Uncertainty-Aware Elastic Virtual Machine Scheduling for Stream Processing Systems

Shigeru Imai, Stacy Patterson, and Carlos A. Varela
Department of Computer Science, Rensselaer Polytechnic Institute
{*imais,sep,cvarela*}*@cs.rpi.edu*

***Abstract*—Stream processing systems deployed on the cloud need to be elastic to effectively accommodate workload variations over time. Performance models can predict maximum sustainable throughput (MST) as a function of the number of VMs allocated. We present a scheduling framework that incorporates three statistical techniques to improve Quality of Service (QoS) of cloud stream processing systems: (i) uncertainty quantification to consider variance in the MST model; (ii) online learning to update MST model as new performance metrics are gathered; and (iii) workload models to predict input data stream rates assuming regular patterns occur over time. Our framework can be parameterized by a QoS satisfaction target that statistically finds the best performance/cost tradeoff. Our results illustrate that each of the three techniques alone significantly improves QoS, from 52% to 73-81% QoS satisfaction rates on average for eight benchmark applications. Furthermore, applying all three techniques allows us to reach 98.62% QoS satisfaction rate with a cost less than twice the cost of the optimal (in hindsight) VM allocations, and half of the cost of allocating VMs for the peak demand in the workload.**

I. Introduction

In stream processing, the rates of incoming data streams can fluctuate over time, creating dynamically changing computational demand. To guarantee Quality-of-Service (QoS) under the varying computing demand in a cost-effective way necessitates elastically changing the number of VMs allocated to a cloud-hosted application.

One way to classify elastic VM scheduling techniques is by whether they are *reactive* or *proactive*. Reactive scheduling allocates or deallocates VMs in response to some resource usage metric changes (*e.g.*, CPU, memory, network), such as is done in AWS AutoScaling [1]. On the other hand, proactive scheduling makes scaling decisions using predictions of application performance and future workloads, to scale-up the number of VMs before performance degrades or to scale-down the number of VMs to reduce cost while maintaining QoS. Proactive scheduling approaches that incorporate measures of application performance and workloads have the potential to achieve better QoS at lower cost, when compared to reactive approaches. However, their cost-effectiveness and QoS depend on the accuracy of the performance and workload prediction mechanisms.

Challenges to accurate performance prediction arise from factors such as skewed data distribution [2] and resource contention with other applications sharing the same VMs [3]. It has been shown that if a VM scheduler incorporates an application's performance variability, it can achieve higher QoS objective satisfaction compared to a scheduler that is agnostic of the performance variability [4], [5]. It is also common to use manual over-provisioning to compensate for potential inaccuracies in performance models [6], [7], [8].

Workload prediction, such as ARIMA [9], has been used in elastic VM scheduling [10], [7], [11], [6], [8]. The VM schedulers in these works select the number of VMs based on the prediction value with the minimum expected error, but they do not consider the variability of workloads. Even though real-world time series often have recurring patterns, they can be affected by irregular events, resulting in larger variance. Despite the evidence of the impact of uncertainties in both application performance and future workload, existing works in VM scheduling have not addressed these uncertainties in a holistic manner.

This work proposes a framework for proactive elastic VM scheduling for stream processing systems in cloud computing environments that explicitly models uncertainties in 1) VM performance, 2) application performance, and 3) workload variability. We quantify the uncertainty from #1 and #2 through the variance of an application performance model and #3 through the variance of a workload prediction model. By incorporating uncertainties from both prediction models into scaling decisions, our elastic stream processing framework achieves high QoS objective satisfaction rates without human intervention at far lower cost than a static over-provisioning approach. We focus on the application performance metric of *maximum sustainable throughput* (MST) [8]. MST represents the maximum data input rate a stream processing application can process without backlogging any input data. Using future workload and MST prediction models, our VM scheduler proactively schedules VMs to satisfy the following *throughput-QoS* objective:

$$\lambda_t < \tau(m_t), \tag{1}$$

where λ_t is the input workload data rate to the stream processing system at time t, and $\tau(m_t)$ is the MST for a number of VMs m_t. This objective ensures that the system maintains an MST that is greater than the data rate of input workloads.

Through simulations following real-world workload distributions, we show that our scheduling framework achieves 98.62% of QoS satisfaction rate (see Eq. (22) for definition), at up to 48% lower cost compared to static scheduling that covers the peak workload.

The rest of the paper is organized as follows. In Section II, we present background on sustainable stream processing, including the concept of MST, and we formulate the problem we address in this work. Section III presents the proposed scheduling techniques. Section IV describes our workloads and evaluation setup, and Section V shows the evaluation results. We present related work in Section VI and conclude the paper in Section VII.

II. Background and System Architecture

A. Stream Processing Environment

Figure 1 shows a common stream processing environment which consists of a data producer, a message broker (*i.e.*, Kafka), a stream processing system, and a data store. Data streams flow from left to right, starting from the data producer to the data store. The data producer sends events at the input data rate of λ_t Mbytes/sec at time t, and they are appended to message queues in Kafka. The input data rate fluctuates over time. Kafka is a message queuing system that is scalable, high-throughput, and fault-tolerant [12] and is supported by major stream processing frameworks including Apache Storm [13] and Apache Flink [14]. The stream processing system pulls data out of Kafka as quickly as it can at the throughput of $\mu(m_t)$ Mbytes/sec, where m_t is the number of allocated VMs at time t. After the stream processing system processes events, it optionally stores results in the data store (*e.g.*, file system or database).

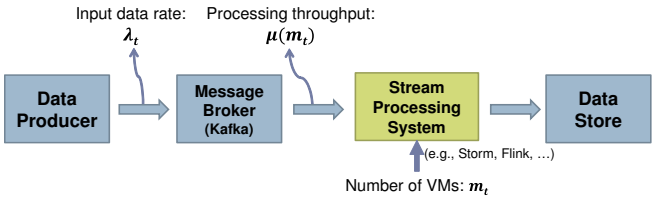

Figure 1. Stream processing environment consists of data producer, message broker (Kafka), stream processing system, and data store.

In our previous works [8], [15], we introduced the concept of MST. MST is the maximum throughput that a stream processing system can ingest indefinitely with a given number of VMs. If the input workload data rate is consistently greater than the MST (*i.e.*, if we keep violating the throughput-QoS objective in Inequality (1)), backlogged data in Kafka eventually exceeds the system resource capacity and makes the entire system inoperable. Thus, it is important that the MST of the allocated number of VMs be consistently greater than the fluctuating input workloads over time.

To measure MST, we pre-fill Kafka with enough data and monitor outgoing throughput from it. Since Kafka provides data to the stream processing system with insignificant overhead, the stream processing system can process data at its maximum rate. To detect the convergence of throughput, the following K out of N method is used: we monitor the latest N samples, and if K samples are within $\pm\delta\%$ (*e.g.*, 5%) from the previous sample, we determine that the throughput has converged at the MST.

B. Elastic VM Scheduling Framework

Figure 2 shows the architecture of the stream processing system in which the elastic VM scheduling framework works. We run a stream processing framework such as Storm on a cloud computing environment. To satisfy the throughput-QoS objective (1), the elastic VM scheduling framework adjusts the number of VMs to accommodate the input data rate and reconfigures the application.

The MST prediction model is trained with performance metrics obtained from the application monitor. This model predicts the number of VMs that will be needed to process the workload. The workload predication model obtains workload information (*i.e.*, input data rates) from the workload monitor, and it uses this information to predict future workloads. Both models can be trained offline using small workloads as we show in Section IV and then updated adaptively as the application executes. Inside the scheduling framework, the VM scheduler makes scaling decisions using the MST and the workload prediction models. Since it takes tens of seconds to minutes to launch a VM, predicting future workloads and proactively allocating VMs to cover the predicted workloads can be effective in achieving a high QoS satisfaction rate.

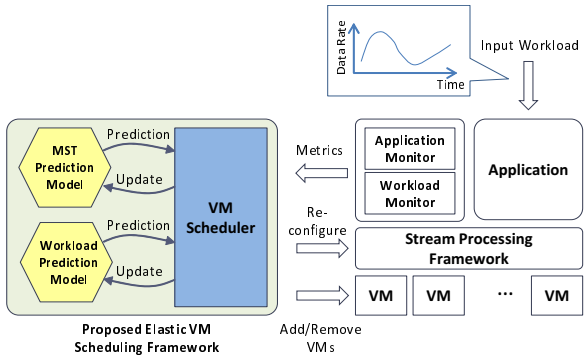

Figure 2. Stream processing system architecture including elastic VM scheduling framework.

C. Problem Formulation

We model the workload changes λ_t over time as a discrete sequence, with timesteps $t = 1, 2, 3, \ldots$. Every S timesteps, we make a scaling decision whether to allocate or shut down VMs. This means that we have a constant VM configuration

for the duration of S timesteps, whereas the workload changes every timestep. When we start a new VM, we assume it takes $U(< S)$ timesteps until it becomes available. For example, we use $S = 10$ minutes and $U = 2$ minutes.

The problem is to find a cost-minimal sequence of VM allocations such that the MST is greater than or equal to the input data rate λ_t for all t. Thus, for each update interval T, we must determine $\hat{m}_T$ such that,

$$\hat{m}_T = \arg \min_{m \in \{1,2,...,\text{Max VMs}\}} \left(\max_{t=T,...,T+(S-1)} \lambda_t < \tau(m) \right).$$

III. VM Scheduling Techniques

In this section, we describe our methods for estimating the number of VMs the scheduler should allocate at each update interval. A novel feature of our approach is that we explicitly incorporate uncertainty into our models; this includes uncertainty in the estimation process as well as uncertainty about the change in workloads. By incorporating uncertainty, our framework is better able to meet QoS requirements, as demonstrated in Section V.

We first present our baseline VM scheduling approach. We then describe three methods to incorporate uncertainty. These methods can be used independently or in combination. We evaluate the effectiveness of these policies in Section V.

A. Baseline MST

This technique schedules VMs based on a MST model without workload forecasting [8], [15].

1) Offline Model Training: Given a training set $\mathcal{D}_{\text{train}} = \{(m_1, \tau_1), (m_2, \tau_2,), ..., (m_n, \tau_n)\}$, where m_i is the i-th number of VMs and τ_i is the measured MST for m_i, we train an MST prediction model $\hat{\tau}(m)$ using linear regression. In [8], we proposed the following two models to predict MST for m homogeneous VMs:

- Model 1:

$$\hat{\tau}_1(m) = \frac{1}{w_0 + w_1 \cdot \frac{1}{m} + w_2 \cdot m + w_3 \cdot m^2} \quad (2)$$

- Model 2:

$$\hat{\tau}_2(m) = w_0 + w_1 \cdot m - w_2 \cdot m^2 \quad (3)$$

To save cost and time to collect MST samples while achieving high prediction accuracy, we statistically determine the most effective set of VMs $\hat{\mathcal{S}}$ using the simple resource (*i.e.*, CPU, memory, and network intensive) benchmarks from Intel Storm Benchmarks [16] . Thus, for all $m_i \in \hat{\mathcal{S}}$, we collect a corresponding MST sample τ_i and use them to train both Models 1 and 2.

2) Model Selection: Once we train both Models 1 and 2 using MST samples obtained for $\hat{\mathcal{S}}$, we then select the best-fitting model for the target application through the evaluation of extra validation samples [15]. Since it may be difficult to find one prediction model that works well for multiple stream processing applications with various scalability characteristics, this process may improve the selected model's prediction accuracy. The validation samples are selected from VMs that are not included in $\hat{\mathcal{S}}$, but from where there is a discrepancy between the prediction values from the two models. If the discrepancy at m VMs is greater than a threshold, we add m to the validation samples. If the discrepancy is never greater than the threshold, we select the model with smaller training error.

3) VM Scheduling: At the beginning of every scheduling cycle time t, we estimate the minimum number of VMs to cover the given workload λ_t using a MST prediction model $\hat{\tau}(m)$ as follows:

$$\hat{m}_t = \min m \text{ s.t. } \hat{\tau}(m) > \lambda_t. \quad (4)$$

When scaling up, new VMs will become ready after the VM startup delay of U timesteps. On the other hand, when scaling down, VMs will be immediately shut down.

B. Online Model Learning

Online learning can be used to update MST models as we obtain new training samples. These training samples can reduce uncertainty in the performance models that are trained offline. Even without any offline training, the prediction accuracy with solely online training can gradually approach the accuracy with offline training.

By definition, MST can be obtained when excessive workloads saturate the system capacity. Thus, when the scheduler under-provisions VMs for the application workload (*i.e.*, when the throughput-QoS objective is violated), the application monitor can obtain a new training sample pair (m_i, τ_i). This sample is added to the training data set $\mathcal{D}_{\text{train}}$ to update the selected MST model. Note that the formulation (4) does not change even if we apply online learning to the MST model $\hat{\tau}(m)$.

C. Uncertainty-Awareness for MST

We quantify uncertainty from VM and application performance through the variance of an MST model. As explained in Section III-A1, we train Models 1 and 2 using a training set $\mathcal{D}_{\text{train}}$, and then select the model that better fits the training data. Let $\hat{\tau}(m)$ be the MST model trained using linear regression. For the i-th pair (m_i, τ_i) in the training data set $\mathcal{D}_{\text{train}}$, we assume the following relationship holds between the measured MST τ_i and predicted MST values $\hat{\tau}(m_i)$ for $i = 1, 2, ..., n$, where n is the size of the training data set:

$$\tau_i = \hat{\tau}(m_i) + \varepsilon_i, \quad (5)$$

where $\varepsilon_i \sim \mathcal{N}(0, \sigma_\tau^2)$. $\hat{\sigma}_\tau$ is an estimate of the true variance σ_τ^2 [9]:

$$\hat{\sigma}_\tau^2 = \frac{\sum_{i=1}^{n} (\tau_i - \hat{\tau}(m_i))^2}{n - l}, \quad (6)$$

where l is the number of regression coefficients in the MST prediction model.

Based on Eq. (5), we assume that MST follows the normal distribution:

$$\tau(m) \sim \mathcal{N}(\hat{\tau}(m), \hat{\sigma}_\tau^2). \tag{7}$$

Given a workload at time t λ_t, we introduce a new random variable $\delta = \tau(m) - \lambda_t$. Since λ_t is a constant for a time period of $[t, t+1)$ (*i.e.*, one minute in this work), δ follows $\mathcal{N}(\hat{\tau}(m) - \lambda_t, \hat{\sigma}_\tau^2)$. We can estimate the probability of $\delta > 0$:

$$\Pr[\delta > 0] \quad = \quad \int_0^\infty f(\delta \mid \hat{\tau}(m) - \lambda_t, \hat{\sigma}_\tau^2)\, d\delta,$$

where $f(\delta|\hat{\tau}(m) - \lambda_t, \hat{\sigma}_\tau^2)$ is the probability density function for the normal distribution (7). Figure 3 shows this probability density function. In the figure, the shaded area corresponds to the value of $\Pr[\tau(m) - \lambda_t > 0]$. We estimate

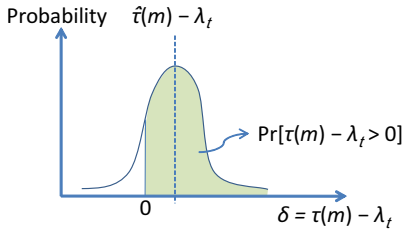

Figure 3. Probability density function for normal distribution $\mathcal{N}(\hat{\tau}(m) - \lambda_t, \hat{\sigma}_\tau^2)$. The shaded area corresponds to the value of $\Pr[\tau(m) - \lambda_t > 0]$.

the minimum number of VMs $\hat{m}$ that are required to satisfy the throughput-QoS objective $\tau(m) > \lambda_t$ as follows:

$$\hat{m}_t = \min(m) \text{ s.t. } \Pr[\tau(m) - \lambda_t > 0] \geq \rho, \tag{8}$$

where $\rho \in [0.5, 1.0)$ is a *QoS satisfaction target*.

In this formulation, uncertainty is quantified in terms of variance, as shown in Eq. (6), and larger variance leads to require a larger number of VMs to satisfy the constraint in Eq. (8). When $\rho \approx 1.0$, we expect δ to be large to prevent QoS violations, which means that the mean of the normal distribution in Figure 3 is far away from zero. When $\rho = 0.5$, however, we expect δ to be very close to zero. When $\rho = 0.5$, this is equivalent to finding the smallest $\hat{m}$ that satisfies $\hat{\tau}(m) > \lambda_t$ (*i.e.*, it is equivalent to the baseline MST in Section III-A).

D. Uncertainty-Awareness for Workload Forecasting

In this section, we add uncertainty-awareness for workload forecasting to the baseline MST shown in Section III-A. We make multiple forecasts into the next scheduling cycle and determine the highest expected workload we should cover to satisfy the throughput-QoS objective.

1) Workload Forecasting: We use Autoregressive-moving-average (ARMA) models [9] to forecast future workloads. An ARMA(p, q) model consists of an autoregressive (AR) term of order p and a moving-average (MA) term of order q as follows:

$$\lambda_t = \phi_1\lambda_{t-1} + \cdots + \phi_p\lambda_{t-p} + \varepsilon_t + \theta_1\varepsilon_{t-1} + \cdots + \theta_q\varepsilon_{t-q}, \tag{9}$$

where $\lambda_t, \lambda_{t-1}, ..., \lambda_{t-p}$ is a series of observed input data rates; $\varepsilon_t, \varepsilon_{t-1}, ..., \varepsilon_{t-q}$ is a series of error terms $\varepsilon_t \sim \mathcal{N}(0, \sigma_\lambda^2)$; and $\phi = (\phi_1, ..., \phi_p)$ and $\theta = (\theta_1, ..., \theta_q)$ are model parameters. Suppose we have a series of n input data rates $\lambda_t, \lambda_{t-1}, ..., \lambda_{t-n-1}$; maximum likelihood estimation is used to estimate ϕ and θ. Once we estimate ϕ and θ, we calculate a k-step ahead forecast $\hat{\lambda}_{t+k}$ by recursively applying Eq. (9) as follows:

$$\begin{aligned} \hat{\lambda}_{t+k} \quad = \quad & \phi_1\lambda_{t+k-1} + \cdots + \phi_p\lambda_{t+k-p} + \\ & \varepsilon_{t+k} + \theta_1\varepsilon_{t+k-1} + \cdots + \theta_q\varepsilon_{t+k-q}. \end{aligned} \tag{10}$$

An estimate of σ_λ^2 is given by:

$$\hat{\sigma}_\lambda^2 = \frac{\phi_1^2 + \cdots + \phi_p^2 + \theta_1^2 + \cdots + \theta_q^2}{n}. \tag{11}$$

Once we obtain the estimates for ϕ and θ, we can define a new set of parameters $\psi = (\psi_1, \psi_2, ..., \psi_j)$ as follows:

$$\begin{aligned} \psi_1 \quad &= \quad \phi_1 - \theta_1 \\ \psi_2 \quad &= \quad \phi_1\psi_1 + \phi_2 - \theta_2 \\ &\vdots \\ \psi_j \quad &= \quad \phi_1\psi_{j-1} + \cdots + \phi_p\psi_{j-p} - \theta_j, \end{aligned} \tag{12}$$

where $\theta_j = 0$ for $j > q$. Using $\hat{\sigma}_\lambda^2$ and ψ, we can estimate the variance for the k-step ahead forecast $\hat{\sigma}_{\lambda|t+k}^2$ as follows [9]:

$$\hat{\sigma}_{\lambda|t+k}^2 = \left\{1 + \sum_{j=1}^{k} \psi_j^2\right\}\hat{\sigma}_\lambda^2. \tag{13}$$

2) Highest Workload Estimation: Figure 4 shows how we determine the highest expected workload in the next scheduling cycle. For the scheduling cycle starting at time t, we make a scaling decision at $t - U$ to make all newly created VMs ready at the beginning of the next cycle at time t. First, we forecast workloads and variances for the next scheduling cycle in $\hat{\lambda}_{t+k}$ and $\hat{\sigma}_{\lambda|t+k}^2$ for $k \in [0, S)$ using Eqs. (10) and (13), respectively. We then determine the timestep h with the largest expected workload at $t + h$:

$$h = \underset{k \in [0,S)}{\operatorname{argmax}} \ \hat{\lambda}_{t+k} + 2\hat{\sigma}_{\lambda|t+k}. \tag{14}$$

We add $2\hat{\sigma}_{\lambda|t+k}$ to the forecasted value in Eq. (14) to yield a conservative VM allocation.

3) VM Scheduling: Once we determine h from Eq. (14), we estimate the minimum number of VMs needed to cover the workload $\hat{\lambda}_{t+h}$ using the method described in Section III-C to incorporate the predicted workload variance. Assuming the h-step ahead forecast follows the normal distribution:

$$\lambda_{t+h} \sim \mathcal{N}(\hat{\lambda}_{t+h}, \hat{\sigma}_{\lambda|t+h}^2), \tag{15}$$

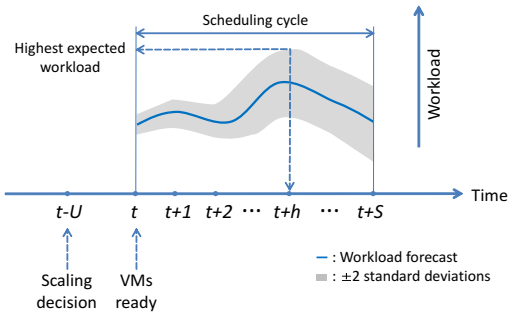

Figure 4. Finding the highest expected workload in the scheduling cycle $[t, t+S)$.

the difference between MST and the λ_{t+h}, $\delta = \tau(m) - \lambda_{t+h}$, is also expected to follow a normal distribution:

$$\delta \sim \mathcal{N}(\hat{\tau}(m) - \hat{\lambda}_{t+h}, \hat{\sigma}^2_{\lambda|t+h}). \quad (16)$$

Similar to Eq. (8), we estimate the probability of $\delta > 0$:

$$\Pr[\delta > 0] = \int_0^\infty f(\delta|\hat{\tau}(m) - \hat{\lambda}_{t+h}, \hat{\sigma}^2_{\lambda|t+h})\, d\delta. \quad (17)$$

We estimate the minimum number of VMs $\hat{m}$ required to satisfy the throughput-QoS objective $\tau(m) > \lambda_{t+h}$ as:

$$\hat{m}_t = \min(m) \text{ s.t. } \Pr[\tau(m) - \lambda_{t+h} > 0] \geq \rho. \quad (18)$$

E. Uncertainty-Awareness for both MST and Workload Forecasting

We now show how to incorporate uncertainties from both MST and workload forecasting into the VM scheduling. Following the method in Section III-D2, we first determine the future timestep $t+h$ that is expected to give the largest workload in the next scheduling cycle. Assuming that the MST and workloads are independent, the difference between MST and the h-step ahead workload, $\delta = \tau(m) - \lambda_{t+h}$, also follows a normal distribution:

$$\delta \sim \mathcal{N}(\hat{\tau}(m) - \hat{\lambda}_{t+h}, \hat{\sigma}^2_\tau + \hat{\sigma}^2_{\lambda|t+h}). \quad (19)$$

Similar to Eqs. (8) and (17), we estimate the probability of $\delta > 0$ as follows:

$$\Pr[\delta > 0] = \int_0^\infty f(\delta|\hat{\tau}(m) - \hat{\lambda}_{t+h}, \hat{\sigma}^2_\tau + \hat{\sigma}^2_{\lambda|t+h})\, d\delta, \quad (20)$$

where $f(\delta \mid \hat{\tau}(m) - \hat{\lambda}_{t+k}, \hat{\sigma}^2_\tau + \hat{\sigma}^2_{\lambda|t+k})$ is the probability density function for the normal distribution (16). We then estimate the minimum number of VMs $\hat{m}$ required to satisfy the throughput-QoS objective $\tau(m) > \lambda_{t+h}$ as:

$$\hat{m}_t = \min(m) \text{ s.t. } \Pr[\tau(m) - \lambda_{t+h} > 0] \geq \rho. \quad (21)$$

IV. Evaluation Setup

In this section, we describe the test applications and workloads used in our evaluations. We also describe the offline training phase and parameterizations for our application performance and workload models.

A. Test Applications and Workloads

We evaluate our proposed framework using the following applications:

- *Grep, Rolling Count, Unique Visitor, Page View*, and *Data Clean*: Typical use-case benchmarks from Intel Storm Benchmarks [16].
- *Vertical Hoeffding Tree (VHT)*: Stream machine learning application from Apache SAMOA [17].
- *Rolling Hashtag Count*: Typical word count-like application which counts the number of Twitter hashtags. It outputs hashtag counts every five seconds for the results computed in the 60 seconds moving window.
- *Rolling Flight Distances*: This application computes distances between all flights in the near future. It outputs flight pairs which distance is less than a threshold every five seconds for the results computed in the 60 second moving window.

We use the three workloads as shown in Figure 5. All workloads are time series of data rates in Kbytes/sec over one week of time. These workloads were created as follows:

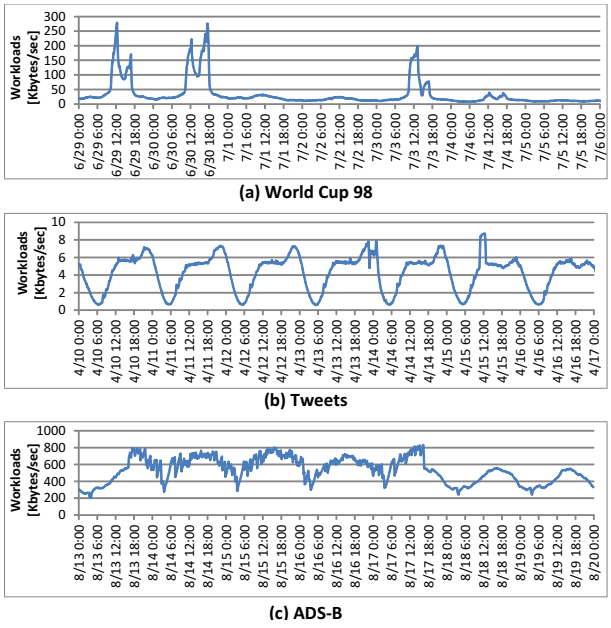

Figure 5. Workloads used for evaluation. Each workload has one week of data: (a) World Cup 98 (6/29/1998-7/5/1998), (b) Tweets (4/10/2016-4/16/2016), and (c) ADS-B (8/13/2017-8/19/2017).

- *World Cup 98* in Figure 5(a) was created from the FIFA World Cup 1998 website access logs [18] (duration: 6/29/1998-7/5/1998, time in UTC). The workload spikes corresponds to the days when there were popular matches. For example, there were two matches of round of 16 on June 30 and two quarter finals on July 3.
- *Tweets* in Figure 5(b) was created from tweets downloaded from twitter.com using Twitter APIs [19] (du-

ration: 4/10/2016 to 4/16/2016, time in EDT). Downloaded tweets were those in English from the U.S. contiguous 48 states. Even though we were only allowed to obtain a fraction of all tweets from that time period, the time series shows clear daily recurring patterns.

- *ADS-B* in Figure 5(c) was created from the Automatic Dependent Surveillance-Broadcast (ADS-B) data from ADS-B Exchange [20] (duration: 8/13/2017-8/19/2017, time in UTC). The data are provided by worldwide community of volunteers and contains flight data from all over the world. Just as the Tweets workload, it also shows daily recurring patterns.

B. Offline MST Model Training

We obtain MST samples as described in Section II-A with the following parameters for convergence detection: $N = 5, K = 4, \delta = 5\%$. All the test applications are implemented with Apache Storm v0.10.0 and are configured to process stream data from Kafka v2.11. Both Storm and Kafka are running on Amazon EC2, where we use the m4.large VM instance type for all the worker nodes of Storm. As we have explained in Section III-A, we collect MST samples for each test application for the pre-determined effective set of VMs $\hat{\mathcal{S}} = \{3, 4, 6, 8, 24\}$ to predict MST for up to 128 VMs. We then train both Models 1 and 2 and select the better-fitting model for each application. For Rolling Flight Distances, we use the following $\mathcal{S}_{\text{flight}} = \{1, 2, 3, 5, 6, 8, 9, 11, 12, 14\}$ since it is designed to process the actual ADS-B workloads in Figure 5(c) and the MST of 14 VMs is sufficient to process the peak workload of 800 Kbytes/sec. Figure 6 shows the selected MST models. Predicted MST values are plotted together with actual MST measurements for up to 128 VMs except for Rolling Flight Distances, which shows measurements up to 14 VMs.

C. Workload Forecasting Model Training

As shown in Eq. (9), an ARMA(p, q) model consists of an autoregressive (AR) term of order p and a moving-average (MA) term of order q. We determine the values for p and q using Akaike information criterion (AIC), which is commonly used in time series analysis. AIC evaluates the fitness of a model to data as well as the simplicity of the model. Using the first 100 values from each test workload, we perform a grid search to find the (p, q) pair that gives the lowest AIC: we compute AIC for all possible pairs of $p = 0, 1, 2, 3$ and $q = 0, 1, 2, 3$ except for $(p, q) = (0, 0)$. As the result of the search, we obtained models as shown in Table I. For online update of the model, we use a sliding window of the past 100 workload values to estimate ϕ and θ in Eq. (9) by maximum likelihood estimation.

Table I
SELECTED ARMA MODELS FOR THE TEST WORKLOADS.

Workload	Model
World Cup 98	ARMA(1, 2)
Tweets	ARMA(1, 1)
ADS-B	ARMA(1, 3)

V. EVALUATION

We evaluate the VM scheduling techniques presented in Section III with the real-world workloads shown in Figure 5. Evaluations are simulation-based, however, MST prediction models are trained with real measurements as described in Section IV-B. We first describe the experiment settings that are common to all evaluations. We then give the experimental results.

A. Common Experimental Settings

1) Simulation Time Horizon: We perform simulations over discrete timesteps $t = 1, 2, ..., T$, where a workload value λ_t is given for each t. The workloads change every minute and have duration of one week, so $T = 7 \times 24 \times 60 = 10,080$ for all three workloads. Our VM scheduler performs VM allocation or deallocation every S timesteps and consider U timesteps of VM startup time. In the following simulations, we use $S = 10$ and $U = 2$, which correspond to 10 and 2 minutes of physical time, respectively.

2) Workloads and Test Applications: We test the workloads against the MST models created for the test applications as shown in Table II. The World Cup 98 workload is used for Grep, Rolling Count, Unique Visitor, Page View, Data Clean, and VHT to evaluate applications with different scaling patterns with a consistent real-world workload. For Rolling Hashtag Count and Rolling Flight Distances, we use the Tweets and ADS-B workloads that are actually processed by these two applications to create their MST models, respectively. Since data rates in the test workloads and throughput generated from the trained MST models are not directly comparable to the World Cup 98 and ADS-B workloads, we artificially create more workload data proportionally keeping workload distribution patterns to reach a maximum throughput as shown in the "Peak Throughput" column in Table II. For the ADS-B workload, actual peak of the workload and the MST model for Rolling Flight Distances match, so there is no need to recreate workload.

Table II
WORKLOADS AND TEST APPLICATIONS (MST MODELS) USED FOR EVALUATIONS.

Workload	Test Application (MST models)	Peak Throughput [Mbytes/sec]
World Cup 98	Grep	32
	Rolling Count	10
	Unique Visitor	48
	Page View	46
	Data Clean	5
	VHT	49
Tweets	Rolling Hashtag Count	200
ADS-B	Rolling Flight Distances	None

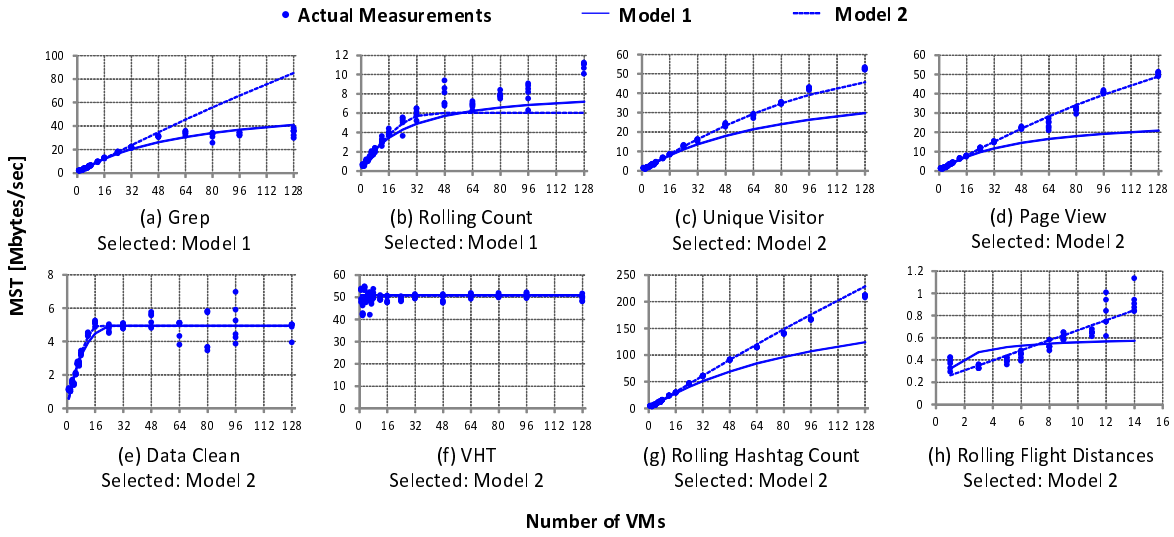

Figure 6. Selected MST prediction models after training. Models from (a) Grep to (g) Rolling Hashtag Count are trained with samples obtained from up to 24 VMs in $\mathcal{S} = \{3, 4, 6, 8, 24\}$, whereas (h) Rolling Flight Distances is trained with samples obtained from up to 14 VMs in $\mathcal{S}_{\text{flight}} = \{1, 2, 3, 5, 6, 8, 9, 11, 12, 14\}$.

3) Ground Truth MST: To evaluate a series of allocated VM counts $\hat{m}_t (t = 1, 2, ..., T)$, we need to know the ground truth MST for m_t for a given application. We create a ground truth MST model $\tau_{\text{tru}}(m)$ from actual MST sampled when running the application under test. Given a VM count m, $\tau_{\text{tru}}(m)$ returns an MST value drawn from a normal distribution $\mathcal{N}(\mu_m, \sigma_m^2)$. The mean μ_m and variance σ_m^2 are created from pairwise linear interpolation of the mean and variance of the MST for measured VMs m_1 and m_2, where m_1 and m_2 are the two closest VM counts to m, with $m_1 < m < m_2$.

4) Baseline Scheduling: We use the following scheduling policies as the baseline.

Ground Truth: Optimal scaling policy based on the actual MST values. It allocates the minimum number of VMs that generates a larger MST than a given workload.

Static Peak: Static VM allocation policy that covers the peak workload. The peaks are equal to the normalization factors in Table II (for Rolling Flight Distances, the peak is 830 Kbytes/sec).

5) Evaluation Metrics: We use the following metrics to evaluate scheduling results.

$$\textbf{QoS Satisfaction Rate [\%]} = \frac{100}{T} \sum_{t=1}^{T} \mathcal{I}(\tau_{\text{tru}}(\hat{m}_t) > \lambda_t), \quad (22)$$

where T is the total timesteps (=10,080) of a simulation, $\hat{m}_t$ is the allocated VM counts using our VM scheduling technique at time t, λ_t is the workload at time t, and $\mathcal{I}(\cdot)$ is the indicator function that returns 1 if the argument is true and 0 otherwise. Note that since the scheduling cycle is $S = 10$ timesteps, we will compare ten workload values against a single MST value.

Relative VM costs: Since total VM cost depends on a sequence of workloads, we cannot directly compare absolute costs obtained from different workloads. Thus, we compute VM allocation cost relative to the cost obtained from the ground truth scheduling and the cost obtained from the static scheduling as follows:

$$RelativeCost_{\text{tru}} = \sum_{t=1}^{T} \hat{m}_t / \sum_{t=1}^{T} m_t, \quad (23)$$

$$RelativeCost_{\text{static}} = \sum_{t=1}^{T} \hat{m}_t / T \cdot \max_t m_t, \quad (24)$$

where $\hat{m}_t$ is the number of allocated VM at time t by the prediction framework and m_t is the true required number of VMs obtained by the ground truth scheduling at time t. Since we use homogeneous VMs, a relative cost is the ratio of the numbers of VMs.

B. Evaluation: Scheduling Policy vs. QoS & Cost

We first evaluate the impact of the different scheduling policies on QoS and Cost.

1) Experimental Settings: Depending on how we incorporate uncertainty awareness of MST, workload forecasting, and online learning techniques into the VM scheduler, we have multiple scheduling policies, as shown in Table III. Each column has the following meaning:

UA: This option enables uncertainty-awareness for MST.

OL: If this option is chosen, online learning for the MST model is performed as described in Section III-B, otherwise the MST model is fixed during the simulation.

ARMA: If this option is chosen, workload forecasting with an ARMA model is enabled and the scheduling is uncertainty-aware for workload forecasting and scaling, otherwise workload forecasting is not used.

Eq.: Depending on the combination of the three options, it shows a reference to the equation used for scheduling.

Table III
VM SCHEDULING POLICIES FOR EVALUATION.

Policy ID	MST		ARMA	Eq.
	UA	OL		
0: MST				(4)
1: MST + UA	✓			(8)
2: MST + OL		✓		(4)
3: MST + UA + OL	✓	✓		(8)
4: MST ; ARMA			✓	(18)
5: MST + UA ; ARMA	✓		✓	(21)
6: MST + OL ; ARMA		✓	✓	(18)
7: MST + UA + OL ; ARMA	✓	✓	✓	(21)

Policy IDs are named after the corresponding scheduling options. We ran simulations five times per application per policy. Since there are eight applications, the number of simulation runs per policy is $8 \times 5 = 40$. For each policy, we took the average of 40 runs for QoS satisfaction rates and relative costs against the ground truth and static peak scheduling policies. The QoS satisfaction target ρ is 0.95.

2) Scheduling Results: Figure 7 shows average (a) QoS satisfaction rates and (b) relative costs across all the applications for the scheduling policies in Table III, in descending order of the QoS satisfaction rate. Error bars show $\pm$ 1 standard deviation. We can see the overall trend where the QoS satisfaction rate improves with increasing cost as the policy becomes more complex. The most complex #7 policy achieved 98.62% QoS satisfaction rate. While it cost 84% more than the ground truth scheduling, it was 48% less when compared to static scheduling that covers the peak workload. We also notice that each of the three scheduling options significantly improved QoS satisfaction rates from 52% to 73-81% QoS satisfaction rates compared to the base #0 policy. Policies #7 and #5 show very close QoS satisfaction rates: 98.62% vs. 98.35%; however, in terms of the relative cost to ground truth, #7 costs 7% less than #5 (#7: 1.85 vs. #5: 1.99). This result suggests that the online learning technique contributes to lower the cost. Regardless of using the online learning technique or not, when we consider uncertainties from both MST and workload forecasting models in #7 and #5, we successfully achieved high QoS satisfaction rates. These satisfaction rates are also close to the QoS satisfaction target of $\rho = 0.95$.

Figure 8 shows application-wise QoS satisfaction rates and relative costs for four scheduling policies, #0, #1, #5, and #7. These policies were chosen so that a new scheduling technique was incrementally added. From #0 to #1, UA was added; from #1 to #5, ARMA was added; and from #5 to #7, OL was added. Overall, we can see that as we add a new technique, the QoS satisfaction rates improve. The first two rates improvements (*i.e.*, #0 to #1 and #1 to #5) can be explained in terms of the scheduling techniques: we

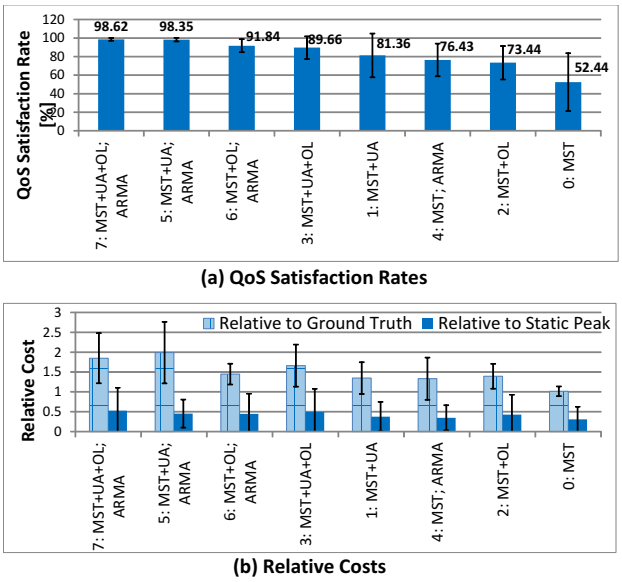

Figure 7. Average (a) QoS satisfaction rates and (b) Relative costs across the all applications for different scheduling policies in Table III. QoS satisfaction target: $\rho = 0.95$. Error bars show $\pm$ 1 standard deviation.

add variance for MST in (7) and then add another variance for workload forecasting incrementally in (16). Since larger variance leads to more conservative VM allocation, improvement in QoS satisfaction rates can be expected. For the third improvement (*i.e.*, #5 to #7), Unique Visitor, Page View, and Rolling Hashtag Count show at least 20% of QoS rate increase. This is most likely due to the relationship between initial MST models after offline training and actual MST sample values. As shown in Figure 6, these three models have some VM counts where predicted values exceed actual sample values. This means that the models over-estimate their MST, and thus they tend to under-provision VMs. By updating their MST models online, the models become more accurate, which can lead to better QoS satisfaction rates.

For all the applications except for VHT, if we spend more cost, we are rewarded with higher QoS satisfaction rates. However, VHT shows 100% QoS satisfaction rates even with the the most basic #0 policy. This is due to the fact that VHT does not scale at all as shown in Figure 6. To achieve a 100% QoS satisfaction rate, both static peak and ground truth policies needed 1 VM for the entire simulation period. However, our scheduling techniques with ARMA unnecessarily over-provision especially when spikes occur.

C. Evaluation: QoS satisfaction target vs. QoS & Cost

To investigate how the QoS satisfaction target ρ affects actual QoS and cost, we tested $\rho = \{0.5, 0.7, 0.9, 0.95\}$ with the most complex "#7:MST+UA+OL;ARMA" policy. We ran simulations five times per application per ρ and took the average QoS satisfaction rate and relative cost across all the applications. Figure 9 shows the results. We can

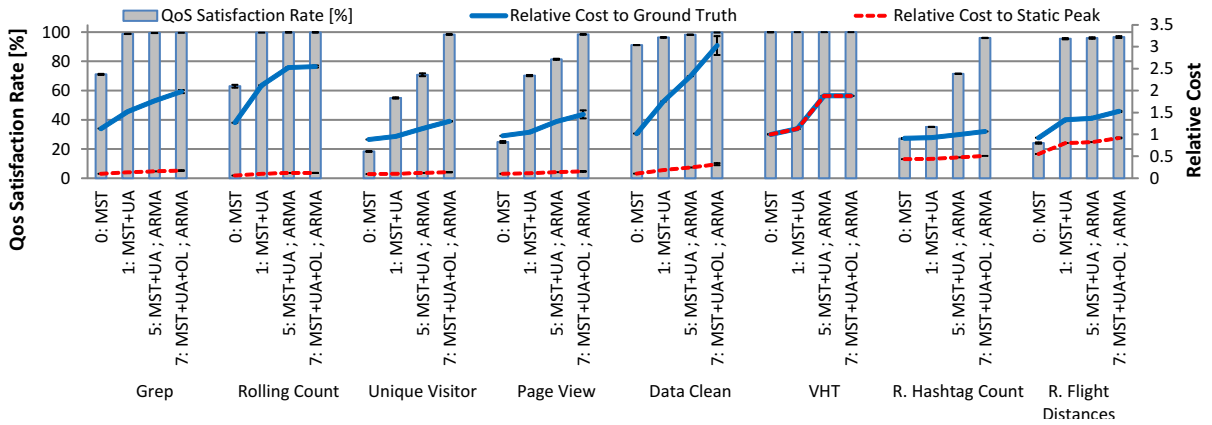

Figure 8. QoS satisfaction rates and relative costs for the #0, #1, #5, and #7 scheduling policies in Table III. QoS satisfaction target: $\rho = 0.95$. Error bars show $\pm$ 1 standard deviation.

see that ρ and QoS satisfaction rate positively correlate. In fact, Pearson's correlation coefficient between ρ and the QoS satisfaction rate was 0.652. Looking at the slope from $\rho = 0.9$ to $\rho = 0.95$ on the relative cost to ground truth, it is steeper than that from $\rho = 0.7$ to $\rho = 0.90$. This result matches the following fact: since ρ is a constraint on the probability of a normal distribution, when the value of ρ approaches 1.0, it will be required to allocate infinitely many VMs to satisfy the constraint.

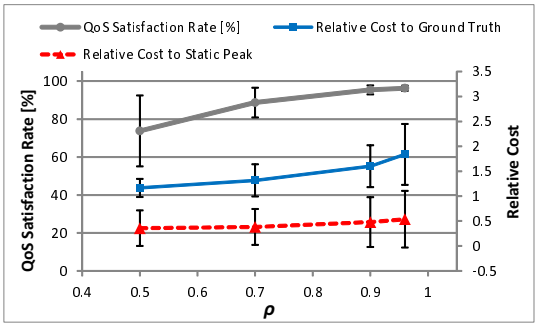

Figure 9. QoS satisfaction target ρ vs. actual QoS satisfaction rates and relative costs. Error bars show $\pm$ 1 standard deviation.

D. VM Allocation Sequence

To confirm the effectiveness of scheduling policies on VM allocation sequences, we evaluated the four scheduling policies #0, #1, #5, and #7 with $\rho = 0.95$ for the Grep application as shown in Figure 10. We plot (a) Input workload and allocated MST, (b) allocated number of VMs, and (c) backlogged data. Since the difference is hardly visible, in Figures 10(a) and (b), we only plot #0 and #7 together with the ground truth and static policies. From Figure 10(a), it appears #0 closely follows the input workload; however, from Figure 10(c), we can confirm that it accumulates up to 12.8 Gbytes of backlogged data when the input workload spikes. Since #0 does not have any uncertainty consideration or workload forecasting, this result is unavoidable. Unlike #0, #5 and #7 proactively react to the workload spikes and successfully avoid backlogging.

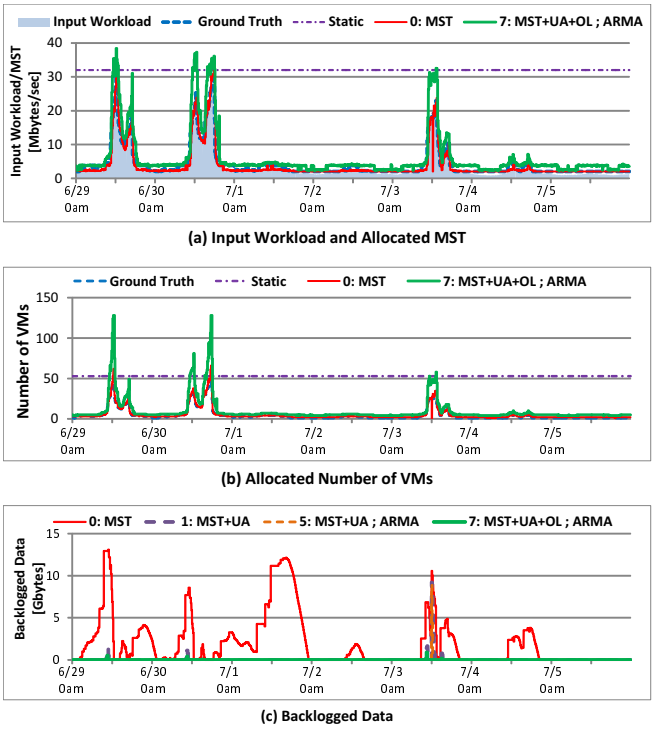

Figure 10. Scheduling results for the Grep application with the FIFA world cup 1998 website access workload (6/29/1998-7/5/1998): (a) Input workload and allocated MST, (b) Allocated number of VMs, and (c) Backlogged data.

VI. Related Work

Workload forecasting is widely used in proactive elastic VM scheduling. Roy et al. [10] used an ARIMA model to forecast the World Cup 98 workload, which is also used

in this work. Hu et al. proposed KSwSVR, a workload forecasting method based on Kalman filters and Support Vector Regression and applied it to elastic resource provisioning [6]. Jian et al. used a simple linear model to forecast future workloads, where parameters are estimated by linear regression, and to proactively allocate VMs estimated by a M/M/m queuing theory model [7]. ProRenaTa, developed by Liu et al. [11], combines proactive and reactive approaches for elastic scaling on distributed storage systems. In each scheduling cycle, the proactive scheduler first predicts workload using an ARIMA model and makes an initial scaling decision. To address inaccurate workload prediction, after the reactive controller observes the actual workload, it allocates extra VMs if needed. One of the major differences between our work and these works is that we explicitly account for estimated uncertainty from workload forecasting and use that information to achieve high QoS satisfaction rate in a cost efficient manner.

There are several previous works that require manual parameter configurations to achieve high QoS. Rodriguez and Buyya [5] considered performance variation of VMs for workflow applications with a performance degradation parameter defined for each VM type. This parameter effectively adjusts the over-provisioning rate per VM type. Even though KSwSVR [6] showed good prediction accuracy, they needed to specify a parameter for over-provisioning to reduce QoS violations. Similarly, in our own previous work [8], we introduced over-provisioning rate parameters to account for the inaccuracy of application performance prediction models. Our scheduling framework also provides ρ, a parameter to control QoS satisfaction rate; however, due to the reasonably positive correlation of 0.652 to the actual QoS satisfaction rate and also due to the fact that it is bounded by 1.0, the cost for finding the right ρ value can be significantly lower than the cost for parameter tuning in these existing works.

VII. Conclusion

We have proposed a framework for proactive elastic VM scheduling for stream processing systems in cloud computing environments that explicitly incorporates uncertainty in application performance and workloads. The framework estimates the required number of VMs that satisfies a certain probability criteria for the throughput-QoS objective, using expected variances from application performance and workload prediction models. We have shown that our scheduling framework can achieve 98.62% of QoS satisfaction in simulations using real-world workloads. Further, the scheduler achieves up to 48% lower cost compared to a static scheduling that covers the peak workload. For future work, we plan to incorporate reconfiguration costs into the VM scheduler.

References

[1] Amazon Web Services, "Amazon Auto Scaling," http://aws.amazon.com/autoscaling/.

[2] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "SkewTune: Mitigating Skew in Mapreduce Applications," in *Proc. ACM SIGMOD Conf. on Manage. of Data*, 2012, pp. 25–36.

[3] C. Delimitrou and C. Kozyrakis, "HCloud: Resource-efficient provisioning in shared cloud systems," in *ACM SIGOPS Operating Systems Review*, vol. 50, no. 2, 2016, pp. 473–488.

[4] N. Rameshan, Y. Liu, L. Navarro, and V. Vlassov, "Augmenting elasticity controllers for improved accuracy," in *Proc. IEEE Conf. on Autonomic Computing*, 2016, pp. 117–126.

[5] M. A. Rodriguez and R. Buyya, "Deadline Based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds," *IEEE Trans. on Cloud Computing*, vol. 2, no. 2, pp. 222–235, 2014.

[6] R. Hu, J. Jiang, G. Liu, and L. Wang, "KSwSVR: A new load forecasting method for efficient resources provisioning in cloud," in *Proc. IEEE Conf. on Services Computing*, 2013, pp. 120–127.

[7] J. Jiang, J. Lu, G. Zhang, and G. Long, "Optimal cloud resource auto-scaling for web applications," in *in Proc. IEEE Symp. on Cluster, Cloud and Grid Computing*, 2013, pp. 58–65.

[8] S. Imai, S. Patterson, and C. A. Varela, "Maximum Sustainable Throughput Prediction for Data Stream Processing over Public Clouds," in *Proc. IEEE Symp. on Cluster, Cloud and Grid Computing*, May 2017.

[9] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[10] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *Proc. IEEE Conf. on Cloud Computing*, 2011, pp. 500–507.

[11] Y. Liu, N. Rameshan, E. Monte, V. Vlassov, and L. Navarro, "ProRenaTa: Proactive and reactive tuning to scale a distributed storage system," in *Proc. IEEE Symp. on Cluster, Cloud and Grid Computing*, 2015, pp. 453–464.

[12] J. Kreps and L. Corp, "Kafka : a distributed messaging system for log processing," *ACM SIGMOD Workshop on Networking Meets Databases*, p. 6, 2011.

[13] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy, "Storm@twitter," in *Proc. of SIGMOD Conf. on Manage. of Data*, 2014, pp. 147–156.

[14] P. Carbone, S. Ewen, S. Haridi, A. Katsifodimos, V. Markl, and K. Tzoumas, "Apache Flink: Stream and batch processing in a single engine," *IEEE Data Eng. Bulletin*, vol. 38, no. 4, Dec 2015.

[15] S. Imai, S. Patterson, and C. A. Varela, "Maximum sustainable throughput prediction for large-scale data streaming systems," Department of Computer Science, Rensselaer Polytechnic Institute, Tech. Rep., November 2017, (Extended journal version of [8] in review).

[16] Intel Corporation, "Storm benchmark," https://github.com/intel-hadoop/storm-benchmark.

[17] G. D. F. Morales and A. Bifet, "SAMOA: Scalable Advanced Massive Online Analysis," *Journal of Machine Learning Research*, vol. 16, pp. 149–153, Jan 2015.

[18] M. Arlitt and T. Jin, "A workload characterization study of the 1998 world cup web site," *IEEE Network*, vol. 14, no. 3, pp. 30–37, May 2000.

[19] Twitter, "Twitter API reference," https://dev.twitter.com/docs.

[20] ADS-B Exchange, "ADS-B Exchange - World's largest co-op of unfiltered filght data," https://www.adsbexchange.com/.