Recovering Request Patterns to a CPU Processor From Observed CPU Consumption Data

Hugo Lewi Hammer, Anis Yazidi, Alfred Bratterud, Hårek Haugerud, and Boning Feng

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

`hugo.hammer@hioa.no`, `anis.yazidi@hioa.no`, `alfred.bratterud@hioa.no`, `harek.haugerud@hioa.no`, `boning.feng@hioa.no`

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 are 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.

Keywords: classification, co-occurrence information, text mining, tweets

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 follow some statistical queuing model, see e.g. [14, 5, 22, 24, 23, 11, 16] for a few representative examples. The typical approach in such papers is to run artificial stress tests on the systems 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 live settings. Unfortunately, such observations do not reveal the properties of the underlying real live request process to the system which contains important information on how the computer system should be constructed, 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.

2   Related work

Applying queuing models for analyzing resource usage for different computer systems have received a lot of attention. We shall review some representative studies on this topic.

In [17], a double renting scheme that combines short term renting and long term renting is proposed. The service system is modeled as $M/M/m + D$ queuing model. An optimal configuration for profit optimization is derived subject to guaranteeing the service quality of all requests. Many optimization factors are considered for the profit optimization problem which includes the market demand, the workload of requests, the server-level agreement, the rental cost of servers, the cost of energy consumption etc. The authors treat the case of homogeneous cloud environment and the case of heterogeneous is left for future work as it is more involved.

In [15], Liu and his colleagues study the effect of cloud scheduling on service performance. Unlike most legacy models, the service rate is not considered fixed but rather depends on the schedule strategy. Each server is modeled as an $M/G/1$ queuing system, while a semi-Markov model is used to model the cloud computing center.

In [21], an open Jackson network is used to model a cloud platform and used to provide SLA guarantee in terms of response time. A sequential model composed of $M/M/1$ and $M/M/m$ in sequence was proposed to model the cloud platform. The work is particularly useful for dimensioning the cloud as it is able to determine the system bottleneck parameters that needs to be adjusted so that to guarantee the desired response time. The work can be applied to enable dimensioning cloud platform to host different types of services (education, production, e-health, etc.) .

In [19], a queue model for multimedia cloud is proposed. The model is composed of three concatenated queues: are the schedule queue, the computation queue, and the transmission. The aim is to efficiently host multi-media services in the cloud by optimizing the quality of service QoS and resource cost. queue.

Bouterse and Perros [4] 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.

Parameter estimation in queuing models have a long tradition going back to the David Cox paper from [6]. Most papers rely on the likelihood principle in one way or another, see e.g. [13, 1, 8], but there are some situations where standard estimation techniques can not be used. E.g. Heggland and Frigessi [12] 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. [3, 7] 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.

3   CPU consumption data

In this paper we apply the methodology described in the introduction on real CPU consumption data collected from a laptop with a Windows 7 operating system used by an office worker. We observed the office worker for 19 whole working days (i.e. weekends removed) with the average CPU usage in five minutes 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 constantly running. We observe that the CPU consumption typically is largest during the day and with some occasional computations during the evening. We observe little CPU consumption during the night. Large tasks are visible in the real data with
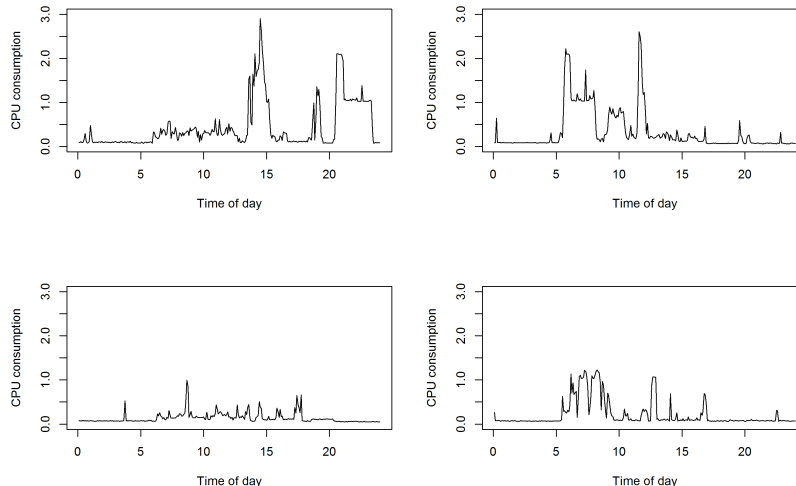
Fig. 1. The CPU consumption data for four arbitrary days.

a constant CPU usage around 1, e.g. in the evening of the upper left panel. In the same panel we observe that the CPU consumption is around 2 right after 8 PM (value 20 on the x axis) meaning that two CPU cores run some larger tasks. We also observe large time spans in which the CPU usage is far below one meaning that in this periods a number of smaller requests are sent to the CPU processor. Even though we do not observe the CPU requests directly we see that we still are able to say something about the amount of request and the sizes of them 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. The model will then be used to recover unobserved properties of the underlying request process to the CPU processor.

4 Data generating model

The laptop, in which the data was collected from, was equipped with a four core CPU processor. We assume that the number of available cores are more than the number of current requests that need to processed such that requests are never queued. More specifically we assume that the request process can be modeled by a $M/G/\infty$ queue process.

4.1 Notation

We assume that we have CPU consumption data for D days. We divide each day in 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 $\bar{y}_{dt}, \ d = 1, \ldots D, \ t = 1, \ldots T$ denote the average CPU consumption on the time interval $[\tau_{t-1}, \tau_t]$ at day d. Recall from the sections above that this is in fact our observations. Further let $y_d(\tau)$ denote the current CPU usage at time τ on day d and recall that is unobservable since we only observe the average CPU consumptions $\bar{y}_{dt}$. 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 request. Finally let $s_{d1}, \ldots, s_{dN_d}$ denote the size (processing time) for the requests $a_{d1}, \ldots, a_{dN_d}$. Since we assume a $M/G/\infty$ model, the departure time for request, h_{dn}, will simply be $h_{dn} = a_{dn} + s_{dn}$.

4.2 Statistical queuing model

We assume that $N_d, \ d = 1, \ldots, D$ 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 analyzed in this paper were we observed most of the requests during the day (Section 3). In stead 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 happen at different times during the day. E.g. if high values are chosen for both shape parameters, e.g. $\alpha = \beta = 20$, almost all arrivals (CPU requests) will happen 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 on the Beta distribution, see e.g. [10]. Finally we assume that the processing time of each 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 analyzed in this paper, the Log-normal distribution performed well. Since the requests to the CPU can be 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 thus the Log-normal distribution is suitable. We also experimented with both an exponential and gamma distribution, but the results were less promising.

Given the arrivals and processing times as 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

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

where $I(A)$ is the indicator function returning one if A is true and zero else. 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 on the time interval $[\tau_{t-1}, \tau_t]$ which gives

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

where

$$
F(\tau; a_{dn}, h_{dn}) = \begin{cases}
0 & \text{if } \tau \leq a_{dn} \\[2mm]
\dfrac{\tau - a_{dn}}{\Delta\tau} & \text{if } a_{dn} < \tau \leq h_{dn} \\[3mm]
\dfrac{h_{dn} - a_{dn}}{\Delta\tau} & \text{if } \tau > h_{dn}
\end{cases}
$$

4.3 Likelihood function

Given both the observations (Section 3) and the statistical model presented in the previous section, the natural next step is the use the observations to estimate the unknown parameters in the statical model $\lambda, \alpha, \beta, \mu$ and σ. The parameters characterize the properties of the underlying request process to the CPU processor and estimating these parameters is thus the goal of this paper. The most common way to estimate parameters in statistical models is to optimize the likelihood functions which in this case can be written as

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

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

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

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

5 Approximate Bayesian computing

Since the likelihood function is infeasible, we resort to a more indirect way to estimate the parameters. Intuitively, if we generate outcomes from the statistical model in Section 4.2 using the true parameter values (that generated the real data), we expect that outcomes from the statistical model should be similar to the real data. Fortunately, it is simple to generate outcomes from the statistical model in Section 4.2 which means that such an indirect approach is possible. We rely of the framework called Approximate Bayesian Computing (ABC). See [3, 7] for nice reviews on the 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, which 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 samples $\theta_1, \ldots, \theta_M$ from the prior distribution $p(\theta)$.
2. For each sample 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 summarize 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 is close to the unknown true parameter θ_o.
5. Fit a statistical model to the relation between the parameters and the statistics using the samples 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 get inference on the 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 as the estimate of θ_o.

The interested reader is referred to [3, 7] 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 4.2, i.e. $\overline{\mathbf{y}}_d$, $d = 1, \ldots, D$.

6   CPU data statistics

A cruical part of the ABC methodology, described in the previous section, is to choose statistics that are able to distinguish properties of samples from the model in Section 4.2 for different values of the parameters $\lambda, \alpha, \beta, \mu, \sigma$. Thus we started by generating samples from the model for different values of the parameters. We fixed the expected CPU consumption for outputs from the model to be equal to the value in the observations, i.e.

$$E(N_d)E(s_{dn}) = \frac{1}{D} \sum_{d=1}^{D} \overline{\mathbf{y}}_d$$

$$\lambda \exp\left(\mu + \sigma^2/2\right) = \frac{1}{D} \sum_{d=1}^{D} \overline{\mathbf{y}}_d \tag{2}$$

where the expectations on the left hand side follows from the properties of the Poisson and Log-normal distributions. We thus choose combinations of λ, μ and σ satisfying 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 (2). Finally we sat $\alpha = \beta = 3$ which resulted in a little more requests happening during day which is in accordance with the real data. To study the effects of adjusting the parametric values, we ran several samples from the model for the different parametric choices. One sample from each of the values of μ are shown i 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 in an even larger degree. Comparing to the real data in Figure 1, it seem like the lower left panel in Figure 2 is the most similar to the real data. However it seem to be more spikes in the real data, indicating that $\lambda = 50$ is a too low 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 seem 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 finding suitable statistics of the real data. In the experiments we use the following the statistics.
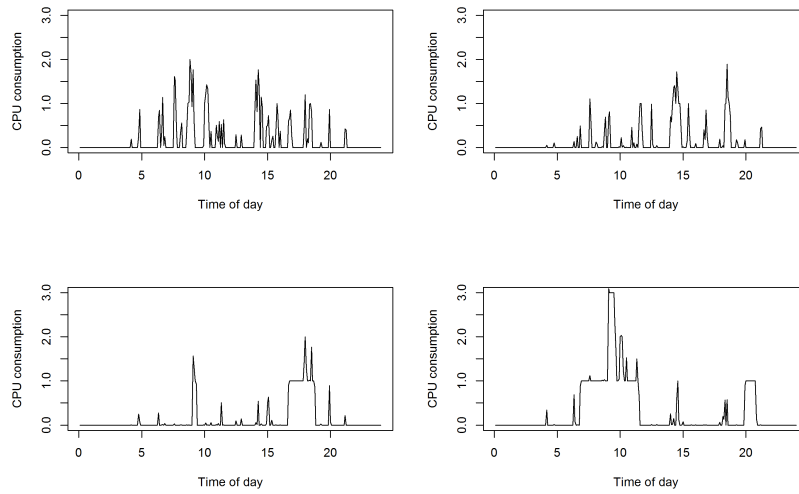
Fig. 2. Samples from the model in Section 4.2 for $\lambda = 50$. For the panels from upper left to lower right, $\mu = -6, -7.5, -9$ and -10.5, respectively.
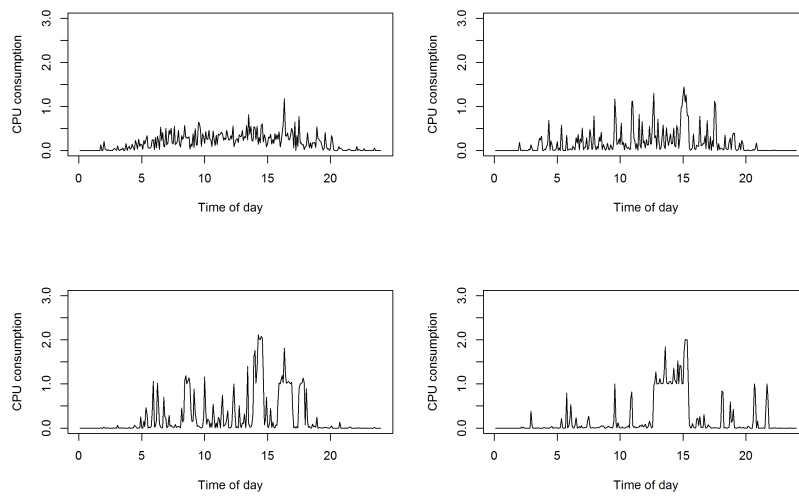

Fig. 3. Samples from the model in Section 4.2 for $\lambda = 1000$. For the panels from upper left to lower right, $\mu = -9, -11, -13$ and -15, respectively.

– Average CPU consumption.

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

– Cumulative probabilities. Define

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

being the portion of observations with a value less than q (quantile). We see that with $\lambda = 50$ compared to $\lambda = 1000$ it is far more common with a CPU consumptions equal to zero, one and two 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 seem 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 and 1.99. 2.01. 2.1 and 2.5.

– To measure the degree of spikyness we compute the change in CPU consumption from one time step to the next

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

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

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

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

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

In the experiments we used $L = 5$. We use the moving average in stead of the CPU consumption observations directly since the observations is 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 much from one day to another and we should have the same property in the model. We there include the standard deviation in the daily averages

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

Typically choosing μ and σ such that the Log-normal distribution gets 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

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

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

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

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

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

7 Experiments

In the previous sections we have built a reliable model for the CPU consumption data and an estimation strategy based on the ABC framework. In this Section we will evaluate to what extent we are able to recover the request patterns to the CPU processor. To use the ABC procedure, we establish the following.

– Prior distributions. The ABC procedure requires that we have prior distributions for the unknown parameters. We assume that we have little prior information and chose wide uniformly distributed priors as follows

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

where $\text{Unif}(a, b)$ denotes the uniform distribution on the interval between a and b.

– Generate samples. We know follow the first three steps of the ABC procedure described in Section 5 by generating $M = 4 \cdot 10^6$ samples and computing the statistics presented in Section 6.
– In the fifth step of the procedure different statistical models can be used like partial least square [20], neural net [2] or ridge regression [18]. We evaluated both the neural net and the ridge regression approach and both 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 $T = 288$ meaning 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.

7.1 Synthetic data example

We start by generating a synthetic dataset from the model with parameter values $\lambda = 500, \alpha = 8, \beta = 8, \mu = -10, \sigma = 2.2$ and compute the statistics from Section 6. If the estimation procedure performs well we should get reliable estimates of the parameters used to generate the synthetic data. Figure 4 shows histogram of outcomes from the ABC posterior distribution for the different parameters. We see that the procedure, quite astonishingly, estimates the parameters of the underlying queue process very well.
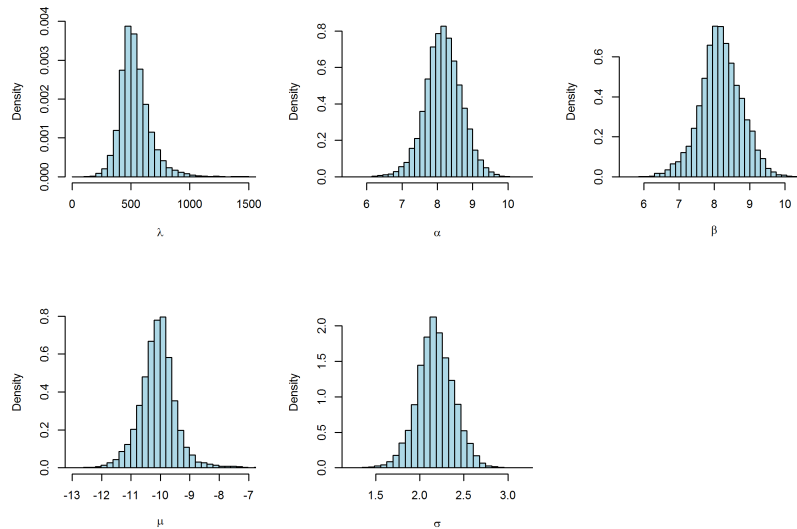

Fig. 4. Synthetic example: Samples from the posterior distribution for the different parameters of the model.
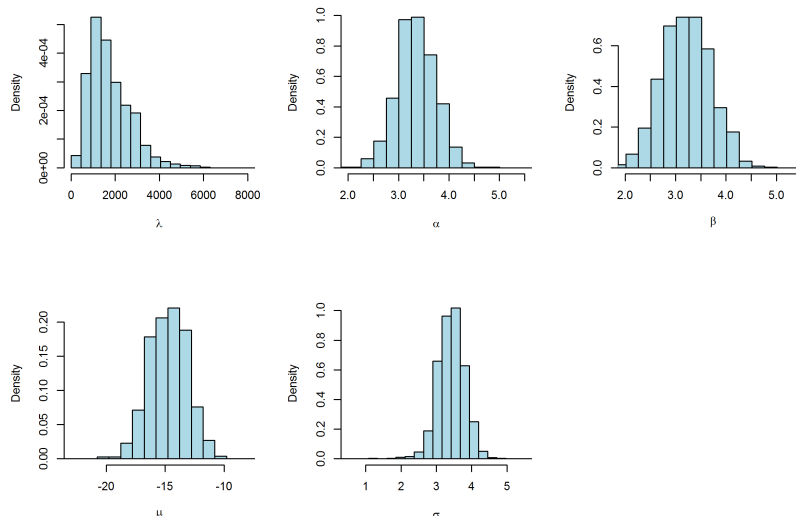
Fig. 5. Real data example: Samples from the posterior distribution for the different parameters of the model.

7.2 Real CPU consumption data example

We now run the estimation procedure for the real data. Figure 5 show samples from the posterior distribution for the different parameters. We see that we are able to get fairly precise and reliable estimates of the underlying request process to the CPU processor. 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 minute) and 5000 (a request about every 15 seconds). We see $\alpha \approx \beta \approx 3.5$ meaning that most of the requests to the CPU processor happens 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 happens at other times during the day.

Figure 6 shows four arbitrary samples from the stochastic model using the estimated parameters with the highest posterior probability (the MAP parameters). We see that outcomes from the statistical model replicates the properties of the real CPU consumption data very well.

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 from the model have different properties. Next we build statistics that quantify these differences which are used to estimate the parameters of the statistical model to replicate the properties of
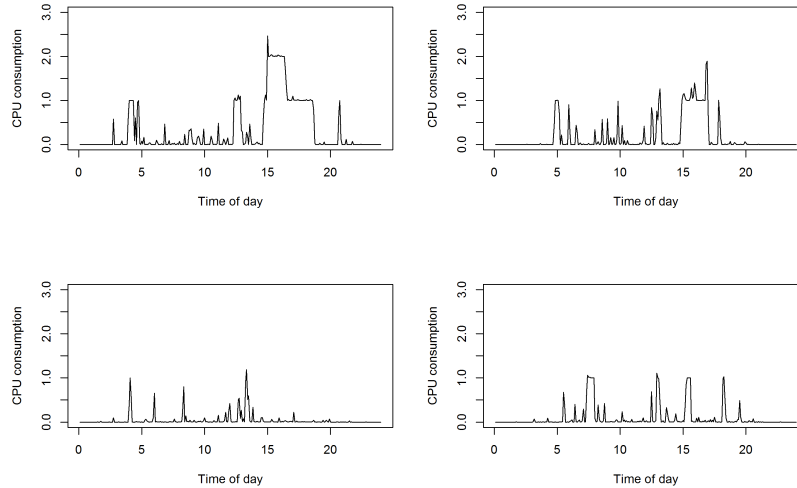
Fig. 6. Real data example: Samples from the statistical model in Section 4.2 using the MAP estimate of the unknown parameters.

real CPU consumption data. Our results show that we are able to recover the properties of the underlying request patterns to the CPU processor.

There are several interesting directions for future research. We want to apply the framework to other types of data like single core CPU processors. This adds another complexity since processes may be queued until the CPU is finished with other tasks. If the load is so high that the CPU processor runs continuously, of course recovering of the request pattern is impossible. We may also apply the methodology to other types of data like network data.

References

1. SK Acharya. On normal approximation for maximum likelihood estimation from single server queues. *Queueing systems*, 31(3-4):207–216, 1999.
2. Michael GB Blum and Olivier François. Non-linear regression models for approximate bayesian computation. *Statistics and Computing*, 20(1):63–73, 2010.
3. 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.
4. Brian Bouterse and Harry Perros. Scheduling cloud capacity for time-varying customer demand. In *Cloud Networking (CLOUDNET), 2012 IEEE 1st International Conference on*, pages 137–142. IEEE, 2012.
5. 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.

6. DR Cox. The statistical analysis of congestion. *Journal of the Royal Statistical Society. Series A (General)*, 118(3):324–335, 1955.
7. 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.
8. Paul Fearnhead. Filtering recursions for calculating likelihoods for queues based on inter-departure time data. *Statistics and Computing*, 14(3):261–266, 2004.
9. William Feller. An introduction to probability theory and its applications. vol. i. 1950.
10. Catherine Forbes, Merran Evans, Nicholas Hastings, and Brian Peacock. *Statistical distributions*. John Wiley & Sons, 2011.
11. 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.
12. 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.
13. 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.
14. Jong Kim and Kang G Shin. Execution time analysis of communicating tasks in distributed systems. *Computers, IEEE Transactions on*, 45(5):572–579, 1996.
15. Xiaodong Liu, Songyang Li, and Weiqin Tong. A queuing model considering resources sharing for cloud service performance. *The Journal of Supercomputing*, 71(11):4042–4055, 2015.
16. 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.
17. 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.
18. Gisela Muniz and BM Golam Kibria. On some ridge regression estimators: An empirical comparisons. *Communications in StatisticsSimulation and Computation®*, 38(3):621–630, 2009.
19. Xiaoming Nan, Yifeng He, and Ling Guan. Queueing model based resource optimization for multimedia cloud. *Journal of Visual Communication and Image Representation*, 25(5):928–942, 2014.
20. 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.
21. Jordi Vilaplana, Francesc Solsona, Ivan Teixidó, Jordi Mateo, Francesc Abella, and Josep Rius. A queuing theory model for cloud computing. *The Journal of Supercomputing*, 69(1):492–507, 2014.
22. Wei Xiong and Tayfur Altiok. Queueing analysis of a server node in transaction processing middleware systems. *Computers & Operations Research*, 35(8):2561–2578, 2008.
23. Mustafa Yuzukirmizi and J MacGregor Smith. Optimal buffer allocation in finite closed networks with multiple servers. *Computers & Operations Research*, 35(8):2579–2598, 2008.
24. Zhongju Zhang and Weiguo Fan. Web server load balancing: A queueing analysis. *European Journal of Operational Research*, 186(2):681–693, 2008.