Dynamic VM allocation in a SaaS environment

Brian Bouterse and Harry Perros
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
{bmbouter,hp}@ncsu.edu

Abstract

Given the costs associated with a cloud infrastructure, dynamic scheduling of virtual machines (VMs) can significantly lower costs while providing an acceptable service level. We develop a series of forecasting models for predicting demand for VMs in a cloud-based software used as a SaaS. These models are then used in a periodic-review provision model which determines how many VMs should be provisioned or de-provision at each inspection interval. A simple provisioning heuristic model is also proposed, whereby a fixed reserve capacity of VMs is continuously maintained. We evaluate and compare the performance of these models for different model parameters using historical data from the Virtual Computing Laboratory (VCL) at North Carolina State University.

Keywords

Provision of virtual machines (VMs), time-series forecasting models, hidden Markov models (HMM), Virtual Computing Laboratory (VCL), periodic-review model, capacity planning.

1. Introduction

Historically, a service provider owns the entire infrastructure, which is typically dimensioned for peak demand. Cloud computing provides an alternative way to use an infrastructure without owning it, where a service provider rents computing resources by the hour. The cloud infrastructure is provided by a cloud provider. This frees a service provider from the financial burden of owning equipment, but requires scheduling of resources in order to satisfy dynamically varying customer demand. In this paper, we address the issue of provisioning dynamically the capacity of a software offered as a Software-as-a-Service (SaaS). Capacity is expressed in VMs, where each VM can process N customers (i.e., service requests) concurrently.

There have been several auto-scaling related studies on the prediction of resources of a software running on a cloud, expressed in terms of servers, CPU, memory and storage, in order to satisfy given QoS constraints. Urgaonkar et al. [1] presented a dynamic resource provisioning of servers for Internet applications that employ a multi-tier architecture using a queueing network that depicts the flow of customers through a multi-tier system. In addition, they used reactive provisioning to predict short-term fluctuations. Roy et al. [2] considered how to auto-scale a given multi-tier software. They first proposed a second-order auto-regressive moving average (ARMA) model to predict demand, and then they used a queueing model to determine the required number of machines so that to satisfy a given response time. D. Minarolli and B. Freisleben [3] studied the problem of predicting the CPU and memory requirements of a multi-tier application using a support vector machine (SVM) model. Gong et al. [4] developed a system for dynamic fine-grained CPU and memory allocation to VMs in order to reduce resource costs and avoid application SLA violations. It achieves this by predicting CPU demand for VMs using signal processing and a Markov chain model. Hu et al [5], described an improved version of support vector regression (SVR) algorithm and a Kalman filter for predicting CPU, memory and storage in a cloud so that to ensure QoS requirements. Islam et al. [6] used a 3-layer neural network model and linear regression to predict CPU usage. The models were trained using sampled CPU usage data from an EC2-

based application. A prediction interval of 12 minutes was used because the setup time of VM instances in the cloud was typically around 5−15 minutes. The predicted CPU was used to auto-scale the software. Sotomayor et al [7], considered capacity planning techniques with the assumption that demand knowledge is known through a priori reservation of users. Silva et al. [8] presented a heuristic to optimize the number of machines that should be allocated to process tasks. The proposed heuristic ensures that the minimum required charged time fits in a predefined budget, while obtaining the maximum speedup possible with the allocated machines. For a review on auto-scaling techniques and further references, see Lorido-Botran et al [9].

Jiang et al [10] obtain an optimum cloud configuration using the genetic algorithm for a given a workload characteristic and SLAs. Mao and Humphrey [11] report on a comprehensive study of VM startup time across different cloud providers. The authors show that boot times vary by operating system, provider, and time of day, but in the worst-cases VMs become available for use between 450 and 800 seconds after they are requested. Additionally, they report that VM boot time is independent of the number of virtual machines requested. Jiang et al [12] proposed a method for determining the capacity of a SaaS application, expressed in VMs. For this, they used the ensemble learning method to combine a number of different prediction models in order to estimate the number of arrivals in a unit time, taken to be a day. Customer departures during a unit time are estimated based on the life-span distribution of a VM constructed from historical data over a period of time. Finally, Bouterse and Perros [13] studied several forecasting methods for predicting the required cloud capacity of a SaaS application, expressed in VMs, in the presence of time-varying customer demand.

In this paper, we consider the issue of provisioning dynamically the capacity of a SaaS application, where capacity is expressed in VMs, where each VM can process N customers concurrently. The paper builds on some preliminary work reported in [13]. A periodic-review model is described which determines how many VMs should be provisioned or de-provision in the next inspection interval. For this, the number of arrivals and departures

in the next inspection interval have to be estimated. The arrivals are estimated using different forecasting techniques, and the departures are estimated based on the residual service times of each customer in service. In addition, a simple provisioning heuristic model is also proposed, whereby a fixed reserve capacity of VMs is continuously maintained. We evaluated and compared the performance of these models using historical data from the Virtual Computing Laboratory (VCL) at North Carolina State University [14]. This is a cloud computing platform consisting of around 2,500 blades, and it offers computing services to 42,000 students and faculty who use it for teaching and research. We also evaluated and compared the performance of these models by varying some of the system parameters and in particular the length of the inspection interval. The contributions of the paper are: a) an evaluation of different forecasting models, namely, a moving average model, an exponential moving average model, an autoregressive model, a mixed autoregressive model, and an autoregressive hidden Markov model; b) a new simple heuristic method that is shown to perform very well; and c) an analysis of the impact of the length of the inspection period on the performance of the proposed models.

The paper is organized as follows. In the next section we describe the VM provision model, and in section 3, we describe how the number of departures in an inspection period is calculated. The forecasting models and the heuristic method are presented in section 4. Numerical results and model comparisons are given in section 5. Results from varying the boot time, shutdown time, and service time are presented in section 6. The case of reducing the inspection interval is taken up in sections 7 and 8. Finally, the conclusions are given in section 9.

2. The VM provision model

The SaaS provider requests dynamically VMs from the IaaS provider, who can create an unlimited number of VMs. Each VM contains N *seats*, that is it can process N customers (i.e., requests) concurrently. VCL uses VMs that have two seats, and in view of this we will also assume that $N = 2$. The impact of different values of N is not reported in this paper, but it follows the same trends as described in Bouterse [17]. VMs are requested by the SaaS

provider dynamically as customer demand increases, and they are also released dynamically as demand decreases. The increases and decreases are in integer numbers of VMs. We assume an infinite population of customers.

A VM is assumed to have a 300-second boot time from the time it is requested to the time it becomes available. This assumption is based in the multi-provider boot time study reported in Mao and Humphrey [11], and it includes configuration in addition to booting. A shutdown delay is introduced whereby a VM that is to be de-provisioned is first placed in an inactive state for a fixed period of time and then it is returned back to the IaaS provider. Inactive VMs are available for use immediately. The shutdown delay is introduced to avoid a thrashing behavior whereby resources are released and immediately requested, see Groskinsky et al. [15]. A VM can only be released from active service when all its seats are empty.

When a new customer arrives, it is assigned immediately to a free seat in a VM. The customer is forced to join a queue, served on a FIFO basis, if it arrives at a time when no seats are available. We assume that the software application is controlled by a scheduler which is activated at fixed inspection intervals. Due to the boot time being set to 300 sec, we will first assume that the length of the inspection interval is also 300 sec. The inspection interval will be varied in sections 7 and 8.

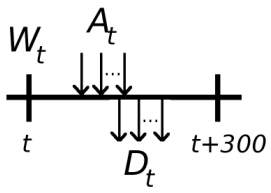


Figure 1: The inspection interval [t, t+300]

Let t be the beginning of the inspection interval [t, t+300], and let A_t and D_t be the number of arrivals and departures (i.e., service completions) during the interval, as shown in figure 1. Also, let W_t be the number of customers waiting for service at time t, the beginning of the inspection interval. At time t the periodic scheduler is activated and it first calculates

the number of customer seats C_t needed for the interval [t, t+300], where $C_t = W_t + A_t - D_t$. Where A_t and D_t are estimated as described below. The number of seats to provision or de-provision, O_t and Q_t respectively, is determined by the current number S_t of empty seats, the number of VMs O_{t-1} that were ordered in the previous inspection interval [t-300, t] and are available at time t, and Q_{t-1} the number of VMs that were taken out of service in the previous inspection interval [t-300, t] but not decommissioned yet. The previously ordered VMs O_{t-1} are put into service at the beginning of the interval [t, t+300], and the Q_{t-1} VMs taken out of service at the previous interval are removed all (or partially) if they are not put back into service. O_t and Q_t are given as follows:

$$O_t = max\left\{\left\lceil\frac{C_t - S_t - N*O_{t-1} - N*Q_{t-1}}{N}\right\rceil, 0\right\} \quad (1)$$

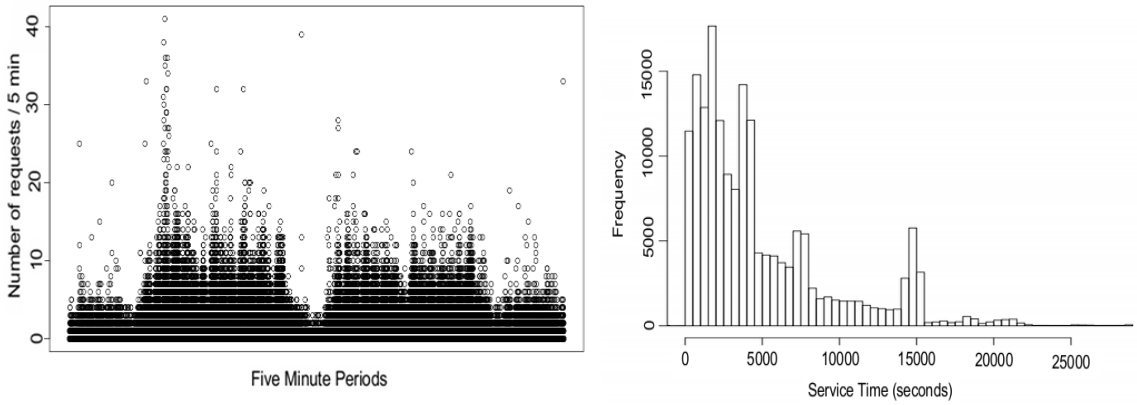
$$Q_t = max\left\{\left\lceil\frac{S_t + N*O_{t-1} - C_t}{N}\right\rceil, 0\right\} \quad (2)$$

As discussed above, in order to provision for each inspection interval, the scheduler has to be able to predict A_t and D_t. In the following section we discuss how D_t is calculated and in the subsequent section 4 we present a number of forecasting models that we used to predict A_t. Numerical results and comparisons of these models are given in section 5.

Performance is expressed in two metrics, the first of which is the customer waiting time. This is the time a customer waits in the queue until it is allocated a seat, and it is expressed in sec. In this paper, we use the 99th percentile of the waiting time as a metric of the waiting time, and the mean waiting time. Percentiles are preferred SLA metrics, as they reflect better the variability of a performance measure than the mean. The percentiles of the waiting time are measured on hourly, daily, and weekly timescales since different timescales are used in a service level agreement. The second performance metric is the VM utilization. Utilization is defined as the proportion of time that the seats of a VM are in use. Utilization is summarized as an average over the period of the evaluation.

The forecasting models used to predict the number of arrivals A_t in the next inspection interval [t, t+300], are trained and tested using VCL data for an entire year. All

service times longer than or equal to 8 hours were removed resulting to a total of 175,554 requests for service, i.e. customers. Figure 2 shows the number of arrivals per 300 sec for the entire year starting on July 1, 2008 and the histogram of the service times (sec). The models were trained on the first six months and the testing was done on the remaining six months. This division does not introduce any seasonal biases since both the training and evaluation periods include an equal quantity of active semesters, summer semesters, exams, and break periods. The VM provision model with the forecasting models used to predict the number of arrivals A_t were implemented in a simulation, which was used to obtain the waiting time and utilization performance metrics.



a) Number of requests per 300 sec b) Service time histogram (sec)

Figure 2: VCL data

3. Estimating the Departure Process D_t

We estimate the average number of departures during an inspection period by determining the probability that each customer in service will depart during the inspection period. For this, we need to know the service time distribution of the customers. To adhere to the methodology of the training and testing sets, we only used the training data to construct a histogram of the VCL service times. In addition, we also fitted a theoretical distribution to this histogram which allow us to modify its parameters in order to obtain service time distributions other than that of VCL. After experimenting with a variety of distributions, we

selected a Gamma distribution with parameters α=1.1401, β=4158.3 (χ^2 = 42838.8, with a *p*-value of 2.774E-9037).

To estimate the number of departures, D_t, during an inspection interval [t, t+300], we sum the probability that each customer in service will depart within the next 300 seconds, given that it was in service at time t. This probability is equal to $F(s + 300) - F(s)$, where $F(s)$ is the CDF of the service time distribution and s is the amount of service the customer has received until time t. This technique is computationally intensive, since it has to be done every 300 sec of simulation time for all the customers in service, and in view of this, the quantity $F(s + 300) - F(s)$ was pre-computed for discrete values of s.

Below, we only give the simulation results for the Gamma distribution, since they are almost identical to those obtained using the actual histogram.

4. Predicting the number of arrivals A_t

In this section, we present five different forecasting models for predicting the number of arrivals A_t during an inspection interval [t, t+300]. These are: a moving average model, an exponential moving average model, an autoregressive model, a mixed autoregressive model, and an autoregressive hidden Markov model. We also present a simple provisioning heuristic model whereby a fixed reserve capacity of VMs is continuously maintained. Each model was trained using the training data, i.e., the first six months of the VCL data. The model was then used in the VM provision model, described above, which was simulated using the test data, i.e., the second six months of the VCL data. We note that an estimate of A_t is denoted as $\hat{A}_t$. We now proceed to present the parameters of the forecasting models.

a. The pre-known demand model.

In addition to the forecasting models, we also ran the VM provision model using the actual trace of the VCL arrivals and corresponding service times. We refer to this model as the *pre-known demand* model. This model operates with $\hat{A}_t = A_t$, and is used as a base line model for comparison with the other models. The results of simulating the VM provision model with the actual VCL trace are given in table 4.

b. Moving average model

The moving average model is defined as follows: $\hat{A}_t = (A_{t-1} + A_{t-2} + \ldots + A_{t-k}) / k$, where $\hat{A}_t$ is the predicted number of arrivals in the [t, t+300] inspection interval. The model was trained using the training data set by varying k and keeping track of the SSE, the sum of the squared differences of the number of actual arrivals minus the estimated one. The lowest SSE was observed when k = 12. The results of simulating the VM provision model with the moving average model are given in table 4.

c. Exponential moving average model

The exponential moving average model is as follows:

$$\begin{cases} \hat{A}_0 = 0 \\ \hat{A}_t = aA_{t-1} + (1-a)\hat{A}_{t-1} = \hat{A}_{t-1} + a\left(A_{t-1} - \hat{A}_{t-1}\right), \qquad t > 0. \end{cases}$$

where A_{t-1} is the actual number of customers who arrived during the [t - *300*, t) interval, and $\hat{A}_t$ is the predicted number of arrivals during the period [t, t + 300). The model was trained using the training data, and the value of α = 0.13 was selected since it minimizes SSE. The results of simulating the VM provision model with the exponential moving average model are given in table 4.

d. Autoregressive model

A q order autoregressive model with parameters c, $\{\varphi_1,\varphi_2,\ldots,\varphi_q\}$, and ε_t is defined as follows:

$$\hat{A}_t = c + \sum_{i=1}^{q} \varphi_i A_{t-1} + \varepsilon_t$$

where $\hat{A}_t$ is the estimated value for the [t, t+300] interval, and $\varepsilon_t \sim N(0, \sigma^2)$. Using the Yule-Walker method on the training set, an AR(2), i.e. q=2, with c=0 was obtained with parameters φ_1=0.4258, φ_2=0.3435, and σ^2=3.372. Based on experimentation, increasing the modeling order beyond 2 yields only marginal improvements. The results of simulating the VM provision model with the autoregressive model are given in table 4.

Time Period	C	φ_1	φ_2	σ^2
Fall/Spring Classes	0	0.4089	0.3241	4.967
Summer Sessions	0	0.2523	0.1747	0.7864
Exams	0	0.2549	0.2411	1.18

Table 1: c, φ_1, φ_2, and σ^2 values for three different calendar periods

e. Mixed autoregressive model

A mixed autoregressive model is used whereby three autoregressive models are trained corresponding to a) Fall and Spring semesters, b) Summer sessions 1 and 2, and c) exam calendar periods. These three models together are referred to as the *mixed autoregressive* model and are used according to the academic calendar. Three autoregressive models of order 2 have their parameters optimized using the Yule-Walker method for each of these three calendar periods using the training data. The model parameters c, φ_1, φ_2, and σ^2 for each period are given in table 1. The results of simulating the VM provision model with the mixed autoregressive model are given in table 4.

i	j	$p_{i,j}$	p-value
0	0	0.9583615	0
0	1	0.0001809	0.585
0	2	0.0414576	0
1	0	0.0117296	0
1	1	0.7786373	0
1	2	0.2096332	0
2	0	0.0421590	0
2	1	0.2244591	0
2	2	0.7333819	0

Table 2: Branching probabilities $p_{i,j}$ with p-values.

f. Autoregressive hidden Markov model

An autoregressive hidden Markov model AR(q)-HMM(M) was also developed to predict the number of arrivals during each 300-second interval, where q is the order of the autoregressive model and M is the number of hidden states. Estimation of the model parameters of a given AR(q)-HMM(M) was done using Grocer [16], which estimates parameters using a maximum likelihood method based on a Kittagawa-Hamilton filter. This estimation requires the user to pre-select q and M, and training of the remaining model parameters is done on the first six months of data.

A variety of models with different q and M values were tested and the selection of the best model was based on the p-values of the estimated parameters, which indicates their statistical significance, and the SSE values. For brevity, an in-depth presentation of each model is not given, but two important observations are noted. First, using 5 hidden states revealed that several states had autoregressive coefficients which were very similar to each other, for the same autoregressive order q. Consequently, we settled for M=3. Second, given M=3, using too small or too large of a value of q increased the SSE; selecting q = 5 produces a model that demonstrates less prediction error than models with other values for q. These two observations together indicated that an AR(5)-HMM(3) model was the best. Tables 2 and 3 give the estimated parameters for this model.

	State 1		**State 2**		**State 3**	
Parameter	**Coefficient**	***p*-value**	**Coefficient**	***p*-value**	**Coefficient**	***p*-value**
AR(-1)	0.1863004	0	0.4107128	0	-0.3574271	0
AR(-2)	0.1273634	0	-3.307E-11	0.8764984	0.0491597	0
AR(-3)	0.1388058	0	3.572E-11	0.8642294	0.0518409	0
AR(-4)	0.1197142	0	2.503E-12	0.9904083	0.0574256	0
AR(-5)	0.1217175	0	-3.806E-11	0.8541158	0.0702153	0
Constant	1.2427012	0	2.266E-11	0.8761246	0.9607062	0
Error variance	5.0576163	0	2.221E-16	0.9999966	0.4464271	0

Table 3: AR coefficients and *p*-values for each state

We note that the *p*-values of all estimated parameters in table 3 are zero, except for p_{01} whose values is zero for all practical purposes. Likewise table 4 has all zero *p*-values except for state 2, where non-zero *p*-values have corresponding coefficients that are effectively zero anyway. The state 1 autoregressive model has all positive coefficients and a constant of 1.24 and it can be seen as a growth state. The state 2 autoregressive model is for all practical purposes a lag-1 model with a coefficient of 0.4 and zero constant and error variance, and it can be seen as a state of declining arrivals. The state 3 autoregressive model has a negative dependency on the lag-1 value, a weak positive dependency on the other lags and a constant close to 1, and it can be seen as a state of random fluctuation of arrivals. Finally, we note that the steady-state probability of being in state 1, 2, and 3 is 0.3918647, 0.3063385, 0.3017968, respectively. That is, the system spends about one third of the time is each state, indicating a strong influence of the states on the model.

The trained AR(5)-HMM(3) model was used in the provision model which, as in the above models, was simulated using the test data. For this we needed to know the state of the HMM at the first 300-second period of the test data, which can be obtained by applying Viterbi's algorithm on the training data. For simplicity, we assumed that the initial state was state 1. (Through experimentation, the starting state does not have any impact on the results due to the long estimation period.) The results of simulating the VM provision model with the AR(5)-HMM(3) autoregressive model are given in table 4.

g. Fixed reserve capacity model

In addition to the above models, we have also used a simple heuristic model. We recall that at time *t*, we estimate the required number of seats $C_t = W_t + A_t - D_t$ and then we either order O_t or transition Q_t VMs out-of-service per expressions (1) and (2) respectively. The idea behind this model is that the number of virtual machines is scaled such that *R* unused virtual machines will be available. This is equivalent to always planning for *R* arrivals and is effectively $\hat{A}_t = R$ which causes *R* seats to be predicted as needed for all $t \geq 0$. This model

leaves headroom for new customers by assuming R customers will arrive in each interval. We refer to this model as the *fixed reserve capacity model*.

The best value was obtained by simulating the provision model with the fixed reserve capacity heuristic using the training data and letting R take values in the integer set [1, 50]. The lowest SSE was observed when $R = 2$. The results of simulating the VM provision model with the fixed reserve capacity model are shown in table 4.

5. Results and model comparisons

The simulation results from all of the above models are summarized in table 4. The results given are: the average utilization of the VMs computed over the six-month period, and the 99^{th} percentile of hourly, daily and weekly waiting time of a customer. The waiting time is the time elapsing from the instance a customer arrives at the SaaS system to the instance it is allocated to a seat. The hourly 99^{th} percentile is computed by first computing the 99^{th} percentile of all the waiting times for each hour, and then take the average of all these 99^{th} percentiles. The other percentiles are similarly computed. The average waiting time, though not a performance metric of interest, is also given so that it can be contrasted with the percentile results.

Model	**Average utilization**	**Waiting time 99th percentile per hour**	**Waiting time 99th percentile per day**	**Waiting time 99th percentile per week**	**Average waiting time**
Pre-known demand	0.8662	164.95	43.12	24.81	15.07
Moving Average	0.8641	380.21	75.08	50.71	33.89
Exponential moving average	0.8623	195.13	74.22	48.88	32.74
Auto-Regressive	0.8650	447.26	101.53	67.07	35.34
Mixed autoregressive	0.8676	428.63	105.23	69.80	38.84
AR(5)-HMM(3)	0.8643	257.21	86.96	59.51	47.39
Fixed reserve capacity	0.8599	179.78	74.47	55.29	43.59

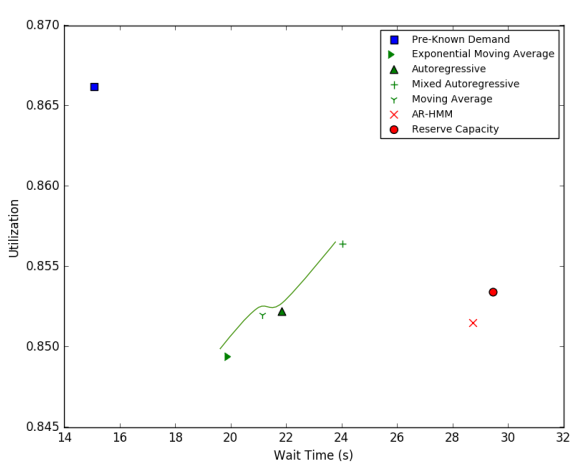
Table 4: Summary of results

The performance of the provision models presented above is measured in terms of waiting time and utilization. We note that shortening the timescale causes the 99th waiting time percentile to increase. This means that we need to allocate more resources if we want to satisfy a given 99th percentile constraint on shorter timescales. In the case of the hourly timescale, there are typically a few observations within an hour, which means that the 99th percentile is most likely the highest observed value. This is not the case for the larger timescales, where typically there are a sufficient number of observations and therefore the 99th percentile is not the largest observed value.

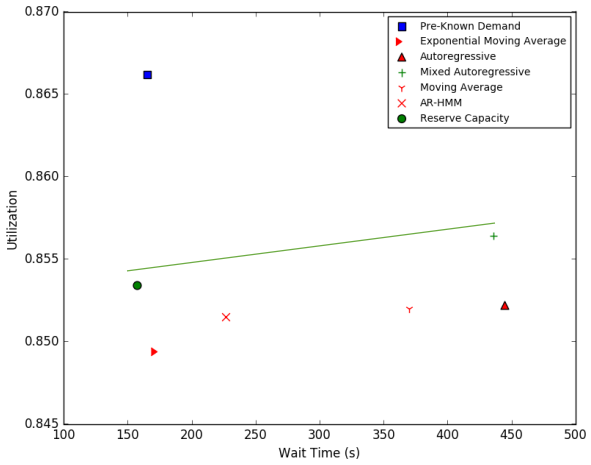
We note that the utilization across all models is about the same. (This is due to the fact that all models for predicting A_t have approximately the same mean.) Consequently, we can compare the models using only the waiting time percentiles. In this case, depending on the time scales we have different winners. Specifically, the fixed reserve capacity model clearly outperforms all models for the hourly 99th percentile with the exponential moving average model coming second. The exponential moving average and the fixed reserve capacity models give the best results for the daily 99th percentile, and for the weekly 99th percentile, the exponential moving average model performs the best.

A 1% drop in the utilization calculated over the evaluation period corresponds to about 700 VM hours. The cost of renting these additional VM hours with current level prices is not significant, but it may be significant at the aggregate level if the service provider offers other SaaS applications. In view of this, we carried out a comparison using both the waiting time and utilization metrics. In the absence of additional subjective preferences of the service provider regarding the relative importance of these two metrics, we compare these models using a Pareto front, as shown in figure 3. The pre-known demand model is indicated in blue because it is not a candidate model for evaluation. The Pareto front is drawn in green and the Pareto optimal models are colored in green as well while Pareto sub-optimal models are colored in red. We see that for the hourly 99th percentile of the waiting time, the Pareto optimal models are the fixed reserve capacity and mixed autoregressive models. For the daily 99th percentile of the waiting time, the exponential moving average, the fixed reserve

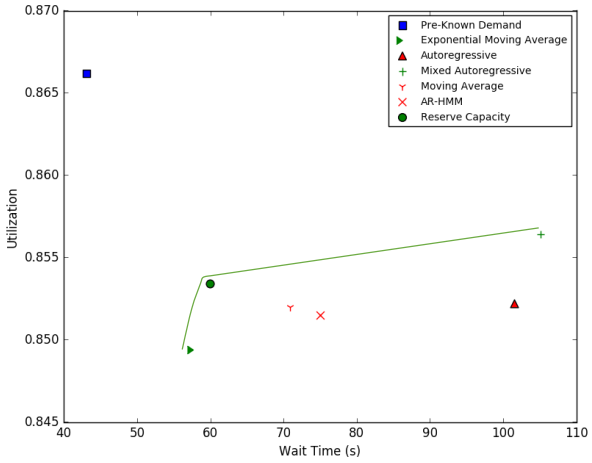
capacity, and mixed autoregressive models are Pareto optimal. Finally, for the weekly 99th percentile of the waiting time, the fixed reserve capacity and mixed autoregressive models are Pareto optimal. In general, the reserve capacity model is Pareto optimal for all the three timescales.



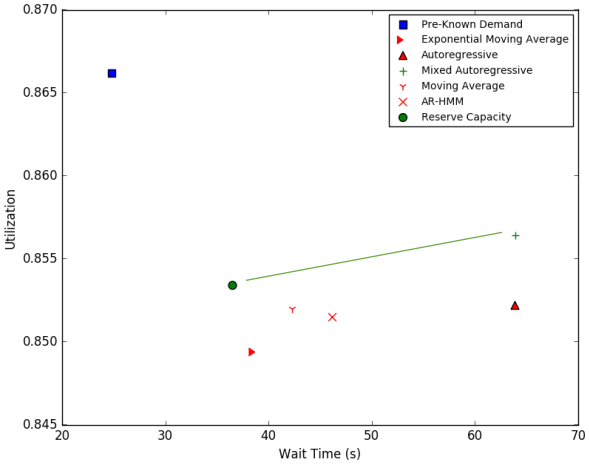
a) Average waiting time vs utilization



b) Percentile waiting time per hour vs utilization



c) Percentile waiting time per day vs utilization



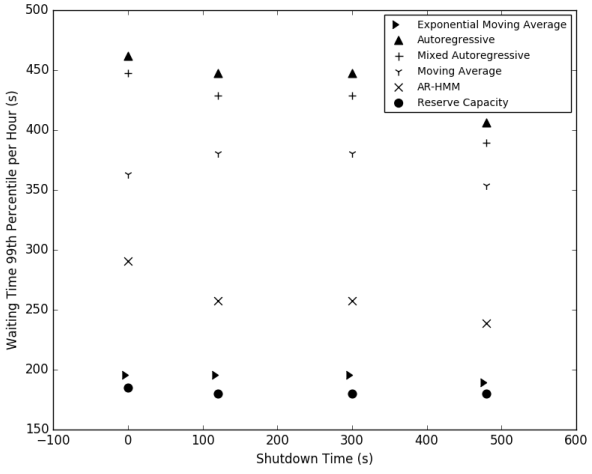
d) Percentile waiting time per week vs utilization

Figure 3: Pareto optimality for all models

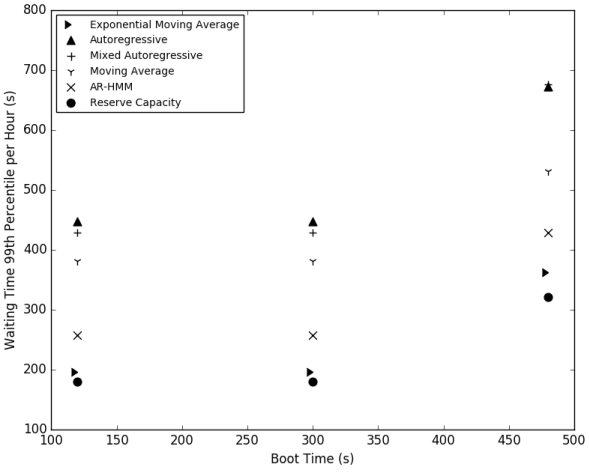
6. Varying the boot time, shutdown time, and service time

We first examine the impact of varying the boot and shutdown times on the waiting time and utilization, assuming that the rest of the model parameters remain the same. Previously, we assumed that these two times are equal to 300 sec, which is also the length of the inspection interval. We allowed the boot time to take the values {120, 300, 480} sec and the shutdown time the values {0, 120, 300, 480} sec. A shutdown time of 0 causes a VM to be deleted immediately if it is determined that it is no longer needed. A boot time of 0 represents the case where a requested VM becomes available for service immediately. VMs take time to become ready so a boot time value of 0 is unrealistic. (We are not considering the case, where a system is over-provisioned with a large number of VMs so that there is always a free seat available whenever a customer arrives. Such a system will have a zero waiting time, but a very low utilization.)

In the interest of space, we only present results for the 99th percentile of the waiting time per hour for all the forecasting models and the fixed reserve capacity model. The remaining metrics for the waiting time follow the same pattern. The average utilization did not vary much, and therefore it is not given here. These results are summarized in figure 4. Specifically, in figure 4a we vary the shutdown time and keep the boot time fixed to 300 sec, and in figure 4b, we vary the boot time and keep the shutdown time fixed to 300 sec.



a) Varying shutdown time, boot time = 300 sec



b) Varying boot time, shutdown time = 300 sec

Figure 4: Varying boot time and shutdown delay for all models

We first note that the fixed reserve capacity and the exponential moving average models produce the lowest 99th percentile waiting times, which is consistent with the results in the previous section. We also note that there is a stepwise performance based on the boot time or the shutdown time falling within the regions, (0, 300], and (300, 600], or the shutdown time being 0. The value 0 is only used by the shutdown time and it represents its own performance region where VMs are deleted immediately. The stepwise performance behavior is due to the fact that the scheduler is activated every 300 sec.

A boot time change that stays within the range (0, 300] or (300, 600] causes no change in the waiting time metrics because the VM is placed into service when the scheduler is activated at the beginning of the next inspection interval. For example, a VM with a boot time of 120 sec is put into service 300 sec after it is requested, at the beginning of the next inspection interval. Consequently, any variations within the range (0, 300] will not affect the waiting time metrics. However, increasing the boot time from within the range (0, 300] to (300, 600] and thereafter, significantly increases the waiting time metrics. The same holds for the shutdown time. For example, a shutdown time of 200 sec causes the VM to be deleted at 300 sec after it is requested due to the fact that VMs can only be de-provisioned when the scheduler is activated at the beginning of the next inspection interval. On the other hand, increasing the shutdown time so that it changes range, for instance from 0 to (0, 300], or from (0, 300] to (300, 600] and thereafter, will increase the waiting time, but experimentation shows that this increase is very small.

We now examine the impact of reducing the customer service time on the waiting time and utilization. The mean service time considered so far was taken from the VCL data and it is 4957 sec or 1.377 hours, which is significantly larger than the periodic review time of 300 sec. Here, we consider the opposite case where the mean service time is smaller than the 300 sec inspection period. Specifically, we assume that the service times follow a Gamma distribution with $\alpha = 24$ and $\beta = 5$ that correspond to a mean service time of 120 sec and a variance of 600 sec. The mean of 120 sec ensures that most of the customers will get served

within a single inspection period, and the variance of 600 allows for some variation of this scenario. The remaining model parameters and arrival process are the same as before.

Model	**Average utilization**	**Waiting time 99th percentile per hour**	**Waiting time 99th percentile per day**	**Waiting time 99th percentile per week**	**Average waiting time**
Moving Average	0.2195	494.21	67.05	30.73	8.4
Exponential Moving Average	0.2174	70.94	42.37	20.19	7.04
Auto-Regressive	0.2321	564.21	285.23	167	31.4
Mixed autoregressive	0.2435	555.21	281.86	162.25	31.12
AR(5)-HMM(3)	0.2169	430.47	117.49	79.05	30.01
Fixed reserve Capacity	0.1979	58.86	29.83	16.71	11.3

Table 5: Summary of the results for the reduced service time

A summary of the results is given in table 5. We note that the utilization for all models is significantly lower than the utilization reported in table 4. This is because customers complete their service within a periodic review interval thus giving rise to more idle VMs which cannot be taken out of service until the end of the periodic review interval. The fixed reserve capacity model is the best overall performer with the exponential moving average model being very close. For these two models, we observe a reduction in the percentile and mean waiting times relative to the waiting times of table 4. The remaining models which include the moving average, autoregressive, mixed-autoregressive, and AR(5)-HMM(3) all show a reduction in mean waiting time relative to table 4, but they each have higher percentile waiting time for all timescales.

Similar results (not reported in the paper in the interest of space) were obtained for other reduced services. Low volume traffic environments and reduced service times coupled with the fact that some of the forecasting models take into account only recent information, cause the increases in percentile waiting times of table 5 for the moving average, autoregressive, mixed-autoregressive, and AR(5)-HMM(3) relative to table 4. The moving

average model, autoregressive model, and mixed autoregressive model each predict $\hat{A}_t$ from two previous actual arrival observations A_{t-1} and A_{t-2}, which is 10 minutes of wall clock time. The AR(5)-HMM(3) estimates $\hat{A}_t$ from five previous, actual arrival observations A_{t-1}, A_{t-2}, A_{t-3}, A_{t-4}, A_{t-5} , which is 25 minutes of wall clock time. The percent of time the test data has periods where no arrivals occur in 10 or 25 minutes is 5.6% and 1.6% respectively, which causes $\hat{A}_t$ to be 0 during those portions of time. The reduced service times cause the probability P($W_t = 0$) to be effectively 1 as customers depart quickly allowing new arrivals to be served. With $\hat{A}_t = 0$ and $W_t = 0$ the autoregressive, mixed-autoregressive, AR(5)-HMM(3) and moving average models scale down the number of VMs to 0. A customer that arrives when there are 0 VMs, has to wait up to 300 seconds for the next periodic review time and then waits another 300 seconds for the VMs to become available for a total worst case waiting time of 600 seconds. This gives rise to extreme values that influence the 99th percentile. As the timescale increases, the 99th percentile decreases for the reasons given in section 4. In the case of the original service times reported in table 4, the probability P($W_t >$ 0) is larger than in the case of reduced service times due to seats not becoming available through customer departures. This results in the overall cluster size being larger and thus reducing the waiting time percentiles through the availability of more resources.

On the other hand, the exponential moving average and the fixed reserve capacity model produce positive estimates of $\hat{A}_t$, i.e. $\hat{A}_t > 0$, during low volume traffic conditions, which causes a decrease in the waiting time in table 5 relative to table 4. This is because the exponential moving average model incorporates all previous A_t values with exponentially decreasing coefficients for older observations. This creates a hysteresis whereby low volume traffic conditions still predict $\hat{A}_t > 0$ due to medium or high volume traffic conditions having been seen before. The fixed reserve capacity model always predicts $\hat{A}_t = R$ in all types of traffic environments. Non zero $\hat{A}_t$ predictions during low volume traffic environments maintain VMs provisioned, which allow waiting times to decrease when service times are reduced.

7. Decreasing the inspection interval

As shown in table 4, the case of the pre-known demand does not produce a zero waiting time and 100% utilization, due to the periodic ordering and placement into or removal from service of VMs every 300 sec. In this section, we examine the case of reducing the inspection interval such that the scheduler can run multiple times within a 300 second boot time. In order to avoid partial inspection intervals within the 300 sec boot time, the inspection interval d was set to d = 150, 100, 75, 60, 50, 30, 25, 30, 15, 12, and 10.

Resources requested at time t will not become available until time t + 300. In view of this, even though the inspection interval is shorter than 300 seconds, planning must occur for the entire interval [t, t+300), that is over n successive inspection intervals of length d, where n = 300 / d. This approach is referred to as the n-step prediction. Let $\hat{A}_t$, $\hat{A}_{t+d}$, … , $\hat{A}_{t+(n-1)d}$ be the predicted number of arrivals in each of the n inspection intervals obtained using one of the arrival prediction models**,** where t is the beginning of the first inspection interval. Then, the predicted number of arrivals for the entire 300-second period is the sum $\hat{A}_t + \hat{A}_{t+d} + \ldots + \hat{A}_{t+(n-1)d}$. Based on the total predicted number of arrivals, we estimate the number of required seats for the entire 300-second period, which is then used to calculate O_t and Q_t given by (1) and (2). Because of the short inspection interval, ordered VMs over several previous inspection intervals may not have arrived yet. In view of this, O_{t-1} in expressions (1) and (2) is adjusted to indicate all the ordered, but not yet delivered, VMs. Likewise, Q_{t-1} is altered to indicate all VMs that have been taken out of service but are not deleted yet.

Time t is moved forward by d to the beginning of the next inspection interval, and the same analysis is carried out again. That is, we predict the number of arrivals for the next n inspection intervals starting from t+d, which now extends to include the new inspection interval [t+$(n-1)d$, t+nd], and then calculate O_t and Q_t as described above. This process continues in this manner for the entire length of the simulation.

Because of this n-step prediction, the arrival models have to be re-trained so that to

predict the number of arrivals for each inspection interval d = 150, 100, 75, 60, 50, 30, 25, 20, 15, 12, 10 sec. As described in section 4, model parameter training is done using the first six-months of the data, and the simulation results are based on the second six months of the data. During training, the n-step SSE is computed as the squared difference of the ceiling of the n-step prediction and the corresponding number of the actual arrivals over the same n inspection intervals. (The ceiling is used in order to closely approximate the Q_t and O_t expressions used by the scheduler, which both take the ceiling given that partial VMs cannot be ordered.)

Inspection interval (sec)	***k***	**Average utilization**	**Waiting time 99th percentile per hour**	**Waiting time 99th percentile per day**	**Waiting time 99th percentile per week**	**Average waiting time**
300	12	0.8640	380.21	75.08	50.70	33.89
150	37	0.8689	222.27	58.15	42.39	26.44
100	49	0.8695	249.41	51.29	36.74	23.15
75	50	0.8698	316.21	51.59	32.57	20.58
60	50	0.8690	318.00	47.93	31.28	19.02
50	50	0.8692	312.63	48.36	30.11	17.97
30	50	0.8667	315.21	53.09	29.81	15.47
25	50	0.8663	310.00	52.53	31.85	14.76
20	50	0.8654	309.00	52.14	34.36	13.87
15	50	0.8631	307.00	55.11	39.23	13.09
12	50	0.8600	306.00	58.26	42.68	11.99
10	50	0.8569	305.00	65.36	44.74	11.25

Table 6: Summary of results - n-step moving average

8. Numerical results and model comparison

In this section we provide numerical results and model comparisons for a subset of the models, specifically, the moving average model, the exponential moving average model, the autoregressive model, and the fixed reserve capacity models. The mixed autoregressive model is not considered, as it is very similar to the autoregressive model, and the AR(5)-HMM(3) consistently showed poor performance relative to other models.

We note that in the results given, we have included the case of $d = 300$ for comparison purposes. Also, the average waiting time, although not a performance metric of interest, is also given for comparison purposes. As in section 5, we use Pareto curves to compare all the models. (In the interest of space we do not show the Pareto curves for each individual model.)

a. n-step moving average model

For each inspection interval d, the k value that minimizes the n-step SSE is first determined, and then the VM provision model is simulated using the test data. The results are given in table 6. Different inspection intervals with different k values are more Pareto efficient for different waiting time percentiles. For instance, the 150-second inspection interval with k = 37 minimizes waiting time for the 99th percentile per hour. Likewise, the 60-second inspection interval with $k = 50$ minimizes waiting time for the 99th percentile per day.

Inspection interval (sec)	***α***	**Average utilization**	**Waiting time 99th percentile per hour**	**Waiting time 99th percentile per day**	**Waiting time 99th percentile per week**	**Average waiting time**
300	0.13	0.8623	195.13	74.22	48.88	32.74
150	0.01	0.8709	172.46	68.87	47.86	33.32
100	0.01	0.8718	153.98	55.98	40.93	28.31
75	0.01	0.8708	134.10	50.65	35.72	24.76

60	0.01	0.8703	131.69	47.12	33.69	22.82
50	0.01	0.8702	124.81	46.06	32.46	21.57
30	0.01	0.8691	113.76	42.01	28.50	18.12
25	0.01	0.8685	110.20	43.02	27.18	17.14
20	0.01	0.8679	107.81	40.93	25.82	16.11
15	0.01	0.8666	112.92	39.12	25.43	14.92
12	0.01	0.8658	130.32	36.66	24.48	14.15
10	0.01	0.8654	141.81	39.24	23.74	13.51

Table 7: Summary of results - *n*-step exponential moving average

b. n-step *exponential moving average model*

For each inspection interval d, the value of α, where $0.01 \leq \alpha \leq 1$, that minimizes the n-step SSE is first determined. The optimum value of α along with the simulation results are given in table 7. For the average case and all percentile timescales explored, a Pareto frontier is formed from periodic review times where $20 \leq d \leq 100$. Periodic review times of $d < 20$ are on the Pareto frontier for select percentile timescales. The periodic review times $d \geq 150$ sec is suboptimal. The waiting time percentiles show a Pareto front which is expanded but not outperformed with each decrease in periodic review times. Exploring smaller periodic review times is not expected to yield performance that is more Pareto efficient than the review times and α values already explored.

Insp. intervl (sec)	φ_1	φ_2	**Average utilizati on**	**Waiting time 99th percentile per hour**	**Waiting time 99th percentile per day**	**Waiting time 99th percentile per week**	**Average waiting time**
300	0.3345	0.4095	0.8661	455.47	101.53	66.82	36.40
150	0.2968	0.3311	0.8661	370.21	89.13	62.49	29.73
100	0.2573	0.2863	0.8748	350.42	83.87	60.07	31.40
75	0.2243	0.2555	0.8789	341.05	82.39	58.22	35.54
60	0.2053	0.2269	0.8814	326.00	79.65	57.52	39.33
50	0.1856	0.2068	0.8836	321.21	82.53	60.95	43.03
30	0.1406	0.1507	0.8874	317.00	78.85	61.01	51.91

25	0.124	0.1326	0.8903	311.21	80.05	62.58	54.23
20	0.1059	0.1142	0.8903	310.00	82.78	63.93	56.89
15	0.0848	0.0932	0.8918	307.00	88.52	66.04	58.78
12	0.0708	0.0809	0.8931	306.00	90.42	67.73	59.94
10	0.0618	0.0696	0.8922	305.00	92.09	67.08	59.99

Table 8: Summary of results - *n*-step autoregressive model

c. n-step autoregressive model

The values for φ_1 and φ_2 are obtained as before for each inspection interval. Table 8 gives a summary of the results for the *n*-step autoregressive model. We first observe that φ_1 and φ_2 decrease as the inspection interval decreases. This is because as the expected number of arrivals per inspection interval also decreases, which causes the coefficients to decrease as well. The Pareto front for each 99th percentile includes most points where $d \leq 60$. As above, the trend suggests that smaller inspection intervals may extend the Pareto frontier, but they will not produce performance that is more Pareto optimal than the existing points.

d. n-step fixed reserve capacity model

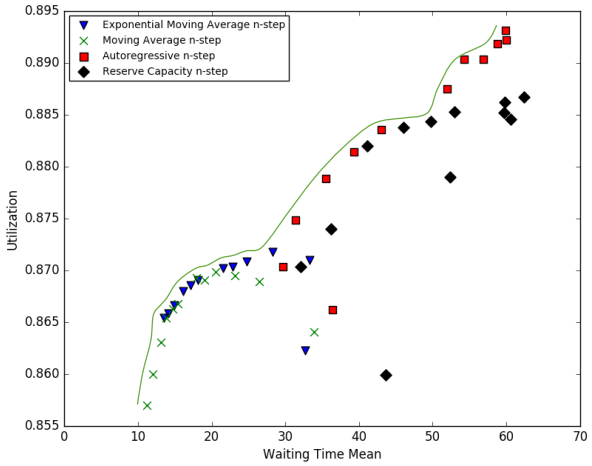
For each inspection interval, the *R* value that minimizes the *n*-step SSE is first determined, and then the simulation is run against the second six months. The results are summarized on table 9.

Insp. intervl (sec)	***R***	**Average Utilization**	**Waiting time 99th percentile per hour**	**Waiting time 99th percentile per day**	**Waiting time 99th percentile per week**	**Average waiting time**
300	2	0.8599	179.78	74.47	55.29	43.59
150	0.5	0.8790	187.58	79.37	64.46	52.38
100	0.15	0.8845	195.78	85.24	69.70	60.59
75	0.1	0.8852	193.27	83.22	69.87	59.73
60	0.05	0.8867	195.55	84.87	72.52	62.43
50	0.05	0.8862	188.09	81.47	68.27	59.84
30	0.05	0.8852	168.04	74.06	59.95	53.00
25	0.05	0.8843	166.20	72.27	57.97	49.72
20	0.05	0.8838	155.34	68.23	54.71	46.03
15	0.05	0.8820	142.85	67.85	51.26	41.15

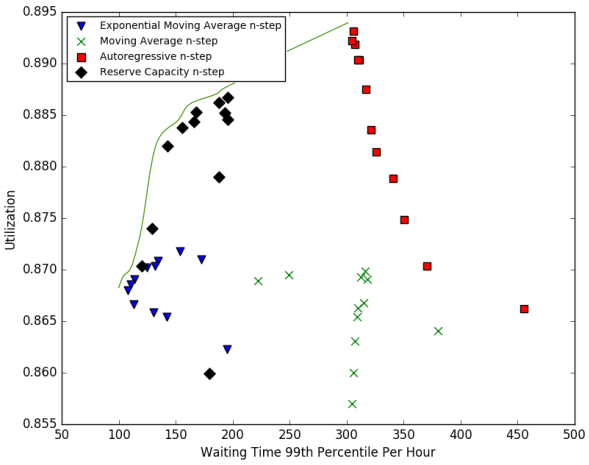
12	0.05	0.8740	129.38	58.74	45.90	36.20
10	0.05	0.8703	120.27	53.03	40.00	32.13

Table 9: Summary of results - *n*-step reserve capacity model

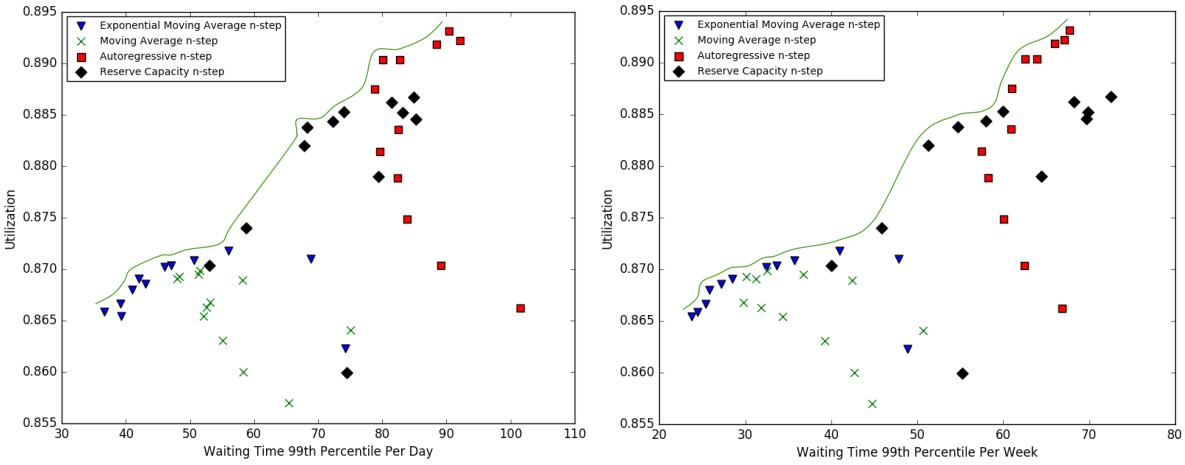
For all 99th percentile timescales, the Pareto frontier is comprised of inspection intervals with $d \leq 60$. As in the above models, further reduction of the inspection intervals will likely add points to the Pareto frontier in the area of lower utilization with lower waiting times, but these new points are not expected to outperform the existing Pareto frontier. Finally, similar to the parameters of the autoregressive model, as the inspection interval decreases, the optimized values of R also decrease. As d decreases, the traffic within 300 seconds is spread over more periodic review intervals, which causes the overall traffic per interval to decrease As such, it is expected that R decrease as n is increases.



a) Average waiting time vs utilization



b) Percentile waiting time per hour vs utilization



c) Percentile waiting time per day vs utilization

d) Percentile waiting time per week vs utilization

Figure 5: Pareto optimality for the *n*-step models

e. Comparison of the n-step model

Figure 5 shows the Pareto optimality for the *n*-step models for all the reported inspection intervals. We note that for the percentile timescales the exponential moving average *n*-step model and the reserve capacity *n*-step model occupy almost the entire Pareto frontier and therefore are the highest performers. This is consistent with the conclusions from the 1-step model comparison in section 6. For the average waiting time case in figure 4a, the exponential moving average, moving average, and autoregressive *n*-step models define the Pareto frontier.

9. Conclusions

In this paper, we described a periodic-review model for provisioning VMs for a cloud-based software used as a SaaS, where the arrival of customers varies over time. The provision model requires knowledge of the number of arrivals within each inspection period, and for this we developed and compared the following forecasting models: a moving average model, an exponential moving average model, an autoregressive model, a mixed autoregressive model, and an autoregressive hidden Markov model. In addition, we have also proposed the fixed reserve capacity model, a simple heuristic whereby at the beginning of each inspection

interval, the required number of available VMs is adjusted up or down so that *R* seats are always available. We trained and compared these models for a variety of system parameters using data from the Virtual Computing Laboratory (VCL) at North Carolina State University, a cloud-based computing service to 42,000 students and faculty who use it to run applications for teaching and research. We concluded that the exponential moving average and the fixed reserve capacity models are generally the highest performing. We also conclude that evaluating system capacity more regularly produces generally higher performance but eventually performance gains stop with further reduction in periodic review interval times.

The applicability of the models was tested by varying the main parameters of a SaaS environment, namely, the service time, boot time and shutdown time. Obviously, it is not clear how these models will perform given a different demand trace. However, it is expected that the methodology developed in this paper should be applicable even if the conclusions maybe different.

References

[1] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood, "Agile dynamic provisioning of multi-tier internet applications," *ACM Trans. Auton. Adapt. Syst.*, vol. 3, no. 1, 2008.

[2] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*, Washington, DC, USA, 2011, pp. 500–507

[3] D. Minarolli and B. Freisleben, "Cross-correlation prediction of resource demand for virtual machine resource allocation in clouds", *Computational Intelligence, Communication Systems and Networks* (CICSyN), 2014 Sixth International Conference, 27-29 May 2014

[4] Z. Gong, X. Gu, and J. Wilkes, "Press: Predictive elastic resource scaling for cloud systems", *International Conference on Network and Service Management,* pp 9–16. IEEE Press, 2010.

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

[6] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," *Future Gener. Comput. Syst.*, vol. 28, no. 1, pp. 155–162, 2012.

[7] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Capacity leasing in cloud systems using the opennebula engine." *Workshop on Cloud Computing and its Applications 2008 (CCA08)*, October 22-23, 2008,

[8] J. N. Silva, L. Veiga, and P. Ferreira, "Heuristic for resources allocation on utility computing infrastructures," in *Proceedings of the 6th international workshop on Middleware for grid computing*, 2008.

[9] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A Review of auto-scaling techniques for elastic applications in cloud environments", *Journal of Grid Computing*, pp 1–34, 2014.

[10] J. Jiang, J. Lu, and G. Zhang, "An innovative self-adaptive configuration optimization system in cloud computing", *Dependable, Autonomic and Secure Computing* (DASC), 2011 IEEE Ninth International Conference, 12-14 Dec. 2011, pp 621 – 627.

[11] M. Mao and M. Humphrey, "A performance study on the VM startup time in the cloud," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 2012, pp. 423–430

[12] Y. Jiang, C.-S. Perng, T. Li, and R. Chang. "Intelligent cloud capacity management", *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2012.

[13] B. Bouterse and H. Perros, "Scheduling cloud capacity for time-varying customer demand," in *Cloud Networking (CLOUDNET), 2012 IEEE 1st International Conference on*, 2012, pp. 137–142.

[14] H. E. Schaffer, S. F. Averitt, M. I. Hoit, A. Peeler, E. D. Sills, and M. A. Vouk, "NCSU's virtual computing lab: a cloud computing solution," *Computer*, vol. 42, no. 7, pp. 94–97, 2009.

[15] B. Groskinsky, D. Medhi and D. Tipper, "An investigation of adaptive capacity control schemes in a dynamic traffic environment," *IEICE Trans. on Commun. E Series B,* vol. 84, pp. 263-274, 2001.

[16] É. Dubois and E. Michaux, "Grocer 1.64: an econometric toolbox for Scilab", 2001.

[17] B. Bouterse, "VM Capacity Planning for Software-as-a-Service Environments", Ph.D. Thesis, North Carolina State University, 2016.