Event-Based Control and Estimation of Server Systems

Cervin, Anton

Published in:
Proceedings of the 4th International Conference on Event-Based Control, Communication, and Signal Processing, EBCCSP 2018

2018

Document Version:
Peer reviewed version (aka post-print)

Event-Based Control and Estimation of Server Systems

Anton Cervin
Department of Automatic Control LTH
Lund University, Sweden
Email: anton@control.lth.se

***Abstract*—Feedback control is increasingly being applied in server systems to make them more robust and efficient. This includes managing quality of service, minimizing power consumption, and adapting to varying workloads. Successful adaptation and control in turn relies on accurate tracking of workload variations and timely detection of changes in the computing infrastructure. Given that server systems are inherently event based, it is natural to consider event-based control and estimation schemes for them. As a prototypical problem, we study a single server system with time-varying arrival rate and derive optimal switching rules for the service rate. The goal is to keep the response time within bounds while minimizing the energy consumption of the server. We also design an event-based estimator of the server states using a particle filter approach. Finally, we outline some research challenges related to event-based control and information fusion in server systems.**

I. Introduction

Within large server systems such as cloud computing platforms there is a need to handle workload variations and to adapt to unpredictable structural changes. The idea of the self-adaptive or elastic cloud [1], [2] is to handle the time-varying supply and demand of computing resources using feedback. The goal is to provide just the right amount of resources at all times, so that the operational cost is minimized, while still delivering good performance to the customers. This can be viewed as a classical feedback control loop (see Fig. 1), where the server system is the plant under control and the adaptation mechanism is the controller. Workload variations and temporary hardware failures are viewed as disturbances that should be countered by adjustments in the resource provisioning. Server performance can be measured by, e.g., average or Xth percentile response times, throughput, utilization, and power usage. One way to perform the actual resource adjustments is to scale the number of virtual machines that run the server software in question.

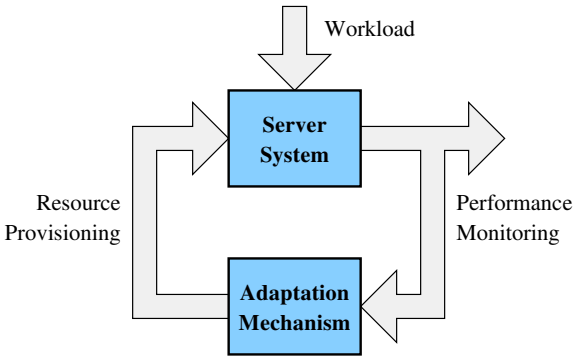

Fig. 1. Automatic resource provisioning in a server system.

The control loop in Fig. 1 looks fairly conventional, but if we zoom in, some interesting features can be noted. The arrows that connect the different blocks in the diagram do not represent continuous signals but rather discrete events. Measurement information is available only when something happens in the system, e.g., when a new job arrives to the server or when a request is completed. Likewise, the resources are typically quantized and can only be set at fixed levels. To deal with these special features, we must turn to control and estimation techniques that can handle discrete events rather than continuous signals. In the past two decades there has been much theoretical development in both event-based control and event-based estimation, e.g., [3]–[12]. Little of the research so far has however focused on event-based techniques for server systems.

Previous research on control of server systems in particular has been conducted along two more or less separate lines. Within operations research and queueing theory, controlled server systems have been modeled and optimized using the theory of Markov decision processes [13], [14], typically assuming fixed model parameters. Within control engineering and soft real-time computing, servers have often been modeled as low-order transfer functions, obtained either from system identification or using fluid-flow approximations of the queueing dynamics [15]. It is however well known that servers exhibit nonlinear dynamics and require other performance metrics than classical control systems [16]. In the last decade, a large number of control structures and design methods have then been applied and evaluated, see [17] for a survey. Striking a middle path, a few researchers have combined queueing models with stochastic control theory, e.g., [18]. The main idea is that the plant model should capture both the random discrete events and the integrator-like queue length dynamics. The resulting stochastic differential equations share some similarities to those that arise in event-triggered control of first-order stochastic systems [3].

In the literature on controlled queueing systems it is usually assumed that the system parameters are known and that the system states are directly accessible for feedback [14]. In real servers, it can be difficult to access internal queues

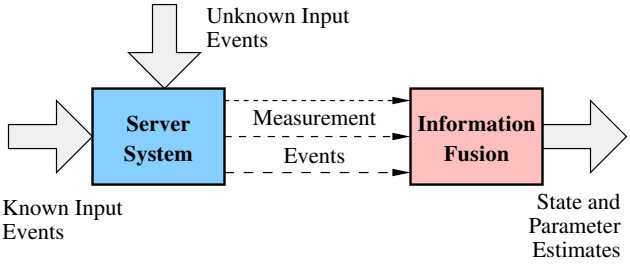

Fig. 2. Event-based information fusion in a server system.

for measurements and to have a-priori knowledge of service time distributions and other system parameters. Therefore estimators are needed. A common approach in practice is to measure relevant events and quantities over fixed time intervals and calculate averages using exponential smoothing [19].

An alternative to exponential smoothing and similar windowing techniques is to use event-based information fusion. Similar to a Kalman filter, the general idea is to estimate states and parameters of the server system using a model of the system together with various asynchronous measurements. The basic idea is illustrated in Fig. 2. Known inputs to the system are, e.g., the control commands from the Adaptation Mechanism in Fig. 1, while the unknown inputs represent, e.g., customer arrivals that cannot be directly measured. Combining a-priori model knowledge with measurements, the Information Fusion system needs to take all types of events (and also the absence of events) into account when forming its estimates of parameters and states.

The event-based information fusion problem is challenging due to the non-linear behavior of the system and because new information is only available at discrete events. One promising approach to tackle the problem is to use particle filters [20], [21], which is a family of Monte Carlo-based inference methods that have gained much attention in recent years. Applying particle filters for estimation in server systems is however not straightforward. New dynamical system models need to be developed, and the filters need to be adapted to handle event-based rather than time-based measurements. Another challenge is to weigh together the information from many different types of events in an optimal way.

In the rest of this paper, we study a simple but prototypical system, modeled as a single-server queue with a time-varying arrival rate and an adjustable service rate. Being a gross simplification of real server systems, the model is still rich enough to offer some control and estimation challenges. In Section II the system model is given and some basic properties are analyzed. In Section III we derive optimal event-based switching rules for the service rate, given complete knowledge of system parameters and states. In Section IV we design an event-based estimator for the system based on particle filtering. Conclusions are given in Section V.

II. A Simple Model of a Server System

As a starting point for our investigation, we consider a single server with a compute-intensive workload, modeled as

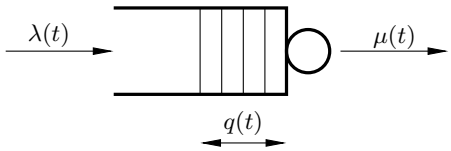

Fig. 3. M/M/1 server model with queue length $q(t)$, time-varying arrival rate $\lambda(t)$ and adjustable service rate $\mu(t)$.

an M/M/1 queueing system[1] (see Fig. 3). Jobs arrive according to a Poisson process with time-varying intensity[2] $\lambda(t)$. Jobs are served in FIFO order with exponentially distributed service times with an adjustable service rate of $\mu(t)$. The queue length, including the job currently being served, is denoted $q(t)$.

Different from standard queueing models, we explicitly model the dynamics of the varying arrival rate. Any Markov model could be used, but we here assume that $\lambda(t)$ evolves according to a bounded random walk with incremental variance σ_λ^2 and lower and upper bounds λ_{min} and λ_{max}. To match the time-varying demand, we allow the service rate to be set to any value in a discrete set, $\mu(t) \in \{\mu_1, \mu_2, \ldots, \mu_M\}$.

The control objective two-fold: to keep the job response times in check and to minimize the power consumption of the server. According to Little's Law [13], the average response time is proportional to the average queue length (as long as the system is not overloaded). Hence, we can focus on controlling the queue length rather than the response time of the server, which greatly simplifies the problem. The conflicting goals of having a short queue and using as little energy as possible is captured in a cost function

$$J = \mathop{\mathrm{E}}_{t}\Big\{\max(0,\, q(t) - q_{tol}) + c(\mu(t))\Big\} \qquad (1)$$

where q_{tol} is the maximum tolerable queue length and $c(\mu)$ is a positive and increasing discrete function that describes the operational cost of each service rate. To quantify how well the queue length regulation works in practice, we will look at the 99th percentile of q, denoted $q_{99\%}$. The 99th percentile indicates that the queue length will stay below this value 99% of the time.

As a basic scenario in the paper, we assume that the arrival rate can vary between $\lambda_{min} = 0.5$ and $\lambda_{max} = 1.5$ with an average rate of change $\sigma_\lambda = 0.05$. The tolerable queue length is given as $q_{tol} = 20$, which is also the assumed target for $q_{99\%}$. We further assume that the server can operate at two different rates, $\mu_1 = 1$ and $\mu_2 = 2$, and that the power consumption is proportional to the service rate in each mode.

Using a fixed rate of $\mu_1 = 1$ can clearly lead to overload situations since the traffic $\rho = \lambda/\mu$ may at times exceed 1. The situation is illustrated in Fig. 4, where it can be seen that even a short overload interval can lead to large queue buildups. Running a long simulation ($10,000$ time units) reveals that $q_{99\%} > 100$, which is not acceptable. On the other hand, using

[1] "M/M/1" is according to Kendall's notation referring to a queueing system with memoryless arrivals, memoryless service times, and one server [13].

[2] To simplify the notation, throughout this paper we shall assume that time is unitless.

$\mu_2 = 2$ gives much shorter average queue lengths as seen in Fig. 5. A long simulation reveals $q_{99\%} = 9$, which is well below the target. The downside is that the server always runs at maximum speed, consuming twice as much power as in the previous case. It is hence natural to seek service rate switching rules that optimize the criterion (1). This will be studied in the next section.

III. Optimal Event-Based Switching

A. Discrete Markov Model for Dynamic Programming

To derive optimal switching rules for the server we first discretize the system dynamics in both time and space and then apply dynamic programming. The Markov state of the system is given by $x = \begin{bmatrix}\lambda & q\end{bmatrix}^T$, where

$$\lambda \in \{\lambda_{\min},\, \lambda_{\min} + \Delta\lambda,\, \ldots,\, \lambda_{\max}\}$$
$$q \in \{0,\, 1,\, \ldots,\, q_{\max}\}$$

The bounded random walk of λ is discretized using a simple Euler approximation with interval h. The probability of the queue evolving from length i to j, assuming fixed values of λ, μ during the interval h, is given by

$$P_{ij}(h) = e^{-(\lambda+\mu)h}\Bigg[\rho^{\frac{j-i}{2}} I_{j-i}(ah) + \rho^{\frac{j-i-1}{2}} I_{j+i+1}(ah) + (1-\rho)\rho^j \sum_{k=j+i+2}^{\infty} \rho^{-k/2} I_k(ah)\Bigg] \tag{2}$$

where $a = 2\sqrt{\lambda\mu}$ and I_k is the modified Bessel function of the first kind of order k, see [13], [22]. Using value iteration, the optimal stationary control policy $\mu(x)$ can then be calculated.

Applying the method to our example server with two service levels, $\mu \in \{1, 2\}$, we discretize the model with $\Delta\lambda = 0.05$, $h = 1$, and $q_{max} = 40$ and then calculate the optimal policy when the running cost is chosen as $c(\mu) = 4\mu$. The cost function was tuned to achieve $q_{99\%} = 20$. The resulting switching law is shown in Fig. 6. It can be seen that both the arrival rate and the queue length affect the optimal choice of the service rate: At lower arrival rates longer queues can

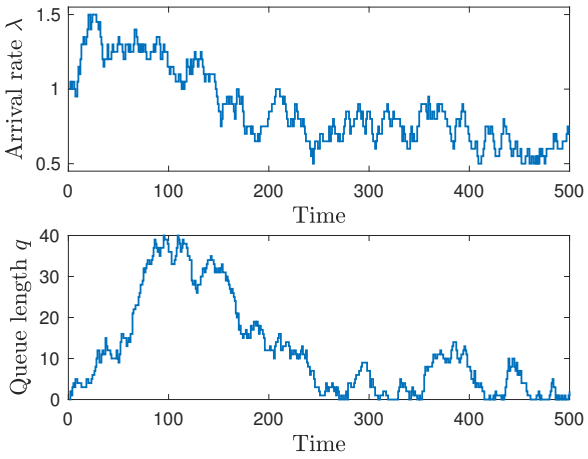

Fig. 4. Simulation of the server with fixed service rate, $\mu = 1$.

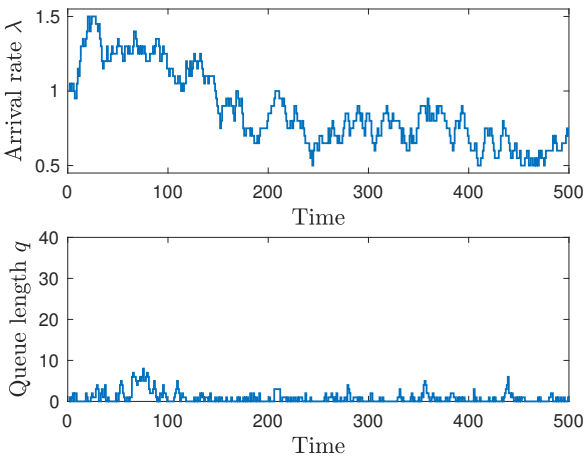

Fig. 5. Simulation of the server with fixed service rate, $\mu = 2$.

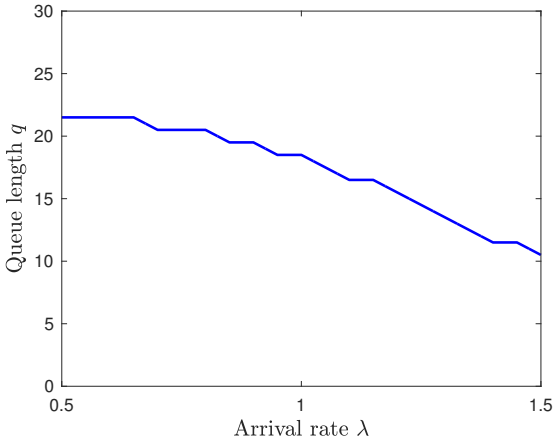

Fig. 6. Optimal switching policy for two service levels, $\mu \in \{1, 2\}$, when $q_{\lim} = 20$. The low service rate is used below the line and the high rate above the line.

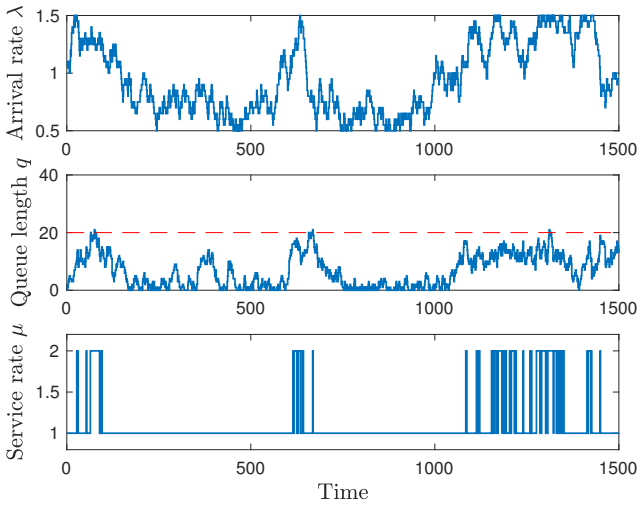

Fig. 7. Simulation of the server using the switching policy in Fig. 6.

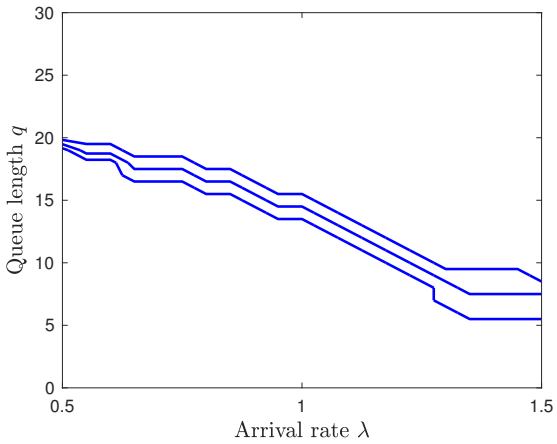

Fig. 8. Optimal switching policy for four service levels, $\mu \in \{0.1,\ 1,\ 2,\ 3\}$, when $q_{\lim} = 20$. The lowest rate is used in the bottommost region, the highest rate in the uppermost region, and the intermediate levels in the two bands inbetween.

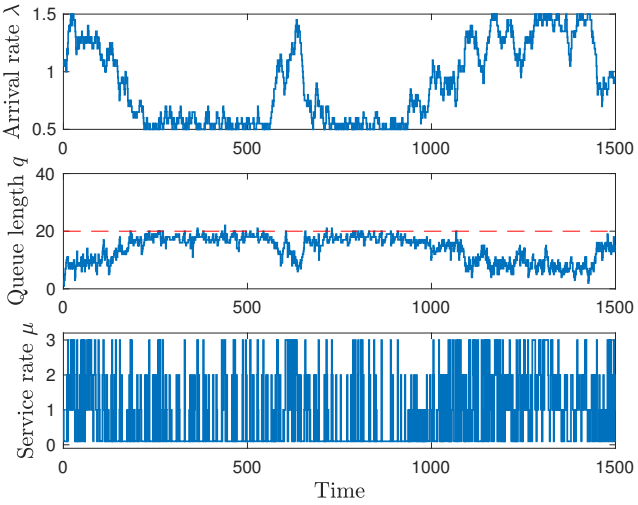

Fig. 9. Simulation of the server using the switching policy in Fig. 8.

be tolerated, and vice versa. A simulation of the server with the control policy is shown in Fig. 7. It is seen that the queue is controlled to be shorter than 20, but that switches can be quite frequent in some intervals. A long simulation reveals that the average service rate (and hence power consumption) is $\bar{\mu} = 1.12$, which is a great savings compared to $\mu_2 = 2$.

The method can easily be extended to handle more service levels, without increasing the dimensional complexity of the dynamic programming problem to be solved. Assuming four service levels, $\mu \in \{0.1,\ 1,\ 2,\ 3\}$, and the running cost $c(\mu) = 0.4\mu$ (again tuned to achieve $q_{99\%} = 20$) we obtain the control policy in Fig. 8. It is seen that the server either operates at its lowest rate (at the botton) or the highest rate (at the top), except around a quite small band in the middle where the intermediate levels are used. A simulation of the multi-rate policy is shown in Fig. 9. Again, the target $q_{99\%} = 20$ is kept, the power consumption is now even lower ($\bar{\mu} = 0.87$), but server rate changes are now extremely frequent, which may not be realistic for real server systems.

B. Limiting the Switching

In the examples above we saw that optimal control laws can generate many switches. This is undesirable if service rate changes are expensive; if it means, e.g., starting up or closing down virtual servers, migrating tasks, etc. In reality these operations can take considerable amounts of time, during which no new control actions can be taken. We will therefore consider different ways to limit the switching.

The most straightforward way to achieve fewer switches is to incorporate an explicit cost per switching event in the cost function. In general, we can let the server running cost c be a function of both the old rate, μ_{old}, and the new rate, μ. The modified cost function then becomes

$$J = \mathop{\mathrm{E}}_{t}\Big\{\max(0,\ q - q_{\lim}) + c(\mu_{\text{old}}, \mu)\Big\} \tag{3}$$

To keep track of the switching cost in the dynamic programming, the Markov state needs to be extended accordingly; $x = \begin{bmatrix}\lambda & q & \mu_{\text{old}}\end{bmatrix}^T$. The downside of this method is that the real switching delay is ignored in the model.

Returning to the case with two service rates, we design the running cost as

$$c(\mu_{\text{old}}, \mu) = \begin{cases} 2\mu, & \mu_{\text{old}} = \mu \\ 2\mu + 1, & \mu_{\text{old}} \neq \mu \end{cases} \tag{4}$$

to achieve $q_{99\%} = 20$. As a result we obtain a switching rule that depends on μ_{old}, see Fig. 11.

We see that the switching cost introduces hysteresis in the control law. To switch to the higher service level, we need to cross the higher threshold (indicated by $\mu_{\text{old}} = 1$) and vice versa. A corresponding simulation is shown in Fig. 12, where the reduction in the number of switches is clearly visible. A long simulation shows that $\bar{\mu} = 1.39$, so the reduced switching comes at the price of higher power consumption.

There may however still be switches arbitrarily close in time—something that is not realizable in a real system. Another option, which may better reflect reality, is to model the actual switching delay in the server, thereby introducing an *indirect* switching cost. One possible model is shown in Fig. 10. Here, a change in μ makes the system go from the Normal to the Transit state, where the server must stay until the time $T(\mu_{\text{old}}, \mu)$ has passed. The transition delay does make the dynamic programming problem slightly more complicated, but it does not make the discrete state space larger compared to having a simple switching cost.

Again considering the case with $\mu \in \{1,\ 2\}$, we remove the explicit switching cost by setting $c(\mu_{\text{old}}, \mu) = 2\mu$ and at

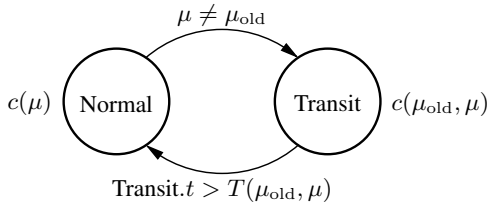

Fig. 10. Modelling service rate switching delays using two discrete modes.

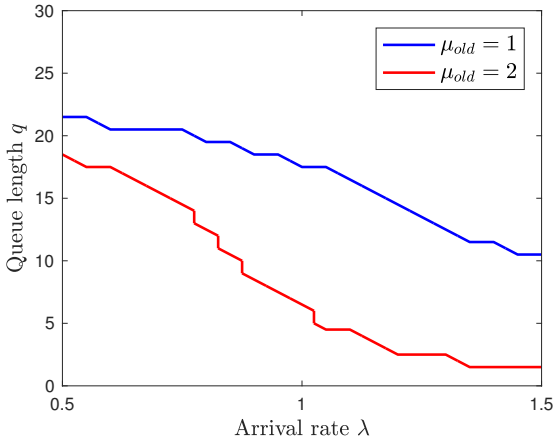

Fig. 11. Optimal switching policy with hysteresis for two service levels, $\mu \in \{1, 2\}$, when $q_{\lim} = 20$ and using the switching cost $c(\mu_{\text{old}}, \mu) = 2\mu + 1(\mu_{\text{old}} \neq \mu)$. The high rate is used after the upper curve has been crossed in the upward direction, and the low rate is used after the lower curve has been crossed in the downward direction.

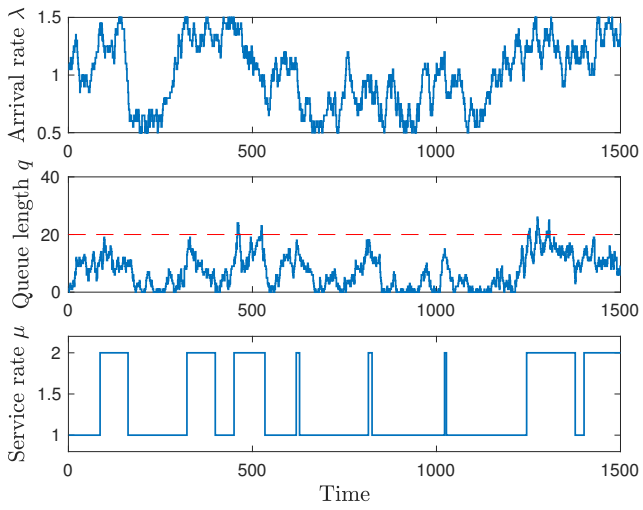

Fig. 12. Simulation of the server using the switching policy in Fig. 11.

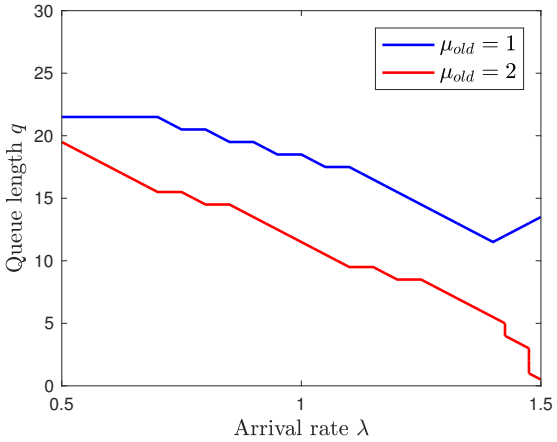

Fig. 13. Optimal switching policy with hysteresis for two service levels, $\mu \in \{1, 2\}$, when $q_{\lim} = 20$ and using a switching delay $T = 10$. The high rate is used after the upper curve has been crossed in the upward direction, and the low rate is used after the lower curve has been crossed in the downward direction.

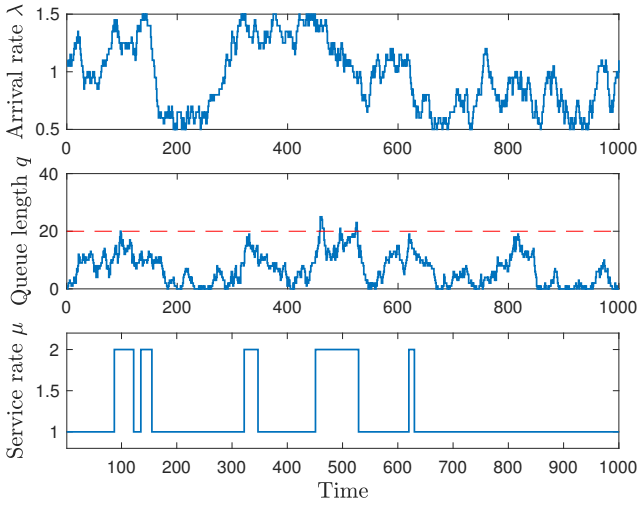

Fig. 14. Simulation of the server using the switching policy in Fig. 13.

the same time introduce a switching delay of $T = 10$. This yields the switching policy in Fig. 13. Again can we see that there is hysteresis that will limit the switching. This behavior is also visible in the simulation in Fig. 14. Now switches are guaranteed not to occur closer than 10 time units apart, which gives room to shut down or start up virtual machines, migrate jobs, etc. Keeping the target $q_{99\%} = 20$, the price is again a slightly higher power consumption, $\bar{\mu} = 1.34$.

IV. Event-Based Estimation

We now turn to the problem of estimating states and parameters in server systems using event-based measurements. There are several general challenges: The amount of generated events can be massive, events can be generated at very different timescales, and also the absence of events gives some information about the system that should ideally be taken into account. It may be too costly run the estimation algorithm at every measurement event, invoking the need of designing additional triggers for the estimator itself.

Estimation in server systems are today often handled using exponential smoothing and other windowing techniques that operate over fixed time intervals [19]. As a general technique to tackle non-linear estimation problems for dynamical systems, particle filters have gained much popularity in recent years [20], [21]. We propose to extend and adapt this method to handle event-based estimation in server systems. Merging several kinds of different measurements from various sources in the same filter, we refer to this as *event-based information fusion*.

The general idea of particle filtering is to approximate the (non-Gaussian) state probability distribution of the unknown states (or parameters) by a cloud of particles. Each particle represents one hypothesis about the current state of the system. The larger number of particles being used, the better the accuracy of the filter will be. As the particle count goes

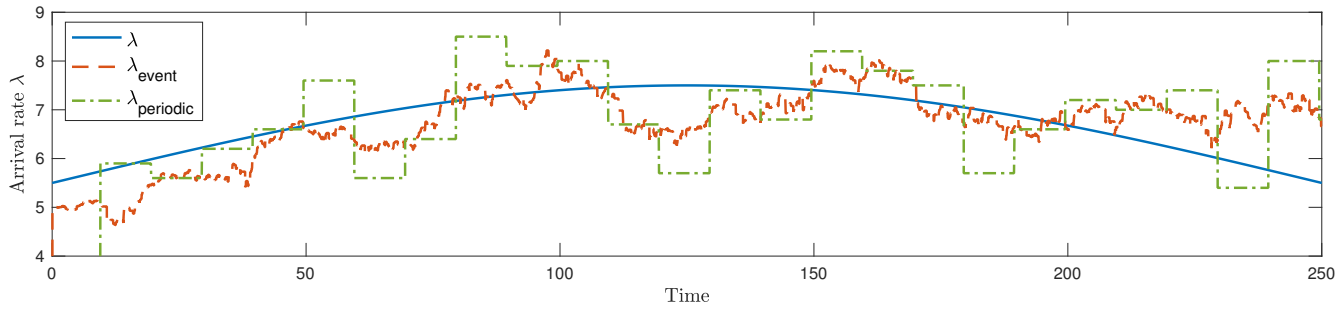

Fig. 15. Estimation of the server arrival rate from response times using an event-based particle filter or a standard averaging periodic filter.

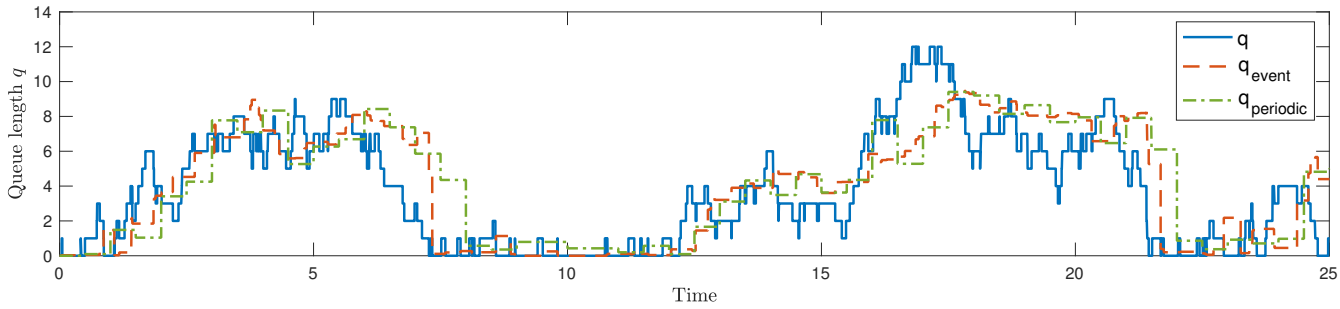

Fig. 16. Estimation of the server queue length from response times using an event-based particle filter or a standard averaging periodic filter. Note the different time scale from Fig. 15.

to infinity, the behavior of an optimal Bayesian estimator is approached. Of course, increased accuracy comes at the price of heavier on-line computations.

For a standard discrete-time model, the basic procedure to be executed every time step is:

1) *Simulate* the particles forward one time step according to the nonlinear stochastic model.
2) Given the measurement in the current time step, *assign weights* to the particles based on their statistical likelihood.
3) *Resample* the particle cloud based on their weights, so that hypotheses that are more likely are kept while less likely ones are discarded.

Resampling can be performed in a multitude of ways; one of the simpler algorithms is *systematic resampling* (e.g., [21]), which we employ here. More details on the procedure above can be found in our companion paper [23].

We now return to the server system example with time-varying arrival rate and adjustable service rate. In the previous section we assumed that both the queue length q and the arrival rate λ were exactly known at all times. To make the problem more interesting (and also more realistic) we will now assume that only the output of the server can be observed. More specifically, we will have information about each departing job from the system, including its response time. The service rate μ is assumed known, while q and λ have to be estimated. (A different setup, where q and μ should be estimated is considered in [23].)

Using some basic facts from queueing theory [13], we can use the response time measurement events in three complementary ways in our information fusion algorithm:

- Assuming $\lambda < \mu$, i.e., that the server is not overloaded, the departure process has the same statistical properties as the arrival process. The time between two consecutive output events is then exponentially distributed with parameter λ.
- The response time in an M/M/1 system with queue length q is Erlang distributed with shape q and rate μ.
- It is possible to conclude that the server has been idle if the response time is smaller than the time between the two latest departure events.

In our implementation of the event-based particle filter we use $N = 100$ particles. Between each output event, we forward simulate the distribution of λ and q using Markov model in Section III.A. At the event, we assign a likelihood of measurement to every particle using the statistical properties listed above. Then we calculate the sample mean estimate of λ and q, and finally we resample the particle cloud.

A simulation of the server and the particle filter is illustrated in Figures 15 and 16. In this simulation we let the actual arrival rate be slowly time-varying rather than random to make the results more visible in the plots.

As a comparison we have also implemented simple, periodic averaging filters for both parameters: The arrival rate is calculated as the number of departures during the interval, and the queue length is calculated as the average response time during the interval times the service rate. For both of these averages, there is a trade-off between the noise level and the time interval. For the arrival rate we choose 10 time units, and for the queue length (which has much faster dynamics) we choose 0.5 time units.

In the figures it can be seen that both the periodic and

event-based filters manage to track the server states, while the periodic filter either has larger variance (in case of λ) or is slower at tracking the transient behavior (in the case of q). A long simulation ($10,000$ time units) reveals that the absolute integrated tracking error is around 10% smaller for the event-based filter for the given setup. However, the periodic averaging filters require much less on-line computations than the particle filter, making it hard to declare a clear winner in this example.

V. Conclusion

We have studied a simple but non-standard queueing model of a single server with dynamical input and output rates and described by example how tools such as dynamic programming and particle filters can be used to obtain event-based switching rules and observers. Event-based techniques hold the promise of lower resource consumption and higher responsiveness, but more thorough analysis, simulations as well as real implementations are needed to draw conclusions about the magnitude of such benefits. Dynamic programming is for complexity reasons limited to a few state variables, so approximations are needed for more complicated scenarios.

Real servers and real traffic show more complex behavior than the simple M/M/1 model studied in this paper. A natural method extension would be to let the particle filter estimate a larger number of server and traffic parameters, including the computational efficiency of the server. Of course, the estimation and control schemes must also be tested and evaluated in real server systems.

There are also general theoretical challenges for particle filter design, e.g., on how to combine the information from various types of events (and lack of events) in an optimal way.

Acknowledgments

This work was supported by the Swedish Research Council. The author is a member of the LCCC Linnaeus Center and the ELLIIT Excellence Center at Lund University.

References

[1] Y. Jiang, C. s. Perng, T. Li, and R. Chang, "Self-adaptive cloud capacity planning," in *Ninth IEEE International Conference on Services Computing*, June 2012, pp. 73–80.

[2] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *IEEE Network Operations and Management Symposium*, April 2012, pp. 204–212.

[3] K. J. Åström and B. Bernhardsson, "Comparison of periodic and event based sampling for first-order stochastic systems," in *Preprints 14th World Congress of IFAC*, Beijing, P.R. China, July 1999.

[4] P. Tabuada, "Event-triggered real-time scheduling of stabilizing control tasks," *IEEE Transactions on Automatic Control*, vol. 52, no. 9, pp. 1680–1685, Sept 2007.

[5] M. Rabi, K. H. Johansson, and M. Johansson, "Optimal stopping for event-triggered sensing and actuation," in *Proc. 47th IEEE Conference on Decision and Control*, Cancún, Mexico, Dec. 2008.

[6] R. Cogill, "Event-based control using quadratic approximate value functions," in *IEEE Conference on Decision and Control*, 2009.

[7] J. Sijs and M. Lazar, "On event based state estimation," in *Proc. Hybrid Systems: Computation and Control*, 2009, pp. 336–350.

[8] T. Henningsson, "Sporadic event-based control using path constraints and moments," in *50th IEEE Conference on Decision and Control and European Control Conference*, Orlando, Florida, USA, Dec. 2011.

[9] G. M. Lipsa and N. C. Martins, "Remote state estimation with communication costs for first-order LTI systems," *IEEE Transactions on Automatic Control*, vol. 56, no. 9, pp. 2013–2025, Sept 2011.

[10] W. P. M. H. Heemels, M. C. F. Donkers, and A. R. Teel, "Periodic event-triggered control for linear systems," *IEEE Transactions on Automatic Control*, vol. 58, no. 4, pp. 847–861, April 2013.

[11] C. G. Cassandras, "The event-driven paradigm for control, communication and optimization," *Journal of Control and Decision*, vol. 1, no. 1, pp. 3–17, 2014.

[12] S. Trimpe and J. Buchli, "Event-based estimation and control for remote robot operation with reduced communication," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 5018–5025.

[13] L. Kleinrock, *Queueing Systems, Vol. I: Theory*. Wiley Interscience, 1975.

[14] M. Y. Kitaev and V. V. Rykov, *Controlled Queueing Systems*. CRC Press, 1995.

[15] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[16] M. Kihl, A. Robertsson, and B. Wittenmark, "Performance modelling and control of server systems using non-linear control theory," *Teletraffic Science and Engineering*, vol. 5, pp. 1151–1160, 2003.

[17] T. Patikirikorala, A. Colman, J. Han, and L. Wang, "A systematic survey on the design of self-adaptive software systems using control engineering approaches," in *Proc. 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2012, pp. 33–42.

[18] R. W. Brockett, W. Gong, and Y. Guo, "Stochastic analysis for fluid queueing systems," in *Proc. 38th IEEE Conference on Decision and Control*, Phoenix, Arizona, Dec. 1999.

[19] D. Prangchumpol, P. Sophatsathit, C. Lursinsap, and P. Tantasanawong, "Resource allocation with exponential model prediction for server virtualization," *Journal of Digital Information Management*, vol. 13, no. 5, pp. 385–398, Oct 2015.

[20] S. Arulampalam, S. Maskell, and N. Gordon, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, pp. 174–188, 2002.

[21] T. B. Schön, "Solving nonlinear state estimation problems using particle filters—an engineering perspective," Div. Automatic Control, Linköping University, Sweden, Tech. Rep. ISY-R-2953, May 2010.

[22] J. Abate and W. Whitt, "Calculating time-dependent performance measures for the $M/M/1$ queue," *IEEE Trans. Communications*, vol. 37, no. 10, Oct 1989.

[23] J. Ruuskanen and A. Cervin, "Internal server state estimation using event-based particle filtering," in *Proc. 4th International Conference on Event-Based Control, Communication and Signal Processing*, Perpignan, France, 2018.