An Inhomogeneous Hidden Markov Model for Efficient Virtual Machine Placement in Cloud Computing Environments

Hugo Lewi Hammer[1], Anis Yazidi and Kyrre Begnum
Department of Computer Science
Oslo and Akershus University College of Applied Sciences

Abstract

In a cloud environment virtual machines are created with different purposes like providing users with computers, handling web traffic etc. A virtual machine is created in such a way that a user will not notice any differences from working on a physical computer. A challenging problem in cloud computing is how to distribute the virtual machines on a set of physical servers. An optimal solution will provide each virtual machine with enough resources and at the same time not using more physical serves (energy/electricity) than necessary to achieve this.

In this paper we investigate how forecasting of future resource requirements (CPU conspumption) for each virtual machine can be used to improve the virtual machine placement on the physical servers. We demonstrate that a time dependent Hidden Markov model with an autoregressive observation process replicates the properties the CPU consumption data in a realistic way and forecasts future CPU consumption efficiently.

Keywords: cloud computing, cpu consumption, inhomogeneous hidden markov model, stochastic bin packing

1 Introduction

The use of cloud computing is increasing at tremendous speed as a result of the use of virtual machines having exploded in recent years. More companies are centralising their resources to data centres to ensure uninterrupted power, better security, greater opportunities and availability. The number of data

[1]Adress: Pilestredet 35
0160 OSLO
Norway
Email: hugo.hammer@hioa.no

centres increased by 56 per cent worldwide from 2005 to 2010 (Koomey; 2008), and more hardware is being installed to handle the rapidly increasing demand. Cloud computing now consumes more electricity every day than India (Green peace; 2010), and Google alone consumes the same amount of energy as Norway's capital city (Statistisk sentralbyrå; 2014; The New York Times; 2011).

The need to make cloud environments more energy-efficient is a major driver for data centres and cloud providers. The idea is simple in principle: always align the number of running physical servers with current demand. This cannot be solved just by knowing the virtual machine creation time, which is where the state-of-the art currently is in cloud solutions such as OpenStack (OpenStack; 2015). Virtual machines are created and removed by the cloud's tenants, leaving "gaps" of vacant capacity on physical servers. It is not guaranteed that new virtual machines will properly fill these gaps, leaving an overhead of waste on many servers. Furthermore, since not all virtual machines are active all the time (in fact, the opposite is much more likely), one can *migrate* them and *pack* them into a smaller number of physical machines in order to power the remaining servers down, thus saving energy. Conversely, in the case of increased demand for capacity and computational power, more physical servers can be booted and virtual machines redistributed, so that their requirements are met.

Even though most cloud technologies and virtual environments support the technical building blocks of dynamic power-savings, the remaining part consists of the algorithms by which the process can be automated. It is a major challenge that most cloud environments have virtual machines that behave on the basis of the local context of their users and user demands, making it difficult to re-use workload profiles. For example, a cloud deployment at a college or university would be used to run computational jobs and student labs, but not to run commercial services. Its usage profiles would not match that of a commercial cloud vendor whose virtual machines follow other usage patterns to a greater extent. The problem therefore fits well with the domain of machine-learning and statistical forecasting, where local behaviour has to be learned so that accurate predictions can be made as the basis for action.

A lot of effort has been devoted to designing optimal solutions for virtual server consolidation, see, e.g., Ferdaus and Murshed (2014) for a positive review. The majority of the works consider the problem as a deterministic resource allocation problem and typically as a bin packing problem (John-

son; 1974). In reality, resource demands change over time, which makes the consolidation problem a stochastic one. A fair amount of research has already been done on stochastic resource allocation, which involves modelling the different problem variants of Stochastic Bin Packing. Such algorithms are inspired by the deterministic counterpart solutions of the Bin Packing problem. When dealing with the realm of Stochastic Bin Packing, the most common solution is that the resource demand follows a normal distribution. However, such a simplistic hypothesis does not capture the fast dynamics of the system, since a proactive approach is required to make instantaneous consolidation decisions. In fact, normal distribution is known to be viable for modelling the average resource demand, but not the instantaneous demand. Thus, these are better suited for long time horizon provisioning over fairly long periods of time, so that the average behaviour converges to normal, and not for timely decision-making, where the main concern is the instantaneous evolution of the resources. In order to tackle this problem, we model the resource consumption of CPUs using an inhomogeneous (time-dependent) version of the hidden Markov model (HMM).

The inhomogeneous HMM is sound and plausible, as it can accommodate two main and intuitive properties of real CPU consumption data:

- The transitions between high activity CPU consumption (active state) and low activity CPU consumption (inactive state) depend on the time of day. In this sense, we assume that the transition probabilities of the underlying Markov chain are time-dependent, and thus an inhomogeneous Markov chain.

- The CPU consumption at a given time is dependent on the CPU consumption at previous time steps. Under the same state of the HMM, namely the active state, or the inactive state, we thus assume that the CPU consumption follows an autoregressive process.

We demonstrate that a time-varying HMM that accommodates the properties above replicates the properties and real CPU consumption data in a very realistic way, and that the model can be applied to efficiently consolidate virtual machines in a cloud environment. In contrast to earlier work on virtual machine consolidation, we introduce the important concept of distinguishing between migration of active and inactive virtual machines. Migration of active virtual machines is typically referred to as live migration. Our

results show that, by migrating the virtual machines with the least probability of being active in future, the number of migrations of active virtual machines can be reduced to almost zero. For inactive virtual machines, live migration, with the associated risk of affecting the users, is not necessary, and safer and more efficient migration strategies can be applied.

We have not found any research paper that uses a time-varying HMM to model resource usage in cloud computing, as proposed in this paper. On the other hand, there are examples in other fields of research. Durland and McCurdy (1994); Masson and Ruge-Murcia (2005); Kim et al. (2008); Banachewicz et al. (2007); Meligkotsidou and Dellaportas (2011) apply time-varying HMMs in economics, and see, e.g., Robertson et al. (2004) and Betr et al. (2008) for applications in environmental studies.

2 Properties of CPU consumption over time

Typical resource consumption variables for virtual machines are CPU, disk I/O, network I/O and memory, where the first three are considered to be more dynamic and also more constrained by shared resources. For example, it is common that all virtual machines share the same network link as well as the same disk controller. It is also normal to overprovision the number of virtual CPUs (VCPU) to a much higher degree than memory. In OpenStack, the default values at the time of writing are 16:1 for VCPU:CPU and 1.6:1 for memory OpenStack (2015). Furthermore, actual CPU consumption will have the greatest influence on the power demand of a physical server in traditional cloud installations, making it the most important variable on which to base an algorithm.

To be able to construct a good consolidation algorithm, we need an understanding of the properties of resource consumption for users. We logged the CPU consumption of a typical (hard-working) office worker every fifth minute for 15 working days. The data are shown in Figure 1, where the three upper panels show CPU consumption for three arbitrary days, and the bottom panel the average CPU consumption over the 15 working days during which we monitored the office worker. We will use the data to develop a suitable statistical model for such data. We can make the following observations from the data.

A. The office worker has high activity during the day, starting from about 5:00 in the morning at the earliest and ending around 8:00 (20:00) in

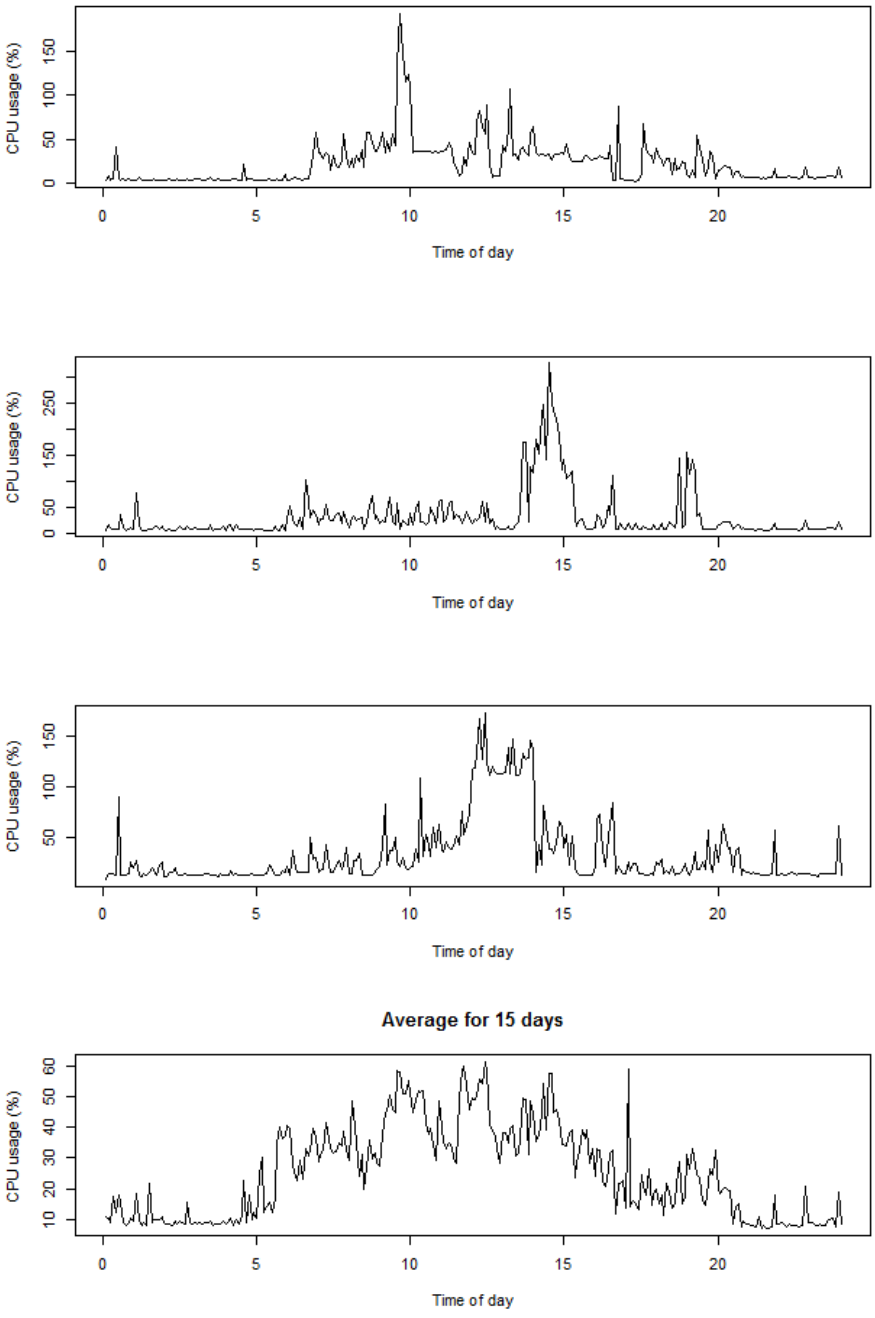

Figure 1: CPU consumption for a typical office worker. Three upper panels: CPU consumption for three arbitrary days. Bottom panel: Average CPU consumption over the 15 working days5 100% CPU consumption is equivalent to one CPU kernel running at full capacity.

the evening. On average, the highest CPU consumption is between 10:00 a.m. and 3:00 p.m. (15:00). There is almost no activity before 5 a.m. and after 8 p.m.

B. There is also a clear dependence between subsequent observations (the autocorrelation is larger than zero).

Our model will use the assumption that virtual machines display a periodic pattern of either high or low CPU activity that is unique to them, as in the above case. This pattern can be observed and learned, so that predictions of resource demands are possible on a per virtual machine basis, as well as being aggregated into next-timeframe, cloud-wide resource demands on which to act.

3 Statistical model for CPU consumption

In this section, we present a suitable statistical model for the CPU data described in Section 2. We start with some relevant theory related to the hidden Markov model before the model for CPU consumption is introduced in Section 3.3.

3.1 Hidden Markov models

Let X_t be a discrete stochastic variable with possible outcomes $\{0, 1, \ldots, K\}$ representing the states of a Markov chain at time point $t \in \{1, 2 \ldots, T\}$. The distribution of another stochastic variable Y_t depends on the state of X_t, $Y_t \sim P(Y_t|X_t)$. Given $X_1, X_2, \ldots, X_T$, we assume that $Y_t, t = 1, 2, \ldots, T$ are independent. Overall, the model can be written as

$$P(X_{1:T}, Y_{1:T}) = P(X_1)P(Y_1|X_1) \prod_{t=2}^{T} P(X_t|X_{t-1})P(Y_t|X_t) \qquad (1)$$

where $X_{1:T} = X_1, X_2, \ldots, X_T$ and $Y_{1:T} = Y_1, Y_2, \ldots, Y_T$. Figure 2 shows a graphical representation of the model where arrows represent conditional dependencies. We assume that $Y_t, t = 1, 2, \ldots, T$ are observed and $X_t, t = 1, 2, \ldots, T$ unobserved. We say that the states of the Markov chain are hidden, thus the name hidden Markov model (HMM). The most common task is

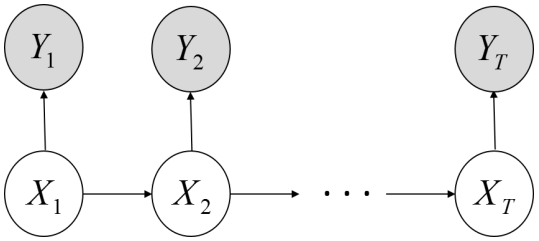

Figure 2: A graphical representation of the hidden Markov model.

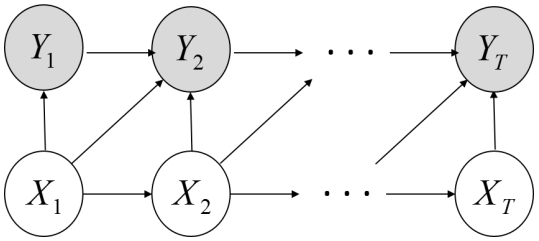

Figure 3: A graphical representation of the generalised Hidden Markov model.

to predict the states of the underlying Markov chain given the observations $Y_t, t = 1, 2, \ldots, T$.

The conditional independence of $Y_{1:T}$ given $X_{1:T}$ is often unrealistic in many applications. One possible generalisation is to let Y_t depend on the previous states, e.g. change $P(Y_t|X_t)$ to $P(Y_t|X_{t-1:t}, Y_{t-1})$ in (1). One example could be that observations are generated from an autoregressive model of lag one

$$\begin{aligned} Y_t &= m(X_t) + a(X_t)(Y_{t-1} - m(X_t)) + \epsilon_{0,t}(X_t), \quad \text{if } X_{t-1} = X_t \\ Y_t &= m(X_t) + \epsilon_{1,t}(X_t), \quad \text{if } X_{t-1} \neq X_t \end{aligned} \tag{2}$$

where the time series depends on the state of the underlying Markov chain. A graphical representation of such a model is shown in Figure 3.

3.2 Forward Backward algorithm

A hidden Markov model can be efficiently evaluated using the Forward Backward algorithm. In the forward part, the unknown $X_{1:T}$ will be integrated out in $O((K+1)^2 T)$ time. In the backward part, different important properties

can be computed in $O(T)$ time such as the maximum posterior probability

$$\arg\max_{\mathcal{X}} P(X_{1:T} \,|\, Y_{1:T}) = \arg\max_{\mathcal{X}} P(X_{1:T}, Y_{1:T})$$

where $\mathcal{X}$ refers to the sample space of possible outcomes of $X_{1:T}$. The backward part can also generate samples from the posterior distribution $P(X_{1:T} \,|\, Y_{1:T})$.

The Forward Backward algorithm can also be used for efficient prediction of futures values (forecasting). Suppose we have a given observation $Y_1, \ldots, Y_t$ and want to predict the next value Y_{t+1} when $X_{1:t+1}$ is unknown. Integrating out the unknown $X_{1:t+1}$ (forward part), we get

$$\begin{aligned}
&P(Y_{t+1}|Y_{1:t}) \propto \\
&\propto \sum_{X_1} \cdots \sum_{X_{t+1}} P(Y_{t+1} \,|\, X_{t+1}) P(X_{t+1} \,|\, X_t) \cdots P(X_2|X_1) P(Y_1|X_1) P(X_1) \\
&= \sum_{X_{t+1}} P(Y_{t+1} \,|\, X_{t+1}) P(X_{t+1} \,|\, X_t) \sum_{X_t} \cdots \sum_{X_1} P(X_2|X_1) P(Y_1|X_1) P(X_1) \\
&= \sum_{X_{t+1}} P(Y_{t+1}) \,|\, X_{t+1}) H(X_{t+1}; Y_{1:t})
\end{aligned}$$

We see that $P(Y_{t+1} \,|\, Y_{1:t})$ becomes a mixture of the conditional distributions $P(Y_{t+1} \,|\, X_{t+1})$ with weights $\propto H(X_{t+1}; Y_{1:t})$. The Forward Backward algorithm also applies to the generalised hidden Markov model shown in Figure 3, but $P(Y_{t+1} \,|\, Y_{1:t})$ will then be a mixture of $(K+1)^2$ distributions.

3.3 Hidden Markov model for CPU consumption

We now apply a hidden Markov model to the CPU consumption data. As expected, the CPU consumption data for office workers shows a periodicity of 24 hours (Figure 1). We assume that the observations for different days are independent outcomes from the same model. We assume that the hidden Markov chain has two states

- The user is using the computer at time t (active), $X_t = 1$. From Figure 1, we see that this is typically from the morning to the afternoon.
- The user is not using the computer at time t (inactive), $X_t = 0$. This is typically in the evening and during the night.

For hidden Markov models, it is most common by far to assume that the transition probabilities of the hidden Markov chain are constant over time. This is not a realistic assumption for this application. For example, it is more likely to go from an inactive to an active state in the morning than in the evening or from an active to an inactive state in the evening than in the morning. Thus, we assume a time-dependent (inhomogeneous) Markov chain with a transition matrix denoted as

$$\mathcal{P}_t = \begin{bmatrix} P_t^{00} & P_t^{01} \\ P_t^{10} & P_t^{11} \end{bmatrix}$$

where P_t^{ij} is the probability of going from state i to j in time step t. Since $P_t^{01} = 1 - P_t^{00}$ and $P_t^{10} = 1 - P_t^{11}$, the transition matrix consists of two free variables in each time step. To reduce the number of unknown parameters, we assume that the transition probabilities are related through some functions. The following functions performed well in our experiments

$$\begin{aligned} P_t^{00} &= \text{logit}^{-1}(\gamma_{00} + \gamma_{01} t + \gamma_{02} t^2) \\ P_t^{11} &= \text{logit}^{-1}(\gamma_{10} + \gamma_{11} t + \gamma_{12} t^2) \end{aligned} \tag{3}$$

Using a parabola within the inverse logit function, we are able to model a user during a 24-hour cycle going from a high probability of being inactive (night) to a high probability of being active (e.g. working hours) and back again to being inactive (evening/night). Using (3), the number of unknown parameters is reduced from $2(T-1)$ to six.

Next, we turn to the model assumptions for the observations. Inspecting the CPU data in Figure 1, the conditional independence between subsequent observations given the hidden state does not seem to be realistic, and we therefore use the generalised version of the HMM based on autoregressive processes of lag 1, see (2) and Figure 3. The distribution of the CPU data is far from a normal distribution, but the logarithm of the data seems to be fairly normal. Thus, we let $Y_t, t = 1, \ldots, T$ represent the logarithm of the CPU data. We assume that the properties of the AR(1) process depend on the state of the underlying Markov chain, see equation (2). We assume that $\epsilon_{i,t}(X_t) \sim N(0, \sigma_i(X_t)), i = 0, 1$, where $N(\mu, \sigma)$ denotes a normal distribution with expectation μ and standard deviation σ.

We may not have many days with observations, which could result in uncertainty in the estimation of the parameters. We deal with uncertainty by casting the problem in a Bayesian framework. Let

$P(\gamma_{ik}), P(m(i)), P(a(i)), P(\sigma_i(j))$ $i, j = 0, 1, k = 0, 1, 2$ denote prior distributions for the unknown model parameters. We assume that we have limited prior information and choose wide uniform distributions for the priors

$$P(m(i)) = P(a(i)) = P(\gamma_{ij}) = \text{Uniform}(-1000, 1000),\ i = 0, 1, j = 0, 1, 2$$
$$P(\sigma_i(j)) = \text{Uniform}(0, 1000),\ \ i, j = 0, 1$$

The given model has an obvious identifiability problem, namely that the state zero of the Markov chain could denote both an active and inactive state. See, e.g., Jasra et al. (2005) for a review of different approaches to dealing with the identifiability problem. In our model, the labelling of the active and inactive state is not important, and the estimation of the parameters worked fine for the general model above without adding any prior information about the active and inactive state.

We assume that, *a priori*, all the hyperparameters are independent. Further we assume that we have observations for D days (24-hour cycles) and that the observations for each day are independent given the hyperparameters. The posterior distribution becomes

$$\begin{aligned}
&P(\boldsymbol{\theta}_X, \boldsymbol{\theta}_Y, X_{1:T,1}, \ldots, X_{1:T,D} \,|\, Y_{1:T,1}, \ldots, Y_{1:T,D}) \propto \\
&P(\boldsymbol{\theta}_X, \boldsymbol{\theta}_Y, X_{1:T,1}, \ldots, X_{1:T,D}, Y_{1:T,1}, \ldots, Y_{1:T,D}) = \\
&P(\boldsymbol{\theta}_X)P(\boldsymbol{\theta}_Y) \prod_{d=1}^{D} \Bigg[P(X_1, \boldsymbol{\theta}_X)P(Y_{1,d}|X_{1,d}, \boldsymbol{\theta}_Y) \times \\
&\qquad\qquad \prod_{t=2}^{T} P(X_{t,d}|X_{t-1,d}, \boldsymbol{\theta}_X)P(Y_{t,d}|X_{t,d}, \boldsymbol{\theta}_Y) \Bigg]
\end{aligned} \tag{4}$$

where $X_{t,d}$ and $Y_{t,d}$ denote the state of the Markov chain and CPU consumption at time t on day d, $\boldsymbol{\theta}_X = [\gamma_{00}, \ldots, \gamma_{12}]$,
$\boldsymbol{\theta}_Y = [a(0), a(1), m(0), m(1), \sigma_0(0), \sigma_1(0), \sigma_0(1), \sigma_1(1)]$,
$P(\boldsymbol{\theta}_X) = \prod_{i=0}^{1} \prod_{j=0}^{2} P(\gamma_{ij})$ and $P(\boldsymbol{\theta}_Y) = \prod_{i=0}^{1} P(a(i))P(m(i)) \prod_{j=0}^{1} P(\sigma_i(j))$.

3.4 Inference

To estimate the parameters of the model in (4), we resort to Markov chain Monte Carlo (McMC) simulation. Suppose that we want to generate realisations from some probability distribution $\pi(x)$. The idea behind McMC algorithms is to construct a Markov chain with $\pi(x)$ as the limiting distribution. For an introduction to McMC simulation, see, e.g., Robert and Casella

(2004). The two most used McMC algorithms are the Gibbs sampler and the Metropolis–Hastings (MH) algorithm. In this paper, we use MH simulation.

If we want to generate realisations from (4), we can handle the unknown variables $X_{1:T,1}, \ldots, X_{1:T,D}$ in two ways. We can either generate realisations of $X_{1:T,1}, \ldots, X_{1:T,D}$ in a Gibbs step (using the Forward Backward algorithm) in addition to the unknown parameters $\boldsymbol{\theta}_X$ and $\boldsymbol{\theta}_Y$ in each iteration of the algorithm, or integrate out $X_{1:T,1}, \ldots, X_{1:T,D}$ from the posterior distribution and only simulate $\boldsymbol{\theta}_X$ and $\boldsymbol{\theta}_Y$ from

$$P(\boldsymbol{\theta}_X, \boldsymbol{\theta}_Y \,|\, Y_{1:T,1}, \ldots, Y_{1:T,D}) = \\ = \sum_{X_{1,1}} \cdots \sum_{x_{t,d}} P(\boldsymbol{\theta}_X, \boldsymbol{\theta}_Y, X_{1:T,1}, \ldots, X_{1:T,D} \,|\, Y_{1:T,1}, \ldots, Y_{1:T,D})$$

The latter strategy was the most successful in our experiments. $X_{1:T,1}, \ldots, X_{1:T,D}$ were integrated out using the Forward Backward algorithm. For the unknown parameters, we chose normally distributed random walk proposals. By choosing the prior distributions as normal distributions, more efficient simulation algorithms can be constructed, see, e.g., Holmes and Held (2006); OHagan and Forster (2004). For our application, the random walk approach performed well and also gave us the possibility of choosing the prior distributions more freely.

To evaluate the convergence of the MH algorithm, we follow the suggestions in Robert and Casella (2004) to start the McMC algorithm from different initial start values of the unknown parameters. If the algorithm converges, the posterior distributions for the parameters should be the same independently of the start values.

4 Using statistical models for efficient consolidation of virtual machines

In this section, we describe how statistical forecasting models can be used to efficiently consolidate virtual machines. We argue that, by using statistical models to forecast future resource consumption, more efficient bin backing can be achieved. This will be analysed thoroughly in the experiments (Section 5). We follow the common strategy of modelling the problem as a bin packing problem, and since resource consumption is modelled using statistical distributions, we fall within the less studied problem of stochastic bin

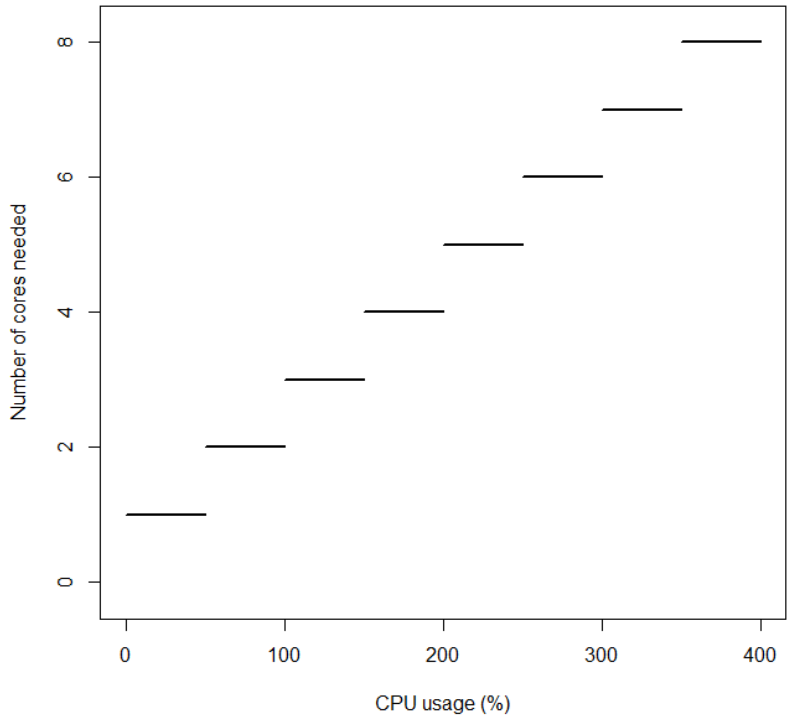

Figure 4: Relation between CPU consumption on a virtual machine and the number of cores the virtual machine should be equipped with.

packing.

Suppose that, at time t, the virtual machines VM_1, $\text{VM}_2, \ldots$ VM_m run on a physical machine consisting of C_{PM} CPU cores. Let $P(Y^i_{t+k}|Y^i_{1:t})$ denote the probability distribution for CPU resource consumption at time $t+k$ for VM_i, $i = 1, \ldots, m$ conditioned on resource consumption so far this day. Further, the number of CPU cores $c \in \{1, 2, \ldots, C_{PM}\}$ a virtual machine should be equipped with is related to the CPU resource consumption y through some function $c = f(y)$. In the experiments, we used the function in Figure 4. See, e.g., OpenStack (2015); Beloglazov and Buyya (2015) for others dealing with such relations. In the plot, e.g., 200% denotes a CPU consumption equivalent to two cores running at full capacity.

The consolidation algorithm consists of an expansion part and a merging

part. This is not an uncommon strategy, see, e.g., Takahashi et al. (2012).

- **Expansion:** We do the following for each physical machine. We compute the probability that the virtual machines on the physical machine need more than C_{PM} cores at time $t+k$

$$P(f(Y^1_{t+k}) + f(Y^2_{t+k}) + \cdots + f(Y^m_{t+k}) > C_{PM} \,|\, Y^1_{1:t}, Y^2_{1:t}, \ldots, Y^m_{1:t}) \quad (5)$$

  If this probability is above some threshold τ (say 0.05), we are above capacity, and some virtual machines must be migrated to other physical machines. We migrate virtual machines until the probability in (5) is below τ. To decide which physical machines the virtual machines should be migrated to, there are typically two main strategies in the bin packing theory. In the *first fit* strategy, a virtual machine is migrated to the first physical machine with enough capacity. In the *best fit* strategy, a virtual machine is migrated to the physical machine with the highest probability in (5) below τ, i.e. the physical machine which the virtual machine fits best. If no running physical machines have enough capacity, a new physical machine must be started up and the virtual machine migrated to it.

- **Merging:** If the expansion step above at time t does not result in any migrations, we try to merge virtual machines to fewer physical machines. We find the physical machine with the lowest probability of exceeding capacity at time $t+k$. We start migrating virtual machines from this physical machine to other physical machines with sufficient available capacity according to (5). If we are able to migrate all the virtual machines, we turn the physical machine off.

The probability in (5) is computed using Monte Carlo simulation.

5 Experiments

In the rest of the paper, we denote the model presented in Section 3.3 CPU HMM. The parameters of the model are estimated based on the data in Section 2 using the MH algorithm presented in Section 3.4. The burn-in period of the MH algorithm was five to 45 minutes for the different parameters, and a good representation of the posterior distribution was achieved in five to

ten hours. Figure 5 shows trace plots from the McMC runs for the parameters $m(0), \sigma_1(1), \gamma_{00}$ and γ_{12}, which are representative of the convergence and mixing properties of all the parameters estimated by the MH algorithm. We see that the McMC algorithm converges fast and mixes well. Note that it is not necessary to estimate the parameters of the model often in this application, and once a day or week is normally sufficient. We estimate the model parameters using the average of all the states from the MH algorithm after the burn-in period.

Figure 6 shows the estimated curves for P_t^{00} and P_t^{11} in equation (3) based on the samples from the MH algorithm. We see that the estimated transition matrix based on P_t^{00} and P_t^{11} is highly inhomogeneous as a function of time, which shows the importance of not modelling the hidden Markov chain as homogeneous. We see that, during the night, there is a high probability of staying in the inactive state (P_t^{00}) and a very low probability of staying in the active state (P_t^{11}). As the morning approaches, P_t^{00} decreases, while P_t^{11} increases rapidly, which means that a transition from the inactive to the active state becomes very likely. In the afternoon, we observe the opposite and a transition from the active to the inactive state becomes more and more likely. It is very satisfying to observe how well the given model and algorithm are able to automatically divide the resource consumption into clear active and inactive states despite using priors with no information about how to distinguish between the two states.

Figures 7 and 8 show histograms of the realizations from the MH algorithm after the burn in period for all the parameters in the model.

Figure 9 shows three independent realisations from the CPU HMM model and the average of 15 realisations. The black lines at the bottom of each panel show at which time intervals the underlying Markov chain was in the active state for these realisations. Because of the inhomogeneity of the hidden Markov chain, we see that it is far more likely to be in the active state during working hours compared to other times of the day. Comparing Figure 9 with Figure 1, we see that the CPU HMM replicates the properties of the real CPU consumption data quite impressively.

5.1 Prediction performance of the CPU HMM

In this section, we evaluate the forecasting performance of the CPU HMM. We compare the model with the AR(1) and AR(2) models. We use the 11 first days and the four last days of data as training and testing, respectively,

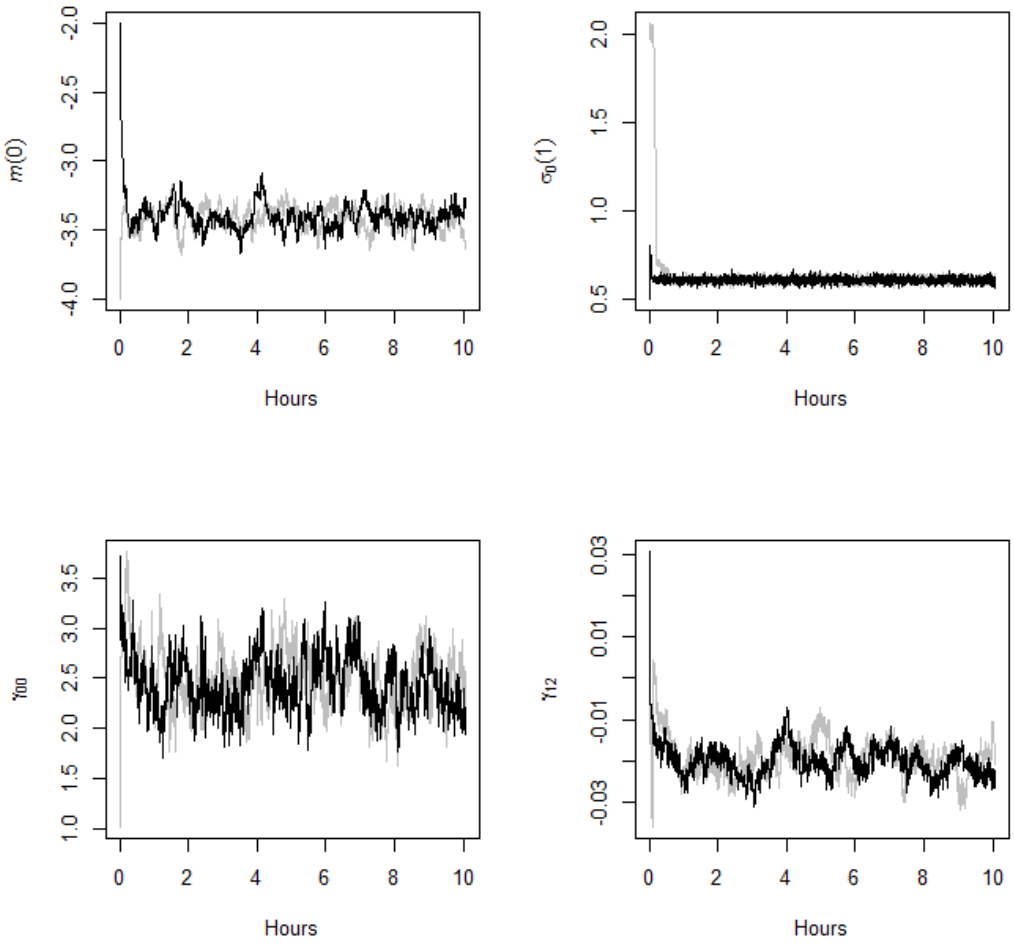

Figure 5: Trace plots for the variables $m(0), \sigma_0(1), \gamma_{00}$ and γ_{12}. The black and the grey curves show two independent McMC runs with different initial states.

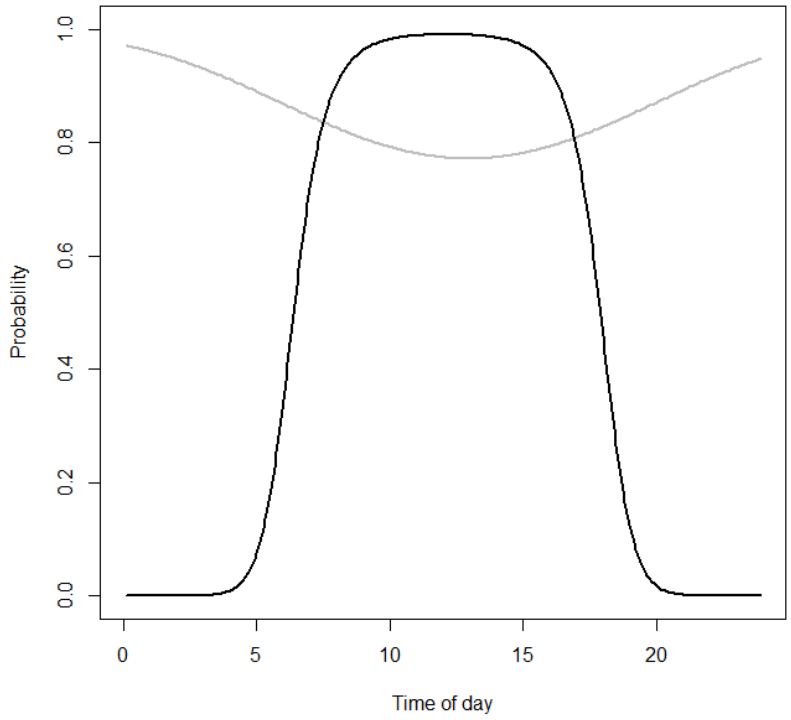

Figure 6: Estimated curves for P_t^{00} (grey curve) and P_t^{11} (black curve).

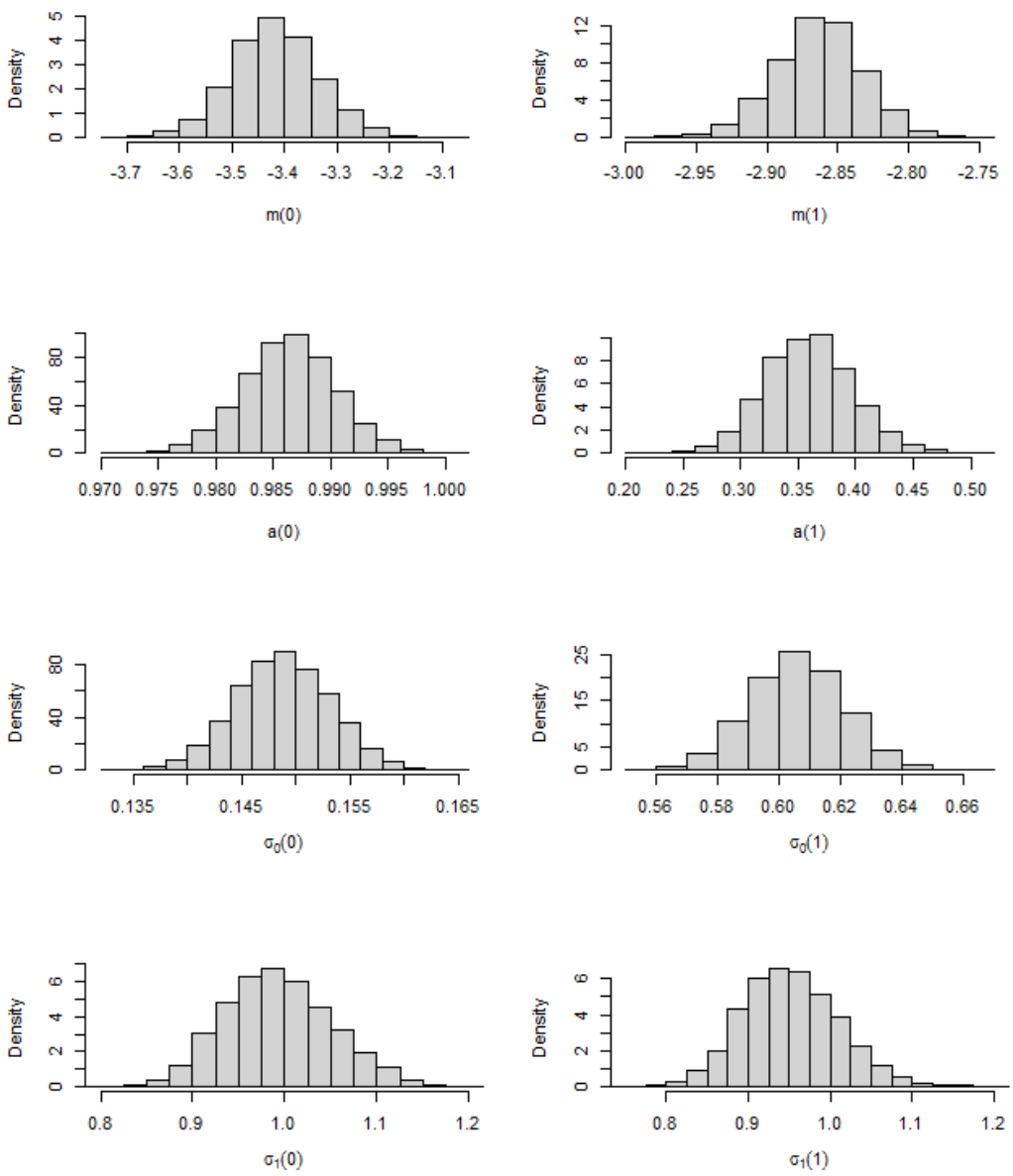

Figure 7: Histogram of realizations from the posterior distribution. The left and the right column show estimates for the active and the inactive state, respectively

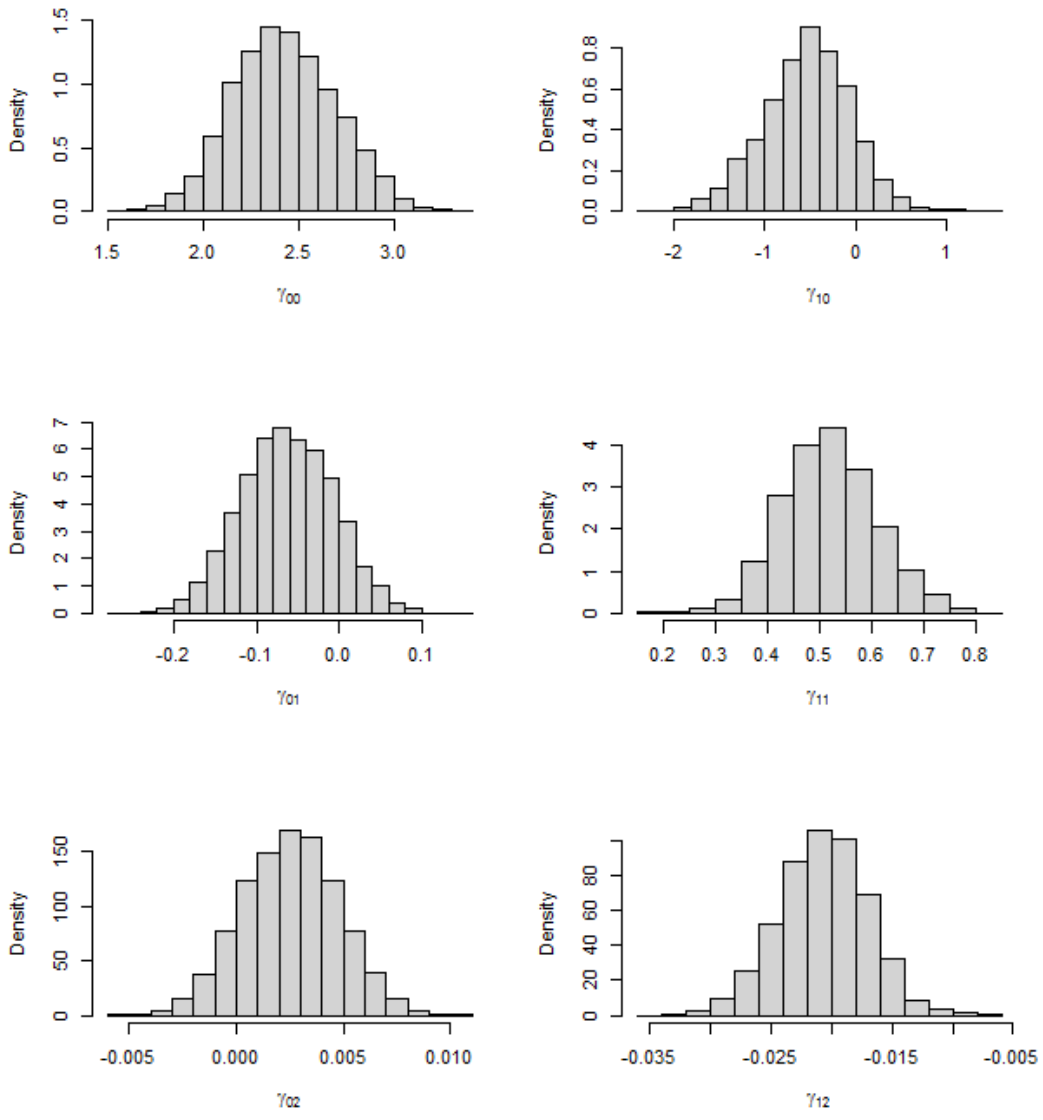

Figure 8: Histogram of realizations from the posterior distribution. The left and the right column show estimates for the active and the inactive state, respectively

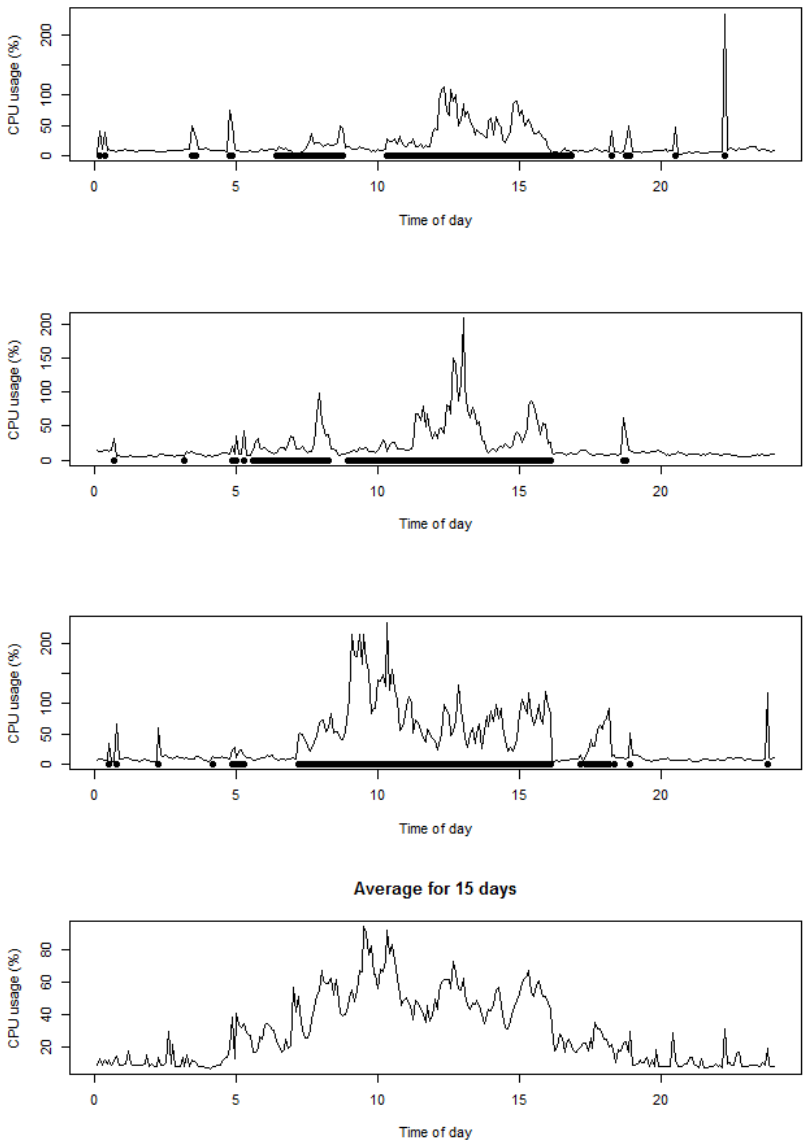

Figure 9: Three upper panels: Three arbitrary realisations from the CPU HMM. Bottom panel: Average of 15 realisations. 100% CPU consumption is equivalent to one CPU kernel running at full capacity. The black lines at the bottom of each panel show at which time intervals the underlying Markov chain was in the active state for these realisations.

Table 1: Forecasting performance of the three models. The error is measured as the average difference in absolute value between the predicted CPU consumption and the true CPU consumption.

Model	Average error (abs. value)
AR(1)	0.332
AR(2)	0.306
CPU HMM	0.298

to measure forecasting performance. We repeatedly forecast the next time step (five minutes). Table 1 shows the forecasting performance of the three models as the average difference in absolute value between the forecasted CPU consumption and the true CPU consumption. Even though the CPU HMM model is only equipped with an AR(1) model within each state, it outperforms both the AR(1) and AR(2) models.

5.2 Consolidation based on CPU HMM

We assume an environment with a set of physical machines consisting of ten CPU cores and a total of 60 virtual machines. The virtual machines should be placed and migrated between these physical machines to reduce the number of physical machines used, while at the same time providing the virtual machines with the resources they need. We assume that each of the virtual machines has a CPU consumption pattern in accordance with the CPU HMM. Recall from Figure 9 that the CPU HMM replicates the properties of real CPU consumption data in a very realistic way, making the experiments in this section realistic from a practical point of view. In this section, we compare different approaches to consolidating virtual machines on physical machines based on CPU consumption. For the different virtual machines, we used parameter values in the CPU HMM based on different adjustments of the estimated parameter values from the MH simulations. For some virtual machines, for instance, we shifted the curves in Figure 6 one to three hours to the right or to the left to capture that some office workers start the day earlier than others. For other virtual machines, we made the office days shorter and longer by shrinking or stretching the curves in Figure 6. We also made adjustments so that some virtual machines have a higher CPU consumption on average than others. In total, we ended up with 60 users (virtual machines) with different, but realistic user patterns and

resource needs. For each virtual machine, we can now generate synthetic CPU consumption data for a 24-hour cycle that can be used to evaluate the performance of different consolidation strategies between the physical machines.

We assume that migration of virtual machines is executed every 15 minutes (if any virtual machines need to be migrated). We consider five different migration strategies. Strategy 1 does not follow the algorithm in Section 4 completely and will be described in further detail. Strategies 2 to 5 follow the algorithm in Section 4 and only some additional details distinguishing these strategies are provided below.

1. Current CPU consumption: In the algorithm in Section 4, virtual machines are migrated if the probability in (5) is above τ. In this strategy, we instead migrate virtual machines if the current total CPU consumption at a physical machine is above some threshold τ_2. For a physical machine above capacity (τ_2), the virtual machines that currently have the highest CPU consumption will be migrated first.

2. Here we provide some further details for the algorithm in Section 4. In the migration step, if (5) is above τ for a physical machine, we need to decide in which order to migrate the virtual machines. In this strategy, we migrate virtual machines by the descending expected number of CPU cores needed at time $t + k$

$$\mathrm{E}[f(Y^i_{t+k}) \,|\, Y^i_{1:t}], \;\; i = 1, 2, \ldots m$$

   i.e. we start by migrating the virtual machine with the highest expected number of CPU cores needed. In the merging step, we also migrate based on this rule.

3. This is the same as strategy 2 except that we start by migrating the virtual machines with the *lowest* expected number of CPU cores needed at time $t + k$. This means that, instead of moving a small number of resource-demanding virtual machines to come below capacity, we move many less resource-demanding virtual machines. We thus expect that the number of migrations will be higher with this strategy compared to strategy 2.

4. In this strategy, the order of migration is based on whether the virtual machines are active or not. We start by migrating the virtual machines that have the highest probability of being active at time $t + k$.

5. This is the same as strategy 4, except that we start migrating the virtual machines with the lowest probability of being active at time $t + k$. We expect that the number of migrations will be higher with this strategy, compared to 4, but that the number of migrations of active virtual machines will be lower.

In all the experiments, we use the first-fit bin packing strategy. We also experimented with the best-fit strategy, and the differences in performance were minimal. For each of strategies 2 to 5, we set $k = 1, 2, \ldots, 6$, which means performing migrations based on the forecast state of the virtual machines 5, 10, ..., 30 minutes in the future.

To generate synthetic data for a virtual machine, we generate a realisation from the CPU HMM with the given parameters for this virtual machine. We start by generating a x_1 from $P(X_1)$, then y_1 from $P(Y_1|X_1)$ and so on until we get the complete data traces $x_{1:T}$ and $y_{1:T}$. Based on $y_{1:T}$ and the function shown in Figure 4, we can count the number of physical machines that are above and below capacity in each time step. To make comparison between the different consolidation strategies easier, we adjust τ and τ_2 so that all the experiments end up with a probability of ten per cent that an arbitrary physical machine at an arbitrary time step will exceed the capacity. The better the consolidation strategy, the fewer physical machines and migrations will be used to achieve this. From $x_{1:T}$, we know whether a virtual machine was in an active or inactive state when it was migrated, and the number of migrations of active and inactive virtual machines can be counted.

To reduce the effect of Monte Carlo error in our experiments, we generated 50 independent synthetic data traces $x_{1:T}$ and $y_{1:T}$ for each virtual machine and ran each migration strategy for 50 24-hour cycles, one for each independent set of data.

Table 2 summarises the results of these experiments. We see that that carrying out consolidation based on the current CPU consumption (strategy 1) performs poorly compared to strategies 2 to 5. While strategies 2 to 5 use about 12 physical machines on average, strategy 1 uses over 18 physical machines on average. We also see that the number of migrations is many times higher for the current CPU strategy, with about 10 migrations of virtual machines per update (15 minutes) compared to between one and two for the other strategies. This emphasises the importance of having a suitable statistical model for forecasting future CPU consumption. We see that the other four strategies perform almost equally well as regards the average

Table 2: Consolidation of virtual machines based on different strategies. Columns from left to right: Migration strategy (Str.), how far into the future the method forecasts (Pred.dist), the average number of physical machines used at a given time step ($\overline{\text{nPM}}$), the average number of migrations per update (nMigrations), the average number of migrations of active virtual machines per update (nActiveMigr) and the probability that an arbitrary physical machine will exceed the capacity at an arbitrary time step (P(Exc)). From top to bottom, we have the different strategies from 1 to 5. The values in parentheses represent 95% confidence intervals based on the 50 independent runs.

Str.	Pred.dist	$\overline{\text{nPM}}$	nMigrations	nActiveMigr	P(Exc)
1	0 min	18.7 (18.3, 18.9)	9.75 (9.56, 9.93)	2.57 (2.51, 2.63)	0.100 (0.098, 0.103)
2	5 min	12.2 (12.1, 12.2)	1.35 (1.31, 1.38)	0.84 (0.82, 0.87)	0.100 (0.098, 0.103)
2	10 min	12.1 (12.0, 12.1)	1.34 (1.31, 1.37)	0.88 (0.86, 0.9)	0.100 (0.099, 0.103)
2	15 min	11.7 (11.6, 11.7)	1.29 (1.25, 1.32)	0.78 (0.76, 0.8)	0.099 (0.097, 0.102)
2	20 min	11.5 (11.4, 11.5)	1.33 (1.30, 1.36)	0.78 (0.76, 0.8)	0.099 (0.097, 0.101)
2	25 min	11.9 (11.9, 12.0)	1.33 (1.29, 1.37)	0.79 (0.76, 0.81)	0.098 (0.097, 0.101)
2	30 min	12.2 (12.2, 12.2)	1.30 (1.27, 1.34)	0.75 (0.73, 0.78)	0.098 (0.097, 0.101)
3	5 min	11.8 (11.7, 11.8)	2.35 (2.30, 2.39)	0.40 (0.38, 0.42)	0.101 (0.099, 0.103)
3	10 min	11.7 (11.6, 11.7)	2.16 (2.12, 2.19)	0.29 (0.27, 0.30)	0.099 (0.098, 0.102)
3	15 min	11.6 (11.5, 11.6)	2.00 (1.96, 2.04)	0.32 (0.30, 0.34)	0.099 (0.097, 0.102)
3	20 min	12.0 (12.0, 12.1)	1.89 (1.85, 1.94)	0.29 (0.27, 0.31)	0.097 (0.095, 0.100)
3	25 min	12.0 (12.0, 12.1)	1.64 (1.60, 1.68)	0.29 (0.28, 0.31)	0.102 (0.097, 0.105)
3	30 min	11.9 (11.8, 11.9)	1.87 (1.83, 1.92)	0.26 (0.25, 0.28)	0.097 (0.096, 0.100)
4	5 min	11.7 (11.6, 11.7)	1.37 (1.33, 1.40)	0.91 (0.88, 0.93)	0.101 (0.098, 0.103)
4	10 min	11.9 (11.9, 12.0)	1.46 (1.42, 1.50)	0.97 (0.94, 0.99)	0.101 (0.099, 0.104)
4	15 min	11.7 (11.7, 11.7)	1.30 (1.26, 1.33)	0.85 (0.83, 0.88)	0.099 (0.097, 0.102)
4	20 min	11.9 (11.8, 11.9)	1.30 (1.27, 1.33)	0.82 (0.8, 0.84)	0.102 (0.099, 0.104)
4	25 min	11.5 (11.5, 11.5)	1.35 (1.32, 1.38)	0.86 (0.83, 0.88)	0.097 (0.095, 0.101)
4	30 min	11.9 (11.9, 11.9)	1.31 (1.28, 1.34)	0.82 (0.8, 0.85)	0.098 (0.096, 0.100)
5	5 min	11.8 (11.8, 11.9)	2.24 (2.19, 2.30)	0.24 (0.22, 0.26)	0.098 (0.095, 0.100)
5	10 min	12.3 (12.3, 12.3)	2.08 (2.04, 2.12)	0.20 (0.18, 0.21)	0.101 (0.099, 0.103)
5	15 min	12.3 (12.2, 12.3)	1.82 (1.77, 1.86)	0.18 (0.17, 0.20)	0.098 (0.096, 0.100)
5	20 min	11.4 (11.4, 11.5)	1.74 (1.70, 1.77)	0.16 (0.14, 0.17)	0.099 (0.096, 0.102)
5	25 min	12.0 (11.9, 12.0)	1.72 (1.69, 1.76)	0.15 (0.14, 0.17)	0.098 (0.095, 0.100)
5	30 min	11.7 (11.6, 11.7)	1.66 (1.62, 1.69)	0.16 (0.14, 0.17)	0.101 (0.098, 0.104)

number of physical machines used.

Furthermore, we see that strategies 2 and 4 (highest expected number of cores needed and most active virtual machines) need fewer migrations than strategies 3 and 5, which is as expected. By migrating large or active virtual machines instead of small and inactive ones, fewer virtual machines, naturally, have to be migrated to get below capacity. On the other hand, comparing the number of active virtual machines that are migrated, we see that this number is many times higher for strategies 2 and 4 compared to 3 and 5. Another important observation is that the number of migrations of active virtual machines is again substantially lower for strategy 5 (least active) than strategy 3. This emphasises the usefulness of being able to predict the probability that a virtual machine is active in the future if the focus is on reducing the number of migrations of active virtual machines. The migration of inactive virtual machines can be done much more efficiently than the migration of active virtual machines and with less chance of disturbing the user.

There is no clear trend between the average number of physical machines used and the number of minutes into the future the model forecasts. On the other hand, for all the four strategies, there is a clear trend that the number of migrations (see both columns 4 and 5) is reduced when forecasting further into the future. Considering both the number of physical machines used and the number of migrations necessary to achieve this, the best migration performance is achieved by forecasting far into the future.

Figure 10 shows the number of physical machines used at different times during the day for strategy 2 when forecasting 30 minutes into the future. We would get more or less the same curve if we chose any of the other strategies, except strategy 1. We see that the system uses few physical machines during the night, since almost all virtual machines are inactive. As morning approaches, the number of physical machines increases before decreasing again in the afternoon.

6 Closing remarks

In this paper, we present a hidden Markov model (HMM) that replicates the properties of real CPU consumption data in a very realistic way. The HMM model presented deviates from the standard homogeneous HMM model in two ways:

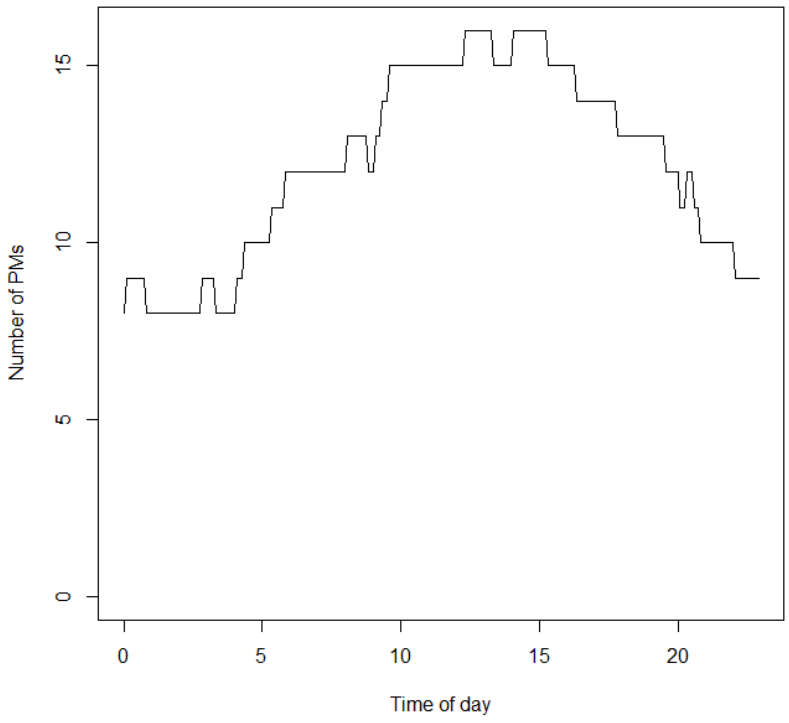

Figure 10: Number of physical machines at different times during a day.

- The model uses an inhomogeneous Markov chain to model the probability that a user switches between active and inactive state throughout the day.

- In a standard HMM model, the subsequent outcomes of the CPU values are assumed to be independent given the underlying Markov chain, and they only depend on the current state of the underlying Markov chain. Such a hypothesis is not realistic as it does not reflect the "smoothness" of the real life demand curve: resource demands at time instant $(t + 1)$ usually lie in the neighbourhood of resource demand at instant (t). Therefore, motivated by the need for a more realistic model for CPU consumption, we adopt a time-varying HMM model that accommodates more realistic dependence between the CPU values at different time instants.

Our experiments show that the model has good forecasting properties and outcompetes both the AR(1) and AR(2) models. We believe the improvement will be even larger when forecasting further into the future, since the underlying inhomogeneous Markov chain captures the long-term trends in the model given the current state.

We also investigated whether a 3-state HMM model results in a better model for the CPU data presented in this paper. One may expect the active state may be better separated into low activity (say, writing emails) and high activity (say, conducting more complicated computational task). In our experiments the MCMC algorithm still converges, but ending up with two almost equal states very similar to the non-active state of the 2-state model. The third state will be very similar to the active state of the 2-state model. Further, computing the Bayesian information criterion for both the 2-state and 3-state model result in a lower value for the 2-state model. Both of these observations indicate that the 2-state model is a better model for the data. A possible explanation is that we use a heavy tailed distribution for the resource data (recall that $P(Y_t \mid X_t)$ is assumed log-normally distributed since we use a log transform of the original CPU data) and thus capturing both low activity (e.g. writing emails) and high activity (running computations). The parameters of the log-normal distribution is fitted such that if fits well with the portion of the active state being low and high. In fact, computing the distribution of CPU usage over the 15 days with observation we do not observe a "low" and "high" active state, but a more unimodal distribution that fits well with the log-normal distribution.

In contrast to earlier work on virtual machine consolidation, we introduce the important concept of distinguishing between migration of active and inactive virtual machines. Our results show that by migrating the virtual machines with the least probability of being active, the number of migrations of active virtual machines can be reduced to almost zero. For inactive virtual machines, live migration is not necessary and much more efficient migration strategies can be used. Live migration always introduces a risk of affecting the user negatively during the migration process.

Possible extensions of our work include considering multiple types of resources and constraints, and mapping the problem to a multi-dimensional bin packing problem. Instead of using a univariate normal distribution, a multivariate normal distribution can be used for the simultaneous distribution of different resources. If the normal assumption is not realistic, the real data can be transformed, as was done in this paper, or one can resort to other multivariate distributions, e.g. using copulas (Jun Yan (2007).

References

Banachewicz, K., Lucas, A. and Vaart, A. (2007). Modeling portfolio defaults using hidden markov models with covariates, *Econometrics Journal* **10**: 1–18.

Beloglazov, A. and Buyya, R. (2015). OpenStack Neat: a framework for dynamic and energy-efficient consolidation of virtual machines in OpenStack clouds, *Concurrency and Computation: Practice and Experience* **27**(5): 1310 – 1333.

Betr, B., Bodini, A. and Cossu, Q. (2008). Using hidden markov model to analyse extreme rainfall events in central-east sardinia, *Envirionmetrics* **19**: 702 – 713.

Durland, J. M. and McCurdy, T. H. (1994). Duration-dependent transitions in a markov model of U.S. GNP growth, *Journal of Business & Economic Statistics* **12**(3): 279–288.

Ferdaus, M. H. and Murshed, M. (2014). Energy-aware virtual machine consolidation in iaas cloud computing, *Cloud Computing*, Springer, pp. 179–208.

Green peace (2010). Click clean: How companies are creating the green internet, http://www.greenpeace.org/usa/Global/usa/planet3/PDFs/clickingclean.pdf.

Holmes, C. C. and Held, L. (2006). Bayesian auxiliary variable models for Binary and multinomial regression, *Bayesian Analysis* **1**(1): 145–168.

Jasra, A., Holmes, C. C. and Stephens, D. A. (2005). Markov chain monte carlo methods and the label switching problem in bayesian mixture modeling, *Statistical Science* **20**(1): 50 – 67.

Johnson, D. S. (1974). Fast algorithms for bin packing, *Journal of Computer and System Sciences* **8**(3): 272–314.

Jun Yan (2007). Enjoy the Joy of Copulas: With a Package copula, *Journal of Statistical Software* **21**(4): 1–21.
URL: *http://www.jstatsoft.org/v21/i04/*

Kim, C.-J., Piger, J. and Startz, R. (2008). Estimation of markov regime-switching regression models with endogenous switching, *Journal of Econometrics* **143**(2): 263–273.

Koomey, J. G. (2008). Worldwide electricity used in data centers, *Environmental Research Letters* **3**(3): 034008.
URL: *http://stacks.iop.org/1748-9326/3/i=3/a=034008*

Masson, P. and Ruge-Murcia, F. J. (2005). Explaining the transition between exchange rate regimes, *The Scandinavian Journal of Economics* **107**(2): 261 – 278.

Meligkotsidou, L. and Dellaportas, P. (2011). Forecasting with non-homogeneous hidden markov models, *Statistics and Computing* **21**(3): 439–449.

OpenStack (2015). https://www.openstack.org/. [Online; accessed Juli 2015].

OHagan, A. and Forster, J. (2004). *Bayesian auxiliary variable models for Binary and multinomial regression*, Kendalls Advanced Theory of Statistics, vol. 2B, Bayesian Inference, Arnold, London.

Robert, C. and Casella, G. (2004). *Monte Carlo Statistical Methods*, Springer Series in Statistics, Springer Science, New York, USA.

Robertson, A., Kirshner, S. and Smyth, P. (2004). Downscaling of daily rainfall occurrence over northeast brazil using a hidden markov model, *Journal of Climate* **17**(22): 4407–4424.

Statistisk sentralbyrå (2014). Statistikkbanken, antall boliger i oslo @ONLINE.
URL: *https://www.ssb.no/statistikkbanken/SelectVarVal/saveselections.asp*

Takahashi, S., Takefusa, A., Nakada, H., Yoshise, A., Shigeno, M. and Kudoh, T. (2012). Virtual machine packing algorithms for lower power consumption, *Proceedings of the 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, CLOUDCOM '12, IEEE Computer Society, Washington, DC, USA, pp. 161–168.

The New York Times (2011). Google details, and defends, its use of electricity@ONLINE.
URL: *http://www.nytimes.com/2011/09/09/technology/google-details-and-defends-its-use-of-electricity.html*