Comprehensive Elastic Resource Management to Ensure Predictable Performance for Scientific Applications on Public IaaS Clouds

In Kee Kim, Jacob Steele, Yanjun Qi, and Marty Humphrey
Department of Computer Science
University of Virginia
Email: {ik2sb, jss2zb}@virginia.edu, {yq2h, humphrey}@cs.virginia.edu

***Abstract*—Scientists have become increasingly reliant on large-scale compute resources on public IaaS clouds to efficiently process their applications. Unfortunately, the reactive nature of auto-scaling techniques made available by the public cloud provider can cause insufficient response time and poor job deadline satisfaction rates. To solve these problems, we designed an end-to-end elastic resource management system for scientific applications on public IaaS clouds. This system employs the following strategies: 1) an accurate and dynamic job execution time predictor, 2) a resource evaluation scheme that balances cost and performance, and 3) an "availability-aware" job scheduling algorithm. This comprehensive system is deployed on Amazon Web Services and is compared with other state-of-the-art resource management schemes. Experimental results show that our system achieves a 9%–32% improvement with respect to the deadline satisfaction rate over other schemes. We achieve this deadline satisfaction rate improvement while still providing improved cost-efficiency over other state-of-the-art approaches.**

***Index Terms*—IaaS Clouds, Job Scheduling and Resource Management, Job Execution Time Prediction;**

I. Introduction

Elasticity and pay-as-you-go pricing models have attracted both industry and scientific communities to cloud computing [4]. Elasticity, one of the key features of cloud computing, enables resources to be scaled up and down based on users' needs. However, determining the exact amount of required resources can be very difficult due to dynamic changes in the workload [17] and variable job execution time. Public cloud service providers like AWS (Amazon Web Services) [1] and Microsoft Windows Azure [2] offer an automatic resource scaling service called auto-scaling, which is a rule/scheduling-based resource manager [1, 3, 17]. This feature allocates resources of VMs (Virtual Machine) based on a scheduled timeline or a set of user defined rules. These rules trigger scaling based on general performance indicators like CPU load, memory usage and I/O bandwidth usage. Auto-scaling is a convenient way to manage resources automatically, but lacks the awareness of deadline considerations. Thus, auto-scaling performs poorly when exposed to unexpected workload demands or highly variable job execution times. For example, the system cannot immediately scale proper resources to handle the demand or job deadline because of delays caused by the monitoring of the VMs and new VM start-up time [16].

The fundamental reason that current auto-scaling approaches have limitations, comes from their reactive nature. The response time and cost efficiency of auto-scaling could be enhanced if there are predictive aspects in the auto-scaling mechanisms. These predictive aspects can help to determine proper resources to be prepared in advance or to provide better information for job scheduling. In order to provide better response time and cost efficiency, predictive techniques [6, 9, 19] for managing cloud resources have been applied, focusing on estimation of future resource demands. However, these approaches perform poorly when applications have high variance of their execution time. Considering this gap, we describe to predict the application execution time in order to achieve a proactive resource management system on public IaaS clouds that provides faster response time and improved cost efficiency.

In this paper, we introduce LCA, a resource management system, which provides predictable performance for job deadline satisfaction and increased cost efficiency for scientific applications on public IaaS clouds. (**LCA** is named after the three starting letters of our approaches.) LCA takes into account scientific applications working with jobs that have user-defined deadlines. Our approach is focused on 1) *estimating job execution time*, 2) *resource evaluation*, and 3) *job scheduling*. For job execution time prediction, we employ LLR (Local Linear Regression) [7] to estimate job execution time, taking into account the size of data required for the computation and type of VM instances since these are key factors that affect job execution time. For resource evaluation, we suggest a cost-performance optimized evaluation scheme, which determines job deadline satisfaction by comparing prediction results with the job deadline, and evaluates cost efficiency of VM instances by computing the cost-performance ratio. Moreover, we describe an availability-aware job scheduling algorithm that assigns jobs to available resources first, instead of starting up a new machine. This improves cost-efficiency and performance for LCA by minimizing the spawning of new instances.

We design and implement the resource manager on top of AWS and use various types of VM instances provided by AWS. We measure the performance of the LLR predictor by comparing with well-known existing approaches such as kNN [15, 20], linear regression [12], and $mean$-based prediction [20]. We measure the job deadline satisfaction rate, total running cost, and VM utilization by comparing with the state-of-art baselines under various workload patterns. The results show that our LLR predictor has 11.15%–17.78% better prediction accuracy than other well-known approaches. LCA has a 9%–32% better job deadline satisfaction rate than the two baselines. LCA shows that it is more tolerable system with respect to workload changes as compared to the two baselines. LCA also shows an average 18% VM utilization improvement and leverages 16%–41% less VMs. LCA achieves a 18%–35% better cost efficiency than SCS [15] with LLR predictor.

The contributions of this paper are:

- We introduce a novel elastic resource management system for scientific applications based on job execution time prediction, cost-performance ratio-based resource evaluation and resource availability-aware scheduling.
- We introduce a simple and effective job execution time prediction method called Local Linear Regression (LLR), which shows better prediction accuracy than other approaches.
- The cost-performance ratio-based resource evaluation method improves better job deadline satisfaction rate. The availability-aware job scheduling algorithm minimizes the time and cost overhead generated by starting new VMs, and it helps users reduce financial cost and execution time for processing their jobs.

The rest of this paper is organized as follows. Section II contains the related work. Section III describes the system design and three main approaches of LCA. Section IV contains the evaluation and discussion, and Section V provides the conclusion.

II. Related Work

A. Resource Management in Cloud Computing

Mao and Humphrey [17] introduced a linear integer programming-based optimization approach to solve auto-scaling problems on IaaS by focusing on job deadline and workflows of an application. Their approach scales the VM resources by computing the optimal amount of resources. This is done using linear programming with variable parameters on cloud applications (e.g. job deadline, budget constraints, application workflows, etc.). Another optimization-based approach was introduced by Chaisiri et al [8]. They focused on resource demands from users and cost of VMs, and used a stochastic programming method to find the optimal amount of resources. They introduced a model using the combination of on-demand and reserved resources trying to solve the under- and over-provisioning problem.

Prediction-based approaches [6, 9, 19] have been conducted to manage cloud computing resources. These works employ statistical learning mechanisms to predict future workload change or system status of the VMs. Their goal is to manage computing resources based on the prediction results. Gong et al [6] used two predictors to forecast short term (unexpected) and long term (seasonal) workload change. Roy et al [19] used ARMA model to predict the expected workload and tried to calculate minimum cost for resource provisioning based on prediction results and response time analysis. Jiang et al [9] used a similar time-series predictor with [19] but they focused on the change of resource cluster's state rather than predicting workload change, and scaled up and down their cluster based on the prediction result.

The difference between optimization-based approaches and this paper is that the optimization approaches employ an assumption that the execution time of the job is already known or easily predictable. The optimization function uses the assumption as one of its input parameters. However, predicting job execution time for real applications is not a trivial research topic and accurate prediction of job execution time provides more sufficient information to improve the job deadline satisfaction rate and cost efficiency of the resource managers. Existing predictive approaches mainly focus on forecasting future workload change or system status of cloud resources, but we focus on the job execution time estimation in order to handle the scientific applications which have high variance of the job execution time. Moreover, this research is also different from [14]. They offer a workflow ensemble approach for resource provisioning to satisfy cost and deadline constraints, but they focused on a single VM type resource model for scientific workflows, because they assume an application has a single cost effective VM type. We cannot use this assumption because of the high job execution time variance of our application, which implies the application's optimal VM type can change due to the size of the data to be processed as well as the current system status.

B. Job Execution Time Prediction

There have been many attempts to predict job execution time since it can improve the performance of a job scheduler or resource manager for large scale distributed systems. In this section, we review two major approaches to predict job execution time: instance-based learning [10, 13, 18, 20] and linear approach [5, 12].

The instance-based learning is widely used in large scale distributed systems such as high-performance parallel computing [18, 20] and Grid [10, 13]. This approach stores N recent jobs to build the experience base and searches for the most similar past jobs from the experience base by using a distance function. Then, this method predicts the job execution time for the new job by using kNN with similar samples.

The linear approach is a simple and intuitive method to estimate the execution time of jobs by finding a linear relation between past executions and the new job or by forecasting future resource status. Lee et al [12] introduced an application-centric estimation approach that uses a linear regression based prediction using execution time history of the target application and the variables affecting the application's execution. [5] introduced a resource-specific approach that focused on estimating the load status of the CPU in order to use it to predict the job execution time. They used the $AR(16)$ method

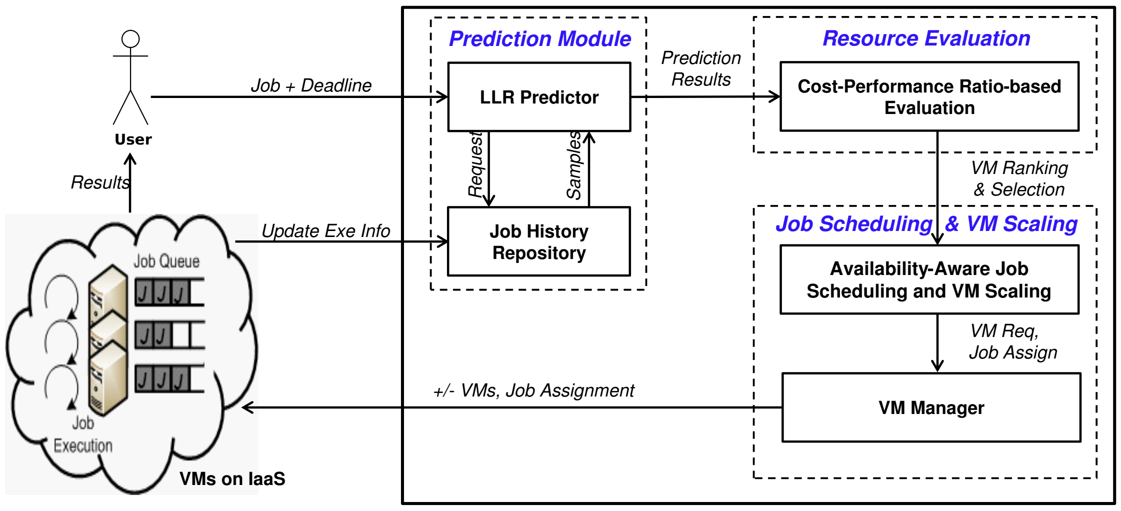


Fig. 1. Overview of System Architecture of LCA

(Auto Regression with 16 order of linear model) to estimate CPU load status.

These two approaches are relevant to this work but we have two different aspects to take into account. First, we are targeting to build a resource manager on public IaaS clouds, which offer various types of VMs that has a different degree of the performance. Moreover, the execution time of the application varies based on the size of data to be processed. We therefore need to consider two factors (type of resources, size of data), and we aim to build more find-grained and suitable prediction model of the job execution time for our cloud application.

III. System Design and Approaches

A. Overview of LCA System Design

LCA is composed of three major components and the overview of system architecture of LCA is shown in Fig. 1.

*1) **Prediction Module**:* This module has two sub-components: the LLR predictor and the job history repository. The LLR predictor estimates execution time for an input job on various types of VM in public clouds. This predictor will be discussed in Section III-B. The job execution history repository stores predicted and real job execution results and, it provides samples to estimate the job execution time to help the LLR predictor reduce prediction error and update the prediction model.

*2) **Resource Evaluation Module**:* This plays two major roles: 1) determining that the job can meet the deadline by comparing the predicted results and the user defined deadline and 2) evaluating the cost-efficiency of VMs for the job execution by using the cost-performance optimized resource evaluation scheme. This evaluation scheme will be discussed in Section III-C.

*3) **Job Scheduling and VM Scaling Module**:* This module consists of two sub-components: the availability-aware job scheduler and the VM manager. The availability-aware job scheduler is in charge of job scheduling taking into account the resource availability, the cost efficiency, and the job deadline satisfaction. The VM manager is responsible for VM scaling. It spawns up more VMs when the job scheduler cannot find a proper resource to assign an input job. The VM manager also terminates a VM if the VM's running time is going to meet hourly bound for pricing and the VM has satisfied a condition for its termination. This job scheduling and VM scaling mechanism will be explained in Section III-D

TABLE I
Specification of General Purpose Microsoft Windows Instances on Amazon EC2, Northern Virginia, USA

VM Type	ECU[1]	# of CPUs	Memory	Hourly Price
$m1.small$	1	1	1.7 GB	\$0.075
$m1.medium$	2	1	3.7 GB	\$0.149
$m1.large$	4	2	7.5 GB	\$0.299
$m1.xlarge$	8	4	15 GB	\$0.598

B. Job Execution Time Prediction

Current auto-scaling approaches cannot handle those applications having high variance of their execution time due to lack of a capability to estimate the job execution time. As one of the major innovations in this work, we describe to estimate the job execution time of the scientific applications, in order to achieve a better proactive resource management system on IaaS clouds. Using a data-driven approach, we describe to estimate the job execution time for an application based on the historical measurements of the execution time from that same application. Using a scientific application which calculates the watershed delineation [11] as the main case study, we first deploy this scientific application with 26 sample executions on four different types of VMs on AWS. The execution results measured with these 26 sample runs on the AWS described in Table I.

TABLE II
PEARSON CORRELATION COEFFICIENT OF TWO OPERATIONS WITH SIZE OF DATA AND TYPE OF VIRTUAL MACHINE

Operation Type	Data Size	VM Type
Non-Data Intensive Operation	0.0973	0.7089
Data Intensive Operation	0.6129	0.3223

We have observed that normally scientific applications include several pipeline procedures where each procedure does either data intensive operation or non-data intensive operation. The data intensive operation focuses on processing data and its execution time seems to be affected by the size of data to be processed. Non-data intensive operations do not process data, and their execution time seems not to be affected by the size of data. Instead, the time of their execution appears to be correlated with the performance of the VMs. However, overall, if a user requests a job for running a scientific application, every pipeline procedure in the application will be executed and the overall job execution time equals to the sum of the execution times for every involved pipeline procedure.

Based on the above observations, we conduct a correlation study to check the following four relationships for running a scientific application on IaaS clouds.

- For the data intensive operations, what is the relationship between the job execution time and the size of the data to be processed?
- For the data intensive operations, what is the relationship between the job execution time and the VM type?
- For non-data intensive operations, what is the relationship between the job execution time and the size of the data to be processed?
- For non-data intensive operations, what is the relationship between the job execution time and the VM type?

Using equation-(1), we investigate the relationship through calculating the absolute value of Pearson's correlation coefficient [22] between the two variables involved in each of the four cases above. In equation-(1), X means the independent feature variables (e.g. size of data, VM type) for the execution time, and Y indicates the execution time of each operation. For Pearson correlation coefficient, 1 means a very strong linear relation between the independent variable X and the execution time Y, and 0 means vice versa.

$$r = \left| \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}} \right| \tag{1}$$

The calculated correlation coefficients for all four cases are shown in Table II. We can see that:

- For non-data intensive operations, there exists strong linear relation between the VM type and the execution time, whereas, a negligible linear relation exists between the size of data to be processed and the measured job execution time.
- Differently, data intensive operations have moderate linear correlation among the size of data vs. job execution time, and weak linear relation with regard to the VM types.

[1]Single ECU (EC2 Compute Unit) provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor [1].

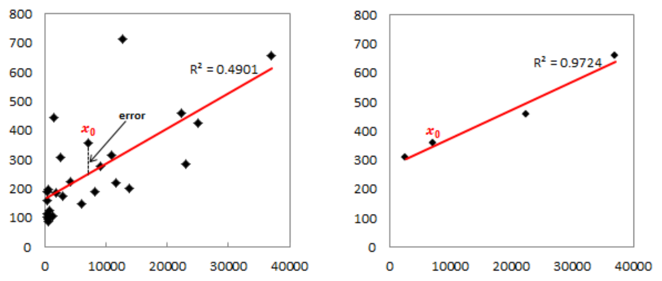
The above correlation analysis clearly indicates that a simple linear regression function mapping linearly from the feature variables X to the execution time variable Y cannot produce reliable predictions, since the linear model produces results with high bias. As discussed in the Related Work, another way to predict the job execution time was the instance-based approach, e.g. k-nearest neighbors (kNN) method. However this method suffers from the issue of producing the predictions with high variance, especially when the sample size is small, e.g. for our case with the number of samples N=26. Considering both these concerns, we describe a novel prediction model named Local Linear Regression (LLR) which aims to learn a function $f()$ transforming (mapping) the feature variables X to Y accurately, and allowing $f()$ to be very flexible, yet smooth at the same time.

The LLR achieves the flexibility by fitting a different and simple linear model separately at every point x_0. This is done utilizing the following three-stage process.

1) For each testing point x_0, apply the kNN method to find a set $V(x_0)$ including its top-k neighboring samples in the training (historical) data set.
2) Using only those observations in the set V, we fit a simple linear regression model, i.e. learning the parameters $\alpha(x_0)$ and $\beta(x_0)$ through minimizing the objective function in equation-(2).
$$\min_{\alpha(x_0),\beta(x_0)} \sum_{x_i \in V(x_0)} [y_i - \alpha(x_0) - \beta(x_0)x_i]^2 \tag{2}$$
3) Then the predicted job execution time $f(x_0)$ for the current testing job x_0 will be derived from the equation-(3).
$$f(x_0) = \alpha(x_0) - \beta(x_0)x_0 \tag{3}$$

The basic idea of local linear regression vs. global linear regression is explained by Fig. 2 using the sample points measured on a $m1.large$ instance on AWS. The line in Fig. 2(a) indicates a regression line derived by using all training samples except the current point x_0. The line in Fig. 2(b) is learned by using only the kNN (using one more application parameter) of point x_0 (here, k=3). Clearly we can see that, using a smaller number of neighboring samples in Fig.2-(b) achieves a more effective prediction of the job execution time for target point x_0, than using all the training samples in Fig. 2(a). Furthermore, we can also evaluate the goodness of predictions through a measure called model fitness (R^2) [15] from the regression line fitting. Fig. 2(b) achieves a better value of R^2 than the fitness value from Fig.2-(a).

In summary, we conclude that local regression with selected neighbors can produce better estimation for the job execution time than linear regression. In our algorithm, selecting the proper distance function and the right value for k when using



(a) Linear regression (using all samples)
(b) Local Linear Regression (using selected neighbors)

Fig. 2. Linear Regression vs. Local Linear Regression. Left graph shows that fitting a global linear regression model to map from the input feature describing the size of the data, to the output Y variable representing the job execution time. X-axis is job execution time (unit second), Y-axis is size of data to be processed (unit bytes). Right graph shows that fitting a local linear regression model using only the neighboring training points for the current target test point: x_0

TABLE III
PREDICTION RESULTS OF JOB EXECUTION TIME FOR THREE DIFFERENT JOBS. DEADLINE FOR ALL THREE JOBS IS 60 MINUTES.

VM Info		**Prediction Result**		
VM Type	**Min. Price**	**Job A**	**Job B**	**Job C**
$m1.small$	\$0.00125	80 min.	80 min.	80 min.
$m1.medium$	\$0.00248	50 min.	55 min.	40 min.
$m1.large$	\$0.00498	40 min.	25 min.	20 min.
$m1.xlarge$	\$0.00997	25 min.	11 min.	10 min.

kNN method to choose the neighbor set V are two critical factors to achieve reliable prediction results. We use k=4 in kNN and the Euclidean distance function. The features being utilized for representing each job operation x_i includes the two most important variables, the size of the data to be processed and the type of VMs, plus one more variable representing the application specific parameter. In the experiment section, we compare the LLR model with both the *global linear regression* baseline and a baseline using kNN for predicting the job execution time.

C. Resource Evaluation

LCA aims to not only satisfy a deadline for each job, but to also reduce the total execution cost for all jobs. To this end, LCA evaluates VMs to select a cost efficient VM that meets the deadline. This is because the prediction model for job execution time produces several options of VMs as discussed in the previous section. As shown in Table I in Section III-B, IaaS clouds providers determine the pricing based on the specification of the hardware. For example, AWS's $m1.large$ instance has double the hardware of $m1.medium$ instance. And $m1.large$ instance costs more than two times of $m1.medium$ instance. This pricing can seem reasonable as $m1.larger$ instance can outperform the $m1.medium$ by two. However, hardware specification itself does not guarantee proportional results so characteristics of applications must be considered.

For resource evaluation of VMs' cost efficiency, we consider the following three cases. Suppose that a user submits three different jobs, each with a 1 hour deadline, for a scientific application. The prediction results of job execution time for the three jobs on four different types of VMs are shown in Table III. Three types of VMs are able to meet the deadline (1 hour), but $m1.small$ cannot meet the user-defined deadline. Thus, $m1.small$ is excluded from the candidate list for the three executions. Note that we use minute-based cost (hourly cost/60) to explain these cases. In the case of job A, $m1.large$ is only 1.25x faster than $m1.medium$ but costs twice that of $m1.medium$. $m1.xlarge$ is 2x faster than $m1.medium$ but it is 4 times more expensive than $m1.medium$. In this case, using the cheapest instance ($m1.medium$) among the three candidates ($m1.medium$, $m1.large$, and $m1.xlarge$) is the best in terms of cost efficiency, because $m1.medium$ has the smallest (best) value for the performance-cost ratio (minute cost * execution time). For processing job B, $m1.xlarge$ is 2.28x faster than $m1.large$ and 5.14x faster than $m1.medium$ but it is only 2 times more expensive than $m1.large$ and 4 times more expensive than $m1.medium$. In this case, using $m1.xlarge$ for processing job B is the most cost efficient. For the last case job C, $m1.medium$, $m1.large$, and $m1.xlarge$ have the same cost-performance ratio, so we can use any instance among three candidates for processing job C.

By considering the above three cases, we suggest a cost-performance optimized resource evaluation. After predicting the execution time of an incoming job, the prediction mechanism in LCA provides several execution options and each option can be expressed as $option_n = \{T_n, C_n, V_n\}$. T_n is the predicted job execution time (minutes) based on VM types, C_n is the execution cost per minutes, and V_n is the VM type. This evaluation scheme calculates a cost-performance ratio $(T_n \times C_n)$ for each option and sorts options in ascending order based on the cost-performance ratio. If every option has the same cost-performance ratio, this scheme sorts options in ascending order based on hourly price. This scheme is expressed in equation-(4).

$$\begin{aligned} &Cost\text{-}Perf.\ Ratio - basedResource\ Evaluation \\ &= sort\{t_1 \times c_1, t_2 \times c_2, ..., t_n \times c_n\} \end{aligned} \quad (4)$$

D. Availability-Aware Job Scheduling and VM Scaling

Availability-aware job scheduling and VM scaling consists of two steps: 1) job scheduling and 2) VM scaling. Job scheduling is used once the resource evaluation is completed, LCA assigns an incoming job to a proper VM. For this task, LCA uses the results of the cost-performance optimized resource evaluation (Section III-C) and builds a VM list from the current running VM pool. For instance, suppose LCA has 8 running VMs and VM index 1, 2 are $m1.small$, VM index 3, 4 are $m1.medium$, VM index 5, 6 are $m1.large$, and VM index 7, 8 are $m1.xlarge$. In the example in Table III, LCA builds a candidate VM list (VM List = [InstanceType: $m1.medium$, VMs: VM_3, VM_4, InstanceType: $m1.large$,

TABLE IV
VM Candidate List for Job A Based on Resource Evaluation

Sorted Subset Order	VM Type	Running VMs
1	$m1.medium$	VM_3, VM_4
2	$m1.large$	VM_5, VM_6
3	$m1.xlarge$	VM_7, VM_8

VMs: VM_5, VM_6, InstanceType: $m1.xlarge$, VMs: VM_7, VM_8]) for Job A. The candidate VM list for job A is shown in Table IV.

Based on the subset order, LCA calculates the expected time of job completion on each VM in the first subset ($\{InstanceType : m1.medium, VMs : VM_3, VM_4\}$) including queue wait time and remaining execution time for the current job on VMs. Then, LCA computes the job completion time that can meet the deadline for the job. Fig. 3 and equation-(5) explain how to calculate the job completion time on each VM. Fig. 3 shows a case for assigning a job to a selected VM. Equation-(5) is a formula to calculate the expected job completion. The remaining execution time (T_{remain}) for the currently running job on the VM is calculated by "the predicted execution time of that job (T_{pred}) – current running time for that job on its VM." ϵ is a buffer to handle the prediction error. If a proper VM which meets the deadline for the job, is found in the first subset, LCA assigns the job to that VM's queue. In this step, if multiple VMs can meet the deadline, LCA selects a VM that offers the earliest job completion time for the job. If LCA cannot find any VM to meet the deadline, it begins searching for a VM in the next order subset ($\{InstanceType : m1.large, VMs : VM_5, VM_6\}$). LCA repeats this procedure for each subset.

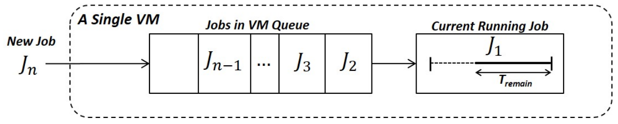
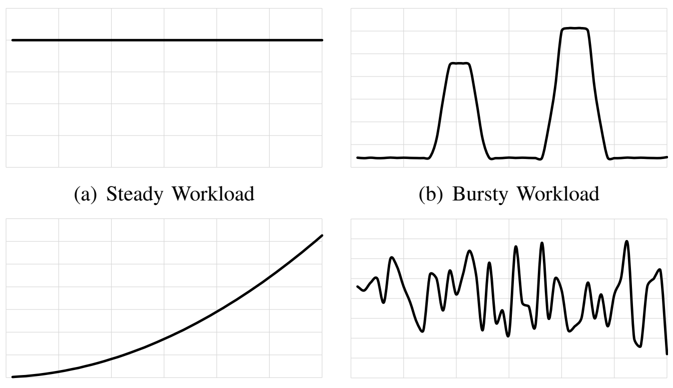


Fig. 3. Job assignment to a VM.

$$Job\ Completion\ Time = \sum_{i=2}^{N} T_{pred,J_i} + T_{remain,J_1} + \epsilon \quad (5)$$

VM scaling mechanism is used when LCA cannot find a proper VM for the assigned job from the above scheduling procedure. In this case, LCA adds a resource for this job (scaling up). LCA updates the predicted execution time for each VM type by adding lag time to start a new VM [16]. Then, LCA re-evaluates cost efficiency for VM types by using the updated job execution time and resource evaluation mechanism. LCA creates a VM, which has the highest cost efficiency and assigns the job to the VM. If the updated job execution time for all VM types cannot meet the deadline, LCA rejects this job. LCA also has a VM scaling down mechanism that considers hourly basis price policy for public IaaS clouds. When a VM's running time is close to its hourly bound, LCA checks the status of VM and its working queue. LCA terminates the VM if the VM's status is idle and the VM's queue is empty, which means the VM has no more jobs to process.



(a) Steady Workload
(b) Bursty Workload
(c) Incremental Workload
(d) Random Workload

Fig. 4. Workload patterns used in the experiments.

TABLE V
Workload Information.

Workload Pattern	Average Job Arrival Time	Standard Deviation of Job Arrival Time
Steady	121.52s	16.73
Bursty	153.84s	140.92
Incremental	72.00s	64.77
Random	119.31s	64.99

The nature of an availability-aware scheduling and VM scaling mechanism is that it maximizes the use of current VMs for the job assignment rather than creating a new VM instance. This mechanism has the following advantages:

- It maximizes the utilization of VM instances.
- It minimizes the performance and cost overhead due to start up lag time.

IV. Evaluation

We conduct a performance evaluation of LCA by comparing with two state-of-the-art baselines: 1) SCS (Scaling Consolidation Scheduling) [15] and 2) SCS + LLR. The reason we make a new baseline (SCS + LLR) is that we want to evaluate how the LLR predictor improves the performance of the resource managers as well as establish a cost comparison between the baselines. We also want to evaluate the performance benefit of LCA's resource evaluation scheme, and availability-aware job scheduling and VM scaling as compared to those of SCS.

We implement and deploy LCA and other resource managers on AWS and use a watershed delineation system [11] as our scientific application. We compare the performance of the approaches under the same workloads. Four workloads patterns (steady, bursty, incremental, and random workloads), shown in Fig. 4, are used in the experiments. Table V and VI show detailed information on generating the four workload patterns and the job requests used in the experiments. We generate identical job requests based on these workloads, and send the jobs to all resource managers on AWS and measure 1)

TABLE VI
JOB REQUEST INFORMATION

Number of Jobs	Job Deadline		Job Duration	
	Average	Std. Dev.	Average	Std. Dev
100	1800s	581	905.40s	748

TABLE VII
PERFORMANCE EVALUATION OF FOUR PREDICTORS (LLR, LR, kNN, $Mean$)

	LLR	Lin. Regression	kNN	mean
Avg. Accuracy	78.77%	67.72%	65.38%	60.99%
MAPE	0.2773	0.3901	0.5012	0.8254

performance of predictor in three approaches, 2) job deadline satisfaction rate, 3) total running cost, and 4) VM utilization. For this experiment, all three approaches use four different types of general purpose VMs running Microsoft Windows Server 2012 offered by AWS. The information on VMs used by resource managers are shown in Table I in Section III-B.

A. Predictor Performance

We first measure the performance of LLR predictor by comparing with linear regression, kNN, and a $mean$-based predictor. Linear regression and kNN are used in SCS. And we use a $mean$-based predictor as a baseline because it is a simple and convenient way to predict the job execution time [21]. Evaluation of the predictor performance is conducted by measuring the two following criteria: *prediction accuracy* and *MAPE* (Mean Absolute Percentage Error) [7]. A higher value of prediction accuracy is better, and a lower value of MAPE means better performance.

$$Pred.\ Accuracy = \begin{cases} \dfrac{T_{actual}}{T_{predicted}}, & T_{predicted} \geq T_{actual} \\ \dfrac{T_{predicted}}{T_{actual}}, & T_{predicted} < T_{actual} \end{cases} \tag{6}$$

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{T_{actual,i} - T_{predicted,i}}{T_{actual,i}}\right| \tag{7}$$

Table 7 shows the performance evaluation results for the predictors. LLR outperforms the others in all three criteria. The average prediction accuracy of LLR is 11.15%, 13.39%, and 17.78% better than linear regression, kNN, and $mean$-based predictor respectively. Correspondingly, MAPE of LLR is 28.91%, 44.67%, and 66.40% less than linear regression, kNN, and $mean$-based prediction.

B. Job Deadline Satisfaction Rate

In this section, we measure the job deadline satisfaction rate for LCA, SCS + LLR, and SCS. The job deadline satisfaction rate is calculated by the following equation.

$$Deadline\ Satisfact.\ Rate = \frac{N_{DeadlineSatisfiedJobs}}{N_{AllJobs}} \tag{8}$$

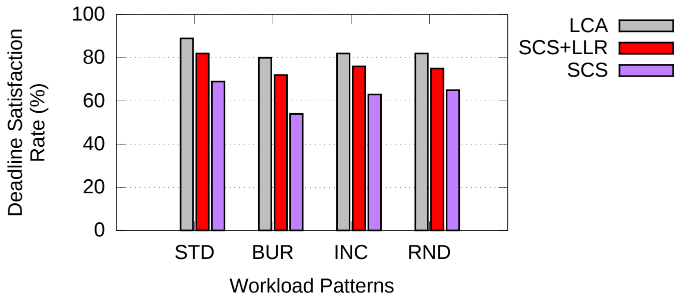


Fig. 5. Job deadline satisfaction rate of LCA, SCS+LLR, and SCS in the four workload patterns. STD: steady, BUR: bursty, INC: incremental, and RND: random workload.

TABLE VIII
PERFORMANCE DEGRADATION BY WORKLOAD CHANGE. (FROM STEADY TO BURST)

	LCA	SCS+LLR	SCS
Performance Degradation	-10.11%	-12.20%	-21.74%

The experimental result is shown in Fig. 5. LCA outperforms other baselines for all workloads and shows an average of 83% job deadline satisfaction rate. LCA has an average improvement of 9.21% and 31.75% over than SCS+LLR and SCS. An interesting result is that the resource managers with LLR (LCA and SCS+LLR) outperform SCS that uses linear regression and kNN for the job execution time estimation. All resource managers rely on a prediction result of the job execution time to manage VMs and assign the incoming job to a specific VM. As we discussed in the previous evaluation, the LLR-based predictor shows better performance than others such as linear regression and kNN. This means that the LLR predictor provides a more accurate estimation of the job execution time. More accurate prediction results enable the resource managers to assign a job to a suitable VM where it is more likely to satisfy the job deadline. This performance gain comes from differences in job scheduling and VM scaling mechanisms considering both resource managers (LCA and SCS+LLR) have the same predictor for the job execution time estimation. This performance benefit implies that the availability-aware job scheduling and VM scaling effectively manages incoming jobs/VM resources to satisfy the user-defined job deadline.

All resource managers show the best performance in the steady workload and the worst performance in the bursty workload. To figure out the tolerance of the resource managers for the various workloads, we measure the performance degradation by changing the workload from steady to bursty. The result is shown in Table VIII. LCA has 10.11% performance degradation due to the workload change from steady to bursty. This is 1.20x and 2.15x less performance degradation than SCS-LLR and SCS This result shows that LCA has higher tolerance to workload change as compared to SCS+LLR, SCS and Optimal Scaling. As LCA outperforms SCS+LLR in both the job deadline satisfaction rate and the performance degradation, this result implies that LCA's two other strategies

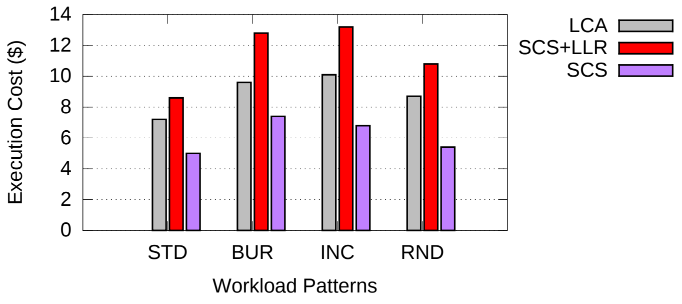


Fig. 6. Overall running cost of VM instances used by LCA, SCS+LLR, and SCS in the four workload patterns (steady, bursty, incremental, and random workload).

(resource evaluation and scheduling) help the system improve both performance metrics.

C. Total Running Cost

We measure the overall running cost of VMs used to process all job requests in steady and bursty workloads. To check the total running cost, we use billing and cost management service in AWS as well as an embedded cost calculation function for all resource managers. Note that we exclude the cost for VMs where the resource managers run. We only measure the cost of VMs for job processing. The purpose of this experiment is to evaluate the cost efficiency of the approaches. Cost efficiency is an important factor for end users in selecting cloud resource managers for their application deployments because many companies and institutions have budget caps on the use of cloud services and applications. Thus, a resource management system with higher cost efficiency is more attractive to users if the approaches have an acceptable performance for job deadline satisfaction rate. Fig. 6 shows the overall running cost for the resource managers. LCA spends \$7.2 for the steady workload and \$9.6 for the bursty workload. It also spends \$10.1 and \$8.7 for the incremental and random workloads. These results mean that LCA has an average of 30.9% less cost efficiency than SCS. However, this is an unfair comparison because SCS does not have comparable performance in the job deadline satisfaction rate. So, we compare LCA with SCS+LLR and the results show that LCA has an average of 27.30% higher cost efficiency than SCS+LLR in all workloads. This implies that LCA could also provide improved cost-efficiency over SCS if it has an acceptable performance for the job deadline satisfaction rate.

D. Number of Created VM and VM Utilization

In order to evaluate the performance of availability-aware job scheduling and VM scaling scheme, we measure the number of VMs created by the resource managers and VM utilization. Table 9 shows the number and type of spawned VMs for LCA, SCS+LLR and SCS to process the job requests. LCA creates a total of 15, 39, 35, and 17 VMs from the steady workload to random workload respectively. SCS+LLR uses a total of 35, 52, 57, and 37 VMs in four workloads. SCS spawns a total of 21, 49, 32, and 25 VMs. LCA uses 16.53%–41.43% less VMs than the baselines for all workloads. These results imply that LCA has less delay time (startup lag) for starting new VMs. This is because the availability-aware job scheduling and VM scaling mechanism helps LCA utilize the current running VMs rather than creating new VMs. Also, we measure the startup in this experiment and we realized that AWS has an average of 507.55 seconds of startup delay.

TABLE IX
AVERAGE VM UTILIZATION FOR FOUR WORKLOAD PATTERNS.

	LCA	**SCS+LLR**	**SCS**
Idle	18.60%	31.09%	26.20%
Job Execution	69.17%	55.21%	62.37%
Startup	12.23%	13.70%	11.43%

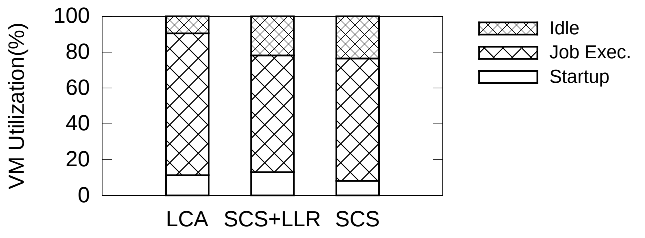
$$VM\ Utilization = \frac{\sum_{i=1}^{n} T_{job\_execution,VM_i}}{\sum_{i=1}^{n} T_{total\_running,VM_i}} \qquad (9)$$

We also measure VM utilization, which is calculated by equation-(9). A VM is created when a resource manager determines that it needs more VMs to process the job requests. Once a VM is created, it runs in one of three states. These states are 1) start up, 2) job execution, and 3) idle. Job execution time ($T_{job\_execution}$) in the VM is calculated by the total running time of VM ($T_{total\_running}$) − (startup time ($T_{startup}$) + idle time (T_{idle})). The VM will be terminated if it satisfies its termination condition. The resource managers have similar termination conditions. The results of this experiment are shown in Fig. 7 and Table IX. In all workloads, LCA outperforms the others. On average (Table IX), LCA has 10.90%–25.29% more utilization than the baselines and 29.01%–40.17% less idle time. These results show that LCA's approaches to schedule jobs are effective in increasing the VM utilization as well as improving LCA's cost-efficiency as LCA has less VM idle time.

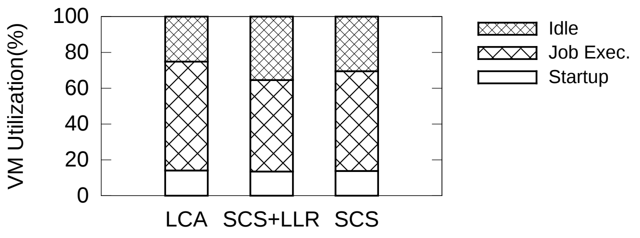
V. Conclusion

A predictive approach for resource management on IaaS clouds is an important issue because it enables the resource m nagers to handle dynamic change in the workloads and high variance in the job execution time. In this paper we describe LCA, a system that employs 1) a LLR-based prediction model to estimate the job execution time, 2) a cost-performance optimized scheme-based resource evaluation, and 3) an availability-aware job scheduling and VM scaling mechanism. LCA effectively predicts the job execution time taking into account the type of VMs and the size of data required for the computation. The cost-performance optimized resource evaluation scheme determines the job deadline satisfaction by comparing prediction results with the job deadline, and evaluates the cost efficiency of VMs by computing the cost-performance ratio. The availability-aware job scheduling and VM scaling mechanism assigns jobs to available resources first, and improves the cost-efficiency and job deadline satisfaction rate for LCA by minimizing the spawning of new VMs.

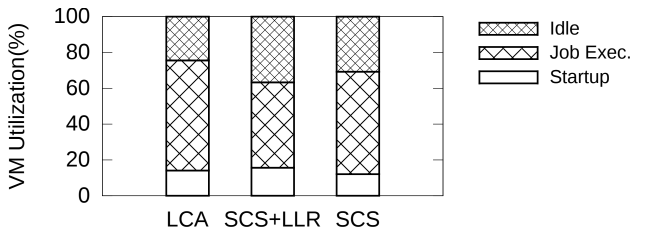
We designed and implemented LCA on top of AWS and use various types of VMs provided by AWS. We measured the performance of LCA in five categories by comparing with two



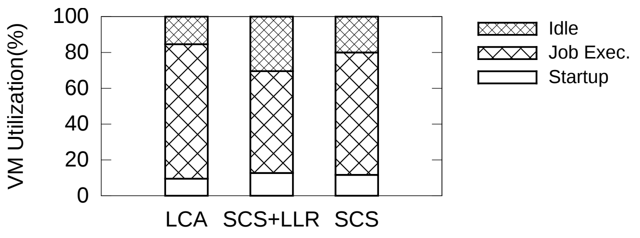
(a) Steady Workload



(b) Bursty Workload



(c) Incremental Workload



(d) Random Workload

Fig. 7. VM Utilization of LCA, SCS+LLR, and SCS in the four workload patterns (stready, bursty, incremental, and random).

state-of-the-art baselines under identical workloads. The results show that LCA has a 28%–73% better job deadline satisfaction rate and achieves a 17.74%–35.10% higher cost efficiency than the baselines. Although we use a single scientific application for the performance evaluation, we believe that the three main approaches are highly relevant to the study of proactive cloud resource management systems for other scientific applications.

References

[1] Amazon Web Services. http://aws.amazon.com.

[2] Microsoft Azure. http://azure.microsoft.com.

[3] Right Scale. http://www.rightscale.com/.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A View of Cloud Computing. *Communications of ACM*, 53(4), 2010.

[5] P. A. Dinda and D. R. O'Hallaron. Host Load Prediction using Linear Models. *Cluster Computing*, 3(4):265–280, 2000.

[6] Z. Gong, X. Gu, and J. Wilkes. PRESS: PRedictive Elastic ReSource Scaling for Cloud Systems. In *6th International Conference on Network and Service Management (CNSM 2010)*, 2012.

[7] T. Hastie, R. Tibshirani, and J. Friedman. The Element of Statistical Learning: Data Mining, Inference, and Prediction. 2011.

[8] ivadon Chaisiri, B.-S. Lee, and D. Niyato. Optimization of Resource Provisioning Cost in Cloud Computing. *IEEE Transactions on Service Computing*, 5, 2012.

[9] J. Jiang, J. Lu, G. Zhang, and G. Long. Optimal Cloud Resource Auto-Scaling for Web Applications. In *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (IEEE/ACM CCGrid 2013)*, 2013.

[10] N. H. Kapadia, J. A. B. Fortes, and C. E. Brodley. Predictive Application-Performance Modeling in a Computational Grid Environment. In *8th IEEE International Symposium on High Performance Distributed Computing (HPDC '99)*, 1999.

[11] F. Karimipour, M. Ghandehari, and H. Ledoux. Watershed delineation from the medial axis of river networks. *Computer & Geosciences*, 59:132–147, 2013.

[12] B.-D. Lee and J. M. Schopf. Run-Time Prediction of Parallel Applications on Shared Environments. In *2003 IEEE International Conference on Cluster Computing (CLUSTER 2003)*, 2003.

[13] H. Li, D. Groep, and L. Wolters. Efficient Response Time Predictions by Exploiting Application and Resource State Similarities. In *6th ACM/IEEE International Workshop on Grid Computing (GRID 2005)*, 2005.

[14] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski. Cost- and Deadline-Constrained Provisioning for Scientific Workflow Ensembles in IaaS Clouds. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2012)*, 2012.

[15] M. Mao and M. Humphrey. Auto-Scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2011)*, 2011.

[16] M. Mao and M. Humphrey. A Performance Study on the VM Startup Time in the Cloud. In *5th International Conference on Cloud Computing (IEEE CLOUD 2012)*, 2012.

[17] M. Mao, J. Li, and M. Humphrey. Cloud Auto-Scaling with Deadline and Budget Constraint. In *11th ACM/IEEE International Conference on Grid Computing (GRID 2010)*, 2010.

[18] T. N. Minh and L. Wolters. Using Historical Data to Predict Application Runtimes on Backfilling Parallel Systems. In *18th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2010)*, 2010.

[19] N. Roy, A. Dubey, and A. Gokhale. Efficient Autoscaling in the Cloud using Predictive Models for Workload Forecasting. In *4th International Conference on Cloud Computing (IEEE CLOUD 2011)*, 2011.

[20] W. Smith. Prediction Services for Distributed Computing. In *21th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007)*, 2007.

[21] W. Smith, I. Foster, and V. Taylor. Using Historical Data to Predict Application Runtimes on Backfilling Parallel Systems. In *3rd International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 1998)*, 1998.

[22] Wikipedia. Pearson correlation coefficient. http://en.wikipedia.org/wiki/Pearson_productmoment_correlation _coefficien.