources are shared among the customers. However, the Cura approach in Figure 5(a) has a much lower resource requirement having up to 80% reduction in terms of the number of servers. This is due to the designed *global optimization* capability of Cura. Where per-job and dedicated cluster services always attempt to place jobs based on per-job optimal configuration obtained from *Starfish*, resources for which may not be available in the cloud, Cura on the other hand can schedule jobs using other than their individual optimal configurations to better adapt to available resources in the cloud.

[4]Here, the 50 MBps throughput is a conservative estimate of the throughput between the storage and compute infrastructures based on measurement studies on real cloud infrastructures [22].

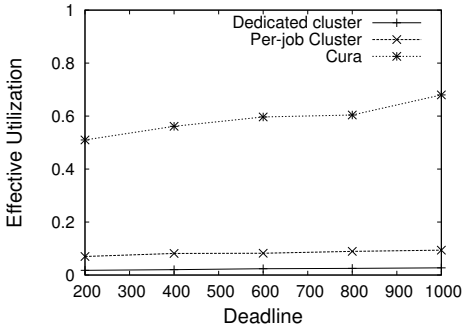


(a) Number of Servers

(b) Response time



(c) Cost



(d) Effective Utilization

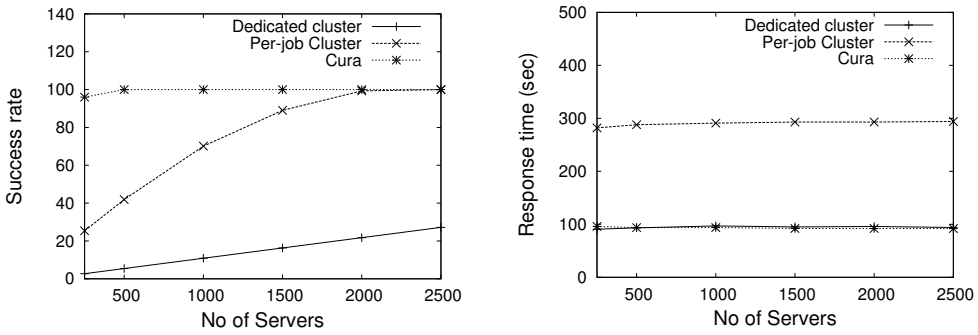
Fig. 5: Effect of Job-deadlines

We also compare the approaches in terms of the mean response time in Figure 5(b). To allow each compared technique to successfully schedule all jobs (and not cause failures), we use the number of servers obtained in Figure 5(a) for each individual technique. As a result, in this response time comparison, Cura is using much fewer servers than the other techniques. We find that the Cura approach and the dedicated cluster approach have lower response time (up to 65%).

In the per-job cluster approach, the VM clusters are created for each job and it takes additional time for the VM creation and booting process before the jobs can begin execution leading to the increased response time of the jobs. Similar to the comparison on the number of servers, we see the same trend with respect to the per-job cost in Figure 5(c) that shows that the Cura approach can significantly reduce the per-job infrastructure cost of the jobs (up to 80%). The effective utilization in Figure 5(d) shows that the per-job cluster services and dedicated cluster approach have much lower effective utilization compared to the Cura approach. The per-job services spend a lot of resources in creating VMs for every job arrival. Especially with short response time jobs, the VM creation becomes a bigger overhead and reduces the effective utilization. The dedicated cluster approach does not create VMs for every job instance, however it has poor utilization because dedicated clusters are sized based on peak utilization. But the Cura approach has a high effective utilization having up to 7x improvement compared to the other techniques as Cura effectively leverages global optimization and deadline-awareness to achieve better resource management.

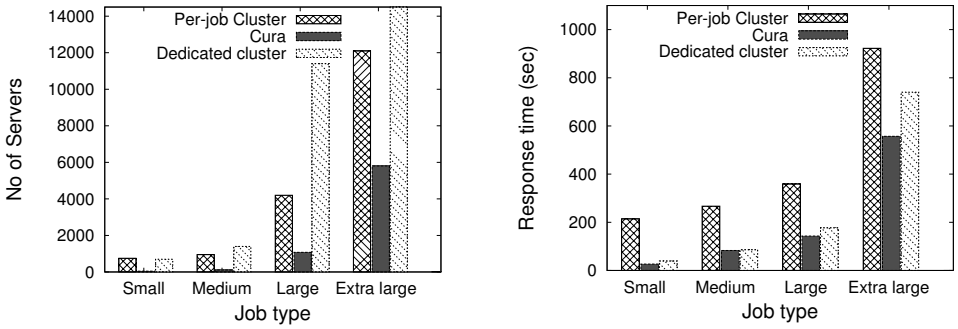
2) Varying number of Servers: We next study the performance of the techniques by varying the number of servers provisioned to handle the workload. Figure 6(a) shows the success rate of the approaches for various number of servers. Here, the success rate represents the fraction of jobs that successfully meet their deadlines. We find that the Cura approach has a high success rate even with 250 servers, whereas the per-job cluster approach obtains close to 100% rate only with 2000 servers. Figure 6(b) shows that the response time of successful jobs in the compared approaches show a similar trend as in Figure 5(b) where the Cura approach performs better than the per-job cluster services.

3) Varying job sizes: This set of experiments evaluates the performance of the techniques for various job sizes based on the size of input data read. Note that small jobs process 100 MB of data, medium jobs process 1 GB of data and large and extra large jobs process 10 GB and 100 GB of data respectively. Also small, medium and large jobs have a mean deadline of 100 second and the extra large jobs have a mean deadline of 1000 second as they are long running. We find that the performance in terms of number of servers in Figure 7(a) has up to 9x improvement for the short and medium jobs with Cura approach compared to the per-job cluster approach. It is because in addition to the VM-aware scheduling and reconfiguration-based VM management, these

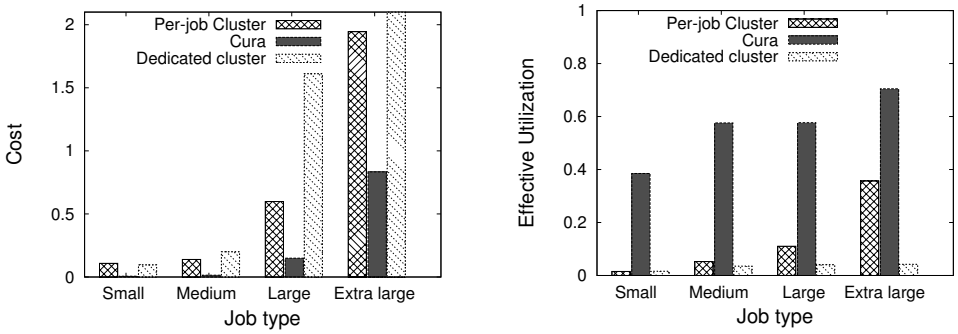


(a) Success Rate (b) Response time

Fig. 6: Effect of servers



(a) Number of Servers (b) Response time



(c) Cost (d) Effective Utilization

Fig. 7: Effect of Job type

jobs benefit the most from the secure instant VM allocation as these are short jobs. For large and extra large jobs, the Cura approach still performs significantly better having up to 4x and 2x improvement for large and extra large jobs compared to the per-job cluster services. The dedicated cluster service requires significantly higher resources for large jobs as the peak workload utilization becomes high (its numbers significantly cross the max Y-axis value). This set of experiments show that the global optimization techniques in Cura are not only efficient for short jobs but also for long running batch workloads.

The response time improvements of Cura and dedicated cluster approach in Figure 7(b) also show that the improvement is very significant for short jobs having up to 87% reduced response time and up to 69% for medium jobs. It is reasonably significant for large jobs with up to 60% lower response time and extra large jobs with up to 30% reduced response time. The cost comparison in Figure 7(c) also shows a similar trend that the Cura approach, although is significantly effective for both large and extra large jobs, the cost reduction is much more significant for small and medium jobs.

V. Related Work

Resource Allocation and Job Scheduling: There is a large body of work on resource allocation and job scheduling

in grid and parallel computing. Some representative examples of generic schedulers include [37], [38]. The techniques proposed in [39], [40] consider the class of malleable jobs where the number processors provisioned can be varied at runtime. Similarly, the scheduling techniques presented in [41], [42] consider moldable jobs that can be run on different number of processors. These techniques do not consider a virtualized setting and hence do not deal with the challenges of dynamically managing and reconfiguring the VM pools to adapt for workload changes. Therefore, unlike Cura they do not make scheduling decisions over *dynamically* managed VM pools. Chard et. al present a resource allocation framework for grid and cloud computing frameworks by employing economic principles in job scheduling [45]. Hacker et. al propose techniques for allocating virtual clusters by queuing job requests to minimize the spare resources in the cloud [46]. Recently, there has been work on cloud auto scaling with the goal of minimizing customer cost while provisioning the resources required to provide the needed service quality [43]. The authors in [44] propose techniques for combining on demand provisioning of virtual resources with batch processing to increase system utilization. Although the above mentioned systems have considered cost reduction as a primary objective of resource management, these systems are based on either per-job or per-customer optimization and hence unlike Cura, they do not lead to a globally optimal resource management.

MapReduce task placement: There have been several efforts that investigate task placement techniques for MapReduce while considering fairness constraints [32], [17]. *Mantri* tries to improve job performance by minimizing outliers by making network-aware task placement [3]. Similar to Yahoo's capacity scheduler and Facebook's fairness scheduler, the goal of these techniques is to appropriately place tasks for the jobs running in a given Hadoop cluster to optimize for locality, fairness and performance. Cura, on the other hand deals with the challenges of appropriately provisioning the right Hadoop clusters for the jobs in terms of VM instance type and cluster size to globally optimize for resource cost while dynamically reconfiguring the VM pools to adapt for workload changes.

MapReduce in a cloud: Recently, motivated by MapReduce, there has been work on resource allocation for data intensive applications in the cloud context [18], [33]. Quincy [18] is a resource allocation system for scheduling concurrent jobs on clusters and Purlieus [33] is a MapReduce cloud system that improves job performance through locality optimizations achieved by optimizing data and compute placements in an integrated fashion. However, unlike Cura these systems are not aimed at improving the usage model for MapReduce in a Cloud to better serve modern workloads with lower cost.

MapReduce Profile and Analyze tools: A number of MapReduce profiling tools have been developed in the recent past with an objective of minimizing customer's cost in the cloud [4], [23], [5], [28], [29]. Herodotou et al. developed an automated performance prediction tool based on their profile and analyze tool *Starfish* [24] to guide customers to choose the best cluster size for meeting their job requirements [26]. Similar performance prediction tool is developed by Verma. et. al [29] based on a linear regression model with the goal of guiding customers to minimize cost. Popescu. et. al developed a technique for predicting runtime performance for jobs running over varying input data set [28]. Recently, a new tool called Bazaar [27] has been developed to guide MapReduce customers in a cloud by predicting job performance using a gray-box approach that has very high prediction accuracy with less than 12% prediction error. However, as discussed earlier, these job optimizations initiated from the customer-end may lead to requiring higher resources at the cloud. Cura while leveraging existing profiling research, addresses the challenge of optimizing the global resource allocation at the cloud provider-end with the goal of minimizing customer costs. As seen in evaluation, Cura benefits from both its cost-optimized usage model and its intelligent scheduling and online reconfiguration-based VM pool management.

VI. Conclusions

This paper presents a new MapReduce cloud service model, Cura, for data analytics in the cloud. We argued that existing cloud services for MapReduce are inadequate and inefficient for production workloads. In contrast to existing services, Cura automatically creates the best cluster configuration for the jobs using MapReduce profiling and leverages deadline-awareness which, by delaying execution of certain jobs, allows the cloud provider to optimize its global resource allocation efficiently and reduce its costs. Cura also uses a unique secure instant VM allocation technique that ensures fast response time guarantees for short interactive jobs, a significant proportion of modern MapReduce workloads. Cura's resource management techniques include cost-aware resource provisioning, VM-aware scheduling and online virtual machine reconfiguration. Our experimental results using jobs profiled from realistic Facebook-like production workload traces show that Cura achieves more than 80% reduction in infrastructure cost with 65% lower job response times.

VII. Acknowledgements

This work is partially supported by an IBM PhD fellowship for the first author. The first and third authors are partially sponsored by grants from NSF CISE NetSE program, SaTC program and I/UCRCs program, an IBM faculty award and a grant from Intel ISTC on Cloud Computing.

References

[1] B. Igou "User Survey Analysis: Cloud-Computing Budgets Are Growing and Shifting; Traditional IT Services Providers Must Prepare or Perish". Gartner Report, 2010

[2] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[3] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha and E. Harris. Reining in the Outliers inMap-Reduce Clusters using Mantri. In *OSDI*, 2010.

[4] K. Kambatla, A. Pathak and H. Pucha. Towards Optimizing Hadoop Provisioning in the Cloud. In *HotCloud*, 2009.
[5] K. Morton, A. Friesen, M. Balazinska, D. Grossman. Estimating the Progress of MapReduce Pipelines. In *ICDE*, 2010.
[6] Pig User Guide. *http://pig.apache.org/*.
[7] A. Thusoo et. al. Hive - A Warehousing Solution Over a MapReduce Framework In *VLDB*, 2009.
[8] D. Borthakur et al. Apache Hadoop goes realtime at Facebook In *SIGMOD*, 2011.
[9] S. Melnik et al. Dremel: interactive analysis of web-scale datasets In *VLDB*, 2010.
[10] C. Curino, E. P. C. Jones, R. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, N. Zeldovich Relational Cloud: A Database-as-a-Service for the Cloud In *CIDR*, 2011.
[11] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, J. Rittinger Multi-Tenant Databases for Software as a Service: Schema-Mapping Techniques In *SIGMOD*, 2008.
[12] P. Xiong, Y. Chi, S. Zhu, J. Tatemura, C. Pu, H. Hacigumus ActiveSLA: A Profit-Oriented Admission Control Framework for Database-as-a-Service Providers In *SOCC*, 2011.
[13] T. Wood, P. Shenoy, A. Venkataramani and M. Yousif Black-box and Gray-box Strategies for Virtual Machine Migration. In *NSDI*, 2007.
[14] Amazon Elastic MapReduce. *http://aws.amazon.com/elasticmapreduce/*
[15] Amazon Elastic Compute Cloud. *http://aws.amazon.com/ec2/*
[16] Hadoop. http://hadoop.apache.org.
[17] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, I. Stoica. Improving MapReduce Performance in Heterogeneous Environments. In *OSDI*, 2008.
[18] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: fair scheduling for distributed computing clusters. In *SOSP*, 2009.
[19] I. Roy, Srinath. Setty, A. Kilzer, V. Shmatikov, E. Witchel Airavat: Security and Privacy for MapReduce *NSDI*, 2010.
[20] SELinux user guide. *http://selinuxproject.org/page/Main_Page*.
[21] M. R. Garey, D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman. ISBN 0-7167-1045-5.
[22] S. L. Garfinkel An Evaluation of Amazon's Grid Computing Services: EC2, S3 and SQS . *Technical Report, TR-08-07, Harvard University, available at: ftp://ftp.deas.harvard.edu/techreports/tr-08-07.pdf*
[23] F. Tian and K. Chen Towards Optimal Resource Provisioning for Running MapReduce Programs in Public Clouds In *CLOUD*, 2011
[24] H. Herodotou and S. Babu On Optimizing MapReduce Programs / Hadoop Jobs. In *VLDB*, 2011.
[25] H. Herodotou et. al Starfish: A Selftuning System for Big Data Analytics. In *CIDR*, 2011.
[26] H. Herodotou, F. Dong and S. Babu No One (Cluster) Size Fits All: Automatic Cluster Sizing for Data-intensive Analytics. In *SOCC*, 2011.
[27] V. Jalaparti, H. Ballani, P. Costa, T. Karagiannis, A. Rowstron Bazaar: Enabling Predictable Performance in Datacenters *MSR Cambridge, UK, Technical Report MSR-TR-2012-38*.
[28] A. Popescu, V. Ercegovac, A. Balmin, M. Branco, A. Ailamaki Same Queries, Different Data: Can we Predict Runtime Performance? *SMDB*, 2012.
[29] A. Verma, L. Cherkasova, and R. H. Campbell Resource Provisioning Framework for MapReduce Jobs with Performance Goals In *Middleware*, 2011.
[30] Y. Chen, A. Ganapathi, R. Griffith, R. Katz The Case for Evaluating MapReduce Performance Using Workload Suites In *MASCOTS*, 2011.
[31] S. Kavulya, J. Tan, R. Gandhi, P. Narasimhan An Analysis of Traces from a Production MapReduce Cluster In *CCGrid*, 2010.
[32] T. Sandholm and K. Lai. Mapreduce optimization using dynamic regulated prioritization. In *ACM SIGMETRICS/Performance*, 2009.
[33] B. Palanisamy, A. Singh, L. Liu and B. Jain Purlieus: locality-aware resource allocation for MapReduce in a cloud. In *SC*, 2011.
[34] Google BigQuery . https://developers.google.com/bigquery/.
[35] Y. Chen, S. Alspaugh, D. Borthakur and R. Katz Energy Effciency for Large-Scale MapReduce Workloads with Significant Interactive Analysis In *EUROSYS*, 2012.
[36] A. Singh, M. Korupolu, and D. Mohapatra. Server-storage virtualization: Integration and load balancing in data centers. In *SC*, 2008.
[37] A. Mu'alem , D. Feitelson, Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. In *TPDS*, 2001
[38] J. Skovira, W. Chan, H. Zhou, D. Lifka The EASY - LoadLeveler API Project In *IPPS*, 1996.
[39] S. Anastasiadis , K. Sevcik Parallel Application Scheduling on Networks of Workstations. In JPDC, 1997
[40] E. Rosti , E. Smirni , L. Dowdy , G. Serazzi , B. Carlson Robust Partitioning Policies of Multiprocessor Systems. In Performance Evaluation, 1993.
[41] S. Srinivasan, V. Subramani, R. Kettimuthu, P. Holenarsipur, P. Sadayappan Effective Selection of Partition Sizes for Moldable Scheduling of Parallel Jobs. In HIPC, 2002.
[42] W. Cirne Using Moldability to Improve the Performance of Supercomputer Jobs. Ph.D. Thesis. UCSD, 2001
[43] M. Mao, M. Humphrey Auto-Scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows. In SC, 2011.
[44] B. Sotomayor, K. Keahey, I. Foster Combining Batch Execution and Leasing Using Virtual Machines
[45] K. Chard, K. Bubendorfer, P. Komisarczuk High Occupancy Resource Allocation for Grid and Cloud systems, a Study with DRIVE. In HPDC, 2010.
[46] T. Hacker, K. Mahadik Flexible Resource Allocation for Reliable Virtual Cluster Computing Systems. In *SC*, 2011

VIII. Appendix A: VM-aware Schedule Example

VMs	t_{run} VM-1	Cost VM-1	t_{run} VM-2	Cost VM-2	t_{run} VM-3	Cost VM-3
10	900	1500	562.5	1875	321.42	2142.85
20	473.68	1578.94	296.05	1973.68	169.17	2255.63
30	333.33	1666.66	208.33	2083.33	119.04	2380.95
40	264.70	1764.70	165.44	2205.88	94.53	2521.00

TABLE II: Job type -1: Optimal with virtual machine type -1 (VM-1)

VMs	t_{run} VM-1	Cost VM-1	t_{run} VM-2	Cost VM-2	t_{run} VM-3	Cost VM-3
10	1250	2083.33	500	1666.66	357.14	2380.95
20	657.89	2192.98	263.15	1754.38	187.96	2506.26
30	462.96	2314.81	185.18	1851.85	132.27	2645.50
40	367.64	2450.98	147.05	1960.78	105.04	2801.12

TABLE III: Job type -2: Optimal with virtual machine type -2 (VM-2)

VMs	t_{run} VM-1	Cost VM-1	t_{run} VM-2	Cost VM-2	t_{run} VM-3	Cost VM-3
10	5000	8333.33	2187.5	7291.66	875	5833.33
20	2631.57	8771.92	1151.31	7675.43	460.52	6140.35
30	1851.85	9259.25	810.18	8101.85	324.07	6481.48
40	1470.58	9803.92	643.38	8578.43	257.35	6862.74

TABLE IV: Job type -3: Optimal with virtual machine type -3

VMs	t_{run} VM-1	Cost VM-1	t_{run} VM-2	Cost VM-2	t_{run} VM-3	Cost VM-3
10	250	416.66	156.25	520.83	89.28	595.23
20	131.57	438.59	82.23	548.24	46.99	626.56
30	92.59	462.96	57.87	578.70	33.06	661.37
40	73.52	490.19	45.95	612.74	26.26	700.28

TABLE V: Job type -4: Optimal with virtual machine type -1

Table VI shows a simple workload of 15 jobs scheduled using the VM-aware scheduler. The workload consists of 4 types of jobs. Tables II, III, IV and V show the performances predictions of these 4 job types made across 3 VM types. VM-1 is assumed to have 2 GB memory and 2 VCPUs and VM-2 and VM-3 are assumed to have 4 GB memory and 4 VCPUs and 8 GB memory and 8 VCPUs respectively. The tables compare 4 different cluster configurations for each VM type by varying the number of VMs from 10 to 40. The running time of the job in each cluster configuration is shown as t_{run} and the resource utilization cost is shown as $Cost$. We find that job type 1 is optimal with the VM-1 and incurs 20% additional cost with VM-2 and 30% additional cost with VM-3. Similarly, job type 2 is optimal with VM-2 and incurs 20% additional cost with VM-1 and 30% additional cost with VM-3. Job type 3 is optimal for VM-3 and incurs 30% additional cost with VM-1 and 20% additional cost with VM-2. Job type

4 is similar to job type-1 which is optimal for VM-1, but it has shorter running time.

In Table VI, the arrival time and the deadline of the jobs are shown. Now, the scheduler's goal is to choose the number of virtual machines and the virtual machine type to use for each job. At time $t = 0$, we find jobs, 1, 2, 3, 4 and 5 in the system. Based on the type of the jobs and by comparing the cost shown in Tables II - V, jobs 1, 2 and 5 are optimal with VM-1 whereas job 3 is optimal with VM-2 and job 4 is optimal with VM-3. The VM-aware scheduler chooses job 1 as the time window deciding job for VM-1 based on the cost-based priority and chooses jobs 3 and 4 as the time window deciding jobs for VM-2 and VM-3 respectively. Once the time windows are decided, it reserves and schedules job 2 in VM-1 based on the cost-based priorities by referring to the performance comparison tables. Similarly it reserves and schedules job 5 in VM-3, however job 5 is optimal only with VM-1. As there is not enough resources available in the VM pool of VM-1, the scheduler is forced to schedule it in VM-3 although it knows that it is less efficient.

At time $t = 5$, job 6 arrives and it is scheduled in VM-2 within the reservation time window as the other permissible cluster configurations using the VM types cannot meet its deadline. When job 7 arrives at time, $t = 105$ it is reserved and scheduled in VM-1 within its reservation time window. At time $t = 160$, when job 8 arrives, the scheduler identifies that it is optimal with VM-1, however as there is not enough VMs in VM-1, it schedules it in VM-3 as the reservation of job 8 starts within the current reservation time window of VM-3. When job 9 arrives, it gets reserved on VM-1 to start at $t = 225$ as it is optimal with VM-1. However, when job 10 arrives at $t = 220$ it overrides job 9 by possessing higher priority and hence job 9's reservation is cancelled and job 10 is reserved and scheduled at $t = 225$.

After job 11 arrives at time $t = 230$ and gets scheduled at $t = 250$, the reservation time window needs to be updated for VM-1. The scheduler compares the priority based on the cost and identifies job 11 as the time window deciding job and schedules it at time $t = 250$. Subsequently, job 9's reservation is also made at the earliest possible, $t = 357$ within the new reservation time window. When job 12 arrives, the scheduler identifies that it is optimal with VM-2 and it is reserved at the earliest possible time $t = 302$ and at that time the reservation time window for VM-2 is also updated with job 12. We note that job 13 is optimal with VM-1, however it gets reserved and scheduled only with VM-3 as it has stronger deadline requirements that only VM-3 can satisfy given the available resources in the other pools. Job 14 arrives at $t = 430$ and gets reserved and scheduled at $t = 450$ which also updates the reservation time window of VM-2. However, Job 15 which is optimal with VM-1 needs to be scheduled with VM-3 due to lack of available resources in VM-1 pool. Thus the VM-aware scheduler minimizes the overall resource usage cost even though some jobs violate their per-job optimality.

VM-aware Schedule with Reconfiguration-based VM management: For the same workload shown in Table VI, with the reconfiguration-based VM pool management, the allocation of the VMs in each pool is based on the current workload characteristics. For the example simplicity, we do not show the reconfiguration process in detail, instead we assume that the reconfiguration is performed and illustrate the example with the efficient schedule obtained by the VM-aware scheduler with the reconfigured VM pools. In Table VII, we note that all the jobs of job type 1 and job type 4 are scheduled using their optimal VM type VM-1. Similarly type 2 and type 3 jobs also obtain their optimal VM types VM-2 and VM-3 respectively.

Job id	type	arrival time	deadline	VM	No VMs	start	end
1	4	0	270	1	10	0	250
2	4	0	150	1	20	0	132
3	2	0	275	2	20	0	264
4	3	0	475	3	20	0	461
5	1	0	185	3	20	0	170
6	1	5	310	2	20	0	302
7	1	105	250	1	30	132	225
8	1	160	500	3	10	170	492
9	1	215	850	1	20	357	831
10	1	220	400	1	20	225	357
11	1	230	650	1	20	250	624
12	2	240	460	2	40	302	450
13	1	400	800	3	10	461	783
14	2	430	730	2	20	450	714
15	4	460	700	3	20	492	662

TABLE VI: VM-aware schedule

Job id	type	arrival time	deadline	VM	No VMs	start	end
1	4	0	270	1	10	0	250
2	4	0	150	1	20	0	132
3	2	0	275	2	20	0	264
4	3	0	475	3	20	0	461
5	1	0	185	1	70	0	172
6	1	5	310	1	40	0	270
7	1	105	250	1	30	132	225
8	1	160	510	1	30	172	502
9	1	215	850	1	20	357	831
10	1	220	400	1	20	225	357
11	1	230	650	1	20	250	624
12	2	240	460	2	20	264	529
13	1	400	800	1	30	400	734
14	2	480	730	2	20	529	773
15	4	460	700	1	20	460	592

TABLE VII: Schedule with Reconfiguration-based VM Management