Mobile Networks and Applications manuscript No.
(will be inserted by the editor)

A Queue Model for Reliable Forecasting of Future CPU Consumption

Hugo Lewi Hammer · Anis Yazidi · Alfred Bratterud · Hårek Haugerud · Boning Feng

Received: date / Accepted: date

Abstract Statistical queuing models are popular to analyze a computer systems ability to process different types requests. A common strategy is to run stress tests by sending artificial requests to the system. The rate and sizes of the requests are varied to investigate the impact on the computer system. A challenge with such an approach is that we do not know if the artificial requests processes are realistic when the system is applied in a real setting. Motivated by this challenge, we develop a method to estimate the properties of the underlying request processes to the computer system when the system is used in a real setting. In particular we look at the problem of recovering the request patterns to a CPU processor. It turns out that this is a challenging statistical estimation problem since we do not observe the request process (rate and size of the requests) to the CPU directly, but only the *average* CPU usage in disjoint time intervals.

In this paper we demonstrate that, quite astonishingly, we are able to recover the properties of the underlying request process (rate and sizes of the requests) by using specially constructed statistics of the observed CPU data and apply a recently developed statistical framework called Approximate Bayesian Computing.

Further we apply the model to forecast future CPU consumption. Our results show that the model forecast future CPU consumption with less error than both the hidden Markov model (HMM) in [16] and an ARIMA model. Another good property of the queue model is that we can forecast the *instantaneous* CPU consumption at *any* time point in the future, while the HMM in [16] and time series models are limited to only forecasting the *average* CPU consumption in disjoint time intervals.

Keywords approximate Bayesian computing · CPU consumption · forecasting · queue process and time series

1 Introduction

Scaling of computer systems is one of the most fundamental tasks in computer science. A popular approach is to assume that requests to a computer system

Department of Computer Science, Oslo and Akershus University College of Applied Sciences

follow some statistical queuing model, see e.g. [22,7,31,37,35,15,24] for a few representative examples. The typical approach in such papers is to run artificial stress tests on the system by sending requests to the system and vary the rate and sizes of the requests. A challenge with such an approach is that we do not know if the rates and sizes used in the stress tests are realistic compared that what the computer system are faced with in a real setting. A natural strategy, of course, is to observe how well the system performs in real life settings. Unfortunately, such observations do not reveal the properties of the underlying request processes (rate and sizes of requests) to the system which contains important information on how the computer system should be scaled, see more below.

In this paper we address this challenge and focus on requests to a CPU processor. We use real observed CPU consumption data to recover the properties of the underlying request processes (queuing process) to the CPU processor. More specifically we want to recover the size and rate of the requests to the CPU processor. If we know the time point for every request to the CPU processor (arrival times) and when the system finished processing the requests (departure times), the estimation of the properties of the queuing process can usually be easily done by standard statistical estimation techniques like maximum likelihood estimation. Unfortunately, such information is far from available for observed CPU consumption data. In fact, it is not even possible to observe the current CPU load (number of active CPU cores), but only the *average* load in disjoint time intervals (e.g. five minute intervals). Consequently, a process consisting of many small requests to the CPU and a process consisting of only a few large requests may look similar since the two processes *on average* consume the same amount of CPU resources. Even though we only observe the average CPU consumption we show that by using the statistical framework called Approximate Bayesian Computing (ABC) we are, quite astonishingly, able to estimate the properties of the underlying queuing process, i.e. rates and sizes of requests to the CPU processor.

From a practical point of view, being able to recover the properties of the underlying queuing process can be quite useful. E.g. if the request pattern turns out to consist of many small tasks (in stead of a few large) it can easily be parallelized and the computer system can be constructed to take advantage of this.

Hammer et al. demonstrated in [16] that consolidation of virtual machines in a cloud computer system can be improved by forecasting future CPU consumption and base the consolidation on these forecasts. In this paper we demonstrate how the queue model presented in this paper can be used to make precise forecasts of future CPU consumption. A very interesting property of the queue model is that we can forecast the *instantaneous* CPU consumption at *any* time point in the future, while the HMM in [16] and time series models, like ARIMA models, are limited to only forecasting the *average* CPU consumption in disjoint time intervals. Only by building a model with an underlying queue process are we able to forecast the instantaneous CPU consumption at any time point.

This paper is a major extension of the conference paper [17]. The related work is extended to adapt to the new main focus of the paper, namely forecasting of future CPU consumption. The results in the conference paper [17] only indicate that the queue model presented in this paper can be useful for analysis of computer systems. In this paper we extend the analysis by presenting an original and novel algorithm for how to apply the queue model to forecast future CPU consumption, please see Section 6. The forecast algorithm is evaluated through extensive experimentation

and the results show impressive performance of the forecasting algorithm (Section 7.2).

2 Related work

Forecasting CPU usage has been a focal research question in the literature. CPU load forecasting is useful in many settings, such as, forecasting the running time of tasks [9], deadline-aware scheduling decision of real-time applications [27,10], achieving elasticity in Cloud Computing via pro-active Virtual Machine (VM) migration [28], data center consolidation [32] etc.

The related work on this CPU load forecasting can be categorized under two main categories: Time-series based Methods and Queue-Theory based Methods.

Time-series based Methods : Network Weather Service (NWS) [30] is one of the most early and successful examples of one-step-ahead CPU forecaster. NWS resorts to a set of parallel linear predictors. The Mean Square Error (MSE) is computed for each forecasting model. The predictor that yields the smallest MSE is chosen to forecast the next CPU value. Those models include running average, sliding window average, last measurement, adaptive window average, median filter, adaptive window median, α-trimmed mean, stochastic gradient, and Auto-Regression(AR). Nevertheless the NWS faces challenges when dealing with peaks [3].

The Dynamic Tendency Predictor [34] is among one of the most simple and yet proved efficient one-step ahead CPU estimators. The approach states that recently increasing load will tend to increase in the future and that recently deceasing load will continue to decrease. The Dynamic Tendency Predictor [34] has been extended in [36] to forecast the tendency based on taking into account more steps backward in time instead of only relying on the last step as originally proposed in [34]. Then, a polynomial fitting approach was proposed for the forecasting.

Furthermore, multi-step-ahead forecasting of CPU load has been investigated in the literature, see for example [33]. Yang et al. [33] achieve the latter objective by finding similar historical patterns to the original pattern based on Hamming distance and Euclidean distance. Then different fusion strategies are used in order to combine the different forecastings of the similar patterns.

At this juncture, it is important to emphasize that our work is distinct for the later stream of approaches as we would to rather recover the CPU request patterns to a CPU processor instead of merely looking at the historical CPU load and trying to forecast future values via time series. We believe that recovering the underlying request patterns is very useful for providing a better understanding of the statistical dynamics of CPU load and thus achieving multi-step ahead forecasting. In addition, such an approach permits to distinguish CPU forecasting from classical time series forecasting. In fact, the informed reader can remark that the aforementioned Time-series based Methods are general and not originally designed for CPU forecasting but rather admits a long list applications ranging from stock market forecasting to weather forecasting.

Queue-Theory based Methods : Applying queuing models for analyzing CPU resource usage have received a lot of attention. We shall review some representative studies on this topic.

In [22], the authors propose to perform task scheduling not based on the worst execution time but rather based on the distribution of the execution time. A task is supposed to be composed of different sub-tasks and it is assumed that the execution time of the task is equal to the execution time of the bottleneck task which finishes last. If the worst-case execution time is known, scheduling based on worst-case execution time is guaranteed to meet the timing requirement, however the worst-case execution policy leads to waste of CPU resources which are underutilized. It is known that scheduling based on distribution yields more effective resource utilization. The authors devised a model so that to forecast the task execution time based on two classes of parameters: intrinsic and extrinsic. The intrinsic factors are related to the computational requirement of the task, while the extrinsic factors are related to communication and synchronization. In order to render the system analysis traceable, the model is simplified thanks to a simplification of the extrinsic factors, and mapped to a distributed system a queuing model which nodes dedicated to computations and nodes dedicated to communication and synchronization.

Stochastic fluid models is used in [2] for CPU capturing and releasing. Distributed software agents have routine tasks that need to be processed by a CPU. The problem is to find an optimal threshold on the workload of the agent to design a release-recapture CPU policy. The problem is modeled using two buffers, one for the routine tasks and one for the non-routine tasks. The process servers a buffer using a threshold-based policy. The threshold based policy is described in simple terms as [2]: "Traffic is emptied at rate c by a single scheduler that alternates between the two buffers. When buffer 1 becomes empty the scheduler switches to buffer 2 (after a switch-over time) and returns back to buffer 1 when the contents of buffer 1 exceeds a."

Bouterse and Perros [6] developed a model inspired from the realm of circuit switched telephony models, namely the model of blocking of Erlang, in order to develop reserve capacity model. Different models are used to forecast the arrival rate and based on this, an number of servers are provisioned so that to meet the target the blocking probability using Erlang formula.

Parameter estimation in queuing models Parameter estimation in queuing models have a long tradition going back to the David Cox paper from [8]. Most papers rely on the likelihood principle in one way or another, see e.g. [21, 1, 12], but there are some situations where standard estimation techniques can not be used. E.g. Heggland and Frigessi [18] estimate the parameters of a $G/G/1$ queue model when only the departure times are known applying a method called indirect inference.

A disadvantage of the indirect inference approach is that it is difficult to get reliable uncertainty estimates of the parameter estimates. Over the recent years the indirect inference framework has been generalized and casted in to a Bayesian framework. The approaches are typically referred to as Approximate Bayesian Computing (ABC). See e.g. [5, 11] for very nice reviews of the different approaches available within the ABC framework. Due to the complexity of the estimation problem considered in this paper, we resort to the ABC framework.

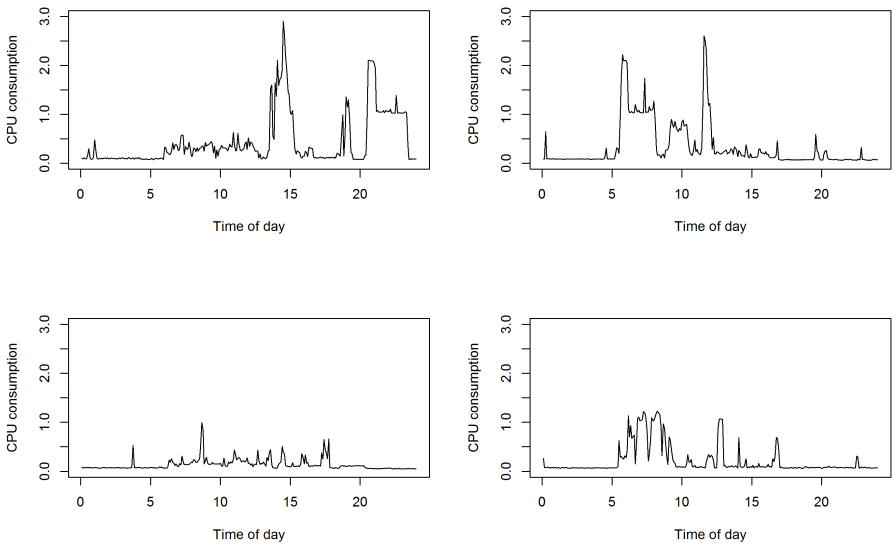

Fig. 1 The CPU consumption data for four arbitrary days.

3 CPU consumption data

Real CPU consumption data were collected on a computer with a Windows 7 operating system operated by an office worker. These are the same data used in [16]. We observed the office worker for 19 whole working days (i.e. weekends removed) and observed the average CPU usage at five-minute intervals, i.e. 288 observations every day. The CPU data for four arbitrary days are shown in Figure 1. A value of one is equivalent to one CPU core running constantly. We observe that the CPU consumption is typically largest during the day, with some occasional computations during the evening. We observe little CPU consumption during the night. Large tasks are visible in the real data with a constant CPU usage of around one, e.g. in the evening as shown in the upper left panel. In the same panel, we observe that the CPU consumption is around two right after 8 p.m. (value 20 on the x axis), meaning that two CPU cores are running some larger tasks. We also observe large time spans in which the CPU usage is far below one, which means that, during these periods, a number of smaller requests are sent to the CPU processor. Even though we do not observe the CPU requests directly, we are still able to say something about the amount of requests and their sizes just by inspecting the CPU consumption data.

In the next section, we will present a statistical model for how such CPU consumption data are generated.

4 Data generating model

The computer from which the data were collected, was equipped with a four core CPU processor. We assume that the number of available cores is higher than the number of current requests that need to processed, so that requests are never queued up. More specifically, we assume that the request process can be modelled by a $M/G/\infty$ queue process. This is a reliable assumption since all the four cores were never used at the same time during the 19 days of observations.

4.1 Notation

We assume that we have CPU consumption data for D days. We divide each day into T equidistant intervals by the time points $\tau_0, \ldots, \tau_T$ and define the length of each time interval by $\Delta\tau = \tau_t - \tau_{t-1}$. We let the measurement unit be in days, so that $\tau_0 = 0$ and $\tau_T = 1$. Let $\overline{y}_{dt},\ d = 1, \ldots D,\ t = 1, and \ldots T$ denote the average CPU consumption in the time interval $[\tau_{t-1}, \tau_t]$ at day d. Recall from the introduction that these are our actual observations. Further, let $y_d(\tau)$ denote the current CPU consumption at time τ on day d and recall that it is unobservable since we only observe the average CPU consumption $\overline{y}_{dt}$. Let N_d be the number of requests to the CPU processor on day d and let $a_{d1}, \ldots, a_{dN_d}$ denote the arrival times for each of these requests. Finally, let $s_{d1}, \ldots, s_{dN_d}$ denote the size (processing time) of the requests $a_{d1}, \ldots, a_{dN_d}$. Since we assume a $M/G/\infty$ model, the departure time for request, a_{dn}, will simply be $h_{dn} = a_{dn} + s_{dn}$. To separate the data and realisations from the queue model, we add a superscript 'R' to realisations, e.g. $\overline{y}^R_{dt}$.

4.2 Statistical queuing model

We assume that $N_d,\ d = 1, and \ldots, D$ are independent Poisson-distributed stochastic variables with expectation λ. For a homogeneous Poisson process, we assume that the requests are uniformly distributed throughout the day. From a practical point of view, this is rarely the case and not for the data analysed in this paper, where we observed most of the requests during the day (Section 3). Instead, we assume an inhomogeneous (time varying) Poisson process and assume that the arrival times of the requests are independent Beta-distributed stochastic variables

$$a_{d1}, \ldots, a_{dN_d} \sim \text{Beta}(\alpha, \beta),\ d = 1, \ldots, D$$

By varying the shape parameters α and β, most of the arrivals (CPU requests) will take place at different times during the day. For instance, if high values are chosen for both shape parameters, e.g. $\alpha = \beta = 20$, almost all arrivals (CPU requests) will take place within a short time period in the middle of the day. Setting $\alpha = \beta = 1$, the arrivals will be uniformly distributed throughout the day (homogeneous Poisson process). For more details about the Beta distribution, see e.g. [14]. Finally, we assume that the processing time for each request (i.e. the size of the request) is an independent stochastic variable from a Log-normal distribution

$$s_{d1}, \ldots, s_{dN_d} \sim \text{LogN}(\mu, \sigma),\ d = 1, \ldots, D$$

If X is a normally distributed stochastic variable with expectation μ and standard deviation σ, then $Y = \exp(X)$ will be Log-normal distributed with parameters μ and σ. Of course, other distributions could be used in the model, but for the data analysed in this paper, the Log-normal distribution performed well. Since the requests to the CPU can comprise both very small (short processing time) and very large tasks, we need a distribution that captures this. Taking the exp of outcomes from the normal distribution, we potentially get both very small values and very large values, and the Log-normal distribution is thus suitable. We also experimented with both an exponential and a gamma distribution, but the results were less promising.

Given the arrivals and processing times described above, and recalling that we assume a $M/G/\infty$ model, the current CPU consumption at time τ on day d is given by

$$y_d(\tau) = \sum_{n=1}^{N_d} I(a_{dn} < \tau < h_{dn}) \tag{1}$$

where $I(A)$ is the indicator function, returning one if A is true and zero otherwise. We see that $y_d(\tau)$ is simply the sum of the requests being processed at time τ. Finally, the CPU observations, $\overline{y}_{dt}$, can be computed from $y_d(\tau)$. Recall that $\overline{y}_{dt}$ is the average CPU consumption in the time interval $[\tau_{t-1}, \tau_t]$, which gives

$$\begin{aligned}
\overline{y}_{dt} &= \frac{1}{\Delta\tau} \int_{\tau_{t-1}}^{\tau_t} y_d(\tau)\ \mathrm{d}\tau \\
&= \frac{1}{\Delta\tau} \int_{\tau_{t-1}}^{\tau_t} \sum_{n=1}^{N_d} I(a_{dn} < \tau < h_{dn})\ \mathrm{d}\tau \\
&= \frac{1}{\Delta\tau} \sum_{n=1}^{N_d} \int_{\tau_{t-1}}^{\tau_t} I(a_{dn} < \tau < h_{dn})\ \mathrm{d}\tau \\
&= \sum_{n=1}^{N_d} [F(\tau_t; a_{dn}, h_{dn}) - F(\tau_{t-1}; a_{dn}, h_{dn})]
\end{aligned} \tag{2}$$

where

$$F(\tau; a_{dn}, h_{dn}) = \begin{cases} 0 & \text{if } \tau \leq a_{dn} \\ \dfrac{\tau - a_{dn}}{\Delta\tau} & \text{if } a_{dn} < \tau \leq h_{dn} \\ \dfrac{h_{dn} - a_{dn}}{\Delta\tau} & \text{if } \tau > h_{dn} \end{cases}$$

5 Parameter estimation based on Approximate Bayesian Computing

Given both the observations (Section 3) and the statistical model presented in the previous section, the natural next step is to use the observations to estimate the unknown parameters in the statistical model $\lambda, \alpha, \beta, \mu$ and σ. The most common

way to estimate parameters in statistical models is to optimise the likelihood functions, which in this case can be written as

$$L(\lambda, \alpha, \beta, \mu, \sigma \,|\, \overline{\mathbf{y}}_1, \ldots, \overline{\mathbf{y}}_D) = \prod_{d=1}^{D} \left(\int \int \sum_{N_d=0}^{\infty} \text{Poisson}(N_d; \lambda) \left[\prod_{n=1}^{N_d} \text{Beta}(a_{dn}; \alpha, \beta) \, \text{LogN}(s_{dn}; \mu, \sigma) \right] G(\overline{\mathbf{y}}_d \,|\, \mathbf{a}_d, \mathbf{s}_d) \mathrm{d}\mathbf{a}_d \, \mathrm{d}\mathbf{s}_d \right)$$

where $\overline{\mathbf{y}}_d = \overline{y}_{d1}, \ldots, \overline{y}_{dt}$, $\mathbf{a}_d = a_{d1}, \ldots, a_{dN_d}$, $\mathbf{s}_d = s_{d1}, \ldots, s_{dN_d}$ and $G(\overline{\mathbf{y}}_d \,|\, \mathbf{a}_d, \mathbf{s}_d)$ refers to the relation in (2). Here $G(\overline{\mathbf{y}}_d \,|\, \mathbf{a}_d, \mathbf{s}_d)$ is a delta function since the relation (2) is deterministic. Hence all possible combinations of $N_d, \mathbf{a}_d, \mathbf{s}_d$ that can give rise to the observed data, $\overline{\mathbf{y}}_d$, need to be identified. If we knew N_d and also in which time interval, $[\tau_{t-1}, \tau_t]$, each a_{dn} and $h_{dn} = a_{dn} + s_{dn}$ occurred, (2) can be solved. Unfortunately, the number of ways of positioning the N_d arrivals and N_d departures on T time intervals is given by

$$\binom{2N_d + T - 1}{2N_d}$$

[13], which explodes as either N_d or T increases, and the likelihood function is thus infeasible except for in the most simple cases.

Since the likelihood function is infeasible, we resort to a more indirect way of estimating the parameters. Intuitively, if we generate outcomes from the statistical model in Section 4 using the true parameter values (that generated the real data), we would expect outcomes from the statistical model to be similar to the real data. Fortunately, it is simple to generate outcomes from the statistical model in Section 4, which means that such an indirect approach is possible. Such an approach dates back to at least [23]. Recently, more efficient approaches have been developed. They are called Approximate Bayesian Computing (ABC). See [5] and [11] for good reviews of different ABC approaches.

The methodology can be described as follows. Suppose that we have a statistical model with parameter θ. Let $p(\theta)$ denote the prior distribution of the parameter and let Υ_o denote the observations that were generated from the statistical model with the unknown parameter θ_o. The goal is thus to estimate θ_o. The estimation procedure can then be described as follows:

1. Generate a large set of realisations $\theta_1, \ldots, \theta_M$ from the prior distribution $p(\theta)$.
2. For each realisation θ_i, generate an outcome from the statistical model Υ_i.
3. Compute a set of K different statistics (sample mean, variance etc.) for each outcome Υ_i, $S_1(\Upsilon_i), \ldots, S_K(\Upsilon_i)$. These statistics summarise the main properties of the data.
4. Compute the same statistics for the observations $S_1(\Upsilon_o), \ldots, S_K(\Upsilon_o)$. Intuitively, if $S_1(\Upsilon_i), \ldots, S_K(\Upsilon_i)$ is close to $S_1(\Upsilon_o), \ldots, S_K(\Upsilon_o)$ using some suitable metric, we expect that θ_i will be close to the unknown true parameter θ_o.
5. Fit a statistical model to the relation between the parameters and the statistics using the realisations θ_i and $S_1(\Upsilon_i), \ldots, S_K(\Upsilon_i)$, $i = 1, \ldots, M$. We let θ be the response of the model (although the opposite is also possible).

6. Use the statistical model to obtain inference on the unknown parameter θ_o. A simple approach is to just plug the observed statistics $S_1(\Upsilon_o), \ldots, S_K(\Upsilon_o)$ into the statistical model and use the output from the model in Step 5 as the estimate of θ_o.

The interested reader is referred to [5] and [11] for more details. In our case, θ refers to the parameters $\lambda, \alpha, \beta, \mu, \sigma$ and Υ to the outcomes from the model in Section 4, i.e. $\overline{\mathbf{y}}_d^R,\ d = 1, \ldots, D$.

5.1 CPU data statistics

A crucial part of the ABC methodology described in the previous section is choosing statistics that are able to distinguish properties of realisations from the model in Section 4 for different values of the parameters $\lambda, \alpha, \beta, \mu, \sigma$. Thus, we started by generating realisations from the model for different values of the parameters. We fixed the expected CPU consumption for outputs from the model as equal to the value in the observations, i.e.

$$\begin{aligned} E(N_d)E(s_{dn}) &= \frac{1}{D}\sum_{d=1}^{D}\overline{\mathbf{y}}_d \\ \lambda \exp\left(\mu + \sigma^2/2\right) &= \frac{1}{D}\sum_{d=1}^{D}\overline{\mathbf{y}}_d \end{aligned} \tag{3}$$

where the expectations on the left-hand side follow from the properties of the Poisson and Log-normal distributions. We thus choose combinations of λ, μ and σ that satisfy this relation. We start by setting $\lambda = 50$, μ equal to the four values $-6, -7.5, -9$ and -10.5 and σ were chosen to satisfy (3). Finally we set $\alpha = \beta = 3$, which resulted in a few more requests happening during the day, which is in accordance with the real data. To study the effects of adjusting the parametric values, we generated several realisations from the model for the different parametric choices. One realisation from each of the values of μ is shown in Figure 2. We see that setting $\mu = -6$ (upper left panel), each CPU request is small and about the same size. For $\mu = -9$ (lower left panel), we see that the output from the model contains both very small tasks and large tasks (the CPU consumption is high for long time periods). For $\mu = -10.5$ (lower right panel), this happens to an even larger degree. Comparing this to the real data in Figure 1, the lower left panel in Figure 2 seems to be most similar to the real data. However, there seem to be more spikes in the real data, indicating that $\lambda = 50$ is too low a value to replicate the properties of the real data.

Figure 3 shows the same as Figure 2 except that we increased λ to 1000. Since we now get far more requests to the CPU, the expected size of each request must be smaller and we set $\mu = -9, -11, -13$ and -15. Comparing Figures 2 and 3, we see that by increasing λ we get more spiky data, which is in accordance with the real data. The lower right panel of Figure 3 seems to replicate properties of the real data quite well.

Figures 2 and 3 show that, by adjusting the parameters of the model, we end up with CPU data with different properties. We saw that the ABC method relies on

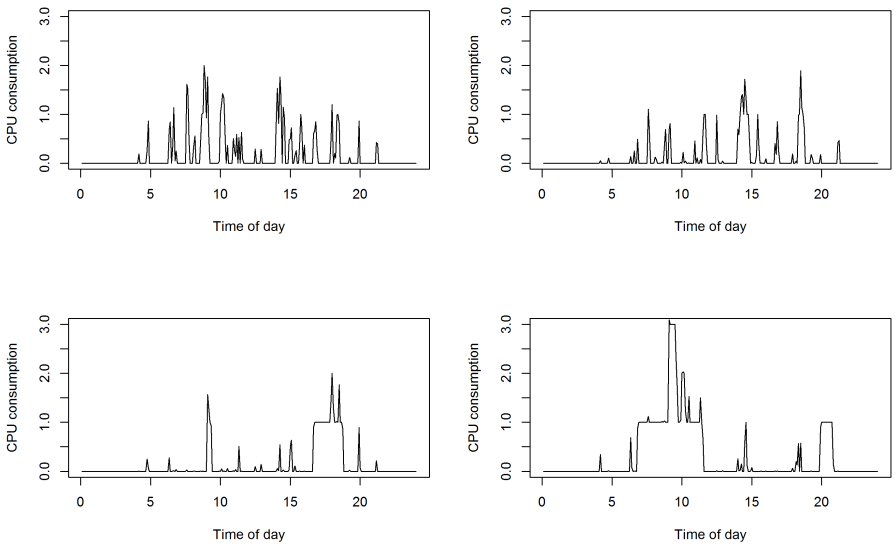

Fig. 2 Realisations from the model in Section 4 for $\lambda = 50$. For the panels from upper left to lower right, $\mu = -6, -7.5, -9$ and -10.5, respectively.

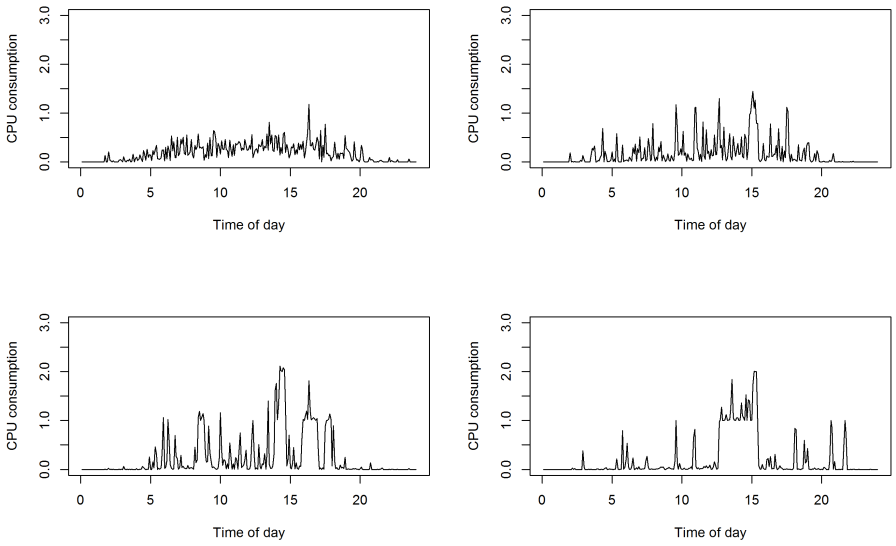

Fig. 3 Realisations from the model in Section 4 for $\lambda = 1000$. For the panels from upper left to lower right, $\mu = -9, -11, -13$ and -15, respectively.

finding suitable statistics of the real data. In the experiments, we use the following statistics.

- Average CPU consumption.

$$\overline{\mathbf{y}} = \frac{1}{D}\sum_{d=1}^{D}\overline{\mathbf{y}}_d$$

- Cumulative probabilities. Define

$$P_q = \frac{1}{DT}\sum_{d=1}^{D}\sum_{t=1}^{T} I(\overline{y}_{dt} < q)$$

  being the proportion of observations with a value less than q (quantile). We see that with $\lambda = 50$ compared to $\lambda = 1000$, a CPU consumption equal to zero, one and two is far more common, since we have less disturbances from zero, one or two cores running. The same is observed if we use a lower value of μ (lower right panel in Figures 2 and 3). Thus, choosing values of q close to these values seems reasonable. In the estimation procedure, we therefore use the following values of q: 0.01, 0.1, 0.5, 0.9, 0.99, 1.01, 1.1, 1.5, 1.9, 1.99, 2.01, 2.1 and 2.5.
- To measure the degree of spikiness, we compute the change in CPU consumption from one time step to the next

$$\frac{1}{D(T-1)}\sum_{d=1}^{D}\sum_{t=2}^{T}|\overline{y}_{dt} - \overline{y}_{dt-1}|$$

- We expect the spikiness measure above to be higher if the average CPU consumption is high. Therefore we also compute the relative change from one time step to the next.

$$\frac{1}{D(T-1)}\sum_{d=1}^{D}\sum_{t=2}^{T}\frac{|\overline{y}_{dt} - \overline{y}_{dt-1}|}{MA(t)}$$

  where $MA(t)$ is a moving average estimate of the expected CPU consumption at different times during the day

$$MA(t) = \frac{1}{D(2L-1)}\sum_{d=1}^{D}\sum_{l=-L}^{L}\overline{y}_{d(t-l)}$$

  In the experiments, we used $L = 5$. We use the moving average instead of the CPU consumption observations directly since the observations are very spiky, resulting in unstable estimates.
- Changes in average load from one day to another. From Figure 1, we see that in the real data the average load varies quite a lot from one day to another, and we should have the same property in the model. There we include the standard deviation in the daily averages

$$\mathrm{SD}_{\overline{y}} = \sqrt{\frac{1}{D-1}\sum_{d=1}^{D}(\overline{\mathbf{y}}_d - \overline{\mathbf{y}})^2}$$

Typically choosing μ and σ, so that the Log-normal distribution becomes more heavy tailed, like the lower right panels in Figures 2 and 3, the standard deviation of the daily average increases.

– Finally, we need some statistics to capture the inhomogeneity in requests during the day, i.e. to estimate the parameters α and β. Let V be a stochastic variable with probabilities

$$P(V = \tau_t) = \frac{MA(t)}{\sum_{t=1}^{T} MA(t)},\ t = 1, 2, \ldots, T$$

which can be interpreted as the probability of requests to the CPU at different times during the day. We now compute the expectation and standard deviations of V and use these values as statistics

$$E(V) = \sum_{t=1}^{T} \tau_t\, P(V = \tau_t)$$

$$\mathrm{SD}(V) = \sqrt{\sum_{t=1}^{T} (\tau_t - E(V))^2\, P(V = \tau_t)}$$

If most of the requests take place early (late) during the day, $E(V)$ will have a low (high) value. The parameters α and β are directly related to the statistics $E(V)$ and $SD(V)$.

6 Forecasting future CPU consumption

In this section we present procedures for forecasting future CPU consumption. The first procedure forecasts the average CPU consumption in disjoint time intervals, while the second procedure forecasts the *instantaneous* CPU consumption at *any* time point in the future. Only by taking into consideration the underlying queue process is it possible to forecast the instantaneous CPU consumption at any time point in the future. This means that the HMM in [16] or any time series models is limited to only forecasting the average CPU consumption and not the instantaneous CPU consumption.

We expect the queue model to forecast well since it is able to capture the complex time dependency patterns of the underlying queue process.

6.1 Forecasting average CPU consumption

We assume that we have estimated the parameters of the queue model from historic CPU consumption data using the procedure above. Further, we assume that we have received observations on a new day, d_0, up to time t, $\overline{y}_{d_0 1}, \overline{y}_{d_0 2}, \ldots, \overline{y}_{d_0 t}$. We now want to forecast the future CPU consumption $\overline{y}_{d_0 t+1}, \overline{y}_{d_0 t+2}, \ldots$. We suggest the following procedure:

1. Start by generating realisations from the queue model for P days, $\overline{\mathbf{y}}_p^R$, $p = 1, 2, \ldots, P$, using parameter values estimated from the historic CPU consumption data.

2. Compare the values of each of the realisations and the observations in a neighbourhood up to time t. We consider two metrics:
   - Interval of length S: $d_p = \sum_{h=0}^{S} \left(\overline{y}_{d_o,t-h} - \overline{y}^R_{p,t-h}\right)^2$.
   - Exponential decay: $d_p = \sum_{h=0}^{t-1} e^{-\gamma h} \left(\overline{y}_{d_o,t-h} - \overline{y}^R_{p,t-h}\right)^2$.
3. Select the Ψ realisations with the smallest d_p. Denote the selected realisations $\overline{\mathbf{y}}^R_{p(\psi)}, \psi = 1, \ldots, \Psi$.
4. Forecast the CPU consumption at time $t+g$, denoted $\widehat{\overline{y}}_{d_0,t+g}$, using one of the following two strategies
   - Average:
   $$\widehat{\overline{y}}_{d_0,t+g} = \frac{1}{\Psi} \sum_{\psi=1}^{\Psi} \overline{y}^R_{p(\psi),t+g}$$
   In the experiments we denote this strategy AV.
   - Weighted average:
   $$\widehat{\overline{y}}_{d_0,t+g} = \frac{1}{W} \sum_{\psi=1}^{\Psi} \frac{1}{d_{p(\psi)}} \overline{y}^R_{p(\psi),t+g}$$
   where $W = \sum_{\psi=1}^{\Psi} d^{-1}_{p(\psi)}$. Intuitively, we should give more weight to the realisations closest to the observations. In the experiments we denote this strategy WAV.

The intuitive thinking behind the procedure is that realisations that are close to the observations in a time period up to time t should have more or less the same underlying arrival and departure times in the neighbourhood of time t, and the realisations should therefore also be close to the observations in the neighbourhood after time t. The forecast procedure is able to take into account the complex dependency structures of the underlying queue model and lay the foundation for precise forecasting.

6.2 Forecasting instantaneous CPU consumption at any time point

The procedure above forecasts the *averge* CPU consumption in disjoint time intervals in the future. It could be argued that it is rather the forecasting of the *instantaneous* CPU consumption at *any* time point in the future that is of main interest. By making small modifications to the procedure above, we can forecast this. Suppose that we want to forecast the CPU consumption at some arbitrary time point in the future $\tau' > \tau_t$. Note that τ' can be *any* time point and not limited by the discretisation of the observations. We modify the procedure above as follows. In the third step above, for each of the selected realisations, we also store the arrival and departure times. We denote them $a^R_{p(\psi),1}, \ldots, a^R_{p(\psi),N_{p(\psi)}}$ and $h^R_{p(\psi),1}, \ldots, h^R_{p(\psi),N_{p(\psi)}}$, $\psi = 1, \ldots, \Psi$. The fourth step is modified as follows. For each selected realisation, we can compute the CPU consumption at time τ' using (1)

$$y^R_{p(\psi)}(\tau') = \sum_{n=1}^{N_{p(\psi)}} I\left(a^R_{p(\psi),n} < \tau' < h^R_{p(\psi),n}\right), \quad \psi = 1, \ldots, \Psi$$

We can further compute a probability distribution for the number of running CPU cores at time τ' (which is the CPU consumption at time τ'). For instance, the probability that x CPU cores will be running at time τ' can be estimated as

$$P(y_{d_0}(\tau') = x) = \frac{1}{\Psi} \sum_{\psi=1}^{\Psi} I\left(y^R_{p(\psi)}\left(\tau'\right) = x\right), \quad x = 0, 1, 2, \ldots \tag{4}$$

or using a weighted sum depending on $d_{p(\psi)}$.

7 Experiments

In the first part of the experiments, we evaluated how well we were able to estimate the parameters of the queuing model. In the next part, we performed a thorough evaluation of the forecasting performance of the methods described in Section 6. We compared it with different ARIMA models and the inhomogeneous hidden Markov model in [16].

7.1 Parameter estimation

In the previous sections, we have built a realistic queue model for the CPU consumption data and an estimation strategy based on the ABC framework. In this Section, we will evaluate the estimation performance of the ABC procedure in Section 5. The ABC procedure requires prior distributions for the unknown parameters. We assume that we have little prior information and chose wide uniformly distributed priors as follows

$$\begin{aligned} p(\lambda) &= \text{Unif}(50, 10000) \\ p(\alpha) &= \text{Unif}(0.5, 10) \\ p(\beta) &= \text{Unif}(0.5, 10) \\ p(\mu) &= \text{Unif}(-25, -5) \\ p(\sigma) &= \text{Unif}(0.5, 7) \end{aligned}$$

where $\text{Unif}(a, b)$ refers to the uniform distribution in the interval between a and b.

We started the parameter estimation by following the first three steps of the ABC procedure described in Section 5 by generating $M = 4 \cdot 10^6$ realisations and computing the statistics presented in Section 5.1.

In the fifth step of the procedure, different statistical models can be used, such as partial least square [29], neural net [4] or ridge regression [25]. We evaluated both the neural net and the ridge regression approach. and the two approaches resulted in very similar results. The results below are based on the neural net approach.

In the rest of the paper, we set $D = 19$ and $T = 288$. which means that we have 288 observations per day (every five minutes) for 19 days. This is in accordance with the real data presented in Section 3.

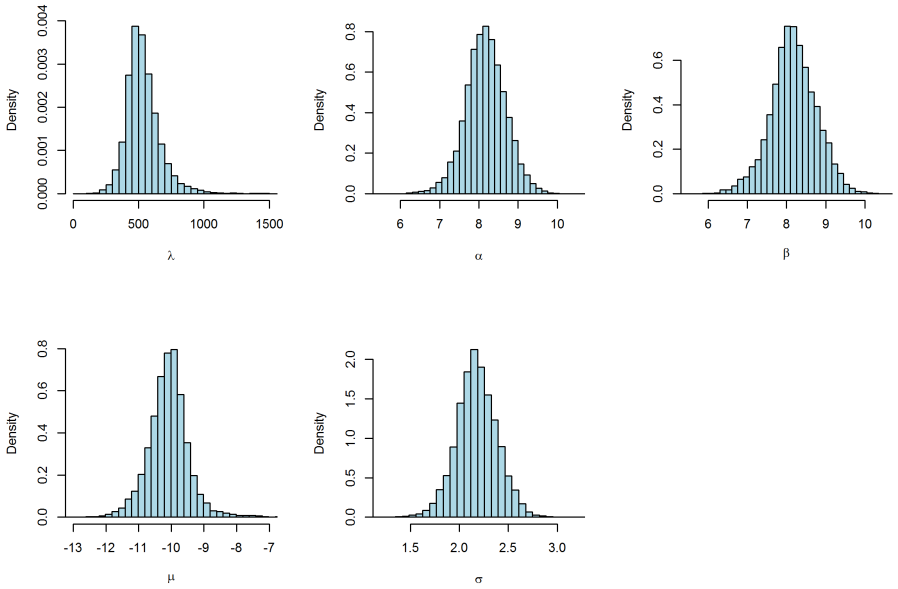

Fig. 4 Synthetic example: Realisations from the ABC posterior distribution for the different parameters of the model.

7.1.1 Synthetic data example

We started by generating a synthetic dataset over $D = 19$ days from the model with parameter values $\lambda = 500, \alpha = 8, \beta = 8, \mu = -10, \sigma = 2.2$ and computed the statistics from Section 5.1. If the estimation procedure performs well, we should get reliable estimates of the parameters used to generate the synthetic data. Figure 4 shows a histogram of outcomes from the ABC posterior distribution for the different parameters. We see that the procedure estimates the parameters of the underlying queue process very well.

7.1.2 Real CPU consumption data example

Next, we ran the estimation procedure for the real data. Figure 5 shows a histogram of outcomes from the ABC posterior distribution for the different parameters. We see that we are able to get fairly precise estimates of the parameters in the queuing model. In particular, the results reveal that the expected number of requests to the CPU processor during a day is somewhere between 1000 (a request about every 1.5 minutes) and 5000 (a request about every 15 seconds). We see $\alpha \approx \beta \approx 3.5$, which means that most of the requests to the CPU processor take place in the middle of the day (office hours), but 3.5 is not a very high value, so a fair amount of the requests also happen at other times during the day.

Figure 6 shows four arbitrary realisations from the queue model using the estimated parameters with the highest posterior probability (the MAP parameters). By comparing Figures 1 and 6, we see that outcomes from the statistical model

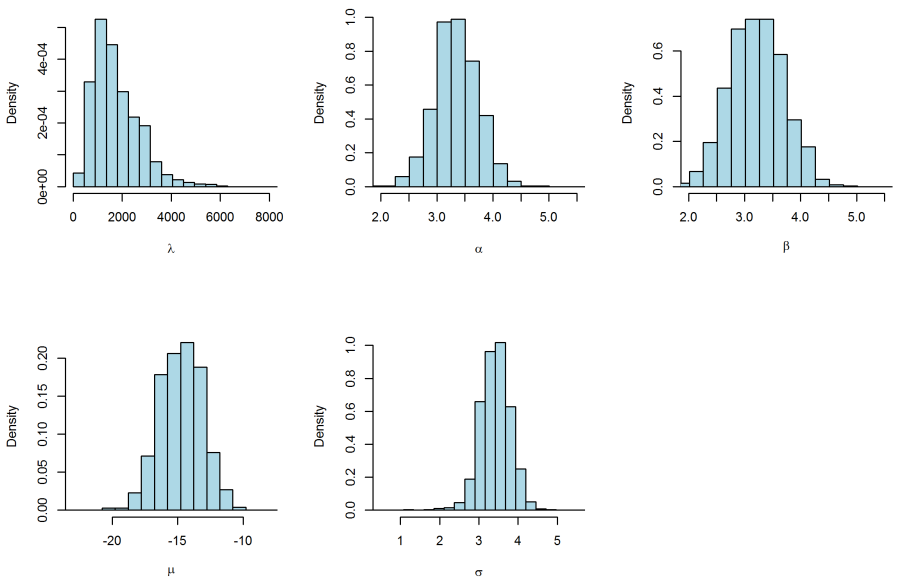

Fig. 5 Real data example: Realisations from the ABC posterior distribution for the different parameters of the model.

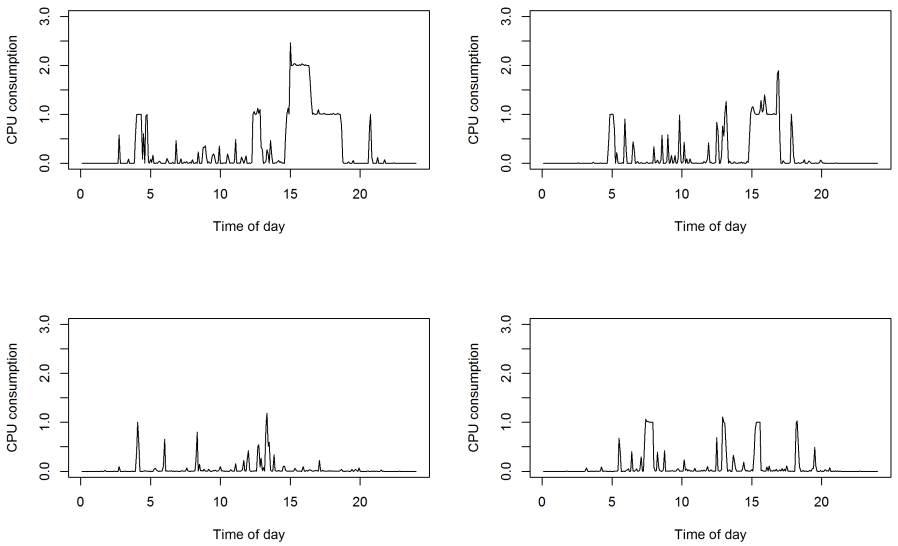

Fig. 6 Real data example: Four arbitrary realisations from the statistical model in Section 4 using the ABC MAP estimate of the model parameters.

replicate the properties of the real CPU consumption data very well.

7.2 Forecasting

In this section, we evaluate the forecasting performance of the queue model. In Sections 7.2.1 and 7.2.2, we evaluate the performance of forecasting average and instantaneous CPU consumption, respectively, using the two procedures in Section 6.

7.2.1 Forecasting of average CPU consumption

We compared it with three alternative models.

- The hidden Markov model from [16], denoted HMM CPU in the rest of the paper
- An autoregressive process of order one, denoted AR(1) in the rest of the paper.
- An ARIMA model. For the ARIMA model, we used the `auto.arima` function in the `forecast` package [19] in the statistical software R [26]. More details on the `auto.arima` function can be found in Section 3 of [19]. There is clearly a daily seasonality in the CPU consumption data that needs to be included in the model. The `auto.arima` function has a method for doing this, but it performed poorly for our data since the data consist of many observations per day (every fifth minute). We therefore followed the recommendations in [20] and modelled the seasonality using a Fourier series approach. We used the Akaike information criterion to choose the number of frequencies in the Fourier series, ending up with three frequencies ($K = 3$ following the notation in [20]).

We forecast future CPU consumption using the procedure in Section 6.1 with the following choices.

- In the first step of the procedure, we generated realisations from the queue model for $P = 100\,000$ days using the MAP estimates of the parameters in the model.
- In the second step of the procedure, we used three different values of S, namely $S = 2$, $S = 6$ and $S = 12$ referring to 10, 30 and 60 minutes backwards in time, respectively. For the exponential decay approach, we chose γ, so that the weight was reduced to 0.5 in 10, 30 and 60 minutes, which was achieved by setting $\gamma = \ln(2)/2, \gamma = \ln(2)/6$ and $\gamma = \ln(2)/12$, respectively.
- In the third step, we used two values for Ψ, namely $\Psi = 50$ and $\Psi = 500$.
- In the fourth step, we used both of the strategies AV and WAV.

We measured forecasting performance for all the combinations of the choices above, ending up with $6 \cdot 2 \cdot 2 = 24$ unique cases.

We evaluated the forecasting performance using leave-one-day-out cross validation. This means that we left out one of the $D = 19$ days to test the forecasting performance and the 18 other days to train the different models above. We repeated this 19 times, so that each day was used as the test set. We measured the forecasting error using the average difference in absolute value between the

Model	Compare	Ψ	AV-WAV	Error
QUEUE	$\gamma = \ln(2)/6$	500	WAV	0.102
QUEUE	$\gamma = \ln(2)/6$	50	WAV	0.103
QUEUE	$S = 6$	500	WAV	0.104
QUEUE	$\gamma = \ln(2)/6$	500	AV	0.104
QUEUE	$S = 6$	50	AV	0.105
QUEUE	$S = 6$	50	WAV	0.105
QUEUE	$S = 6$	500	AV	0.105
QUEUE	$\gamma = \ln(2)/6$	50	AV	0.105
QUEUE	$\gamma = \ln(2)/12$	500	WAV	0.108
QUEUE	$\gamma = \ln(2)/12$	500	AV	0.109
QUEUE	$\gamma = \ln(2)/12$	50	AV	0.111
HMM CPU	–	–	–	0.111
ARIMA	–	–	–	0.112
QUEUE	$\gamma = \ln(2)/2$	500	WAV	0.112
QUEUE	$\gamma = \ln(2)/12$	50	WAV	0.112
QUEUE	$\gamma = \ln(2)/2$	50	WAV	0.113
QUEUE	$\gamma = \ln(2)/2$	500	AV	0.113
QUEUE	$\gamma = \ln(2)/2$	50	AV	0.115
QUEUE	$S = 2$	50	AV	0.117
QUEUE	$S = 2$	500	WAV	0.117
QUEUE	$S = 12$	50	WAV	0.117
QUEUE	$S = 12$	500	AV	0.117
QUEUE	$S = 12$	500	WAV	0.117
QUEUE	$S = 2$	50	WAV	0.118
QUEUE	$S = 12$	50	AV	0.118
QUEUE	$S = 2$	500	AV	0.119
AR(1)	–	–	–	0.135

Table 1 Forecasting performance of the different models. The columns from left to right show: Model, choice in step two of the procedure in Section 6, choice in step three, choice in step four and the forecasting error.

forecasting and the observation

$$\text{Error } = \frac{1}{T}\sum_{t=1}^{T}\left|\widehat{\overline{y}}_t - \overline{y}_t\right|$$

where $\widehat{\overline{y}}_t$ and $\overline{y}_t$ refer to the forecasting and observation at time t during the test day. We computed the forecasting error, forecasting 5, 10, 15 and 20 minutes into the future.

Table 1 shows the results. The forecasting errors in the table are computed as the average of the forecasting error going 5, 10, 15 and 20 minutes into the future. We see that the AR(1) model performs more poorly than the more advanced models. The ARIMA and the HMM CPU models perform about equally well. The performance of the queue model in this paper depends on the choice of the parameters. The choice of S and γ seems to be quite important. The method performs very well using a history of about 30 minutes($\gamma = \ln(2)/6$ or $S = 6$) and outperforms both the ARIMA and the HMM CPU models. Using a history of 10 minutes or one hour, the performance mostly drops below ARIMA and HMM CPU. It also seem that combining $\Phi = 500$ and WAV results in a better performance than $\Phi = 50$ and AV, which means that it is better to select many realisations and weight them than to select only a few without weighting.

Figure 7 shows the forecasting performance of the queue model with the best parameter choices and compared to HMM CPU, ARIMA and AR(1). We see that

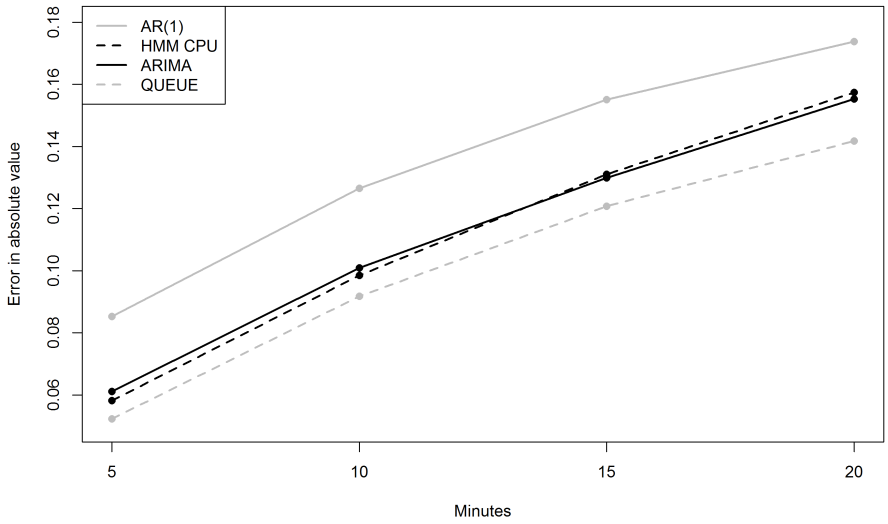

Fig. 7 Forecasting performance of the queue model with the best parameter choices and compared to HMM CPU, ARIMA and AR(1).

the queue model outperforms the other models for all forecasting time horizons.

7.2.2 Forecasting instantaneous CPU consumption

As described earlier in the paper, neither the HMM model in [16] nor the time series models used for comparison in the previous section can be used to forecast instantaneous CPU consumption, and we therefore do not have any models to compare with. Instead of carrying out a comparison, we run a small experiment to demonstrate the results of instantaneous forecasting. We select a time point with high CPU consumption and forecast the instantaneous CPU consumption from 30 seconds to 20 minutes into the future.

We use the forecasting procedure in Section 6.2 with $\gamma = \ln(2)/6$ and $\Psi = 500$, which were the best choices in the experiments above. Further, we use (4) to compute the probabilities. The results are shown in Figure 8. The black curve shows the CPU consumption data in a time span of 50 minutes, 30 minutes before the forecasting starts and 20 minutes after. We see that for the first 30 minutes the CPU consumption is constantly about one, meaning that the CPU is processing a large task. The grey curves show the forecasting of the instantaneous CPU consumption. We see that the procedure forecasts a high probability of one core running, which seems reasonable since the CPU was processing a large task when the forecasting started. Going further into the future, the probability of one core running is gradually reduced to about 0.8 after 20 minutes.

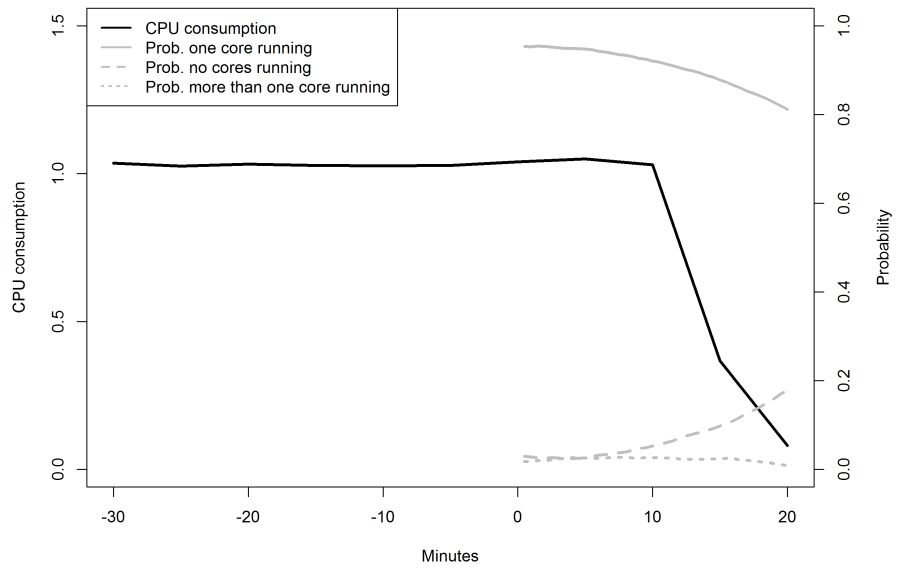

Fig. 8 Forecasting instantaneous CPU consumption from 30 seconds to 20 minutes into the future. The CPU consumption (black curve) is read on the left y-axis, while probabilities (grey curves) are read on the right y-axis.

8 Closing remarks

In this paper, we build a statistical model to replicate how observed CPU consumption data are generated. We demonstrate that, by adjusting the parameters of the statistical model, the outcomes of the model have different properties. Next, we build statistics that quantify these differences and are used to estimate the parameters of the statistical model to replicate the properties of real CPU consumption data. Our results show that realisations from the queue model replicate the properties of real CPU consumption data very well.

Further, we show that the queue model outperforms both the HMM CPU model from [16] and an ARIMA model in forecasting future CPU consumption. Another good property of the queue model is that we can forecast the *instantaneous* CPU consumption at *any* time point in the future following the second procedure in Section 6. This is only possible by taking into account the underlying queuing process that generated the real CPU consumption data. Therefore, neither the HMM CPU in [16] nor ARIMA models can forecast this, but only the *average* CPU consumption at disjoint time intervals in the future.

It could be argued that our $M/G/\infty$ modelling of the CPU processor is a little simplified compared to how modern multi-core CPU processors work. The procedure in this paper only requires that we are able to generate outcomes from the statistical model, which means that the $M/G/\infty$ assumption can be replaced by other and more complex assumptions without affecting the usefulness of the approach.

References

1. SK Acharya. On normal approximation for maximum likelihood estimation from single server queues. *Queueing systems*, 31(3-4):207–216, 1999.
2. Vineet Aggarwal, Natarajan Gautam, Soundar RT Kumara, and Mark Greaves. Stochastic fluid flow models for determining optimal switching thresholds. *Performance Evaluation*, 59(1):19–46, 2005.
3. Farid Benhammadi, Zahia Gessoum, Aicha Mokhtari, et al. Cpu load prediction using neuro-fuzzy and bayesian inferences. *Neurocomputing*, 74(10):1606–1616, 2011.
4. Michael GB Blum and Olivier François. Non-linear regression models for approximate bayesian computation. *Statistics and Computing*, 20(1):63–73, 2010.
5. Michael GB Blum, Maria Antonieta Nunes, Dennis Prangle, Scott A Sisson, et al. A comparative review of dimension reduction methods in approximate bayesian computation. *Statistical Science*, 28(2):189–208, 2013.
6. Brian Bouterse and Harry Perros. Scheduling cloud capacity for time-varying customer demand. In *2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET)*, pages 137–142. IEEE, 2012.
7. Konstantinos Christodoulopoulos, Vasileios Gkamas, and Emmanouel A Varvarigos. Statistical analysis and modeling of jobs in a grid environment. *Journal of Grid Computing*, 6(1):77–101, 2008.
8. DR Cox. The statistical analysis of congestion. *Journal of the Royal Statistical Society. Series A (General)*, 118(3):324–335, 1955.
9. Peter A Dinda. Online prediction of the running time of tasks. *Cluster Computing*, 5(3):225–236, 2002.
10. Peter August Dinda. *Resource Signal Prediction and Its Application to Real-time Scheduling Advisors*. PhD thesis, Pittsburgh, PA, USA, 2000. AAI9986588.
11. Christopher C Drovandi, Anthony N Pettitt, Anthony Lee, et al. Bayesian indirect inference using a parametric auxiliary model. *Statistical Science*, 30(1):72–95, 2015.
12. Paul Fearnhead. Filtering recursions for calculating likelihoods for queues based on inter-departure time data. *Statistics and Computing*, 14(3):261–266, 2004.
13. William Feller. An introduction to probability theory and its applications. Vol. I. 1950.
14. Catherine Forbes, Merran Evans, Nicholas Hastings, and Brian Peacock. *Statistical distributions*. John Wiley & Sons, 2011.
15. Lizheng Guo, Tao Yan, Shuguang Zhao, and Changyuan Jiang. Dynamic performance optimization for cloud computing using m/m/m queueing system. *Journal of Applied Mathematics*, 2014, 2014.
16. Hugo Lewi Hammer, Anis Yazidi, and Kyrre Begnum. An Inhomogeneous Hidden Markov Model for Efficient Virtual Machine Placement in Cloud Computing Environments. *Journal of Forecasting*, 2016.
17. Hugo Lewi Hammer, Anis Yazidi, Alfred Bratterud, Hårek Haugerud, and Boning Feng. Recovering Request Patterns to a CPU Processor from Observed CPU Consumption Data. In *Industrial Networks and Intelligent Systems*, pages 14–28. Springer, 2017.
18. Knut Heggland and Arnoldo Frigessi. Estimating functions in indirect inference. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 66(2):447–462, 2004.
19. RJ Hyndman and Y Khandakar. Automatic time series forecasting: the forecast package for r. *Journal of Statistical Software*, 26(3):1–22, 2008.
20. Hyndman, RJ. Forecasting with long seasonal periods. http://robjhyndman.com/hyndsight/longseasonality/, Sep 2010.
21. Sudha Jain. Relative efficiency of a parameter for a M/G/1 queueing system based on reduced and full likelihood functions. *Communications in Statistics-Simulation and Computation*, 21(2):597–606, 1992.
22. Jong Kim and Kang G Shin. Execution time analysis of communicating tasks in distributed systems. *Computers, IEEE Transactions on*, 45(5):572–579, 1996.
23. Daniel McFadden. A method of simulated moments for estimation of discrete response models without numerical integration. *Econometrica: Journal of the Econometric Society*, pages 995–1026, 1989.
24. Jing Mei, Kenli Li, Aijia Ouyang, and Keqin Li. A profit maximization scheme with guaranteed quality of service in cloud computing. *Computers, IEEE Transactions on*, 64(11):3064–3078, 2015.
25. Gisela Muniz and BM Golam Kibria. On some ridge regression estimators: An empirical comparisons. *Communications in Statistics-Simulation and Computation*, 38(3):621–630, 2009.

26. R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015.
27. Sena Seneviratne and David C Levy. Task profiling model for load profile prediction. *Future Generation Computer Systems*, 27(3):245–255, 2011.
28. Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 5. ACM, 2011.
29. Michael Sjöström, Svante Wold, Walter Lindberg, Jan-Åke Persson, and Harald Martens. A multivariate calibration problem in analytical chemistry solved by partial least-squares models in latent variables. *Analytica Chimica Acta*, 150:61–70, 1983.
30. Rich Wolski, Neil T Spring, and Jim Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5):757–768, 1999.
31. Wei Xiong and Tayfur Altiok. Queueing analysis of a server node in transaction processing middleware systems. *Computers & Operations Research*, 35(8):2561–2578, 2008.
32. Jing Xu and José Fortes. A multi-objective approach to virtual machine management in datacenters. In *Proceedings of the 8th ACM international conference on Autonomic computing*, pages 225–234. ACM, 2011.
33. Dingyu Yang, Jian Cao, Jiwen Fu, Jie Wang, and Jianmei Guo. A pattern fusion model for multi-step-ahead cpu load prediction. *Journal of Systems and Software*, 86(5):1257–1266, 2013.
34. Lingyun Yang, Ian Foster, and Jennifer M Schopf. Homeostatic and tendency-based cpu load predictions. In *2003 IEEE International Parallel and Distributed Processing Symposium*, pages 9–pp. IEEE, 2003.
35. Mustafa Yuzukirmizi and J MacGregor Smith. Optimal buffer allocation in finite closed networks with multiple servers. *Computers & Operations Research*, 35(8):2579–2598, 2008.
36. Yuanyuan Zhang, SUN Wei, and Yasushi Inoguchi. Cpu load predictions on the computational grid. *IEICE TRANSACTIONS on Information and Systems*, 90(1):40–47, 2007.
37. Zhongju Zhang and Weiguo Fan. Web server load balancing: A queueing analysis. *European Journal of Operational Research*, 186(2):681–693, 2008.