Real-Time Anomaly Detection in Software Appliances

Florian Knorn, Douglas J. Leith

Hamilton Institute, NUI Maynooth, Ireland

NUI MAYNOOTH
Ollscoil na hÉireann Má Nuad

Hamilton Institute

1. State of the art

- Software appliances (e.g. firewalls, webservers, etc.) in cases “mission critical”
- Anomaly detection currently usually only covers **hardware** failures
- Only **rudimentary** software failure detection:
  - software monitoring = event logging
  - fault detection only through degradation in service / user complaints
- Available, fine grained information (such as CPU & memory usage) **rarely used**
- Thresholds to trigger alarms can be set, however
  - **few guidelines** / heuristics for setting them
  - strongly **situation dependent**
  - **inappropriate** in many situations (e.g. CPU usage may well reach 100% in normal operation)

2. Objective

Use of fine grained system “health” statistics and time series to create robust failure detection algorithms that:

- are of **little** computational burden / memory footprint
- allow **real-time** operation
- allow **unsupervised** operation (i.e. no manual tuning or other intervention)
- are **robust** enough to operate in a wide range of environments

In this poster, we present a starting point in that direction using linear dynamic models.

3. Linear dynamic models

We use a linear state-space multi-input / single-output system with two noise components:

$$\boldsymbol{x}_{k+1} = \boldsymbol{F}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{g}_k n_k,$$
$$y_k = \boldsymbol{h}^T \boldsymbol{x}_k + v_k,$$

with $\boldsymbol{x}_k$: hidden state vector

y_k: observed variable

$\boldsymbol{u}_k$: know input vector (3 inputs in our application)

n_k, v_k: zero mean Gaussian proc. resp. sys. noise.

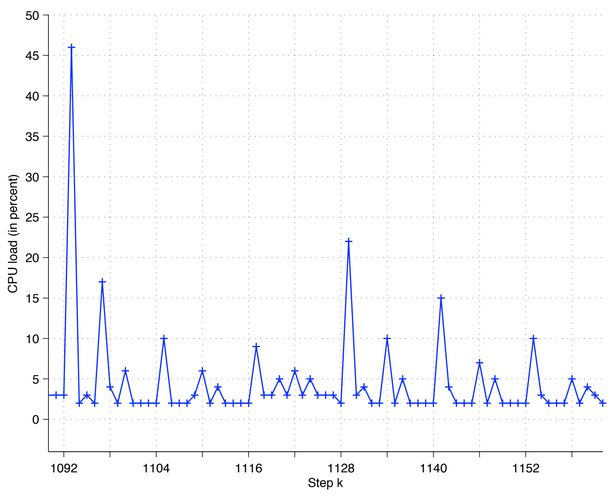

***Fig. 1**: Measured CPU time history*

Fig. 1 illustrates a typical measured time history of CPU utilisation.

- **Periodic** component (with period 12)
- Smaller spikes “travel” relative to the “big” spikes
  - ➡ Use mixture of form-free component ($p{=}12$ states) and
  - ➡ input component (one first order system for each of the $q{=}3$ inputs)

$$\boldsymbol{F} = \begin{bmatrix} \boldsymbol{Z}_p & \boldsymbol{0}_{p\times q} \\ \boldsymbol{0}_{q\times p} & \mathrm{diag}(\boldsymbol{\delta}) \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} \boldsymbol{0}_{p\times q} \\ \boldsymbol{I}_q \end{bmatrix}, \quad \boldsymbol{h} = \begin{pmatrix} 1 \\ \boldsymbol{0}_{(p-1)\times 1} \\ \boldsymbol{1}_{q\times 1} \end{pmatrix}, \quad \boldsymbol{g} = \begin{pmatrix} \gamma_k \\ \boldsymbol{1}_{((p-1+q)\times 1)} \end{pmatrix}$$

where $\boldsymbol{Z}_p$ is the circular permutation matrix, $\boldsymbol{0}$, $\boldsymbol{1}$ and $\boldsymbol{I}$ resp. the zero, one and identity matrix of indicated dimensions, $\boldsymbol{\delta}$ the vector of time constants and γ_k a time dependent variable to increase process noise in case of input.

- Only **few** hyper parameters involved here are:
  - variances of the process and system noise
  - time constants for first order systems for the inputs (may well be 0)
  - the noise increasing factors γ_k

4. Estimation with proposed model

- Classical **Kalman filtering** / smoothing can be used to estimate the model
- Allows efficient **Bayesian inference** for fault detection
- Relatively small computational burden
- Fig. 2 shows the one step ahead predictions of a Kalman filter using our model (solid line).
- The three input events considered are marked on the plot, as well as the confidence band (based on the estimated error covariance of the output).

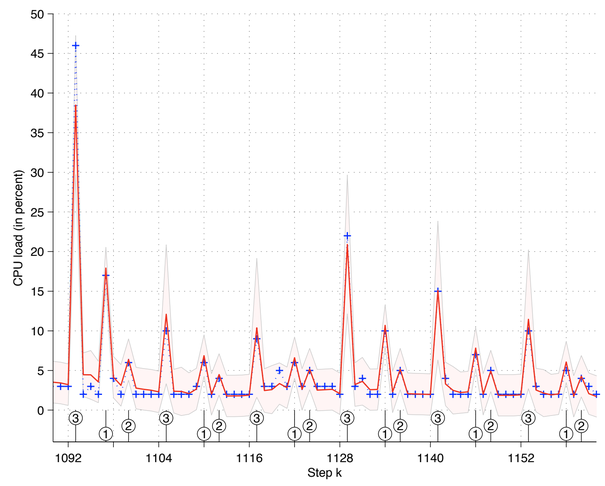

***Fig. 2**: One step ahead predictions using the proposed model*

5. Fault detection

- Fig. 3 shows performance of the filter during a simulated software fault, starting at time $k{=}1131$. The fault caused an additional 15% CPU load on top of the normal load.
- Outliers are marked with a read circle
- A threshold based trigger may **never** have detected this situation since CPU spikes in normal operation are higher than during faulty operation.

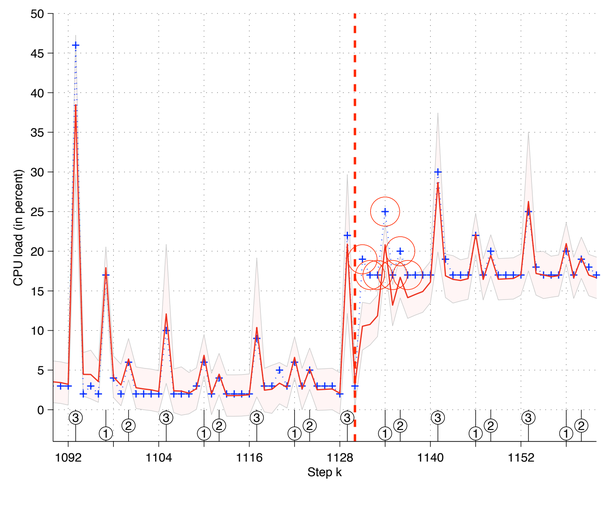

***Fig. 3**: Prediction and detection of a simulated fault*

6. Outstanding questions / future directions

- **Nonlinear combination of events**: by definition, CPU load cannot be higher than 100%. When to events occur that cause high CPU load, the value will saturate and stretch over a longer period of time than each of the events would normally take. It is fair to assume that the *area* under the resulting CPU curve will be the same for to events occurring separately or at the same time.
- **Increasing complexity:** state vector can grow considerably if we consider
  - larger the time-scales
  - more events
  - more complex CPU utilisation shapes caused by an event
  - combination of events (e.g. to cope with combination of events)
- **More sophisticated anomaly detection:** at the moment, faults are triggered simply based on consecutive excursions from the predicted envelope. Better results should be obtained by using cumulative sum test, likelihood ratio base methods, etc. to gain better sensitivity against spurious short excursions.
- **Fault classification:** being able to classify a fault (i.e. beyond detecting a fault, also identifying it) would also be of great use in many applications, for instance in the development stages of a large software application
- **Selection of informative variables:** many variables are available for monitoring. The selection of the most informative subset (beyond CPU usage) has not received much attention yet.
- **Combination of variables:** apart from the selection process, the combined use of different variables should yield better results than the use of single variables
- **Active probing:** passive monitoring may be augmented by active probing. For instance, a process could be given a specific task and its resulting CPU use closely monitored to see if it behaves in line with historical data.
- **Structured environment:** In software appliances the number of processes running are very limited and usually fairly constant. A question remains how well proposed fault detection methods scale with increasing versatility of the appliance / versatility of its tasks.