Cost-Effective Resource Provisioning for MapReduce in a Cloud

Balaji Palanisamy, *Member, IEEE*, Aameek Singh, *Member, IEEE*, and Ling Liu, *Senior Member, IEEE*

Abstract—This paper presents a new MapReduce cloud service model, Cura, for provisioning cost-effective MapReduce services in a cloud. In contrast to existing MapReduce cloud services such as a generic compute cloud or a dedicated MapReduce cloud, Cura has a number of unique benefits. First, Cura is designed to provide a cost-effective solution to efficiently handle MapReduce production workloads that have a significant amount of interactive jobs. Second, unlike existing services that require customers to decide the resources to be used for the jobs, Cura leverages MapReduce profiling to automatically create the best cluster configuration for the jobs. While the existing models allow only a per-job resource optimization for the jobs, Cura implements a globally efficient resource allocation scheme that significantly reduces the resource usage cost in the cloud. Third, Cura leverages unique optimization opportunities when dealing with workloads that can withstand some slack. By effectively multiplexing the available cloud resources among the jobs based on the job requirements, Cura achieves significantly lower resource usage costs for the jobs. Cura's core resource management schemes include cost-aware resource provisioning, VM-aware scheduling and online virtual machine reconfiguration. Our experimental results using Facebook-like workload traces show that our techniques lead to more than 80 percent reduction in the cloud compute infrastructure cost with upto 65 percent reduction in job response times.

Index Terms—MapReduce, cloud computing, cost-effectiveness, scheduling

✦

1 INTRODUCTION

CLOUD computing and its pay-as-you-go cost structure have enabled hardware infrastructure service providers, platform service providers as well as software and application service providers to offer computing services on demand and pay per use just like how we use utility today. This growing trend in cloud computing, combined with the demands for Big Data and Big Data analytics, is driving the rapid evolution of datacenter technologies towards more cost-effective, consumer-driven and technology agnostic solutions. The most popular approach towards such big data analytics is using MapReduce [1] and its open-source implementation called Hadoop [15]. 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 this paper, we discuss the cost-inefficiencies of the existing cloud services for MapReduce and propose a cost-effective resource management framework called Cura that aims at a globally optimized resource allocation to minimize the infrastructure cost in the cloud datacenter. We note that

● *B. Palanisamy is with the School of Information Sciences, University of Pittsburgh, Pittsburgh, PA. E-mail: bpalan@pitt.edu.*
● *A. Singh is with Storage Systems, IBM Almaden Research Center 650, Harry Road, San Jose, CA. E-mail: Aameek.Singh@us.ibm.com.*
● *L. Liu is with the College of Computing, Georgia Institute of Technology, Atlanta, GA. E-mail: lingliu@cc.gatech.edu.*

the existing cloud solutions for MapReduce work primarily based on a per-job or per-customer optimization approach where the optimization and resource sharing opportunities are restricted within a single job or a single customer. For instance, in existing dedicated MapReduce cloud services such as Amazon Elastic MapReduce [13], customers buy on-demand clusters of VMs for each job or a workflow and 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. Here the resource optimization opportunity is restricted to the per-job level. Alternately, one can lease dedicated cluster resources from a generic cloud service like Amazon Elastic Compute Cloud [14] and operate MapReduce on them as if they were using a private MapReduce infrastructure. While this approach enables resource optimization at the per-customer level, we argue that in such an approach, the size of the leased dedicated clusters needs to be chosen based on the peak workload requirements of each customer and hence, the leased clusters are under-utilized for a large fraction of the time leading to higher costs. Cura on the other hand is designed to provide a cost-effective solution to a wide range of MapReduce workloads with the following goals in mind:

First, we observe that existing solutions are not cost-effective to deal with interactive MapReduce workloads that consist of a significant fraction of short running jobs with lower latency requirements. A recent study on the Facebook and Yahoo production workload traces [29], [8], [34] reveals that more than 95 percent of their production MapReduce jobs are short running jobs with an average running time of 30 sec. Unlike existing per-job services that require VMs to be created afresh for each submitted job, Cura deals with such interactive workloads using a secure instant VM allocation scheme that minimizes the job

latency. In addition, Cura results in higher cost-effectiveness compared to an owned cluster in a generic compute cloud that has high costs due to low utilization.

Second, as discussed earlier, existing cloud solutions are largely optimized based on per-job and per-customer optimization which leads to poor resource utilization and higher cost. Additionally, their usage model requires 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. In contrast to existing cloud systems, Cura leverages MapReduce profiling to automatically create the best cluster configuration for the jobs and tries to optimize the resource allocation in a globally cost-effective fashion. Hence, Cura results in requiring much lesser cloud resources than that consumed by the current models.

Finally, Cura's usage model and techniques achieve higher service differentiation than existing models as Cura incorporates an intelligent multiplexing of the shared cloud resources among the jobs based on job requirements. 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 leveraging such opportunities and by accurately understanding each job's performance requirements, Cura multiplexes the resources for significant cost savings.

To the best of our knowledge, Cura is the first effort that is aimed at developing a novel usage model and resource management techniques for achieving global resource optimization in the cloud for MapReduce services. Cura uses a secure instant VM allocation scheme that helps reduce the response time for short jobs by up to 65 percent. By leveraging MapReduce profiling, Cura tries to optimize the global resource allocation in the cloud by automatically creating the best cluster configuration for the jobs. Cura's core resource management techniques include cost-aware resource provisioning, intelligent VM-aware scheduling and online VM reconfiguration. Overall, in addition to the response time savings, Cura results in more than 80 percent savings in the cloud infrastructure cost. The rest of the paper is organized as follows. In Section 2, we present Cura's cloud service model and system architecture. Section 3 discusses the technical challenges of Cura and presents Cura's resource management techniques namely VM-aware scheduling and online VM reconfiguration including a formal discussion of Cura's scheduling and reconfiguration problems. We present the experimental evaluation of Cura in Section 4. We discuss related work in Section 5 and conclude in Section 6.

2 CURA: MODEL AND ARCHITECTURE

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

2.1 Cloud Operational Model

Table 1 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

TABLE 1
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

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 two small instances but takes 20 minutes to complete using a cluster of six 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 two small instances as it has lower resource usage cost compared to the six 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. Typically, the additional risks here include the responsibilities of meeting additional SLA requirements such as executing each job within its deadline and managing the allocation of resources for each job. While the conventional customer-optimized cloud model requires only VMs to be provisioned based on demand, a completely cloud managed model introduces additional role on the cloud provider for resource management. For instance, an inefficient allocation of resources to a particular job can result in higher cost for the cloud provider. Therefore, this model brings higher risk to the cloud while it has high potential cost benefits. 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

---

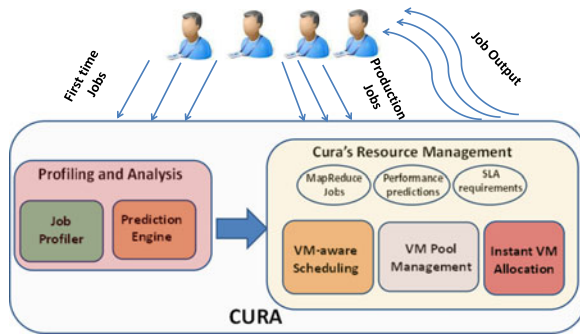1. Example adapted from the measurements in Herodotou et al. paper [25].

Fig. 1. Cura: System architecture.

Fig. 2. VM pool.

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 [4], [5], [24], [27], [28], 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.

2.2 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.

Once a job is submitted to Cura, it may take one of the two paths (Fig. 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 later on. When making scheduling decisions, performance predictions in terms of execution time are made based on the input dataset size, 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 [4], [5], [24] 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
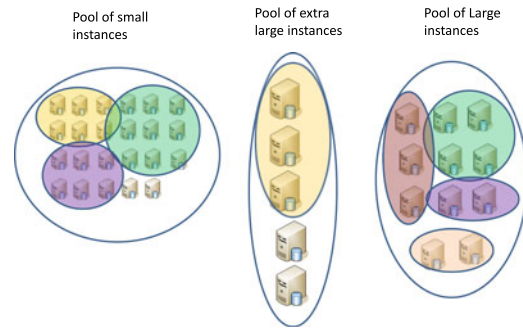
accuracy with less than 12 percent error rate for the predicted running time [27]. Here we would like to note that though Cura's architecture and techniques may apply to other HPC scheduling problems beyond just Map Reduce, we have considered dedicated MapReduce clouds as the target scenario of Cura for the wide popularity of MapReduce and the availability of several open source profile and analyze tools for MapReduce.

2.3 System Architecture

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:

2.3.1 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 Fig. 2 and dynamically creates Hadoop clusters

---

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].

on them. By default, Cura runs the maximum number of pre-created VMs in the cloud (limited by the number of servers) so that all workload can be served instantly.

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 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.

2.3.2   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 3.1) 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.

2.3.3   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 3.2). In addition, this component may perform further optimization such as power management by suitably shutting down VMs at low load conditions.

2.4   Deployment Challenges and Practical Use

In this section, we discuss some basic challenges in deploying a globally optimized resource management model like Cura in a commercial cloud setting. First of all, a global optimization model such as Cura brings additional responsibility to the cloud provider in meeting the SLA requirements for the jobs and to the customers. Though the proposed model is not a must for cloud service providers to function, they can obtain significant benefits by offering such a model. While this model brings attractive cost benefits to both customers and cloud providers, we would need appropriate incentive models for both cloud providers and customers in order to function symbiotically. The emergence of new cloud managed models in commercial services (such as Google Big Query [34]) suggests that the additional management overhead on the cloud providers might be quite practical given the wide range of cost benefits such models bring. Motivated by the huge benefits of global resource management, similar models have also been proposed in the context of database-as-a-service [10], [11], [12] and have been shown to be practical. Another key challenge in globally optimized resource management is that the scheduling and resource allocation techniques need to be highly scalable and efficient to work in even scenarios with thousands of servers and with tens of thousands of customer jobs. This calls for highly scalable scheduling techniques and we believe there is many possible future work along this direction. Finally, we also believe that resource pricing in a globally optimized cloud can be quite a challenge and needs attention from both business perspective as well as from the resource management perspective. When pricing models are closely integrated with resource management techniques, there are huge opportunities for scheduling techniques where resource management decisions are influenced by intimately coupled pricing models. We believe such sophisticated models will be interesting extensions to the Cura global resource management model.

3   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.

3.1   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. As jobs are incrementally submitted to the cloud, scheduling requires an online algorithm that can place one job at a time on an infrastructure already executing some jobs. To better understand the complexity of the problem, we first analyze an *offline* version which leads us to the design of an online scheduler.

3.1.1 Offline VM-Aware Scheduling

In the offline VM-aware scheduling problem, we assume that information about the jobs, their arrival time and deadlines are known *apriori* and the goal of the algorithm is to schedule all jobs to meet their deadline by appropriately provisioning VM clusters and to minimize the overall resource usage in the cloud. We assume each job, J_i is profiled when it first goes to production and based on the profile it has a number of predictions across various cluster configurations, $C^{k,n}$ in terms of instance types denoted by k and number of VMs denoted by n. Let $t_{arrival}(J_i)$ and $t_{deadline}(J_i)$ denote the arrival time and deadline of job, J_i respectively. The running time of the job, J_i using the cluster configuration, $C^{k,n}$ is given by $t_{run}(J_i, C^{k,n})$ and it includes both execution time and the time for loading data into the HDFS. $Cost(J_i, C^{k,n})$ represents the resource usage cost of scheduling job, J_i using the cluster configuration, $C^{k,n}$. Precisely, the cost, $Cost(J_i, C^{k,n})$ represents the product of the number of physical servers required to host the virtual cluster, $C^{k,n}$ and the running time of the job, $t_{run}(J_i, C^{k,n})$. If R_k represents number of units of physical resources in VM type, k and if each physical server has M units of physical resources[4], the resource usage cost can be computed as:

$$Cost(J_i, C^{k,n}) = t_{run}(J_i, C^{k,n}) \times \frac{n \times R_k}{M}.$$

Handling prediction errors. Additionally, $t_{run}(J_i, C^{k,n})$ includes an error bound in the prediction to ensure that the job will complete within its deadline even when there is prediction error. This error can also account for deviations in predicted running time due to interference among multiple MapReduce jobs when they execute in parallel. If $t_{actualrun}(J_i, C^{k,n})$ represents the actual running time and if $t_{error}(J_i, C_{k,n})$ represents the error bound in the predicted running time, we have

$$t_{run}(J_i, C^{k,n}) = t_{actualrun}(J_i, C^{k,n}) + t_{error}(J_i, C^{k,n}).$$

This conservative estimate of $t_{run}(J_i, C^{k,n})$ guarantees that the job completes execution even when there is prediction error.

Let $t_{start}(J_i)$ be the actual starting time of the job, J_i and therefore the end time of job, J_i is given by

---

$$t_{end}(J_i) = t_{start}(J_i) + \sum_{k,n} X_i^{k,n} \times t_{run}(J_i, C^{k,n}),$$

where $X_i^{k,n}$ is a Boolean variable indicating if job, J_i is scheduled using the cluster configuration, $C^{k,n}$ and

$$\forall i, \sum_{k,n} X_i^{k,n} = 1.$$

In order to ensure that all jobs get completed within their deadlines, we have

$$\forall i, t_{end}(J_i) \leq t_{deadline}(J_i).$$

The sum of concurrent usage of VMs among the running jobs is also constrained by the number of VMs, V_k in the VM pools where k represents the VM type. If S_i^t is a Boolean variable indicating if job, J_i is executing at time, t, we have

$$S_i^t = \begin{cases} 1 & \text{if } t_{start}(J_i) \leq t \leq t_{end}(J_i) \\ 0 & \text{otherwise} \end{cases}$$

$$\forall t, \forall k, \sum_i \left(S_i^t \times \sum_n \left(X_i^{k,n} \times n \right) \right) \leq V_k.$$

With the above constraints ensuring that the jobs get scheduled to meet deadlines, now the key optimization is to minimize the overall resource usage cost of all the jobs in the system.

$$Overall cost = \min \sum_{i,k,n} Cost(J_i, C^{k,n}) \times X_i^{k,n}.$$

An optimal solution for this problem is NP-Hard with a reduction from the known NP-Hard multi bin-packing problem [21] with additional job moldability constraints. Therefore, we use a heuristics based VM-aware scheduler which is designed to work in an online fashion.

3.1.2 Online VM-Aware Scheduler

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

---

4. Though we present a scalar capacity value, VM resources may have multiple dimensions like CPU, memory and disk. To handle this, our model can be extended to include a vector of resources or compute dimensions can be captured in a scalar value, e.g. the volume metric [13].

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 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. Concretely, if $Cost(J_i, J_j)$ denotes the resource usage cost of the schedule obtained by reserving job J_i after reserving job J_j and *Jlist* represents the job queue, then the highest priority job is chosen as the job J_i that maximizes the following.

$$\sum_{J_j \in Jlist} Cost(J_j, J_i) - Cost(J_i, J_j).$$

For a job, the least cost cluster configuration (VM type, number of VMs) at a given time is found based on the performance predictions developed for the job during the profile and analyze phase. It is essentially the lowest cost cluster configuration that can meet the jobs deadline and for which resources are available.

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 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 pseudocode for this VM-aware scheduler in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2014.2320498.

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 up of x with x parallel machines, each of them performing $\frac{n}{x}$ pairwise cost comparisons.

Fig. 3a 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 jobs 3, 12, and 14 are optimal with VM-2 and job 4 is optimal with VM-3. Here, 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 B, available in the online supplemental material.
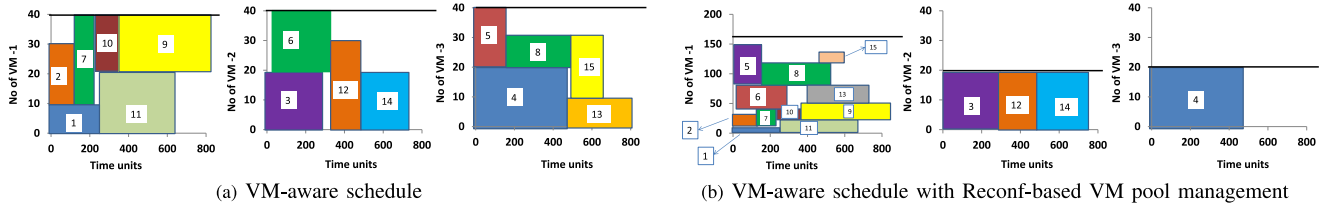
(a) VM-aware schedule (b) VM-aware schedule with Reconf-based VM pool management

Fig. 3. Scheduling in Cura.

3.2 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 Fig. 2 can be reconfigured as shown in Fig. 4 to have more small instances by shutting down some large and extra large instances if the current workload pattern requires more small instances.

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, $C_{opt}(J_i)$. Concretely, the running time of the jobs under a given cluster configuration is predicted by the profile and analyze tool. Based on the amount of resources in the cluster configuration (i.e number of VMs and configuration of each VM) and the running time, Cura computes the total resource cost as described in Section 3.1.1. Cura uses this information to find the optimal cluster configuration as the one which minimizes this resource usage cost while being able to complete the job within its deadline. The reconfiguration manager understands the current demands for each VM instance type in terms of the average number of VMs required for each VM type in order to successfully provision the optimal cluster configuration to the jobs observed in the reconfiguration time window. 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. If $Y_i^{k,n}$ is a Boolean variable indicating if $C^{k,n}$ is the optimal cluster configuration for job, J_i, then the proportion of physical resources, P_k to

be allocated to each VM type k can be estimated based on the cumulative resource usage in each VM pool computed as the product of total running time of the jobs and the size of the cluster used:

$$P_k = \frac{\sum_{i,n}\left(t_{run}(J_i, C_{opt}(J_i)) \times n \times Y_i^{k,n}\right)}{\sum_{i,k,n}\left(t_{run}(J_i, C_{opt}(J_i)) \times n \times Y_i^{k,n}\right)}.$$

The total physical resources, R^{total} in the cloud infrastructure can be obtained as

$$R^{total} = \sum_k V_k \times R_k,$$

where R_k represents the physical resource in VM type, k, and V_k is the number of VMs in the existing VM pool of type k. Therefore, the number of VMs, V_k' in the new reconfigured VM pools is given by

$$V_k' = P_k \times \frac{R^{total}}{R_k}.$$

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 overall observed cost represents the actual cumulative resource cost of the jobs executed during the reconfiguration time window using existing VM pools. Here, $Z_i^{k,n}$ is a Boolean variable indicating if the job J_i used the cluster configuration, $C_i^{k,n}$

$$Overallcost_{observed} = \sum_{i,k,n} Cost(J_i, C^{k,n}) \times Z_i^{k,n}.$$

Next, we compute the estimated overall cost with new VM pools assuming that the jobs were scheduled using their optimal cluster configurations, $C_{opt}(J_i)$. Reconfiguration benefit, $Reconf_{benefit}$ is then computed as the difference between the two

$$Overallcost_{estimate} = \sum_i Cost(J_i, C_{opt}(J_i))$$

$$Reconf_{benefit} = Overallcost_{estimate} - Overallcost_{actual}.$$

Assuming the reconfiguration process incurs an average reconfiguration overhead, $Reconf_{overhead}$ that represents the resource usage spent on the reconfiguration process for each VM that undergoes reconfiguration, the total cost of reconfiguration is obtained as

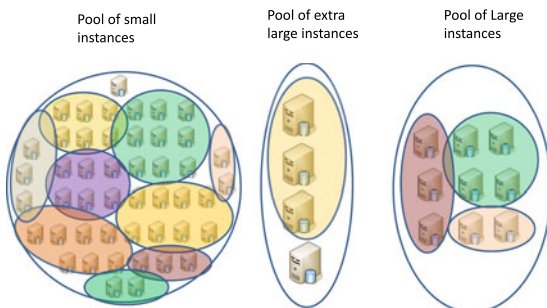$$Reconf_{cost} = \sum_k |(V_k' - V_k)| \times Reconf_{overhead}.$$

Fig. 4. Reconfiguration-based VM management.

The algorithm then triggers the reconfiguration process only if it finds that the estimated benefit exceeds the reconfiguration cost by a factor of β, i.e., if $Reconf_{benefit} \geq \beta \times Reconf_{cost}$ where $\beta > 1$. As $Reconf_{benefit}$ only represents an estimate of the benefit, β is often chosen as a value greater than 1. 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 Fig. 3, we find that the basic VM-aware scheduler in Fig. 3a 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 Fig. 3b 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 B, available in the online supplemental material, for interested readers.

4 EXPERIMENTAL EVALUATION

We divide the experimental evaluation of Cura into two—first, we provide detailed analysis on the effectiveness of Cura compared to conventional MapReduce services and then we present an extensive micro analysis on the different set of techniques in Cura that contribute to the overall performance. We first start with our experimental setup.

4.1 Experimental Setup

Cluster setup. Our profiling cluster consists of 20 CentOS 5.5 physical machines (KVM as the hypervisor) with 16 core 2.53 GHz 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 six VM instance types with the lowest configuration starting from two 2 GHz VCPUs and 2 GB RAM to the highest configuration having 12 2 GHz VCPUs and 12 GB RAM with each VM configuration differing by two 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 six VM types. Each profile is then analyzed using *Starfish* to develop predictions across various hypothetical cluster configurations and input data sizes.

Simulation setup. In order to analyze the performance and cost benefits of Cura on a datacenter scale system, we developed a simulator in Java that uses the profiles and performance predictions developed from the real cluster. The simulator models a cloud datacenter with servers, each having a 16 core 2.53 GHz processors with 16 GB RAM. It implements both the VM-aware scheduling with the instant VM allocation and the reconfiguration-based VM management techniques. The execution time for each job in the simulation is assumed as the predicted execution time (based on the profiles generated from the profiling cluster) and a prediction error which could be either a positive or negative error within an assumed error bound.

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*: techniques that have higher response time provide poor service quality; this metric captures the service quality of the jobs, (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.

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 and reconfiguration-based VM pool management. For the micro-analysis, we also compare the following sub-techniques to better evaluate Cura: *1) Per-job Optimization* technique that uses Cura's secure instant VM allocation but always uses the per-job optimal number of VMs and the optimal VM type, *2) VM-aware scheduler* described in Sections 3.1 and *3) Reconfiguration based VM Management* (Section 3.2).

4.2 Experimental Results

We first present the experimental evaluation of Cura by comparing with the existing techniques for various experimental conditions determined by distribution of the job deadlines, size of the MapReduce jobs, number of servers in the system and the amount of prediction error in the *profile and analyze* process. 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

(a) Number of Servers

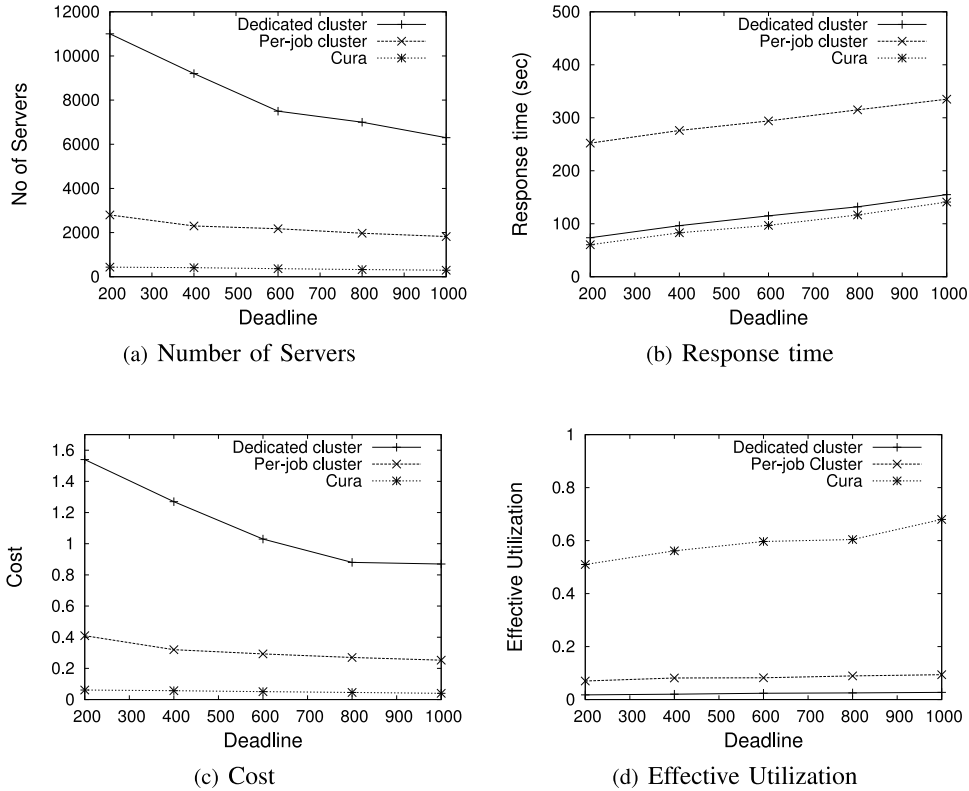(b) Response time

(c) Cost

(d) Effective Utilization

Fig. 5. Effect of job-deadlines.

jobs read 100 MB of data, whereas medium jobs and large jobs read 1 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. By default, we assume that jobs run for the same amount of time predicted in the *profile and analyze* process, however, we dedicate a separate set of experiments to study the performance of the techniques when such predictions are erroneous. 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[5] from the storage server and the Hadoop startup time is taken as 10 sec.

4.2.1 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 (Fig. 5) and then we analyze the performance of Cura in terms of the contributions of each of its sub-techniques (Fig. 6). Fig. 5a shows the performance of the techniques for different maximum deadlines with respect to number of

5. 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].

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 (Fig. 5a) as these resources are shared among the customers. However, the Cura approach in Fig. 5a has a much lower resource requirement having up to 80 percent 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.

We also compare the approaches in terms of the mean response time in Fig. 5b. To allow each compared technique to successfully schedule all jobs (and not cause failures), we use the number of servers obtained in Fig. 5a 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 percent).

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

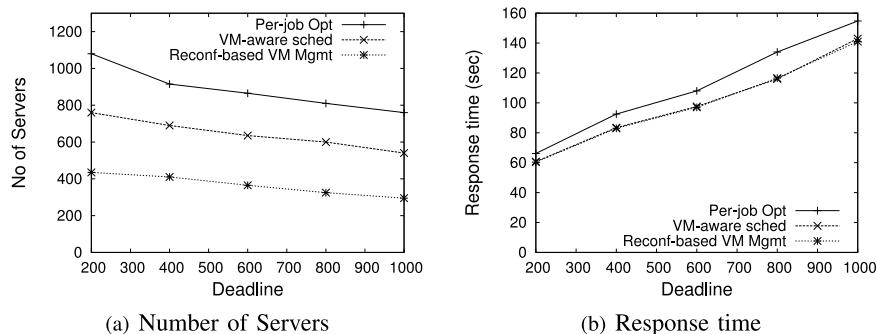(a) Number of Servers

(b) Response time

Fig. 6. Effect of job-deadlines.

trend with respect to the per-job cost in Fig. 5c that shows that the Cura approach can significantly reduce the per-job infrastructure cost of the jobs (up to 80 percent). The effective utilization in Fig. 5d 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 7× improvement compared to the other techniques as Cura effectively leverages global optimization and deadline-awareness to achieve better resource management.

Micro analysis. Next, we discuss the performance of the sub-techniques of Cura and illustrate how much each sub-technique contributes to the overall performance under different deadlines. Fig. 6a shows that with only per-job optimization (which only leverages instant VM allocation), it requires up to 2.6× higher number of servers compared to using reconfiguration-based VM pool management scheme with the VM-aware scheduler. The per-job optimization scheduler always chooses the optimal VM type and the optimal number of VMs for each job and in case the optimal resources are not available when the job arrives, the scheduler keeps on queuing the job until the required optimal resource becomes available when some other jobs complete. It drops the request when it finds out that the job cannot meet its deadline if the optimal resources are provisioned. However, with the VM-aware approach, the scheduler will be able to still schedule the job by provisioning higher resources in order to meet the deadline. Second, with the per-job optimization scheduler, even when some sub-optimal resources are available when the job is waiting, they remain unused as the job is expecting to be scheduled only using the optimal resources. Therefore the per-job optimization results in poor performance. The number of servers required by the VM-aware approach is significantly reduced by up to 45 percent servers by efficient reconfiguration-based VM management that dynamically manages the VMs in each VM pool. Fig. 6b shows the mean response time of the jobs for various sub-techniques. We find that the sub-techniques have similar response times except for the per-job optimization case that has up to 11 percent higher mean response time. As per-job

optimization scheduler keeps the jobs waiting until it finds their optimal resources, it leads to higher queuing time that causes this increase.

4.2.2   Effect of Prediction Error

This set of experiments evaluates the techniques by studying the effect of inaccuracies in the performance prediction. As accurate performance predictions may not always be available, it is important that the techniques can tolerate inaccuracies in performance prediction and yet perform efficiently. Fig. 7 shows the comparison of the techniques while varying the error rate from 0 to 70 percent. Here, the mean deadline of the jobs is taken as 800 second. The error rate means that accurate running time of the jobs can be anywhere within the error range on both sides of the predicted value. The comparison of number of servers in Fig. 7a shows that all the techniques require more number of servers when the prediction error increases. The Cura approach on an average requires 4 percent additional number of servers for every 10 percent increase in prediction error. Note that even the per-job cluster and dedicated cluster schemes require increased number of servers as they also decide the resource requirements based on the performance predictions of the jobs across different cluster configurations.

Fig. 7b shows that the response time of the techniques decreases with increase in the error rate. While the Cura and dedicated cluster approaches have a decrease of 4.2 and 3.7 percent respectively, the per-job cluster approach has a decrease of only 1.4 percent for every 10 percent increase in error rate as the major fraction of the response time in these services is due to the VM creation process. As error rate increases, the techniques provision more resources to ensure that even in the worst case, when the jobs run for the maximum possible time within the error range, the jobs complete within the deadline. Therefore, in cases where the job completes within the maximum possible running time, these additional resources make the job complete earlier than its deadline and therefore it speeds up the execution resulting in lower response time. The cost trend shown in Fig. 7c also shows that the techniques that require fewer servers result in lower per-job cost. Similarly the effective utilization comparison in Fig. 7d shows similar relative performance as in Fig. 5d.

We compare the performance of the sub-techniques of Cura under different error rates in Fig. 8. We find that the

(a) Number of Servers

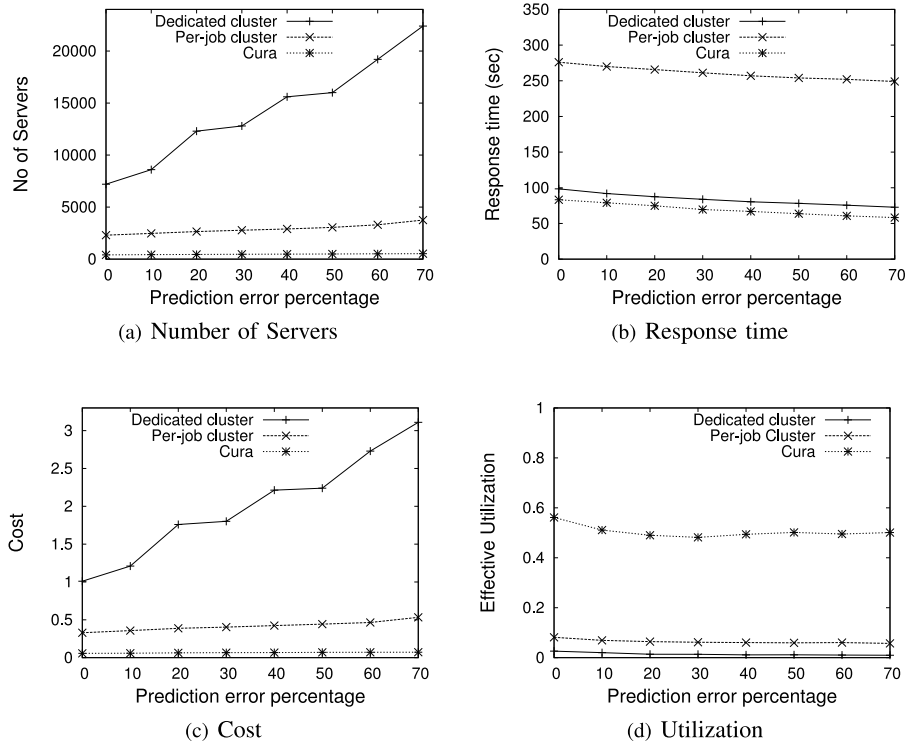(b) Response time

(c) Cost

(d) Utilization

Fig. 7. Effect of prediction error.

number of servers in Fig. 8a shows a similar relative performance among the sub-techniques as in 7a. Here again, the response time as shown in Fig. 8b shows that the per-job optimization scheduler leads to higher response time due to queue wait times and the response time of the sub-techniques increases with increase in error rate.

4.2.3 Varying Number of Servers

We next study the performance of the techniques by varying the number of servers provisioned to handle the workload. Fig. 9a 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 percent rate only with 2,000 servers. Fig. 9b shows that the response time of successful jobs in the compared approaches show a similar trend as in Fig. 5b where the Cura approach performs better than the per-job cluster services.

4.2.4 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 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 1,000 second as they are long running. We find that the performance in terms of number of servers in Fig. 10a 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 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 4× and 2× 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
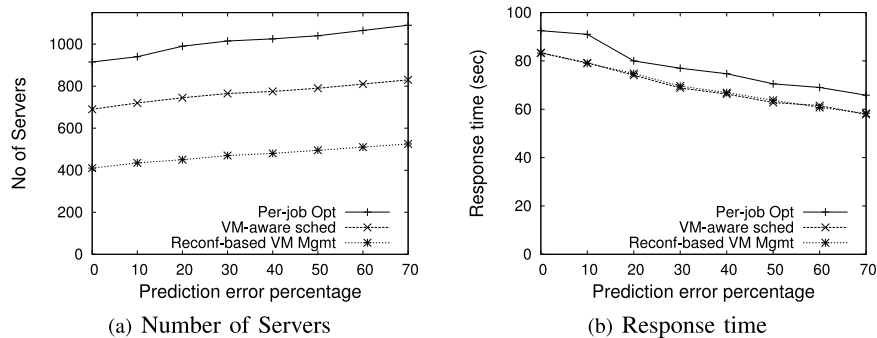

(a) Number of Servers

(b) Response time

Fig. 8. Effect of prediction error.

(a) Success Rate   (b) Response time
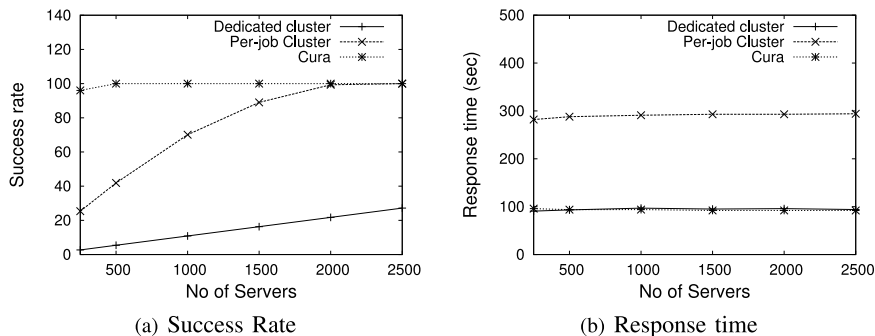
Fig. 9. Effect of servers.

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 Fig. 10b also show that the improvement is very significant for short jobs having up to 87 percent reduced response time and up to 69 percent for medium jobs. It is reasonably significant for large jobs with up to 60 percent lower response time and extra large jobs with up to 30 percent reduced response time. The cost comparison in Fig. 10c 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.

The sub-technique comparison of Cura for various job types in terms of number of servers is shown in Fig. 11a. We find that the sub-techniques have impact on all kind of jobs irrespective of the job size. While secure instant VM

allocation contributes more to the performance of the small jobs compared to large jobs, the sub-techniques in Cura have equal impact on the overall performance for all job categories. The response time comparison of the sub-techniques in Fig. 11b shows that the sub-techniques have similar response time, however, for large and extra large jobs, the per-job optimization leads to increased response time by up to 24.8 percent as large jobs in the per-job optimization require incur longer waiting time in the queue as they often request more resources that may not be immediately available.

4.3   Effect of Deadline Distribution

In Fig. 12, we study the effect of different distributions of deadline on the performance of Cura for different mean deadlines. In Fig. 12a, we find that both the Poisson and uniform distributions require similar number of servers whereas the exponential deadline distribution requires up to 30 percent additional servers as there are more jobs with shorter deadlines in the exponential distribution. The response time comparison in Fig. 12b shows that irrespective
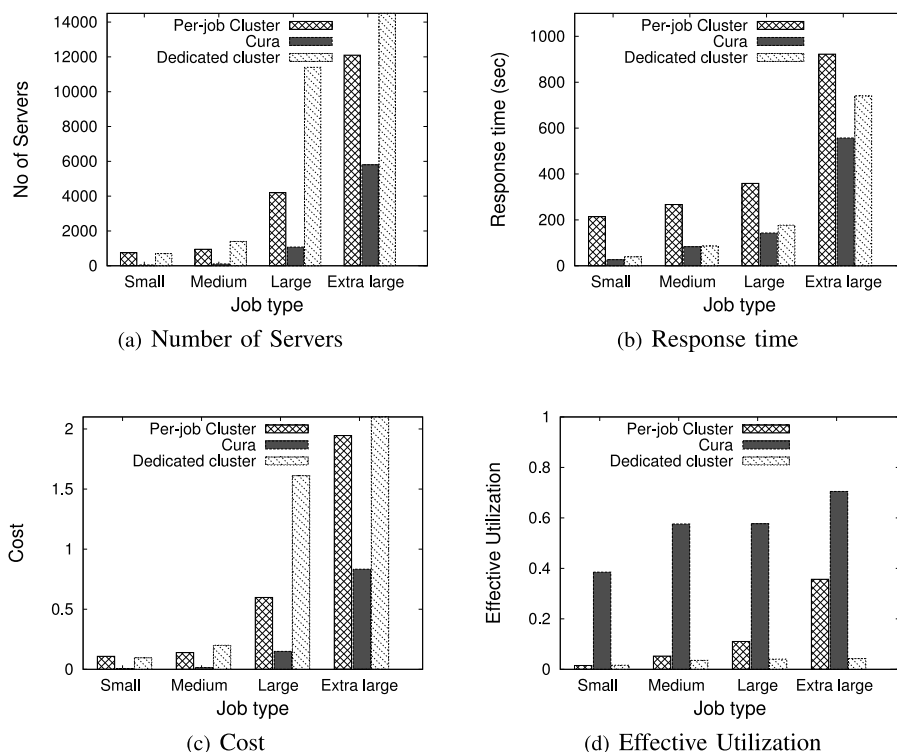

(a) Number of Servers   (b) Response time

(c) Cost   (d) Effective Utilization

Fig. 10. Effect of job type.
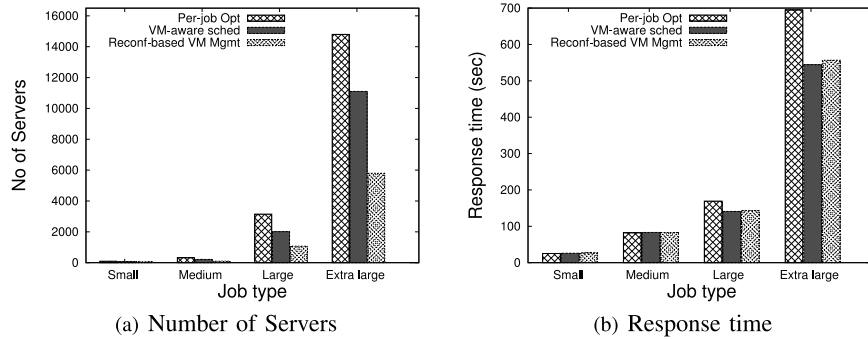
(a) Number of Servers   (b) Response time

Fig. 11. Effect of job type.

of the deadline distribution, the jobs have more or less similar response time.

5 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 and Mahadik 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 [17], [32]. *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], [5], [23], [28], [29]. Herodotou and Babu developed an automated performance prediction tool based on their profile and analyze tool *Starfish* [24] to guide customers to

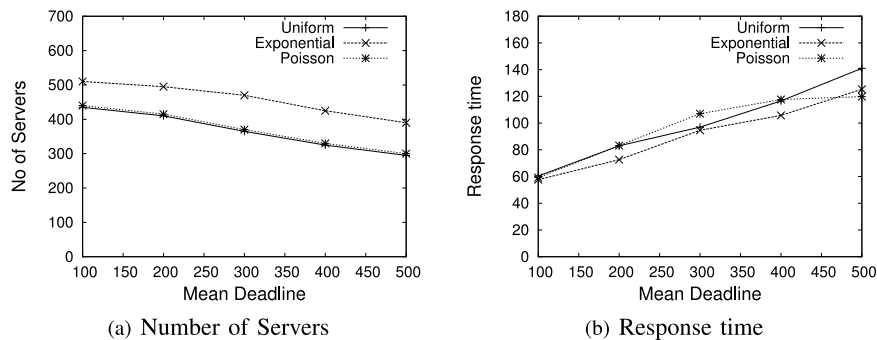(a) Number of Servers   (b) Response time

Fig. 12. Effect of deadline distribution.

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 percent 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.

6   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 percent reduction in infrastructure cost with 65 percent lower job response times.

REFERENCES

[1]   B. Igou, "User survey analysis: Cloud-computing budgets are growing and shifting; traditional IT services providers must prepare or perish," Gartner, Inc., Stamford, CT, USA, Gartner Rep. G00205813, 2010.
[2]   J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Symp. Operating Syst. Des. Implementation*, 2004, p. 10.
[3]   G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in Map-Reduce clusters using Mantri," in *Proc. 9th USENIX Conf. Operating Syst. Des. Implementation*, 2010.
[4]   K. Kambatla, A. Pathak, and H. Pucha, "Towards optimizing Hadoop provisioning in the cloud," in *Proc. Conf. Hot Topics Cloud Comput.*, 2009.
[5]   K. Morton, A. Friesen, M. Balazinska, and D. Grossman, "Estimating the progress of MapReduce pipelines," in *Proc. IEEE 26th Int. Conf. Data Eng.*, 2010, pp. 681–684.
[6]   Pig user guide. [Online]. Available: http://pig.apache.org/, 2014.
[7]   A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive - A warehousing solution over a MapReduce framework," *Proc. VLDB Endowment*, vol. 2, pp. 1626–1629, 2009.
[8]   D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash facebook, R. Schmidt, and A. Aiyer, "Apache Hadoop goes realtime at Facebook," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 1071–1080.
[9]   S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: Interactive analysis of web-scale datasets," in *Proc. 36th Int. Conf. Very Large Data Bases*, 2010, pp. 330–339.
[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 *Proc. 5th Biennial Conf. Innovative Data Syst. Res.*, 2011, pp. 235–240.
[11] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger, "Multi-tenant databases for software as a service: Schema-mapping techniques," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 1195–1206.
[12] P. Xiong, Y. Chi, S. Zhu, J. Tatemura, C. Pu, and H. Hacigumus, "ActiveSLA: A profit-oriented admission control framework for database-as-a-service providers," in *Proc. 2nd ACM Symp. Cloud Comput.*, 2011.
[13] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proc. 4th USENIX Conf. Netw. Syst. Des.Implementation*, 2007, p. 17.
[14] Amazon Elastic MapReduce. [Online]. Available: http://aws.amazon.com/elasticmapreduce/, 2014.
[15] Amazon Elastic Compute Cloud. [Online]. Available: http://aws.amazon.com/ec2/, 2014.
[16] Hadoop. [Online]. Available: http://hadoop.apache.org, 2014.
[17] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *Proc. 8th USENIX Conf. Operating Syst. Des. Implementation*, 2008, pp. 29–42.
[18] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: Fair scheduling for distributed computing clusters," in *Proc. ACM SIGOPS 22nd Symp. Operating Syst. Principles*, 2009, pp. 261–276.
[19] I. Roy, Srinath. Setty, A. Kilzer, V. Shmatikov, and E. Witchel, "Airavat: Security and privacy for MapReduce," in *Proc. 7th USENIX Conf. Netw. Syst. Des. Implementation*, 2010, p. 20.
[20] SELinux user guide. [Online]. Available: http://selinuxproject.org/page/Main_Page, 2014.
[21] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*," San Francisco, CA, USA: Freeman, 1979.
[22] S. L. Garfinkel. (2007). An evaluation of Amazon's grid computing services: EC2, S3 and SQS. School Eng. Appl. Sci., Harvard Univ., Cambridge, MA, USA, Tech. Rep. TR-08-07, [Online]. Available: 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 *Proc. IEEE 4th Int. Conf. Cloud Comput.*, 2011, pp. 155–162.
[24] H. Herodotou et al., and S. Babu, "On optimizing MapReduce programs/Hadoop jobs," *Proc. VLDB Endowment*, vol. 12, 2011.
[25] H. Herodotou, H. Li, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, "Starfish: A selftuning system for big data analytics," in *Proc. 5th Biennial Conf. Innovative Data Syst. Res.*, 2011, pp. 261–272.
[26] H. Herodotou, F. Dong, and S. Babu, "No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics," in *Proc. 2nd ACM Symp. Cloud Comput.*, 2011.
[27] V. Jalaparti, H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Bazaar: Enabling predictable performance in datacenters," Microsoft Res., Cambridge, U.K., Tech. Rep. MSR-TR-2012-38, 2012.
[28] A. Popescu, V. Ercegovac, A. Balmin, M. Branco, and A. Ailamaki, "Same queries, different data: Can we predict runtime performance?" *Proc. 3rd Int. Workshop Self-Manag. Database Syst.*, 2012.

[29] A. Verma, L. Cherkasova, and R. H. Campbell, "Resource provisioning framework for MapReduce jobs with performance goals," in *Proc. ACM/IFIP/USENIX 12th Int. Middleware Conf.*, 2011, pp. 165–186.

[30] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The case for evaluating MapReduce performance using workload suites," in *Proc. IEEE 19th Annu. Int. Symp. Model., Anal., Simul. Comput. Telecommun. Syst.*, 2011, pp. 390–399.

[31] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "An analysis of traces from a production MapReduce cluster," in *Proc. 10th IEEE/ACM Int. Conf. Cluster, Cloud Grid Comput.*, 2010, pp. 94–103.

[32] T. Sandholm and K. Lai, "Mapreduce optimization using dynamic regulated prioritization," *ACM SIGMETRICS/Perform. Eval. Rev.*, vol. 37, pp. 299–310, 2009

[33] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: Locality-aware resource allocation for MapReduce in a cloud," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2011.

[34] Google BigQuery. [Online]. Available: https://developers.google.com/bigquery/

[35] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz, "Energy efficiency for large-scale MapReduce workloads with significant interactive analysis," in *Proc. 7th ACM Eur. Conf. Comput. Syst.*, 2012, pp. 43–56.

[36] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: Integration and load balancing in data centers," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2008, pp. 1–12.

[37] A. Mu'alem and D. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 6, pp. 529–543, Jun. 2001

[38] J. Skovira, W. Chan, H. Zhou, and D. Lifka, "The EASY - LoadLeveler API Project," in *Proc. Workshop Job Scheduling Strategies Parallel Process.*, 1996, pp. 41–47.

[39] S. Anastasiadis and K. Sevcik, "Parallel application scheduling on networks of workstations," *J. Parallel Distrib. Comput.*, vol. 43, pp. 109–124, 1997

[40] E. Rosti, E. Smirni, L. Dowdy, G. Serazzi, and B. Carlson, "Robust partitioning policies of multiprocessor systems," *Perform. Eval.*, vol. 19, pp. 141–165, 1993.

[41] S. Srinivasan, V. Subramani, R. Kettimuthu, P. Holenarsipur, and P. Sadayappan, "Effective selection of partition sizes for moldable scheduling of parallel jobs," in *Proc. 9th Int. Conf. High Perform. Comput.*, 2002, pp. 174–183.

[42] W. Cirne, "Using moldability to improve the performance of supercomputer jobs," Ph.D. dissertation, Dept. Comput. Sci. Eng., Univ. California, San Diego, La Jolla, CA, USA, 2001.

[43] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2011.

[44] B. Sotomayor, K. Keahey, and I. Foster, "Combining batch execution and leasing using virtual machines," in *Proc. 17th Int. Symp. High Perform. Distrib. Comput.*, 2007, pp. 87–96.

[45] K. Chard, K. Bubendorfer, and P. Komisarczuk, "High occupancy resource allocation for grid and cloud systems, a study with DRIVE," in *Proc. 19th ACM Int. Symp. High Perform. Distrib. Comput.*, 2010, pp. 73–84.

[46] T. Hacker and K. Mahadik, "Flexible resource allocation for reliable virtual cluster computing systems," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2011, pp. 1–12.

Balaji Palanisamy received the MS and PhD degrees in computer science from the College of Computing, Georgia Tech, in 2009 and 2013, respectively. He is an assistant professor at the School of Information Science, University of Pittsburgh. His primary research interests lie in scalable and privacy-conscious resource management for large-scale distributed and mobile systems. At University of Pittsburgh, he co-directs research in the Laboratory of Research and Education on Security Assured Information Systems (LERSAIS), which is one of the first group of NSA/DHS designated Centers of Academic Excellence in Information Assurance Education and Research (CAE & CAE-R). He received the Best Paper Award at the Fifth International Conference on Cloud Computing, IEEE CLOUD 2012. He is a member of the IEEE and is currently the chair of the IEEE Communications Society Pittsburgh Chapter.

Aameek Singh received the bachelor's degree from IIT Bombay, India and the PhD degree from the Georgia Institute of Technology. He is currently a research manager in IBM Research-Almaden. His research interests include enterprise systems management, cloud computing and distributed systems. He is a member of the IEEE.

Ling Liu is a full professor in computer science at the Georgia Institute of Technology. She directs the research programs in Distributed Data Intensive Systems Lab (DiSL), examining various aspects of large scale data intensive systems. She is an internationally recognized expert in the areas of cloud computing, database systems, distributed computing, Internet systems, and service oriented computing. She has published more than 300 international journal and conference articles and received the best paper award from a number of top venues, including ICDCS 2003, WWW 2004, 2005 Pat Goldberg Memorial Best Paper Award, IEEE Cloud 2012, IEEE ICWS 2013. She also received the IEEE Computer Society Technical Achievement Award in 2012 and an Outstanding Doctoral Thesis Advisor award in 2012 from the Georgia Institute of Technology. In addition to services as the general chair and PC chairs of numerous IEEE and ACM conferences in data engineering, very large databases and distributed computing fields, she was on editorial board of more than a dozen international journals. She is currently the editor in chief of *IEEE Transactions on Service Computing*, and is on the editorial board of *ACM Transactions on Internet Technology* (TOIT), *ACM Transactions on Web* (TWEB), *Distributed and Parallel Databases* (Springer), and *Journal of Parallel and Distributed Computing* (JPDC). She is a senior member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.