Adaptive Kalman Filtering for Anomaly Detection in Software Appliances

Florian Knorn Douglas J. Leith
Hamilton Institute, Nation University of Ireland, Maynooth, Co. Kildare, Ireland

***Abstract*—Availability and reliability are often important features of key software appliances such as firewalls, web servers, etc. In this paper we seek to go beyond the simple heartbeat monitoring that is widely used for failover control. We do this by integrating more fine grained measurements that are readily available on most platforms to detect possible faults or the onset of failures. In particular, we evaluate the use of adaptive Kalman Filtering for automated CPU usage prediction that is then used to detect abnormal behaviour. Examples from experimental tests are given.**

I. Introduction

Software appliances (e. g. firewalls, web servers) are often viewed as "mission critical" and thus required to have high availability and reliability. Use of redundancy and failover between devices has been the subject of considerable attention, with simple heartbeat monitoring used for failover control. However this primarily targets hardware failure and more detailed software monitoring of operation within an individual appliance is much less well developed.

Current best practice in software monitoring makes use of basic event logging. However, the resulting logs are typically presented in fairly raw form to human operators and, due to the sheer volume of even basic event log data, this data is often largely disregarded. That is, in practice faults are typically detected as a result of a degradation in service or user complaints, with the log data then used primarily for diagnosis after the fact. This is highly inefficient on a number of levels. Firstly, pro-active detection of faults (as opposed to waiting for user complaints) can yield improved quality of service. Secondly, although operators are already swamped with log data, in fact this is only a very small subset of the data which can be readily measured. Software appliances are already instrumented to provide far more detailed measurements than are provided by basic event logging. For example, fine grained time histories of CPU and memory utlisation, file system behaviour etc. are all available but unused (beyond perhaps MRTG graphing of coarse 5 minute average data). The potential wins from using this additional data, in addition to early detection of fault conditions, include acceleration of debugging and software fixes. At present, it can be extremely time consuming (weeks or months) to track back from event logs to the source of errors.

Many software appliances do provide manual setting of threshold values for a few variables, with exceptions triggered when a threshold is exceeded. However, anecdotal evidence suggests that this functionality is rarely used. Not only are guidelines for selection of the appropriate threshold values not available, and expected to be strongly dependent on the specific configuration of an appliance, it is clear that for some important quantities simple thresholds are inappropriate. For example, during normal operation CPU utilisation may reach 100 % and simple thresholding is of little use.

While in this context it is natural to consider the use of time history information to support more discriminating inference, solutions are subject to a number of key constraints. These include: (i) any inference algorithm must have a small computational/memory footprint and admit real-time operation, (ii) be suited to unsupervised operation (i. e. without the need for manual tuning and other intervention) and (iii) robust enough to operate in a wide range of environments (although the range is constrained by the fact that an appliance carries out a limited number of tasks). On the plus side, we have access to plenty of data and measurements are error free (no measurement noise). Moreover, a potentially important feature is that false positives need not carry a prohibitive cost. For example, if the action taken is to log detailed debug information on detecting an anomaly, to allow later debugging, false positives carry a relatively low cost (namely, only increased storage) compared with false negatives.

In this paper we propose a Kalman Filter based framework for software appliance monitoring. This builds on early exploratory work reported in [1]. We investigate a number of modelling options and propose use of a novel non-parametric model structure that is both simpler to implement and significantly easier to tune than other approaches. Our approach is suited to online processing of data and efficient implementation (low memory and CPU burden). Its effectiveness for automated detection of fault conditions is demonstrated using measurements from an experimental testbed under a range of operating conditions and a variety of real faults.

II. Related work

Fault, anomaly or deviation detection is a classical research topic and a great wealth of results has been obtained over the years. See for example survey papers such as [2], [3], [4], [5], [6], [7]. Existing techniques differ mainly in the representation or model of the "normal behaviour". This can be, for example, strings or logic-based profiles, artificial neural networks, clustering information, event transition probabilities, statistical distributions, or stochastic models. Time series models, such as ARMA models or Kalman Filters are used for example in [8], [9], [10], [11]. Once residuals are generated, that is the

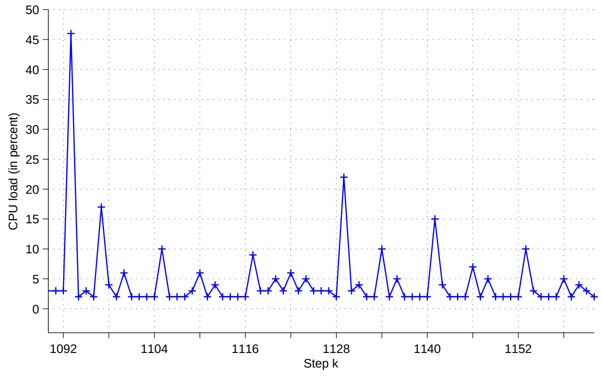


Figure 1. Measured CPU usage as a function of time.

difference between predicted and measured values, statistical significance tests, such as χ^2 or generalized log-likelihood ratios are used to decide whether the observed behaviour is "abnormal" or not.

In the context of software appliances and computer systems, statistical evaluations of variable distributions have been used, [12], [13], [14], [15], [16]. Most of this work, however, makes use of static models rather than time series models. Among the few publications that consider time series models are Soule *et al.* in [17], [18] who consider detecting anomalies in the system wide traffic state of an enterprise or ISP network and [19] who use, among other techniques, discrete time Markov Chain and ARMA models.

III. Predictive Modelling

Our aim is to develop simple predictive models of key software appliance signals in order to capture baseline behaviour which can then be used as the basis for anomaly detection. While a range of signals are potentially of interest, here we focus on modelling of CPU utilisation since this is a primary signal of importance on all appliances. An example time history of measured CPU utilisation on a software appliance is shown in Figure 1. It can be seen that some tasks (e. g. `cron` jobs) are carried out periodically, creating regular spikes in CPU utilisation. The work load varies randomly, however, and this is reflected in the variation in CPU load. Our basic idea is therefore to model the signal of interest (i. e. CPU utilization) y_k at time k as consisting of a periodic component $\mathcal{C}_k$, an event-driven component $\mathcal{E}_k$ and additive noise component $\mathcal{D}_k$ (to cover unmodelled or unexpected signal deviations). That is,

$$y_k = \mathcal{C}_k + \mathcal{E}_k + \mathcal{D}_k \tag{1}$$

While we consider a number of different modelling approaches, all fall within the class of linear time series models with additive Gaussian noise. Statistical inference can therefore be efficiently carried out in an online manner by using standard Kalman Filter tools [20], [21]. In more detail, we assume the following linear state-space model

$$\begin{cases} \boldsymbol{x}_{k+1} = \boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{n}_k, \\ \quad y_k = \boldsymbol{h}^{\mathrm{T}}\boldsymbol{x}_k + v_k, \end{cases} \tag{2}$$

where $\boldsymbol{x}_k$ are the n hidden states, y_k is the observed variable (e. g. CPU load), $\boldsymbol{u}_k$ are the m inputs (e. g. based on the `syslog`) and $\boldsymbol{n}_k$ resp. v_k are zero mean Gaussian process resp. measurement noise with covariances

$$\mathrm{E}\{\boldsymbol{n}_k, \boldsymbol{n}_l\} = \boldsymbol{Q}\,\delta_{kl} = q\boldsymbol{I}\,\delta_{kl}, \quad \mathrm{E}\{v_k, v_l\} = r\,\delta_{kl} \tag{3}$$

where $\boldsymbol{I}$ is the identity matrix of appropriate dimensions and δ_{kl} is the Kronecker delta.

It is mainly the choice of structure for the matrices $\boldsymbol{F}$, $\boldsymbol{B}$ and vector $\boldsymbol{h}^{\mathrm{T}}$ that distinguishes the different models that we consider.

For any model of the form (2), the prediction $\hat{y}_{k+1|k}$ of the signal y_{k+1} given measurements of $y_0, \dots, y_k$ can be calculated using a Kalman Filter. In more detail, we have

$$\hat{\boldsymbol{x}}_{k+1|k} = \boldsymbol{F}\hat{\boldsymbol{x}}_{k|k} + \boldsymbol{B}\boldsymbol{u}_k \tag{4a}$$
$$\boldsymbol{P}_{k+1|k} = \boldsymbol{F}\boldsymbol{P}_{k|k}\boldsymbol{F}^{\mathrm{T}} + \boldsymbol{Q} \tag{4b}$$
$$\hat{y}_{k+1|k} = \boldsymbol{h}^{\mathrm{T}}\hat{\boldsymbol{x}}_{k+1|k} \tag{4c}$$
$$\hat{e}_{k+1|k} = y_k - \hat{y}_{k+1|k} \tag{4d}$$

where $\hat{\boldsymbol{x}}_{k+1|k}$ are predictions of the hidden model state, $\boldsymbol{P}_{k+1|k}$ is the predicted (or *a priori*) covariance matrix of the state, and

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_{k|k}\hat{e}_{k|k-1} \tag{5a}$$
$$\boldsymbol{P}_{k|k} = \left(\boldsymbol{I} - \boldsymbol{K}_{k|k}\boldsymbol{h}^{\mathrm{T}}\right)\boldsymbol{P}_{k|k-1} \tag{5b}$$
$$\boldsymbol{K}_{k|k} = \frac{\boldsymbol{P}_{k|k-1}\boldsymbol{h}}{r + \boldsymbol{h}^{\mathrm{T}}\boldsymbol{P}_{k|k-1}\boldsymbol{h}} \tag{5c}$$

Here, $\boldsymbol{K}_{k|k}$ is called the Kalman Filter gain. Looking at (4b), (5a) and (5c) in more detail, we can see how process noise $\boldsymbol{n}_k$ and measurement noise v_k influence the predicted error covariance matrix $\boldsymbol{P}_{k+1|k}$: If r (the measurement noise variance) is large, i. e. the measurement is corrupted by a lot of noise, the filter gain $\boldsymbol{K}_{k|k}$ will be small, and thus the state estimate $\hat{\boldsymbol{x}}_{k|k}$ will rely more on the prediction $\hat{\boldsymbol{x}}_{k|k-1}$ than on the new measurement y_k. If in turn the process noise $\boldsymbol{n}_k$ (and thus $\boldsymbol{Q}$) is large, the state error covariance matrix in (4b) (and hence the filter gain) will be "large", giving measurements more weight.

A. Form-free model

Given a periodic signal, an intuitive approach to predicting the behaviour of a new cycle is to take, for each sample within the cycle, the average over the corresponding samples from previous cycles. In essence this is the idea underlying the so-called *form-free* modelling approach. It is called form-free because it makes few assumptions about the structure of the data, aside from the fact that it is periodic. Instead the model works directly in terms of the measurements. This represents a key advantage since it provides great flexibility while at the same time being intuitive and relatively easy to understand.

In more detail, recalling the general linear model structure (2), a form-free model is obtained by setting the input matrix $\boldsymbol{B}$ to be zero and selecting the system matrix as [22]

$$\boldsymbol{F} = \boldsymbol{Z}_n = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I}_{n-1} \\ 1 & \boldsymbol{0}^{\mathrm{T}} \end{bmatrix} \tag{6}$$

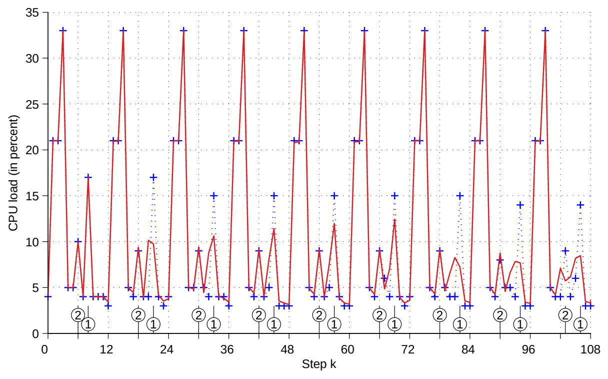


Figure 2. Illustrating application of form-free predictive model. The solid line corresponds to the predictions, the + markers show the measured data, and the circled markers on the bottom indicate two different types of events.

where $\boldsymbol{Z}_n$ is the n dimensional cyclic-permutation matrix and $\boldsymbol{0}$ is the zero matrix (vector) of appropriate dimensions.

In this setting, the state $\boldsymbol{x}_k$ can be interpreted as a vector storing the running average of samples over a period (at $k = 0$ it is initialised with the first n measurements). The system matrix $\boldsymbol{F}$ rotates (permutes) the state vector by one element at each time step, and we have that $\boldsymbol{x}_{k+n} = \boldsymbol{F}^n \boldsymbol{x}_k = \boldsymbol{x}_k$ i. e. the state evolves in a cyclic manner with period n. We can recover each sample by setting the output vector to

$$\boldsymbol{h}^{\mathrm{T}} = \begin{pmatrix} 1 & 0 & \dots & 0 \end{pmatrix} \tag{7}$$

where $\boldsymbol{h}^{\mathrm{T}}$ has n entries. Figure 2 illustrates use of such a form-free model to predict CPU utilisation.

The measurement and process noise covariances in (2) determine, roughly speaking, the number of cycles that the running average is taken over. By averaging over fewer cycles we can adapt our predictions more rapidly to changing conditions, but this comes at the price of increased errors and uncertainty in the predictions. Conversely, by averaging over more cycles we can reduce errors but at the cost of slower adaptation.

In the form-free approach the size of the state vector $\boldsymbol{x}_k$ scales with the number of samples taken over a period and this is a key design driver in the present context. The main difficulty arises when a period involves a mix of time-scales e. g. if the period of a signal is 24 hours but we want to model events occurring at a much finer time scale of seconds or minutes. Note that such situations are likely to be common since there may be regular events at, for example, 5 minute intervals or less plus daily and weekly events (e. g. backups). Furthermore, this approach cannot be used if the periodicity of the signal changes over time.

A related issue can also arise when two (or more) types of events occur periodically, but with periods that are slightly different. This is also illustrated in Figure 2. There are regular spikes in CPU utilisation every 12 time steps — corresponding to tasks processed every 60 s as 5 s time steps are used. However it can be seen that there are also smaller spikes occurring at a slightly different interval (roughly every 12.125 time steps). Since this interval is slightly longer than the first series of spikes, over time these smaller spikes tend to “travel” relative to the larger spikes. As shown in the figure, the form-free model can accurately predict the large spikes but performs less well at predicting the smaller spikes as they drift relative to the large ones. This might be addressed by choosing the model period n to be a value that is an integer multiple of both 12 and 12.125, but the smallest value that satisfies this is $n = 1164$, i. e. a large increase in state dimension is required.

To address these issues, in the following sections we augment the form-free model with an event-driven element to create a new hybrid type of model that combines the advantages of the form-free approach with the low state dimension of event-driven approaches.

B. *Event-driven model*

Before proceeding to consider hybrid models, we briefly introduce purely *event-driven* models. In contrast to form-free models, they do not require any periodicity in the signal of interest. Instead, the requirement is for a model input that signals when an event has occurred and triggers a certain CPU utilisation profile. The CPU utilisation profile can be stored either in the form of parametric or form-free submodels.

1) Parametric event submodel: Suppose that there are m different types of events to be modelled. We associate input $u_{k,i}$ with the ith type of event and set $u_{k,i} = \gamma_i$ when event i occurs at time step k and zero otherwise. The value of γ_i can be adjusted to provide flexibility to capture relevant features of the event. The CPU utilisation profile associated with the ith type of event may then be generated as the output of the dynamic system

$$\boldsymbol{x}_{k+1,i} = \boldsymbol{A}_i \boldsymbol{x}_{k,i} + \boldsymbol{B}_i \boldsymbol{u}_{k,i}, \quad y_{k,i} = \boldsymbol{C}_i \boldsymbol{x}_{k,i} \tag{8}$$

where $\boldsymbol{x}_{k,i}$ is the state vector and the matrices $\boldsymbol{A}_i$, $\boldsymbol{B}_i$ and $\boldsymbol{C}_i$ are design parameters. In the simplest case we use a first order dynamic system

$$x_{k+1,i} = a_i x_{k,i} + u_{k,i}, \quad y_{k,i} = x_{k,i} \tag{9}$$

where $x_{k,i}$ is now a scalar. The parameter a_i determines the shape of the output profile, and by adjusting the magnitude γ_i of the input the height of the pulse generated can be controlled.

2) Form-free event submodel: Event profiles can also be stored in a form-free manner. The idea here is to “play back” a stored profile whenever an associated event occurs. If the shape consists of n_i consecutive points, the dimension of the state vector $\boldsymbol{x}_{k,i}$ would be n_i as well, and the submodel would be

$$\boldsymbol{x}_{k+1,i} = \boldsymbol{A}_{k,i} \boldsymbol{x}_{k,i}, \quad y_{k,i} = \boldsymbol{C}_{k,i} \boldsymbol{x}_{k,i} \tag{10}$$

Here, if the event i occurs at $k = k'$, we have

$$\boldsymbol{A}_{k,i} = \begin{cases} \boldsymbol{Z}_{n_i} & \text{for } k = k', \dots, k' + n_i - 1, \\ \boldsymbol{I}_{n_i} & \text{otherwise} \end{cases} \tag{11}$$

and $\boldsymbol{C}_{k,i}$ of the form in (7) for $k = k', \dots, k' + n_i - 1$, zero otherise. The actual profile, if available, can be stored in the initial condition of the state vector, or the filter can “find it” by adapting the states, over time, from the measurements.

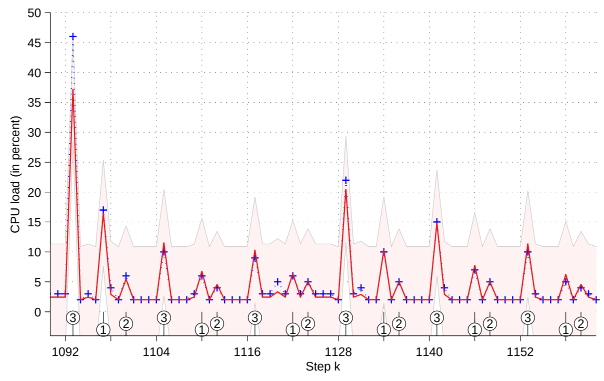


Figure 3. Illustrating predictions using pure event-driven model. Shaded region marks the one standard-deviation confidence interval.

3) Overall model: Combining the individual event submodels we obtain a system of the form (2), with state $\boldsymbol{x}_k^{\mathrm{T}} = \begin{pmatrix}\boldsymbol{x}_{k,1}^{\mathrm{T}} & \dots & \boldsymbol{x}_{k,m}^{\mathrm{T}}\end{pmatrix}$; and system and input matrices having the block diagonal form

$$\boldsymbol{F} = \begin{bmatrix} \boldsymbol{A}_1 & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}_2 & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \cdots & \boldsymbol{A}_m \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_1 & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{B}_2 & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \cdots & \boldsymbol{B}_m \end{bmatrix} \tag{12}$$

and output vector

$$\boldsymbol{h}^{\mathrm{T}} = \begin{pmatrix} \boldsymbol{C}_1 & \boldsymbol{C}_2 & \dots & \boldsymbol{C}_m \end{pmatrix} \tag{13}$$

Note that any mixture of parametric and form-free event submodels can be used as appropriate.

An example result from the pure input model approach is shown in Figure 3. The occurrence of the three events is marked by, respectively, ①, ② and ③. It can be seen than the event-driven model accurately captures the CPU utilisation.

Since the event-driven model responds directly to observed events it can readily accommodate events that happen on a range of time-scales e. g. an event occurring every 5 minutes and an event occurring once every 24 hours are essentially treated in the same way, only the input $u_{i,k}$ changes. However, the event-driven model does require appropriate inputs, and so implicitly requires that the occurrence of all relevant events can be detected. This issue is illustrated, for example, in Figure 3 — around $k = 1120$ there is an increase in CPU utilisation that is not associated with any of the three events modelled.

Further, it can be seen that this purely event-driven approach requires that individual models be defined that store the profile associated with each event. The number of states in the model therefore scales with the number of events, although this might be mitigated by combining inputs i. e., effectively reusing event models. For simple types of stored profile (pulses etc.), the number of states in the individual event models is typically higher in the form-free case than when a parametric model is used. However, while the parametric form is more compact, it has only very limited ability to adapt to changes in observed behaviour (adaptation requires adjustment of the model parameters $\boldsymbol{A}_i$, $\boldsymbol{B}_i$, $\boldsymbol{C}_i$) whereas the form-free model essentially stores the average of the last few instances of an event in the state vector $\boldsymbol{x}_{k,i}$ and so can automatically adapt to changes.

Remark: In the event-driven approach, when two events occur simultaneously their CPU profiles are combined additively. It is this that makes the model linear, and makes statistical inference straightforward. However, in practice we know that the CPU utilisation has a hard "saturation" constraint, namely it cannot exceed 100 % (or go below 0 %). We note that this issue is mitigated when using a coarse sampling interval for CPU utilisation. For example, in our tests we sample CPU utilisation at 5 s intervals. The sampled values are not the instantaneous CPU utilisation at the sample time, but rather the average CPU utilisation over the 5 s interval (CPU utilisation is recorded by a counter, and the average value is just the difference in counter values at sample instants, divided by the interval duration). Unless the CPU utilisation remains at 100 % for the full 5 s interval, the average utilisation will remain less than 100 %.

C. Hybrid Model

The pure form-free model of Subsection III-A is simple and flexible but is only suited to periodic signals and the state dimension can become large when there are a mix of time-scales. The pure event-driven model of Subsection III-B can accommodate non-periodic events and a mix of time-scales with ease, but requires an input trigger for all relevant events plus the state dimension scales in proportion to the number of events. We therefore propose a hybrid approach that plays to the strengths of both approaches and tries to avoid their weaknesses. Namely, regular periodic components of the signal of interest are modelled using a form-free approach and irregular events, or events at a different time-scale from the main periodic components, are modelled using an event-driven approach. The resulting hybrid model has $\boldsymbol{F}$, $\boldsymbol{B}$ matrices of block diagonal form similar to (12), with each block corresponding either to a form-free periodic model as in Subsection III-A or an event submodel (which in turn might be parametric as in III-B1 or form-free as in III-B2), and output vector $\boldsymbol{h}^{\mathrm{T}}$ as in (13).

IV. Automated model tuning

To achieve automated operation we need to extend the Kalman Filter to recursively adapt the input gains in (8) as well as the noise and measurement covariances $\boldsymbol{Q}$ and r in (4b) and (5c).

A. Input gain adaptation

Following a classical extended Kalman Filter approach, e. g. see [23], we add m states (one for each input) to our linear state space model (2)

$$\begin{cases} \begin{pmatrix} \boldsymbol{x}_{k+1} \\ \boldsymbol{\theta}_{k+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{B}(\boldsymbol{\theta}_k)\boldsymbol{u}_k \\ \boldsymbol{\theta}_k \end{pmatrix} + \begin{pmatrix} \boldsymbol{n}_k \\ \boldsymbol{\zeta}_k \end{pmatrix} \\ y_k = \begin{pmatrix} \boldsymbol{h}^{\mathrm{T}} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_k \\ \boldsymbol{\theta}_k \end{pmatrix} + v_k \end{cases} \tag{14}$$

where $\boldsymbol{B}(\boldsymbol{\theta}_k)$ is such that the elements of $\boldsymbol{\theta}_k$ constitute the different input gains $\gamma_{k,i}$ at time k from the different inputs i to the corresponding states, and the new Gaussian white noise sequence $\boldsymbol{\zeta}_k$ has a very small variance covariance matrix $\boldsymbol{S}$, as we assume the true gains to be constant values.

It is straight forward to extend the Kalman Filter equations from Section III with these states, noting that these states do not (directly) affect the output variable y_k. The new *a priori* estimate of the state error covariance matrix of the filter becomes

$$\boldsymbol{P}_{k+1|k} = \begin{bmatrix} \boldsymbol{F} & \boldsymbol{L}_{k|k} \\ 0 & \boldsymbol{I} \end{bmatrix} \boldsymbol{P}_{k|k} \begin{bmatrix} \boldsymbol{F} & \boldsymbol{L}_{k|k} \\ 0 & \boldsymbol{I} \end{bmatrix}^{\mathrm{T}} + \begin{bmatrix} \boldsymbol{Q} & 0 \\ 0 & \boldsymbol{S} \end{bmatrix} \quad (15)$$

with $\boldsymbol{L}_{k|k} = \frac{\partial}{\partial \boldsymbol{\theta}} \left[\boldsymbol{F}\hat{\boldsymbol{x}}_{k|k} + \boldsymbol{B}(\boldsymbol{\theta})\boldsymbol{u}_k \right] \Big|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_{k|k}}$.

B. Noise estimation

Following Jazwinski *et al.* [24], [25] we used the following method for estimating the process noise covariance (3)

$$\hat{q}_k = \alpha_q \hat{q}_{k-1} + (1-\alpha_q)\ h \left(\frac{\hat{e}^2_{k+1|k} - \left(\boldsymbol{h}^{\mathrm{T}} \boldsymbol{F} \boldsymbol{P}'_{k|k} \boldsymbol{F}^{\mathrm{T}} \boldsymbol{h} + r \right)^2}{\boldsymbol{h}^{\mathrm{T}} \boldsymbol{h}} \right) \quad (16)$$

where $\boldsymbol{P}'_{k|k}$ is the top left block of $\boldsymbol{P}'_{k|k}$ corresponding to the states $\hat{\boldsymbol{x}}$, $0 < \alpha_q < 1$ is a smoothing parameter (very close to 1 since, again, we assume the noise covariance to be constant) and $h(\cdot)$ is the ramp function, that is $h(x) = x$ if $x > 0$ and zero otherwise. The update is executed prior to (4).

In a similar fashion, we use a recursion for the measurement noise r, that is updated prior to (5)

$$\hat{r}_k = \alpha_r \hat{r}_{k-1} + (1-\alpha_r)\ h \left(\hat{e}^2_{k|k-1} - \boldsymbol{h}^{\mathrm{T}} \boldsymbol{P}'_{k|k-1} \boldsymbol{h} \right) \quad (17)$$

V. Anomaly detection

With the Kalman Filter deployed and running, one could simply declare an anomaly when a new measurement lies outside the confidence interval around its predicted value, that is $\hat{y}_k \pm \sigma^2_{k|k-1}$, where $\sigma^2_{k|k-1} = (\boldsymbol{h}^{\mathrm{T}} \boldsymbol{P}'_{k|k-1} \boldsymbol{h} + \hat{r}_k)$. However, this simple approach leads to a large number of false positives, as the decision is based only on a single data point / estimate.

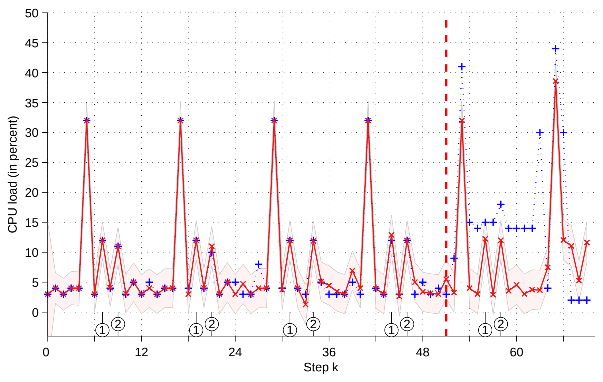
A combination of several measurements yields significantly better (more robust) results. The likelihood of an observed data point can be readily calculated based on the predicted probability distribution provided by the Kalman Filter [24]

$$p(y_k | y_{k-1}, \ldots, y_0) = \frac{1}{\sqrt{2\pi\sigma^2_{k|k-1}}} \exp\left(-\frac{\hat{e}^2_{k|k-1}}{\sigma^2_{k|k-1}} \right) \quad (18)$$

If we now use a moving average (or low pass) filter on the log likelihoods of past measurements, we can obtain a significantly more robust anomaly indicator. Define

$$z_k = \alpha_z z_{k-1} + (1-\alpha_z) \ln p(y_k) \quad (19)$$

with a suitable smoothing factor α_z, and set a threshold for alarm generation in case a certain value (log likelihood) is undercut.



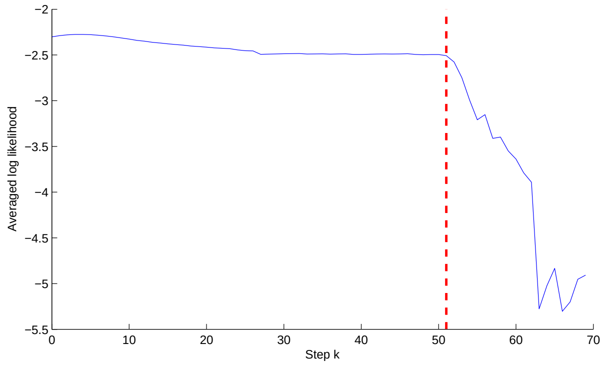


Figure 4. Detection of software fault triggered at $k = 51$.

VI. Experimental Results

We carried out measurements for three different anomalies to explore the utility of the proposed modelling approach for anomaly detection.

(i) Figure 4 shows measurements for a known bug that causes memory usage to grow and CPU load to increase temporarily. Following the fault at $k = 51$ it can be seen that the model prediction deviates significantly from the measured CPU utilisation and the log likelihood drops.

(ii) Figure 5 shows data from a memory overflow fault at $k = 610$. This fault is generated by forcing the appliance to run out of memory, crashing the main software process.

(iii) Figure 6 shows a simulated fault, generated at step $k = 940$, when a static 10 % CPU load is added on top of the measured data.

It can be seen that the proposed modelling approach allows a clear detection of these three anomalies, while generating no false alarms in our tests.

VII. Conclusion

Availability and reliability are often important features of key software appliances such as firewalls, web servers, etc. In this paper we investigate going beyond simple heartbeat monitoring by using more fine grained measurements that are readily available on most platforms to detect possible faults or the onset of failures.

We propose a new self-tuning, extended Kalman Filter based framework and demonstrate the effectiveness of the proposed approach using tests of real faults generated on

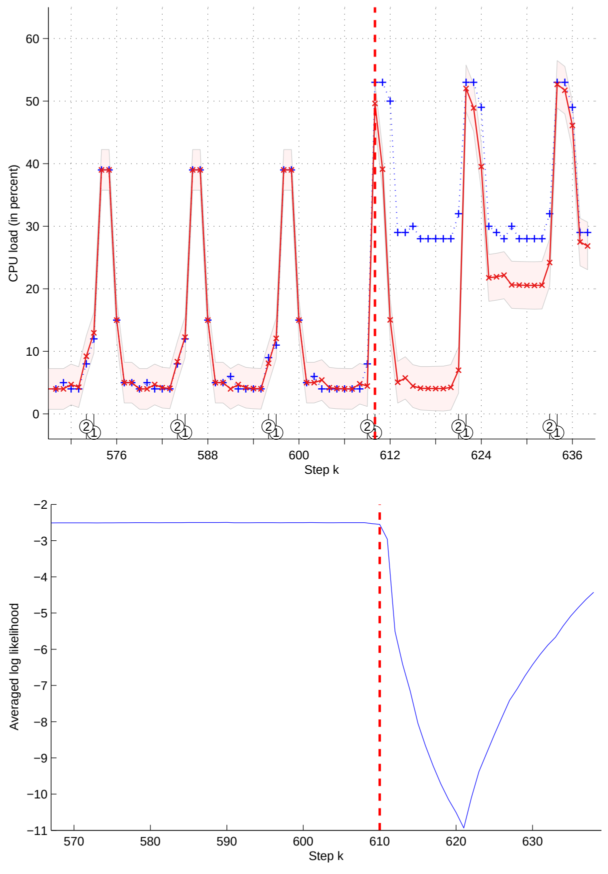


Figure 5. Detection of out of memory fault at $k = 610$.

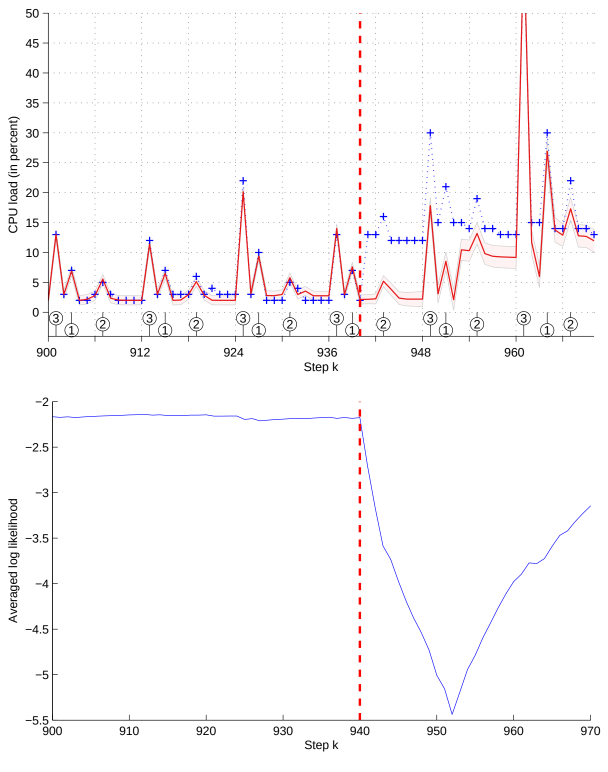


Figure 6. Detection of artificial fault generated at $k = 940$.

an experimental testbed. Our approach is suited to online processing of data and efficient implementation (low memory and CPU burden).

References

[1] F. Knorn and D. J. Leith, "Real–time anomaly detection in software appliances," Poster presented at MLSys'07 Workshop, Whistler, BC, Canada, 2007.

[2] A. S. Willsky, "A survey of design methods for failure detection systems," *Automatica*, vol. 12, pp. 601–611, 1976.

[3] R. Isermann, "Process fault detection based on modelling and estimation methods—A survey," *Automatica*, vol. 20, no. 4, pp. 387–404, 1984.

[4] ——, "Model–based fault-detection and diagnosis — status and applications," *Ann. Rev. Control*, vol. 29, no. 1, pp. 71–85, 2005.

[5] V. Venkatasubramanian, R. Rengaswamy, K. Yin, and S. N. Kavuri, "A review of process fault detection and diagnosis part 1," *Comp. Chem. Eng.*, vol. 27, no. 3, pp. 293–311, 2003.

[6] V. Venkatasubramanian, R. Rengaswamy, and S. N. Kavuri, "A review of process fault detection and diagnosis part 2," *Comp. Chem. Eng.*, vol. 27, no. 3, pp. 313–326, 2003.

[7] V. Venkatasubramanian, R. Rengaswamy, S. N. Kavuri, and K. Yin, "A review of process fault detection and diagnosis part 3," *Comp. Chem. Eng.*, vol. 27, no. 3, pp. 327–346, 2003.

[8] P. M. Frank, "On–line fault detection in uncertain nonlinear systems using diagnostic observers: a survey," *Int. J. Sys. Science*, vol. 25, no. 12, pp. 2129–2154, 1994.

[9] C. M. Hajiyev and F. Caliskan, *Progress in system and robot analysis and control design*. Springer, 1999, ch. Fault Detection in Flight Control Systems via Innovation Sequence of Kalman Filter, pp. 63–74.

[10] L. An and N. Sepehri, "Hydraulic actuator circuit fault detection using extended kalman filter," in *Proc. ACC'03*, vol. 5, Denver, CO, USA, 2003, pp. 4261–4266.

[11] Y. Chetouani, "Fault detection in a chemical reactor by using the standardized innovation," *Process Safety and Envir. Protection*, vol. 84, no. B1, pp. 27–32, 2006.

[12] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, and S. Ma, "Critical event prediction for proactive management in large-scale computer clusters," in *Proc. SIGKDD'03*, Washington, DC, USA, 2003, pp. 426–435.

[13] M. Burgess, H. Haugerud, S. Straumsnes, and T. Reitan, "Measuring system normality," *ACM Trans. Comp. Sys.*, vol. 20, no. 2, pp. 125–160, 2002.

[14] N. Ye, Q. Chen, and C. M. Borror, "EWMA forecast of normal system activity for computer intrusion detection," *IEEE Trans. Rel.*, vol. 53, no. 4, pp. 557–566, 2004.

[15] R. Powers, M. Goldszmidt, and I. Cohen, "Short term performance forecasting in enterprise systems," HP Laboratories, Palo Alto, CA, USA, Tech. Rep. HPL-2005-50, 2005.

[16] V. A. Siris and F. Papagalou, "Application of anomaly detection algorithms for detecting SYN flooding attacks," in *Proc. GLOBECOM'04*, vol. 4, Dallas, TX, USA, 2004, pp. 2050–2054.

[17] A. Soule, K. Salamatian, and N. Taft, "Combining filtering and statistical methods for anomaly detection," in *Proc. IMC'05*, Berkeley, CA, USA, 2005, pp. 331–344.

[18] A. Soule, A. Lakhina, N. Taft, and K. Papagiannaki, "Traffic matrices: balancing measurements, inference and modeling," in *Proc. SIGMETRICS'05*, 2005, pp. 362–373.

[19] G. A. Hoffmann, F. Salfner, and M. Malek, "Advanced failure prediction in complex software systems," Humbold Univ. Berlin, Dep. of Comp. Science, Berlin, Germany, Research Report 172, 2004.

[20] R. E. Kalman, "A new approach to linear filtering and prediction problems," *J. Basic Eng.*, vol. 82, pp. 34–45, 1960.

[21] T. Kailath, A. H. Sayed, and B. Hassibi, *Linear Estimation*. Prentice–Hall International, Inc., 2000.

[22] M. West and J. Harrison, *Bayesian Forecasting and Dynamic Models*. New York, NY, USA: Springer, 1997.

[23] C. K. Chui and G. Chen, *Kalman Filtering with Real–Time Applications*. New York, NY, USA: Springer, 1998.

[24] A. H. Jazwinski and A. E. Bailie, "Adaptive filtering," Interim report, No. 67-6, 1967.

[25] A. H. Jazwinski, "Adaptive filtering," *Automatica*, vol. 5, no. 4, pp. 475–485, 1969.