AROMA: Automated Resource Allocation and Configuration of MapReduce Environment in the Cloud

Palden Lama
Department of Computer Science
University of Colorado at Colorado Springs
Colorado Springs, CO 80918
plama@uccs.edu

Xiaobo Zhou
Department of Computer Science
University of Colorado at Colorado Springs
Colorado Springs, CO 80918
xzhou@uccs.edu

ABSTRACT

Distributed data processing framework MapReduce is increasingly deployed in Clouds to leverage the pay-per-usage cloud computing model. Popular Hadoop MapReduce environment expects that end users determine the type and amount of Cloud resources for reservation as well as the configuration of Hadoop parameters. However, such resource reservation and job provisioning decisions require in-depth knowledge of system internals and laborious but often ineffective parameter tuning. We propose and develop AROMA, a system that automates the allocation of heterogeneous Cloud resources and configuration of Hadoop parameters for achieving quality of service goals while minimizing the incurred cost. It addresses the significant challenge of provisioning ad-hoc jobs that have performance deadlines in Clouds through a novel two-phase machine learning and optimization framework. Its technical core is a support vector machine based performance model that enables the integration of various aspects of resource provisioning and auto-configuration of Hadoop jobs. It adapts to ad-hoc jobs by robustly matching their resource utilization signature with previously executed jobs and making provisioning decisions accordingly. We implement AROMA as an automated job provisioning system for Hadoop MapReduce hosted in virtualized HP ProLiant blade servers. Experimental results show AROMA's effectiveness in providing performance guarantee of diverse Hadoop benchmark jobs while minimizing the cost of Cloud resource usage.

Categories and Subject Descriptors

C.4 [**Computer System Organization**]: Performance of Systems; D.2.6 [**Software**]: Programming Environments

Keywords

MapReduce, resource allocation, auto-configuration

1. INTRODUCTION

Large-scale distributed data processing in enterprises is increasingly facilitated by software frameworks such as Google MapReduce and its open-source implementation Hadoop, which parallelize and distribute jobs across large clusters [1, 5, 27]. There are growing interests in deploying such a framework in the Cloud to harness the unlimited availability of virtualized resources and pay-per-usage cost model of cloud computing. For example, Amazon's Elastic MapReduce provides data processing services by using Hadoop MapReduce framework on top of their compute cloud EC2, and their storage cloud S3.

Existing MapReduce environments for running Hadoop jobs in a cloud platform aim to remove the burden of hardware and software setup from end users. However, they expect end users to determine the number and type of resource sets to be allocated and also provide appropriate Hadoop parameters for running a job. A resource set is a set of virtualized resources rented as a single unit, e.g., virtual machines rented by Amazon web services. Here, we use the term resource set and virtual machine interchangeably. In the absence of automation tools, currently end users are forced to make job provisioning decisions manually using best practices. As a result, customers may suffer from a lack of performance guarantee and increased cost of leasing the cloud resources.

In this paper, we propose to enable automated allocation of heterogeneous cloud resource sets and configuration of Hadoop parameters for meeting job completion deadlines while minimizing the incurred cost at the same time. There are significant challenges in achieving the goal. First, the heterogeneity of available resource sets in terms of hardware configurations and different pricing schemes complicate the resource allocation decision for various jobs and different input data sizes. Second, Hadoop has over 180 configuration parameters that have varying impact on job performance. Examples include the number of reducers to be launched, the size of memory buffer to use while sorting map output, the number of concurrent connections a reducer should use when fetching its input from mappers etc. Furthermore, the optimal configuration of Hadoop parameters is tightly coupled with hardware configuration of selected resource sets as well as the type of jobs submitted.

There are techniques that each addresses resource allocation and parameter configuration in a Hadoop MapReduce framework separately [10, 12, 23]. However, in practice the effectiveness and cost efficiency of those techniques are often inter-dependent. For instance, a Hadoop parameter such as $mapred \cdot tasktracker \cdot map \cdot tasks \cdot maximum$ (the maximum number of parallel mappers) is optimal at different values when the number of virtual CPU cores of a Hadoop node is different [12]. It is important to consider a holistic approach that integrates various aspects of resource provisioning and auto-configuration for Hadoop jobs.

We propose and develop AROMA, a system that automates the allocation of heterogeneous Cloud resources and auto-configuration of Hadoop MapReduce parameters. It can be applied as a Cloud service that manages the provisioning of Hadoop jobs. One chal-

ICAC'12, September 18–20, 2012, San Jose, California, USA.

lenge in automating the resource allocation and configuration of jobs in Hadoop is a lack of performance model for such a complex distributed processing framework. A recent study focused on intensive profiling of routinely executed jobs in the Hadoop environment in order to estimate their performance for various input data sizes [23]. However, such an approach is not feasible for ad-hoc jobs submitted to the system, which have unpredictable execution characteristics. One possible workaround for ad-hoc jobs is to perform intensive job profiling by running them in a staging cluster that is similar to the target Hadoop cluster in size and configuration. However, there are overheads in terms of resources usage and the time spent in job profiling. Hence, it is not suitable for ad-hoc jobs that have deadlines.

AROMA addresses the challenges of Hadoop job provisioning with a novel two-phase machine learning and optimization framework. It exploits an important observation that jobs with similar resource consumption pattern would face similar bottlenecks and as a result they would exhibit similar performance behavior in relation to the changes in resource allocation and Hadoop configuration. In the first offline phase, it groups the information about past jobs into different clusters using the classic k-medoid clustering technique. Each cluster consists of jobs that exhibit similar CPU, network and disk utilization patterns. Then for each cluster, it trains a support vector machine (SVM) model that is able to make accurate and fast prediction of a job's performance for various combinations of resource allocation, configuration parameters and input data sizes.

In the second online phase, it obtains the resource utilization pattern of a newly submitted job by running it in a staging cluster of small VMs with default configuration parameters and using only a fraction of input data to capture its resource utilization signature. This is a lightweight process both in terms of resource usage and the time spent in capturing the signature. AROMA matches the resource utilization signature to those job clusters and identifies the performance model to use for finding the best configuration and resource allocation. Finally, AROMA applies a pattern search based optimization technique on the selected job cluster's performance model to find close to optimal resource allocation and configuration parameters that meet a Hadoop job's completion deadline at the minimal resource cost.

While it sounds intuitive, our approach is non-trivial mainly due to two reasons. First, it is difficult to define a resource utilization pattern in a shared multi-tenant cloud environment. The contention of underlying resources shared between different applications introduces non-negligible disturbance in the resource utilization pattern of the VMs. AROMA addresses this challenge by applying a Longest Common Subsequence (LCSS) based distance metric. Unlike a commonly used Euclidean Distance metric, LCSS is robust against noise and outliers in a resource utilization pattern. It is also effective in comparing the similarity between patterns that are out of phase and of different lengths.

Second, jobs with similar resource utilization signatures show similar changes in the performance in response to the same adjustment in resource allocation and configuration. However in absolute terms, the job execution times may be different. Hence, a performance model for one job can not directly predict the absolute execution time of another similar job for a given resource allocation and configuration. AROMA addresses this challenge by utilizing the concept of relative performance. For instance, assume that two jobs A and B have similar resource utilization patterns. Given the difference between the performance of the two jobs for the same resource allocation R, we can utilize the performance model of job A to predict the performance of job B for various resource allocations. We then evaluate AROMA's adaptiveness to ad-hoc jobs.

We implement AROMA on a testbed of HP ProLiant BL460C G6 blade servers hosting Hadoop MapReduce framework. The testbed uses VMware ESX 4.1 to create a pool of virtual machines which are run as Hadoop nodes. Experimental results show that AROMA makes effective and automated job provisioning decisions to achieve performance guarantee of diverse Hadoop benchmark jobs while minimizing the cost of Cloud resource usage.

To our best knowledge, AROMA is the first automated job provisioning system that meets all the challenges discussed above for MapReduce framework in Clouds. The main contributions of AROMA are as follows:

1. It enables automated job provisioning of Hadoop MapReduce framework in the Cloud so that end users can utilize cloud services without acquiring in-depth knowledge of system internals or going through laborious and time consuming manual but ineffective configuration tuning.

2. It provides a holistic job provisioning facility that integrates resource allocation and parameter configuration to meet the completion deadline guarantee and to improve the cost efficiency of MapReduce jobs.

3. It is able to make effective provisioning decisions for ad-hoc jobs while avoiding extensive and time-consuming job profiling. It does so through innovative and practical techniques that combine machine learning and optimization.

In the following, Section 2 discusses related work. Section 3 gives a case study of the cost and performance impact of resource allocation and parameter configuration in MapReduce. The AROMA architecture and design is presented in Section 4. Section 5 provides the system implementation. Section 6 presents experimental results and analysis. We conclude with future work in Section 7.

2. RELATED WORK

Recently distributed data processing framework MapReduce and its open source implementation Hadoop have gained much research [7, 11, 15, 16, 18, 21]. The pay-per-use utility model of Cloud computing introduces new opportunities as well as challenges in deploying a MapReduce framework [24]. Lee *et al.* designed an architecture to allocate resources to a data analytics cluster in the cloud, and proposed a metric of share in a heterogeneous cluster to realize a scheduling scheme that achieves high performance and fairness [15]. However, their approach allocates resources based on a simple criteria of job storage requirement without considering any performance guarantee.

There are a few studies focusing on performance estimation and guarantee of MapReduce jobs. Polo *et al.* [18] proposed an online job completion time estimator that can be used for adjusting the resource allocations of different jobs. However, their job estimator tracks the progress of the map stage alone and has no information or control over the reduce stage. An interesting approach recently designed by Verma *et al.* uses job profiles of routinely executed Hadoop jobs and a MapReduce performance model to determine the amount of resources required for job completion within a given deadline [23]. However, such an approach does not guarantee the performance of ad-hoc jobs submitted to the system. More importantly, it does not consider the impact of Hadoop configuration parameters on the effectiveness and efficiency of resource allocation. Resource allocation efficiency in datacenters is very important from the economical perspective [8, 25].

Kambatla *et al.* [12] proposed to select the optimal Hadoop configuration parameters for improving the performance of jobs using

a given set of resources. However, there is no guidance on deciding the appropriate number and type of resources to be allocated. There is a need for a holistic system that considers the inter-dependence of various job provisioning decisions in providing performance guarantee in a cost efficient manner.

There are simulation based approaches to systematically understanding the performance of MapReduce setups [9] and tuning the MapReduce configuration parameters [10]. However, designing an accurate simulator that can comprehensively capture the internal dynamics of such complex systems is potentially error-prone.

There are recent studies that address the important issue of improving MapReduce query performance [4, 6]. For example, Lee *et al.* proposed and developed YSmart, a correlation aware SQL-to-MapReduce query translator [16]. Those studies are complimentary to our research in this paper.

AROMA's technical core is based on a novel two-phase statistical machine learning and optimization framework. Statistical machine learning techniques have been used for measuring the capacity of multi-tier Internet sites [20], for online application and virtual machine auto-configuration [2, 19] and for resource allocation for performance and power control in datacenters [13, 14, 17, 22]. AROMA automates resource allocation and configuration of MapReduce environment in the Cloud in a novel and practical way. It is able to provision ad-hoc MapReduce jobs for achieving performance guarantee in a cost-efficient manner.

3. A CASE STUDY

An interesting artifact of the utility nature of the cloud paradigm is that the cost of using 1000 virtual machines (VMs) for one hour is the same as using one virtual machine (VM) of the same capacity for 1000 hours. Thus, a MapReduce job can potentially improve its performance while incurring the same currency cost by acquiring several machines and executing in parallel. However, a user submitting jobs to a Cloud service has to choose from a variety of virtual machines, which have different hardware configurations and corresponding pricing policies usually expressed as $\$perhour$. For a given MapReduce job with the certain input data set, it is challenging to determine how many and what type of virtual machines should be allocated to meet the job completion deadline while minimizing the incurred cost.

Furthermore, the configuration parameters of the Hadoop MapReduce framework for each virtual machine can significantly affect the performance of a running job. There is no single MapReduce configuration that can optimize the performance of all types of VMs and all types of jobs. To illustrate this, we conduct a case study in a testbed of virtualized Hadoop nodes based on VMware vSphere 4.1. A MapReduce benchmark job Sort, which does a partial sort of its input, is executed with 20 GB input data on a cluster of six VMs. We measure the job execution time for various combinations of resource type and configuration parameters as shown in Table 1. For the sake of clarity we use only one configuration parameter and two types of VMs in this case study. We allocate 1 CPU core with 2 GB RAM to a small VM and 2 CPU cores with 4 GB RAM to a medium VM. The usage costs of small and medium VMs are assumed to be $0.85\$perhour$ and $1.275\$perhour$, respectively. We assume that the maximum number of parallel mappers and reducers are fixed to 1 slot per node for small VMs and 2 slots per node for medium VMs.

The results in Table 1 illustrate the impact of resource allocation and configuration on both performance and cost of running a Hadoop job. We observe that the performance impact of MapReduce parameters significantly varies with resource allocation. For example when six small VMs are allocated, the optimal configuration is given by case 3 with $mapred \cdot reduce \cdot tasks$ equal to 70. However for six medium VM allocation, the optimal configuration is Case 5 with $mapred \cdot reduce \cdot tasks$ equal to 35. In this particular study, we have a counter intuitive observation that it may be cheaper to run a Hadoop job using six medium VMs instead of six small VMs. However, the situation can be further complicated by the fact that cost and performance impact can also vary with the different number of VMs and different input size of jobs running in a Hadoop cluster. Hence, it is important to consider all these factors for making a cost efficient and effective job provisioning decision.

Table 1: Cost and performance impact of resource allocation and parameter configuration.

	Hadoop MapReduce Configuration		
small VMs	mapred.reduce.tasks	Exec.time	Cost (cents)
Case 1	7	17.2min	146.2
Case 2	35	14min	119.0
Case 3	70	13min	110.5
medium VMs	mapred.reduce.tasks	Exec. time	Cost (cents)
Case 4	7	11.5min	146.6
Case 5	35	7.36min	93.8
Case 6	70	8min	102.0

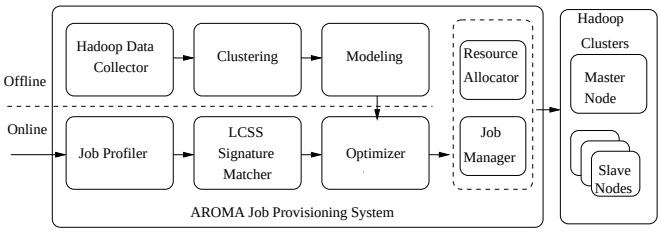
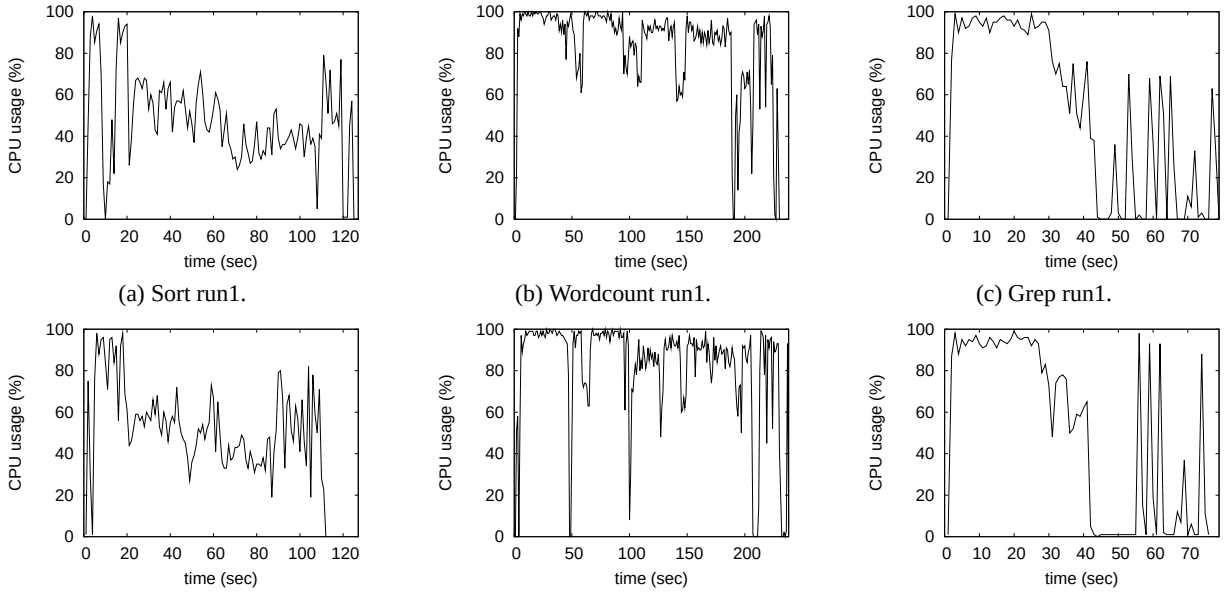


Figure 1: The system architecture of AROMA.

4. AROMA ARCHITECTURE AND DESIGN

AROMA is an automated job provisioning system that integrates resource allocation and parameter configuration to meet job deadline guarantee and improve the cost efficiency of Hadoop MapReduce in Clouds. Figure 1 shows the architecture of AROMA. End users submit jobs as input to AROMA through a command line interface. AROMA first calculates the number and type of VMs to be allocated, and assigns appropriate MapReduce configuration parameters to meet its completion deadline at minimal cost. Then, its resource allocator powers on a pool of dormant VMs pre-installed with Hadoop software. Finally, the job manager submits the job to the Hadoop master node along with its configuration parameters. Here, the key challenges lie in making fast and effective decisions at job submission time, allocating the right amount of resources and finely tuning the MapReduce configuration settings for an ad-hoc job with unpredictable input data size and completion deadline.

The core components of AROMA work in two phases. First the data collector, clustering and performance modeling components process Hadoop log files and resource utilization data to learn the performance models for various types of Hadoop jobs in the offline phase. Next in the online phase, the job profiler, resource signature matcher and optimizer select the appropriate performance model for an incoming job based on its resource utilization signa-



(a) Sort run1. (b) Wordcount run1. (c) Grep run1.

(d) Sort run2. (e) Wordcount run2. (f) Grep run2.

Figure 2: CPU utilization of Sort, Wordcount and Grep jobs during different runs.

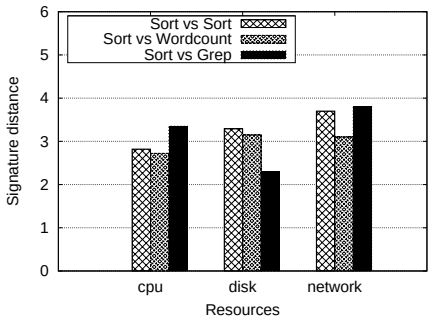


Figure 3: Euclidean Signature Distances for CPU, Network and Disk resources for (i) Sort vs Sort;(ii) Sort vs Wordcount;(iii) Sort vs Grep;

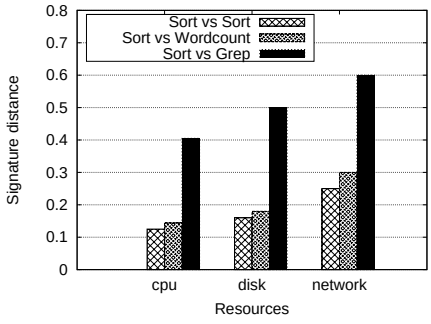


Figure 4: LCSS Signature Distances for CPU, Network and Disk resources for (i) Sort vs Sort;(ii) Sort vs Wordcount;(iii) Sort vs Grep;

ture and calculate the optimal resource allocation and configuration that meet the job completion deadline.

4.1 Machine Learning Performance Modeling

The first challenge for making optimal job provisioning decisions is the lack of the performance models for different Hadoop jobs. AROMA learns the performance models of various types of jobs in the offline phase.

4.1.1 Data collection and clustering

The data collector component extracts the execution time, input data size, resource allocation and MapReduce configuration parameters of various past jobs from the Hadoop JobTracker log files. For each job, it fetches the corresponding series of resource utilization data of the slave VM nodes. We use *dstat* tool at each slave VM to measure CPU usage (in terms of User, System, Idle and Wait percentages), disk usage (number of blocks in and out) and network usage (bytes/sec into and out of network card) every second. All the usage measurements are normalized using their respective maximum values. In our testbed, the JobTracker and slave VM logs are saved in a repository when a Hadoop cluster is decommissioned after job completion.

A core principle that enables AROMA's adaptiveness to ad-hoc jobs is based on the observation that jobs with similar resource utilization pattern exhibit similar performance behavior. As a result, AROMA needs to learn only a single performance model for a group of similar jobs. The resource utilization signature of each job is a combination of time series data of average CPU usage, average memory usage and average disk usage rate. However, it is difficult to compare the resource signatures of two jobs accurately in a multi-tenant cloud environment. It is due to the fact that the contention of shared resources between multiple applications introduces noise and outliers in the resource utilization patterns.

We examine the variability of resource utilization patterns of various Hadoop jobs running on a small cluster of 2 small VMs and using 1 GB input data. The Hadoop configuration parameters are set to the default value. To simulate a multi-tenant cloud environ-

Table 2: Feature Selection for SORT performance model.

Features	t-statistic	p-value
io.sort.factor	-3.6306	0.0211
mapred.job.shuffle.merge.percent	0.9613	0.3423
io.sort.spill.percent	2.13	0.2014
mapred.job.shuffle.input.buffer.percent	-1.5352	0.1328
io.sort.record.percent	-1.0588	0.0191
mapred.job.reduce.input.buffer.percent	1.4043	0.1682
mapred.reduce.parallel.copies	4.3869	0.0342
io.sort.mb	-3.3465	0.0312
mapred.reduce.tasks	-6.4583	0.0101
input size	10.8	0.0011
VM type	8.5	0.0045
number of VMs	-4.2	0.0021

Table 3: Feature Selection for GREP performance model.

Features	t-statistic	p-value
io.sort.factor	-1.1324	0.2682
mapred.job.shuffle.merge.percent	-1.4953	0.1474
io.sort.spill.percent	-1.4527	0.1587
mapred.job.shuffle.input.buffer.percent	-0.8069	0.4273
io.sort.record.percent	3.8211	0.0007
mapred.job.reduce.input.buffer.percent	3.8016	0.0008
mapred.reduce.parallel.copies	-0.1715	0.8652
io.sort.mb	9.487	0.0112
mapred.reduce.tasks	-0.7814	0.4419
input size	9.3	0.0033
VM type	7.5	0.0056
number of VMs	-4.5	0.0011

ment, we host another group of VMs in the same physical server cluster of our testbed and execute Hadoop jobs in them. Figures 2 (a) and 2(d) compare the CPU utilization patterns of a VM node when a Sort job is executed twice. Similarly, the CPU utilization patterns of Wordcount and Grep jobs are also compared. We observed variations in the utilization patterns of the same job when it is executed at different times.

It is challenging to accurately identify the degree of similarity between different jobs due to the disturbance in the resource utilization pattern of VMs. We demonstrate this challenge by measuring the difference between the utilization patterns of various job combinations using a straightforward Euclidean distance metric. Figure 3 shows the Euclidean distance between the resource utilization signatures of different job combinations. We observe that the Euclidean distance based resource utilization signature comparison is misleading. For instance according to this metric, the signature distance of CPU and network utilization between two executions of the same Sort job is found to be greater than that of Sort vs Wordcount job executions.

AROMA addresses this challenge by applying a Longest Common Subsequence (LCSS) based distance metric. Unlike a commonly used Euclidean Distance metric, LCSS is robust against noise and outliers in a resource utilization pattern. It is also effective in comparing the similarity between patterns that are out of phase and of different lengths. Figure (4) shows LCSS distance between the resource utilization signatures of different job combinations. The distances between the CPU, disk and network utilization patterns of two Sort jobs is the smallest and the distances between the utilization patterns of Sort and Grep jobs are the largest.

AROMA applies the classic k-medoid based clustering technique along with LCSS distance metric to group jobs with similar utilization patterns of CPU, network and disk resources. According to the resource utilization patterns of the three jobs, AROMA groups Sort and Wordcount jobs in the same cluster and the Grep job belongs to a different cluster. Resource signatures implicitly capture the potential impact of input data content on a job's behavior.

4.1.2 Performance modeling

AROMA applies a powerful supervised machine learning technique to learn the performance model for each cluster of jobs. It constructs a support vector machine (SVM) regression model to estimate the completion time of jobs belonging to a cluster for different input data sizes, resource allocations and configuration parameters. SVM methodology is known to be robust for estimating real-valued functions (regression problem) from noisy and sparse training data having many attributes. This property of SVM makes it a suitable technique for performance modeling of complex Hadoop jobs in the Cloud environment.

It is important to create a model that utilizes only the statistically significant features and avoids "overfitting" the data. For each job cluster, we apply a stepwise regression technique to determine which set of features are the best predictors for the job performance. It is a systematic method for adding and removing terms from a model based on their statistical significance in a regression. The method begins with an initial model and then compares the explanatory power of incrementally larger and smaller models. At each step, the p value of a t-statistic is computed to test models with and without a potential term. If a term is not currently in the model, the null hypothesis is that the term would have a zero coefficient if added to the model. If there is sufficient evidence to reject the null hypothesis, the term is added to the model. Conversely, if a term is currently in the model, the null hypothesis is that the term has a zero coefficient. If there is insufficient evidence to reject the null hypothesis, the term is removed from the model.

We conduct stepwise regression on the data sets collected from our testbed of virtualized Hadoop nodes. For data collection, we measured the execution times of various Hadoop jobs with different input data sizes in the range 5 GB to 50 GB, using different Hadoop configuration parameters and running on different cluster sizes of Hadoop nodes comprising of small and medium VMs. Tables 2 and 3 show the results of stepwise regression for two different Hadoop jobs, Sort and Grep respectively. Note that the features whose *p-values* are smaller than or equal to the significance level of 0.05 are selected for performance modeling using SVM regression.

The idea of SVM regression is based on the computation of a linear regression function in a high dimensional feature space where the input data are mapped via a nonlinear function. The linear model in an M dimensional feature space $f(x, \omega)$ is given by

$$f(x, \omega) = \sum_{m=1}^{M} \omega_m g_m(x) + b \quad (1)$$

where $g_m(x)$ denotes a set of nonlinear transformations and b is the bias term. The input data x for AROMA's performance model consists of the features selected by the stepwise regression method.

AROMA applies $\varepsilon - SV$ regression technique [3] that aims to find a function $f(x, \omega)$ that has at most ε deviation from the actual output values y for all the training data points. In this case, the output values of the performance model are the job execution times. At the same time, it also tries to reduce model complexity by min-

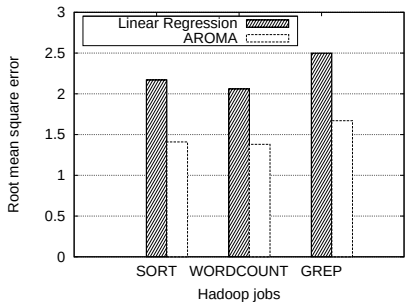


Figure 5: Prediction accuracy of AROMA performance model.

imizing the term $||\omega||^2$. We apply the LIBSVM [3] library to find suitable kernel functions and train the SVM regression model.

SVM regression has a good generalization and prediction power due to the fact that unlike other learning techniques, it attempts to minimize a generalized error bound so as to achieve generalized performance. This error bound is the combination of the training error and a regularization term that controls the complexity of the model. One approach for assessing how accurately a predictive model will perform in practice is to perform *cross-validation*. We apply leave-one-out cross-validation (LOOCV), which involves using a single observation from the original data sample as the validation data, and the remaining observations as the training data. This is repeated such that each observation in the sample is used once as the validation data. Figure 5 shows the results of cross validation performed for three Hadoop jobs in terms of the root mean square error in prediction. AROMA's SVM regression based performance model is able to achieve significantly better prediction accuracy than a linear regression model.

4.2 Online Job Profiling and Signature Match

A MapReduce environment needs to run ad-hoc jobs with unpredictable resource utilization patterns that make it difficult to apply performance models. AROMA addresses this challenge by first running a new job in a small set of Hadoop nodes and using a small chunk of input data. It collects the resource utilization signature of the job using *dstat* tool. Then the signature is compared with the resource utilization signatures of various job clusters' centroid. We observe that signature matching with the cluster centroid rather than each data point is sufficient since the data points within the same cluster are similar. As a result, this process can be executed with less overhead.

However, a straightforward comparison of resource utilization signatures may cause misleading results due to the presence of noise and outliers. AROMA applies the LCSS distance metric to match the signatures corresponding to each resource type for a sequence of measurement samples. It is able to make accurate comparison between the resource utilization patterns of two jobs.

4.3 Cost Efficient Performance Guarantee

It is important but challenging to make effective job provisioning decisions to meet performance guarantee in terms of job completion deadlines and to minimize the cost of resource allocation. AROMA's job provisioning decision is based on a non-linear constrained optimization problem formulated for a given job with input size s and completion deadline D as follows:

$$\text{Minimize} \quad \sum_{i=1}^{N} n_i * rate_i * t_s \tag{2}$$

Subject to Constraints:

$$t_s \leq D \tag{3}$$

$$\forall j \in [1, C], \quad lb_j \leq conf_j \leq ub_j \tag{4}$$

Eq. (2) gives the optimization objective to minimize the total cost of allocating VMs to the Hadoop cluster. Here, n_i is the number of VMs of type i (small,medium) that is charged a cost of $rate_i$ at $\$perhour$ and t_s is the execution time of the given job with input data size s. Eq. (3) defines a constraint that the job execution time must be less than its completion deadline D. Eq. (4) defines the lower bound lb and upper bound ub values of each MapReduce configuration parameter under consideration.

In this optimization problem, the relationship between the decision variables ($n_i, conf_j$) and the dependent variable t_s is given by the SVM regression model corresponding to a given job type. We solve the optimization problem by applying a pattern search algorithm, the generating set search. It is a direct search method that can optimize complex objective functions that are not differentiable.

It computes a sequence of points that approach an optimal point. At each step, the algorithm searches a set of points, called a mesh, around the current point computed at the previous step of the algorithm. The mesh is formed by adding the current point to a scalar multiple of a set of vectors called a pattern. If the pattern search algorithm finds a point in the mesh that improves the objective function at the current point, the new point becomes the current point at the next step of the algorithm.

4.4 Resource Allocator and Job Manager

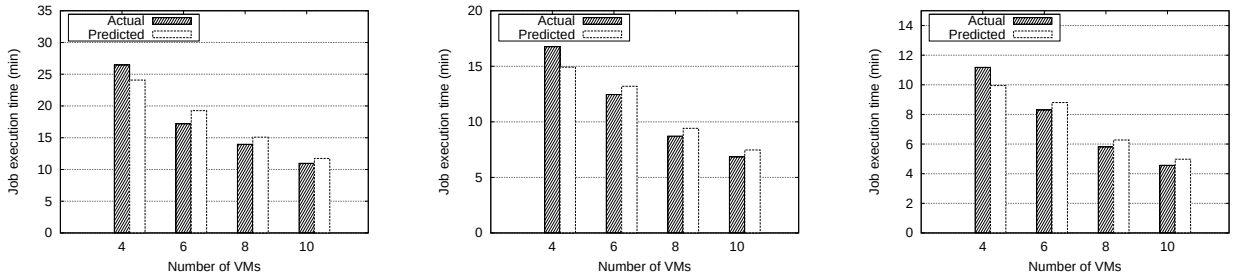
AROMA's optimizer sends its solutions to the resource allocator and the job manager components. The resource allocator is responsible for powering on the appropriate number and types of VMs with pre-installed Hadoop software. It uses VMware vSphere API 4.1 to issue commands for VM management. Note that a VM in its powered off state poses negligible infrastructure cost. Once the VMs are started up, the job manager submits the given job to the Hadoop master node along with the optimized MapReduce configuration parameters. It does so by issuing commands to the hadoop master node using a java *ssh* library.

5. SYSTEM IMPLEMENTATION

5.1 The Testbed

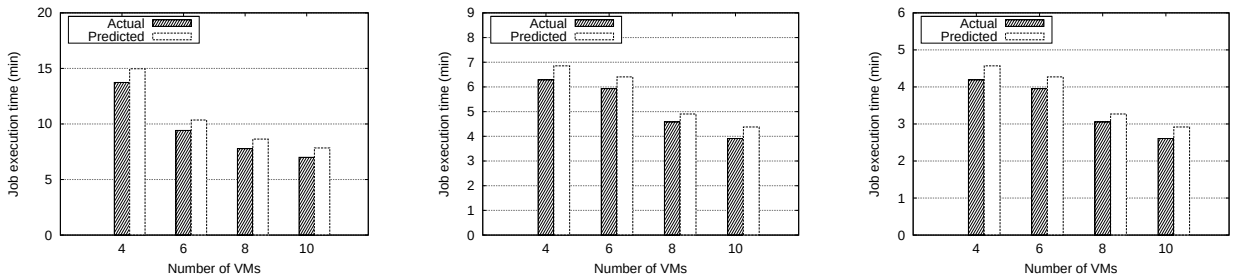
We implement AROMA on a testbed consisting of seven HP ProLiant BL460C G6 blade server modules and a HP EVA storage area network with 10 Gbps Ethernet and 8 Gbps Fibre/iSCSI dual channels. Each blade server is equipped with Intel Xeon E5530 2.4 GHz quad-core processor and 32 GB PC3 memory. Virtualization of the cluster is enabled by VMware ESX 4.1. We create a pool of VMs with different hardware configurations from the virtualized blade server cluster and run them as Hadoop nodes. There are small VMs with 1 vCPU, 2 GB RAM and 50 GB hard disk space. Medium VMs have 2 vCPUs, 4 GB RAM and 80 GB hard disk space. Each VM uses Ubuntu Linux version 10.04 and Hadoop 0.20.2.

We designate one blade server to host the master VM node and use rest of the servers to host the slave VM nodes. The single master node runs the JobTracker and the NameNode, while each slave node runs both the TaskTracker and the DataNode. A small slave VM is configured with one Map slot, one Reduce slot and 200 MB memory per task. Whereas, a medium type slave VM is configured with two Map slots, two Reduce slots and 300 MB memory per task. The data block size is set to 64 MB. The AROMA job provisioning system can be run either on a separate VM or a standalone machine as it manages the Hadoop nodes remotely.



(a) Sort. (b) Wordcount. (c) Grep.

Figure 6: Actual and predicted running times of MapReduce jobs for various VM resource allocations (small VMs).



(a) Sort. (b) Wordcount. (c) Grep.

Figure 7: Actual and predicted running times of MapReduce jobs for various VM resource allocations (medium VMs).

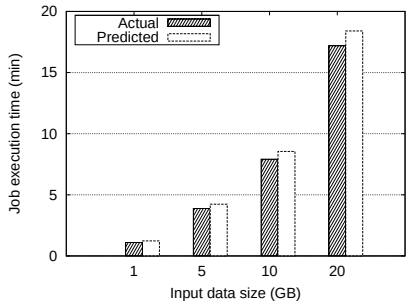


Figure 8: Actual and predicted running times of Sort for various input data sizes.

As related studies [12, 23], we use a number of benchmark jobs that come with the Hadoop distribution for performance evaluation, i.e., Sort, Grep, and Wordcount. We use the RandomWriter and RandomTextWriter tools in the Hadoop package to generate data of various sizes for the Sort, WordCount and Grep programs.

6. PERFORMANCE EVALUATION

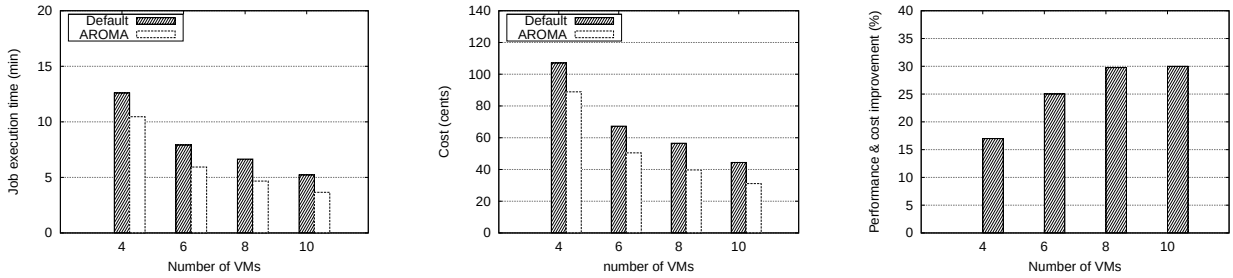
We evaluate AROMA's capability for predicting the performance of Hadoop jobs, auto-configuration of Hadoop MapReduce parameters, joint resource allocation with configuration, and adaptiveness to ad-hoc Hadoop jobs for improving job performance and reducing the incurred cost. For our experiments, the costs of using a small VM and a medium VM are assumed to be $\$0.85$ per hour and $\$1.275$ per hour respectively.

6.1 AROMA Performance Model Accuracy

First, we evaluate AROMA's ability to predict the completion times of Hadoop jobs when they are executed with different number of VM nodes of different types and different input data sizes. We use the default Hadoop configuration for this experiment. Figures 6 (a), (b) and (c) show the actual and predicted running times of Sort, Wordcount and Grep jobs respectively as the number of Hadoop slave nodes are varied. In this case, we use small VMs as Hadoop nodes and fix the input data to be 20 GB. Figures 7 (a), (b) and (c) show the results of a similar study using medium VMs. We observe that AROMA's SVM regression based performance model is able to predict the speedup achieved for each job as we increase the number of VMs. The relative error between the actual and predicted job completion time is less than 12% on all the cases. We observed similar prediction accuracy when the input data size is varied while keeping the number of VMs fixed as shown in Figure 8. Here, we use six small VMs to run a Sort job.

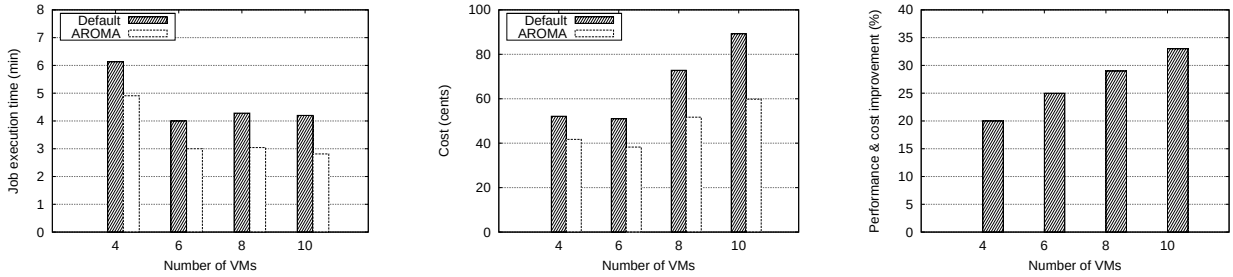
6.2 Auto-Configuration

We evaluate AROMA's capability to automatically configure the Hadoop environment parameters according to variations in the resource allocation. We execute the Hadoop Sort benchmark. The jobs are executed for an input data of 10 GB. Figures 9 (a), (b) and (c) compare the impact of using the default Hadoop configuration with AROMA's auto-tuned configuration on the job execution time and the incurred cost. AROMA's auto-configuration significantly outperforms the default configuration for various allocations of small VMs. The improvement in job performance and cost due to AROMA increases from 17% to 30% when using more number of VMs. It is due to the fact that there are more opportunities for configuration tuning in case of using more VMs. Figures 10 (a), (b) and (c) show the job execution results when medium VMs are used. We observe that using more than six medium VMs shows little improvement in performance and hence increases the cost. It is due to the fact that for a moderately small input data of 10 GB, the overhead of managing a large number of Map/Reduce slots in the cluster can be significant.



(a) performance. (b) cost. (c) improvement.

Figure 9: Impact of AROMA's auto-configuration on job performance and cost for various VM resource allocations (small VMs).



(a) performance. (b) cost. (c) improvement.

Figure 10: Impact of AROMA's auto-configuration on job performance and cost for various VM resource allocations (medium VMs).

Note that in Figure 9(c) and Figure 10(c), the improvement percentage in performance and in cost are the same since the cost of using a VM depends on how long a job runs.

Table 4 compares the Hadoop configurable parameter values due to AROMA's auto-configuration with the default Hadoop values. AROMA uses this configuration for running Sort benchmark with six small VMs and input data of 20 GB. We can observe that there are many differences between two settings. In particular, parameters $io \cdot sort \cdot factor$, $mapred \cdot reduce \cdot tasks$, and $mapred \cdot reduce \cdot parallel \cdot copies$ have significantly different values. Hadoop Sort benchmark sets the default value of $mapred \cdot reduce \cdot tasks$ to 0.9 times the total number of reduce slots in the cluster.

Please note that for the Sort job even if an end user is able to come up with the same parameter configuration as that of AROMA based on best practices, such a parameter configuration is not effective when the Hadoop environment and the submitted jobs change dynamically. For instance, Table 5 shows the parameter configuration differences for running the Sort job with six medium VM and input data of 20 GB. It shows there are a few important differences between two parameter configurations by AROMA.

Next, we evaluate the auto-configuration capability of AROMA when the input data size is varied. Figures 11(a),(b) and (c) show the performance improvement due to AROMA for running the Sort benchmark by using six small VMs.

6.3 Efficient Resource Allocation and Configuration

A key feature of AROMA is its holistic job provisioning approach with optimization for joint resource allocation and auto-configuration. We demonstrate the merit of this feature by comparing the cost efficiency in achieving the job completion deadline by AROMA with and without the integration of resource allocation with auto-configuration of Hadoop parameters.

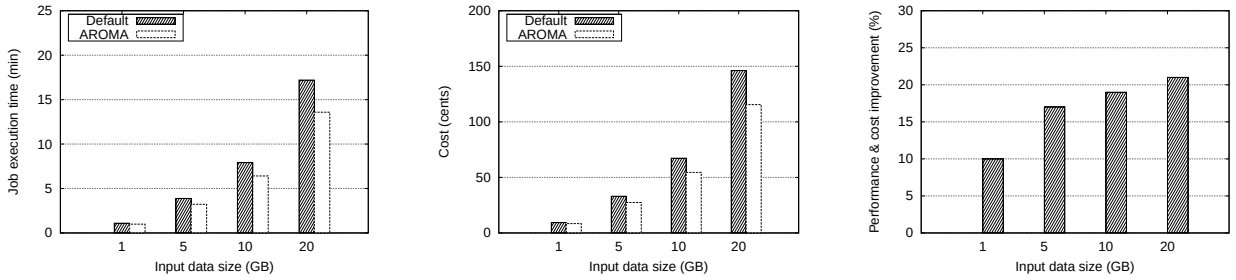
Tables 6 and 7 show the execution time and cost of complet-

Table 4: Hadoop configuration parameters for Sort benchmark (six small VMs and 20 GB input data).

Hadoop parameters	AROMA	Default
io.sort.factor	300	10
mapred.job.shuffle.merge.percent	0.66	0.66
io.sort.spill.percent	0.8	0.8
mapred.job.shuffle.input.buffer.percent	0.7	0.7
io.sort.record.percent	0.15	0.05
mapred.job.reduce.input.buffer.percent	0	0
mapred.reduce.parallel.copies	12	5
io.sort.mb	130	100
mapred.reduce.tasks	120	5

Table 5: Hadoop configuration parameters for Sort benchmark (six medium VMs and 20 GB input data).

Hadoop parameters	AROMA	Default
io.sort.factor	100	10
mapred.job.shuffle.merge.percent	0.66	0.66
io.sort.spill.percent	0.8	0.8
mapred.job.shuffle.input.buffer.percent	0.7	0.7
io.sort.record.percent	0.3	0.05
mapred.job.reduce.input.buffer.percent	0	0
mapred.reduce.parallel.copies	8	5
io.sort.mb	160	100
mapred.reduce.tasks	120	10



(a) performance. (b) cost. (c) improvement.

Figure 11: Impact of AROMA auto-configuration on job performance and cost for various input data sizes.

Table 6: Cost of meeting job execution deadline (360 sec).

	AROMA w/o auto-configuration		
Data input	VMs (number and type)	Execution time (sec)	Cost (cents)
5 GB	3 small	300	21
10 GB	6 medium	240	51
20 GB	12 medium	358	152

Table 7: Cost of meeting job execution deadline (360 sec).

	AROMA w auto-configuration		
Data input	VMs (number and type)	Execution time (sec)	Cost (cents)
5 GB	3 small	210	14.8
10 GB	5 medium	205	36
20 GB	10 medium	357	126

ing a Hadoop Sort job for various input data sizes by AROMA without and with the integration of resource allocation with auto-configuration respectively. Results show that both approaches can complete the job within the completion deadline 360 seconds. However, AROMA with auto-configuration is much more cost efficient for meeting the job completion deadline for all different input data sizes. For example, for the Sort benchmark job with data size 10 GB, AROMA without auto-configuration costs 51 cents to finish the job within the given deadline using 6 medium VMs. On the other hand, AROMA with auto-configuration uses only 5 medium VMs to meet the deadline while incurring the total cost of 36 cents.

The main reason for the cost efficiency of AROMA is two-fold. First, AROMA is able to fine tune the Hadoop configuration parameters corresponding to various resource allocations. As a result, a job running with AROMA's optimized configuration can finish faster than a job using the default Hadoop configuration for the same set of resource allocations. We have observed similar performance improvement results for other job completion deadlines but omitted here due to the space limit.

Second, it performs a cost-aware optimization for the allocation of heterogeneous resources. As shown in Table 7, when the input data is 5 GB, AROMA allocates three small VMs that complete the job in 210 seconds. We note that using two small VMs could still be able to meet the job completion deadline that is 360 seconds. This kind of counter intuitive behavior of AROMA is in fact advantageous. This is due to the fact that running three small VMs for 210 seconds is cheaper than running two VMs of the same capacity for a much longer time, while still meeting the job completion deadline. The cost saving is 12% in this case. On average, AROMA shows cost efficiency of 25%.

6.4 Adaptiveness to Ad-hoc Jobs

We evaluate AROMA's ability to predict the performance of ad-hoc jobs. Assuming that an ad-hoc Wordcount job with 10 GB input data is submitted to the system, AROMA first captures the resource utilization signature of the job by executing it in a staging cluster of two small VMs using 1 GB input data and default Hadoop configuration. As discussed in Section 4.1.1, the resource utilization signature of Wordcount job is found to be similar to that of a Sort job. We assume that the performance model of Sort job is available due to offline data clustering and modeling. AROMA calculates the difference between the measured execution time of Wordcount in the staging cluster and the execution time predicted by the SVM performance model of the Sort job. It applies this difference value to scale the predictions made by the performance model for different resource allocations and configurations.

The accuracy of AROMA's resource signature matching and the performance of ad-hoc jobs depend on the diversity of existing job clusters. Job clustering is initially performed offline based on past data. It can be repeated at regular time intervals to accommodate diverse new jobs.

Figure 12 demonstrates the accuracy of AROMA's performance prediction of Wordcount job for various VM resource allocations. In this case, we use small VMs with default Hadoop configuration for performance evaluation. AROMA is able to achieve a very accurate prediction of job execution time with a relative error of less than 12.5%. Next, we compare the actual and predicted execution times of Wordcount job running on 4 small VMs with 20 various Hadoop configurations as shown in Figure 13.

7. CONCLUSIONS

The software framework MapReduce and its open-source implementation Hadoop are increasingly deployed in the Cloud to harness the unlimited availability of virtualized resources and pay-per-usage cost model of cloud computing. However, currently end users have to make job provisioning decisions manually at the Hadoop environment using best practices. They suffer from a lack of performance guarantee and increased cost of leasing the Cloud resources.

AROMA is proposed and developed to enable automated allocation of heterogeneous cloud resource sets and configuration of Hadoop parameters for meeting job completion deadlines while minimizing the incurred cost. It addresses the significant challenges of automated and efficient Hadoop job provisioning with a novel two-phase machine learning and optimization framework.

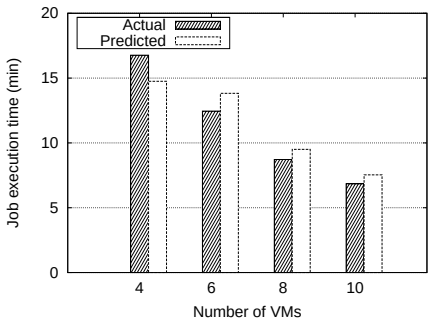


Figure 12: Prediction accuracy for an ad-hoc job for various VM resource allocations.

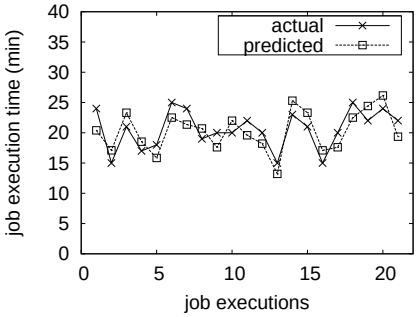


Figure 13: Prediction accuracy for an ad-hoc job for 20 Hadoop configurations.

It is able to make optimal job provisioning decisions with respect to resource allocation efficiency in the face of unpredictable input data sizes and performance expectations. The significance also lies in the fact that it enables automated job provisioning of Hadoop MapReduce framework in the Cloud so that end users can utilize cloud services without acquiring in-depth knowledge about the system internals or going through laborious and time consuming manual but ineffective configuration tuning.

AROMA is developed upon the popular Hadoop MapReduce environment. It is ready to be tailored for other MapReduce running environments. In the future work, we will extend AROMA for running multi-stage job workflows.

Acknowledgement

This research was supported in part by NSF CAREER award CNS-0844983. The authors thank the NISSC for providing HP blade server equipments for conducting the experiments.

8. REFERENCES

[1] A. Abouzid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin. Hadoopdb: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. In *Proc. of the VLDB*, 2009.

[2] X. Bu, J. Rao, and C.-Z. Xu. A reinforcement learning approach to online web system auto-configuration. In *Proc. IEEE Int'l Conference on Distributed Computing Systems (ICDCS)*, 2009.

[3] C. Chang and C. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2, 2011.

[4] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. MapReduce online. In *Proc. USENIX NSDI*, 2010.

[5] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1), 2008.

[6] J. Dittrich, J.-A. Quiané-Ruiz, A. Jindal, Y. Kargin, V. Setty, and J. Schad. Hadoop++: making a yellow elephant run like a cheetah (without it even noticing). *Proc. of the VLDB*, 3:515–529, 2010.

[7] Y. Geng, S. Chen, Y. Wu, R. Wu, G. Yang, and W. Zheng. Location-aware MapReduce in virtual cloud. In *Proc. IEEE Int'l Conference on Parallel Processing (ICPP)*, 2011.

[8] F. Goiri, K. Le, J. Guitart, J. Torres, and R. Bianchini. Intelligent placement of datacenters for internet services. In *Proc. IEEE Int'l Conference on Distributed Computing Systems (ICDCS)*, 2011.

[9] W. Guanying, A. Butt, P. Pandey, and K. Gupta. A simulation approach to evaluating design decisions in MapReduce setups. In *Proc. IEEE Int'l Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2009.

[10] H. Herodotou and S. Babu. Profiling, what-if analysis, and cost-based optimization of MapReduce programs. In *Proc. of the VLDB*, 2011.

[11] b. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proc. USENIX NSDI*, 2011.

[12] K. Kambatla, A. Pathak, and H. Pucha. Towards optimizing hadoop provisioning in the cloud. In *HotCloud Workshop in conjunction with USENIX Annual Technical Conference*, 2009.

[13] P. Lama and X. Zhou. Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee. In *Proc. IEEE/ACM Int'l Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2010.

[14] P. Lama and X. Zhou. PERFUME: Power and performance guarantee with fuzzy mimo control in virtualized servers. In *Proc. IEEE Int'l Workshop on Quality of Service (IWQoS)*, 2010.

[15] G. Lee, B. Chun, and H. K. Randy. Heterogeneity-aware resource allocation and scheduling in the cloud. In *HotCloud Workshop in conjunction with USENIX Annual Technical Conference*, 2011.

[16] R. Lee, T. Luo, F. Wang, Y. Huai, Y. He, and X. Zhang. Ysmart: Yet another SQL-to-MapReduce translator. In *Proc. IEEE Int'l Conference on Distributed Computing Systems (ICDCS)*, 2011.

[17] X. Meng, C. Isci, J. Kephart, L. Zhang, and E. Bouillet. Efficient resource provisioning in compute clouds via vm multiplexing. In *Proc. Int'l Conference on Autonomic Computing (ICAC)*, 2010.

[18] J. Polo, D. Carrera, Y. Becerra, J. Torres, E. Ayguade, M. Steinder, and I. Whalley. Performance-driven task co-scheduling for MapReduce environments. In *Proc. of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2010.

[19] J. Rao, X. Bu, C. Xu, L. Wang, and G. Yin. Vconf: A reinforcement learning approach to virtual machines auto-conguration. In *Proc. IEEE Int'l Conference on Autonomic Computing Systems (ICAC)*, 2009.

[20] J. Rao and C. Xu. CoSL: a coordinated statistical learning approach to measuring the capacity of multi-tier Websites. In *Proc. IEEE Int'l Parallel and Distributed Processing Symposium (IPDPS)*, 2008.

[21] L. Shi, X. Li, and K. L. Tan. S3: An efficient shared scan scheduler on MapReduce framework. In *Proc. IEEE Int'l Conference on Parallel Processing (ICPP)*, 2011.

[22] R. Singh, U. Sharma, E. Cecchet, and P. Shenoy. Autonomic mix-aware provisioning for non-stationary data center workloads. In *Proc. IEEE Int'l Conference on Autonomic Computing (ICAC)*, 2010.

[23] A. Verma, L. Cherkasova, and R. Campbell. ARIA: automatic resource inference and allocation for MapReduce environments. In *Proc. IEEE/ACM Int'l Conference on Autonomic Computing (ICAC)*, 2011.

[24] D. Warneke and O. Kao. Exploiting dynamic resource allocation for efficient parallel data processing in the cloud. *IEEE Trans. on Parallel and Distributed Systems*, 22(6), 2011.

[25] P. Xiong, Z. Wang, S. Malkowski, D. Jayasinghe, Q. Wang, and C. Pu. Economical and robust provisioning of n-tier cloud workloads: A multi-level control approach. In *Proc. IEEE Int'l Conference on Distributed Computing Systems (ICDCS)*, 2011.

[26] J. Xu and J. Fortes. A multi-objective approach to virtual machine management in datacenters. In *Proc. of IEEE/ACM Int'l Conference on Autonomic computing (ICAC)*, 2011.

[27] M. Zaharia, A. Konwinshi, A. D. Josepj, R. Katz, and I. Stoica. Improving MapReduce performance in heterogeneous environments. In *Proc. the USENIX OSDI*, 2008.