Unsupervised Anomaly Event Detection for VNF Service Monitoring using Multivariate Online Arima

Florian Schmidt*, Florian Suri-Payer*, Anton Gulenko*, Marcel Wallschläger*, Alexander Acker* and Odej Kao*
*Complex and Distributed IT-Systems Group, TU Berlin, Berlin, Germany
Email: {firstname.lastname}@tu-berlin.de

***Abstract*—Cloud computing provides companies large scale access to virtual resources, offering cost efficient and flexible usage of digital resources at any time. Thus, companies digitalize their dedicated hardware solutions to virtualized services, which can run in a cloud environment. For example, telecommunication providers move their IP multimedia subsystems, which currently run on dedicated hardware, into the cloud. As the dedicated hardware solutions provided a reliability of 99.999% in the past, the same high reliability is demanded for the virtualized services. But these come with higher complexity due to the fragile computation stack and cannot provide such high requirements. Future zero touch administration systems can help to detect automatically anomalies, find root causes and execute automated remediation actions, providing, providing the opportunity to increase the reliability of the overall system. Thus, this work focusses on the detection of degraded state anomalies.We propose an unsupervised detection approach using a multivariate version of the Online Arima forecasting algorithm consuming realtime monitoring data. This approach is evaluated on a testbed running an open source implementation of the IP multimedia subsystem (Clearwater) executed on a replicated Openstack cloud environment. Results show the applicability of the Online Arima based anomaly detection with high detection rates and low number of false alarms.**

***Index Terms*—anomaly detection; service reliability; NFV**

I. Introduction

Cloud based infrastructure is nowadays used frequently in large scale data centres, providing flexible and scalable resources to their costumers. The virtualization helps to uncouple the dependencies from the physical machines, resulting in virtual services, which can be migrated and scaledout easily among the infrastructure. As the easy usage and flexibility provides access to more and more companies, this popularity also drives the digitalization of previously operated hardware solutions. For example, telecommunication provides softwarize their dedicated hardware solutions (e.g. their IP multimedia subsystems (IMS)) in order to benefit from this flexibility and cost-effectiveness. As these dedicated hardware solutions provided a reliability of 99.999% [1], the companies demand the same high reliability for their virtualized solution. Due to the complexity of the computation model, softwarized components cannot cope with the high demand of reliability. The fragility of such systems cause the usage of large numbers of system administrators, maintaining the continuous operation of the services [2].

The administrators use both active and passive monitoring tools for measuring the systems performance, providing information for the different components of the overall service. Based on these measurements, experts set fixed thresholds triggering automated alarms in order to inform the administrators. As these fixed thresholds are dependent on the individual service and operating system resource usages over time, those may alarm too often or too late, leading a component to fail. Thus, the maintenance of helpful thresholds is difficult as the systems behaviour also might change over time. Finding an anomaly, before it causes a component to fail, is therefore a challenging task as it is highly dependent on the systems individual behaviour. Human administrators need therefore a lot of time to investigate historic data to gain detailed insights for finding the causing host. Only then, they are able to start accurate counter actions, keeping the system high reliable. Self-healing pipelines are proposed to automatically detect and remediate anomalous systems, helping current administrators to detect more efficiently problems and give recommendations to select accurate counter actions [3].

Therefore, we propose an unsupervised anomaly event detection approach in order to detect automatically abnormal changes of service behaviours. Thus, we like to detect degrated state anomalies before components fail and might harm the overall system performance. Our approach is based on the forecasting algorithm Online Arima [4], which is used for learning the normal system behaviour of an individual component of interest using a multivariate model. We assume, that the system runs most of its time in a normal state and anomalies happen rarely. Consequently, we continuously learn each monitored data point leading to a model representing a normal state of that component. Then, we predict with this model the next expected monitoring data and compare it with the actual measured monitoring values. As the model should be able to predict more accurate normal behaviour, the difference between those values should be smaller than those of anomalies. These differences will be detected and recommended to the administrator. As we monitor individual components, the approach is scalable for distributed systems and administrators gain access to individual component behaviours. Furthermore, we concentrate in this work on the service layer and provide an evaluation based on a testbed running an open source implementation of the IP multimedia subsystem executed on a replicated Openstack cloud environment.

The rest of the paper is structured as followed. Next, we present related approaches capturing the state of the art of further anomaly and state change detection methods. Section III describes the background of the Online Arima algorithm, which is used within our approach. Section IV describes our unsupervised, anomaly detection approach. Section V shows the evaluation setup capturing the infrastructure testbed and analysis pipeline. Section VI describes and discusses the results. Lastly, Section VII concludes this paper and gives an outlook for future work.

II. Related Work

Anomaly detection approaches were studied over many years in diverse domains. The approaches can be categorized into supervised and unsupervised methods. Supervised approaches use labelled data from e.g. experts to train their machine learning models. As supervised approaches need labelled training data, those are often insufficient or not able to be provided by experts as this might be too time consuming. Of cause the possibility exists, to inject anomalies into a productive used infrastructure in order to obtain training data, but this may harm the system, which is not desired. For practical usage, unsupervised methods are therefore investigated as they do not consume labelled data, but make assumptions about the structure of data. As this, work focusses on the area of service reliability, we condense the related work in this area for both supervised and unsupervised approaches.

Sauvanaud et al. [5] examine in their work a supervised ensembler approach for detecting anomalies of a Virtual Network Function (VNF) scenario similar to our setup as presented in this work. They combine several different supervised classification algorithms performing individual predictions by using weighted voting mechanism in order to predict for individual services whether the service runs in a normal or abnormal state. Also, Liu et al. [6] propose a supervised anomaly detection strategy. As bases, they use the concept of self organizing maps (SOM). The SOM captures the overall infrastructure with all services and makes predictions about the system's performance. Based on this model, they show to detect anomalies within virtual machines are detected in related regions of the infrastructure.

Dean et al. [7] are also using SOMs, but propose an unsupervised detection model capturing again the overall infrastructure. They aim to predict failures within a cloud infrastructure and also show the applicability to infer the root cause of a failure. Bhattacharyya et al. [8] show the applicability of using recurrent neural networks for learning online wise resource consumption patterns for resource intensive applications like Cassandra or MongoDB databases. In contrast to these deeplearning models, Cotroneo et al. [9] use simple statistical correlation analysis between system components in order to automatically detect anomalies within an network function virtualization (NFV) use case. Changes within these correlation are recommended as anomalies.

Chandola et al. [10] aggregate further concepts for anomaly detection techniques used in this field in a survey.

III. Online Arima

Autoregressive Moving Average Model (ARMA) [11] [12] is a common approach for time series analysis. The ARMA model can identify latent correlation within the data, allowing forecasting future values. But this assumes that we know the complete time series, such that there is no missing data.

In order to overcome the above described problem, Liu et al. [4] propose a game theoretic framework for online learning. An online player sequentially commits in their setting to a decision of forecasting the next data point and consequently suffers a loss. They assume coefficient vectors (α, β) that are set by an adversary and are at no time disclosed to the learner. Since the learning party has no ability to infer the actual noise term ε_t of a time step, this is generated by the adversary, while remaining undisclosed to the learner.

We refer to X_t as the data received at time t. The learner predicts a value $\tilde{X}_t$ and learns the actual value X_t, subsequently suffering a loss l_t:

$$l_t(X_t, \tilde{X}_t) = l_t(X_t, \nabla^d \tilde{X}_t + \sum_{i=0}^{d-1} \nabla^i X_{t-1}) \tag{1}$$

Effectively, the original model ARIMA(k, d, q) will be approximated by another model ARIMA($k+m$, d, 0). The noise terms are dropped and attempted to be compensated by extending the number of lags to regress upon by m. A new $(k+m)$-dimensional coefficient vector γ weighs past observations, formulating the model:

$$\tilde{X}_t(\gamma) = \sum_{i=1}^{k+m} \gamma_i \nabla^d X_{t-i} + \sum_{i=0}^{d-1} \nabla^i X_{t-1} \tag{2}$$

and subsequently the loss:

$$l_t(X_t, \tilde{X}_t) = l_t(X_t, \sum_{i=1}^{k+m} \gamma_i \nabla^d X_{t-i} + \sum_{i=0}^{d-1} \nabla^i X_{t-1}) \tag{3}$$

Liu et al. present two learning appraoches for the Online Arima variant, each of which employs an online convex optimization solver [13] in order to iteratively learn suitable model parameters γ. We refer to γ^t the current coefficients for a given time step t. The first algorithm employs the Online Newton Step (ONS) procedure [14], while the second algorithm utilizes Online Gradient Descent (ODG) [15]. While the ODG variant is computationally more efficient, the ONS approach provides a more favourable regret boundary that entails more accurate predictions. Moreover, choosing a suitable learning rate is harder in the gradient descent setting, yet crucial to achieve a good model approximation. Thus, we consider ONS to be used as learner within this work. Nevertheless, the ODG approach might be favourable if the lag-window is large, wherefore the practical application of both algorithms is entertained.

IV. Unsupervised Anomaly Detection

For anomaly detection, we assume that most data within the monitored system is normal. Thus, we like to learn the normal behaviour of a service component and recommend drastic changes. Let function $s : X \rightarrow X$ represent perfectly the data stream, such that for a given data point $x \in X$ the prediction is always $s(x) = x$. For approximating $s(x)$, we use the Online Arima model to predict the current data point.

We specify our Arima model via two parameters, the window size w and the differencing depth d. The original Arima model utilizes three parameters, k, d and q, to pass lags and errors. We designate $k+m$ as our window size w, as it specifies the amount of relevant lagged data points.

The parameters window size w, as well as differencing depth d are statically defined when initializing the model. Additionally, we note the utilization of a learning rate η necessary for the model learning steps, yet we digress, as its impact is secondary in the ONS approach.

In each time step a new data point X_t is observed for which the model is adapted. In order to train the model, the linear factors γ_i are adjusted according to the ONS procedure given by Hazan et al. [14]. We compare the received data point X_t to its forecasted value $\tilde{X}_t$ using the squared distance as loss metric. The loss gradient according to γ is then computed and utilized to execute the modified Newton Step, where the Matrix A corresponds to the Hessian Matrix H:

$$\gamma^{t+1} = \gamma^t - \eta \cdot H[l_t(\gamma^t)]^{-1} * \nabla l_t(\gamma^t) \tag{4}$$

We recognize the domain space for our coefficients as continuous, and thus omit the projection proposed in the original algorithm. In contrast to Liu et al., we denote no discrete feature space for the coefficients and assume them to be continuous. We therefore randomly initialize $\gamma_i \in [-0.5, 0.5]$, $\forall i \in [1, \dots, w]$ and update them ensuing each observation.

Upon receiving the true value of the data point X_t, we update the model based on the prediction error (loss). In accordance with the algorithm described by Liu et al. we compute:

$$\gamma^{t+1} = \gamma^t - \frac{1}{\eta} \cdot A_t^{-1} * \nabla t \ , \tag{5}$$

where $A_t = A_{t-1} + \nabla t * \nabla t^T$, and $\nabla t := \nabla l_t[\gamma^t](X_t, \tilde{X}_t)$. As the inversion of the pseudo-Hessian Matrix A is computationally expensive, we utilizing the Shermann Morrison formula [16] (eq. (6)) leads to a more efficient computation of the inverse, thus alleviating the computational cost of ONS.

$$(A + u * v^T)^{-1} = A^{-1} - \frac{A^{-1} * u * v^T * A^{-1}}{1 + v^T * A^{-1} * u} \tag{6}$$

We can thus store the matrix A_{t-1}^{-1} instead of A_{t-1} and compute the next A_t^{-1} via:

$$A_t^{-1} = (A_{t-1} + \nabla t * \nabla t^T)^{-1} = A_{t-1}^{-1} - \frac{A_{t-1}^{-1} * \nabla t * \nabla t^T * A_{t-1}^{-1}}{1 + \nabla t^T * A_{t-1}^{-1} * \nabla t} \tag{7}$$

As the Arima computation formulas are indifferent to the dimensionality of the input data, we make a multivariate forecast $\tilde{X}_t$ or the actual data point X_t:

$$\tilde{X}_t = \sum_{i=1}^{w} \gamma_i \nabla^d X_{t-i} + \sum_{i=0}^{d-1} \nabla^i X_{t-1}$$

The Autoregressive part of the model is a linear combination of vectors, weighed by our model coefficients gamma. The differenced vectors are given by $\nabla X_t = X_t - X_{t-1}$, where we perform standard vector subtraction. The consequent differences follow: $\nabla^i X_t = \nabla^{i-1} X_t - \nabla^{i-1} X_{t-1}$.

Upon receiving the true data point X_t, we compute the prediction loss for our forecast $\tilde{X}_t$. We define the incurred loss as the squared Euclidean distance between the two vectors. Let $l_t(X_t, \tilde{X}_t)$ denote our incurred loss in time step t:

$$l_t(X_t, \tilde{X}_t) = \sum_{i=j}^{n} (x_t^j - (\sum_{i=1}^{w} \gamma_i^t \nabla^d x_{t-i}^j + \sum_{i=0}^{d-1} \nabla^i x_{t-1}^j))^2 \tag{8}$$

In order to update the coefficients according to the ONS procedure we require the loss gradient. The loss gradient is consequently:

$$\nabla l_t[\gamma^t](X_t, \tilde{X}_t) = \begin{pmatrix} -2 \cdot \sum_{j=1}^{n} (x_t^j - \tilde{x}_t{}^j) \cdot \nabla^d x_{t-1}^j \\ -2 \cdot \sum_{j=1}^{n} (x_t^j - \tilde{x}_t{}^j) \cdot \nabla^d x_{t-2}^j \\ \vdots \\ -2 \cdot \sum_{j=1}^{n} (x_t^j - \tilde{x}_t{}^j) \cdot \nabla^d x_{t-w}^j \end{pmatrix}$$

We store the data vector $X_t = (x_t^1 \ x_t^2 \dots \ x_t^n)^T$. Figure 1 visualizes the window model. In each step the window slides forward, integrating the new data slice (dashed green) corresponding to data point X_t, as well as its differenced values, to the window head. Meanwhile, the oldest dated slice (dashed red) is dropped off the window tail. The outer slices (bottom slice, righthand slice) are just required for the forecast computation. For the sake of memory optimization, the upper left sub-block of data could be omitted.

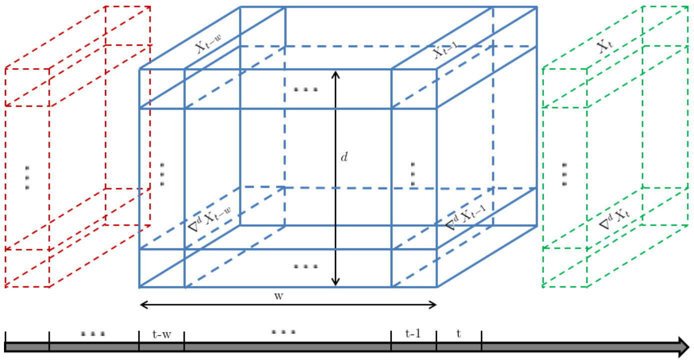

Fig. 1. Multi-dimensional window data model for Online Arima computation.

In order to differentiate anomalies, we calculate a reconstruction error based on the prediction and the measured value using the Euclidean distance. Large errors indicate a shift in the state, which is captured by the learned forecasting model. Thus, we compute a threshold, which indicates whether the error is large enough to be stated as anomaly. The threshold is based on the mean μ and standard deviation σ of the latest w errors. Thus, when the current error is larger than $\mu + s \cdot \sigma$, the data point is stated as anomaly, where s is a user defined sensitivity factor. Additionally, we provide the possibility for smoothing the results using a window of size v. Based on the latest v results, the most common entry will be recommended.

V. Evaluation Setup

For evaluating the applicability of different configurations of our approach, we built a testbed running several different components, which are available as open source solutions. The components are described next in detail.

A. Infrastructure and VNF Service

We use as cloud infrastructure a balanced and replicated Openstack[1] installation consisting of seven compute nodes and three controller nodes. The infrastructure consists of ten physical hosts possessing an Intel Xeon X3450 (4 cores, 2.66GHz), 16GB RAM, 3x 1TB disk, 2x 1GBit Ethernet connection. Through Openstack it is possible to create virtual machines in which we deploy our service use case. The virtual machines (Ubuntu 14.04) are defined to use 2 vCPU cores, 2GB memory, 20GB disk.

We deployed the open source implementation of an IP multimedia subsystem (IMS), called Clearwater[2]. Clearwater is considered as an examples for virtual network functions, which makes it highly interesting to telco-providers. IMS is an emerging architecture for IP-based telecommunication services, such as voice-, video calls or messaging. The core implementation of Clearwater allows users to register, which can then initiate calls within the system. There exists the services named Bono, Sprout, Homestead and Ellis, which are considered within the experiments.

Bono functions as edge proxy, handling clients connections. Clients register via the SIP protocol in order to initiate calls. The requests are then routed to the Sprout service. *Sprout* manages the different communications to the other internal services, e.g. requesting for authentication. *Homestead* contains the client profile information, which are needed to authenticate clients. *Ellis* obtains information for the management unit, such it functions as an account management system.

In order to simulate the load of the IMS, we change every minute the number of client registrations and call initiations randomly. Furthermore, we deployed a replicated version, which is load balanced. Thus, the key services Bono and Sprout are deployed three times each, while single deployments are used for Homestead and Ellis. For each service installation, we created its own virtual machine. For monitoring, we collect inside each virtual machine the resource usage data parsing the proc filesystem, provided by the operating system. The data collection interval is 500ms.

B. Anomaly Injection

In order to evaluate the proposed approaches, we simulate a larger set of anomalous behaviour in the system. Thus, we developed an injection agent handling several different anomalies. The agent is placed inside each virtual machine and can execute the following anomalies:

Disk pollution, temporary disk pollution and HDD stress: Both anomalies are writing data into a log-file. For the temporary case, the files are deleted, while in the other case the file is still available. HDD stress describes an excessive hard disk usage by writing a file.

CPU stress, leak and fluctuation: CPU stress immediately consumes excessively much CPU. CPU leakage describes an anomaly increasing in its intensity over time by consuming more and more CPU. The fluctuation of CPU constantly increases and decreases the CPU usage in order to provide a fluctuation behaviour.

Memory stress, leak and fluctuation: Like the CPU, memory anomalies are also modelled with the same behaviours: immediately consuming high amounts of memory, growing consumption over time and fluctuating allocation of memory.

Fork flodding leak and fluctuation: A process starts to create child processes over time. For the leakage variant, those child processes are again creating more processes, while in the fluctuation variant children are also removed partly.

Large file download: A process is started downloading a large file, resulting in a high network traffic.

File pointer wasting: By requesting file pointers without releasing them, creates a leakage of the pointer IDs over time.

Through using several simulated anomalies with different behaviours, we like to show the robustness of our approaches quality. Notably, the intensity increase of each anomaly can be configured and is randomly selected for the current evaluation.

C. Analysis Pipeline Configuration

As described above, we collect metrics from the proc filesystem, which consists of a large number of values. Before performing the anomaly detection on the data, we do the following preprocessing steps:

Feature selection: We selected these metrics for analysis: CPU percentage, disk-io, disk-io time, load from the previous 1s and 5s, memory usage and percentage, network-io bytes, packets, number of errors, number of dropped packets.

Feature creation: In order to gain more knowledge from the given metrics, we also created new metrics based on the collected ones. The slope of the last 60s of memory providing information about the change of memory over time. Furthermore, we added metrics representing information about the packet size and relations between CPU usage, network traffic and disk usage.

Normalization: Min-max scaling is performed for each metric, resulting in a scaled metric $\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}}$ for an incoming data point x, where x_{min} and x_{max} represent the minimum and maximum values for the individual metrics. These can be inferred over time or set by an expert.

Based on the evaluation setup, we collected data of all service for 72h. The first 20 minutes of data consists of normal data grace time. Afterwards, the anomalies were injected in a round-robin manner one by one to the individual service hosts for 3 minutes. Between the anomalies, a cool down of 1 minute is performed. For evaluation purposes, we removed time frames, were anomalies run on other hosts as we like to know how good the anomaly detection performs without including propagating faults.

[1]https://www.openstack.org/

[2]http://www.projectclearwater.org/

Parameters: Next we show the chosen hyper-parameters on which we perform a grid search for the best configuration.

Window size: $w \in 5, 10, 15, 20, 50$. We refrain from evaluating sizes larger than 50 as we like to learn and predict in real-time expect to performance.

Differencing depth: $d \in 1, 2, 4, 6, 8$. We limit our differencing depth to a ceiling of 8 as we do not expect increased differencing to be beneficial.

Smoothing values: $v \in 2, 4, 6, 8$. We expect for larger smoothing ranges poor detection rates, which can be explained by the fact that Arima models adapt their data view quickly thus making it unlikely that many successive anomaly predictions (which are induced by large deviations between forecast and true observation) occur.

Threshold sensitivity: $s = 1.0$. A fine grained calibration of s could potentially optimize joined performance over detection and normal rate. However, this entails considerable tuning efforts which we leave to further research as our main focus is on the influence of the Arima model parameters (w, d).

VI. Evaluation Results

For evaluation, we measure the percentage of detected anomaly events for the different anomalies with the additional information how many false alarms in total per data point where given by the individual setups. Furthermore, all setups are accompanied by the average detection time (ms) of each anomaly type, after what time period the event was in average detected. As well as the average false alarm duration (multiplied by factor 100 to match the scale, effectively in 0.01 ms), which provides information about the duration of false alarms.

We show a selection of three setups, which are most interesting, each using a fixed smoothing and differencing depth while we iterate across all considered window sizes. In the initial two examples, the smoothing value $v = 4$ was chosen as we observe the progression in behaviour using the differencing depths $d = 4$ and $d = 8$. Next, an example utilizing the smoothing range $v = 6$ with $d = 8$ is outlined.

Figure 2 shows the detection accuracy for $v = 4$ and $d = 4$. The detection accuracy across all window sizes is pretty high, yet there are still individual anomaly types that are missed (e.g. Fork flooding leakage). Also, one can see, that with the increasing window size, the detection accuracy increases. The same behaviour with the decrease of detection times for larger window sizes holds, as seen in Figure 3.

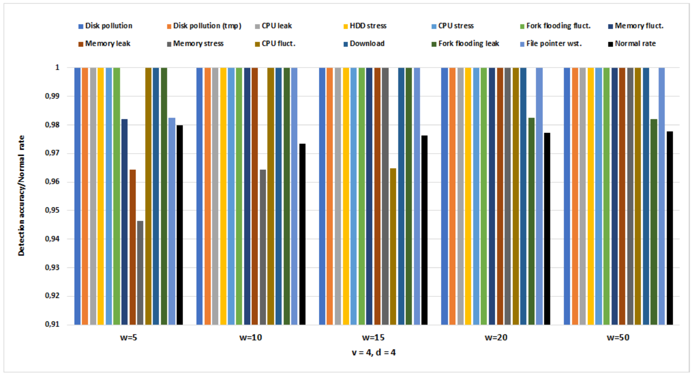

Fig. 2. Detection accuracy/Normal rate, $v = 4$, $d = 4$

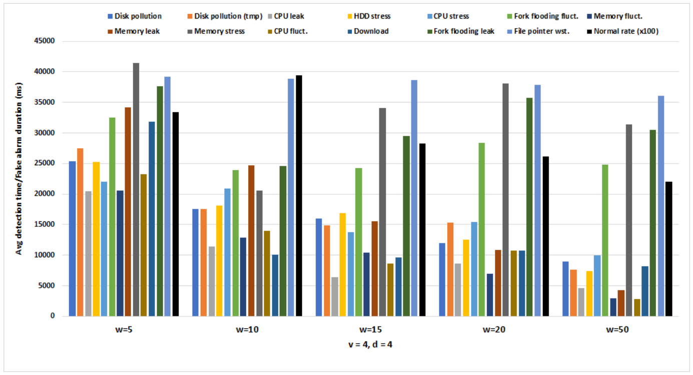

Fig. 3. Detection time/False alarm duration, $v = 4$, $d = 4$

In Fig. 4 and Fig. 5 we depict the final iteration of our progression of differencing orders, showing accuracy and detection time for parameters $v = 4$ and $d = 8$. Window sizes $w >= 20$ now recognize 100% of injected anomaly events, whereas the smaller window sizes perform marginally worse for certain anomaly types. While $w = 5$ has the lowest average detection accuracy (still $> 99.5\%$) it performs arguably the best as the normal rate is considerably higher compared to its competitors. Overall, the detection times have diminished noticeably all across the board as well.

In Figure 6, we show the impact of increased smoothing ($v = 6$). Even though, we employ the most sensitive differencing ($d = 8$), our detection rates drop substantially (note scale difference, as low as 60% for $w = 5$ and 90% for best performing window $w = 50$). Accordingly, the detection times increase (Fig. 7) while normal rates soar, due to the reduced sensitivity. Smoothing ranges too small in size ($v = 2$) produced high detection accuracy at the cost of high false alarm count, which is not appreciated.

All in all, we therefore recommend using a setup with configuration $v = 4$, $d = 8$ and $w >= 20$ for anomaly detection. This configuration showed with respect to all tested combinations the highest detection rates (100%), a low number of false

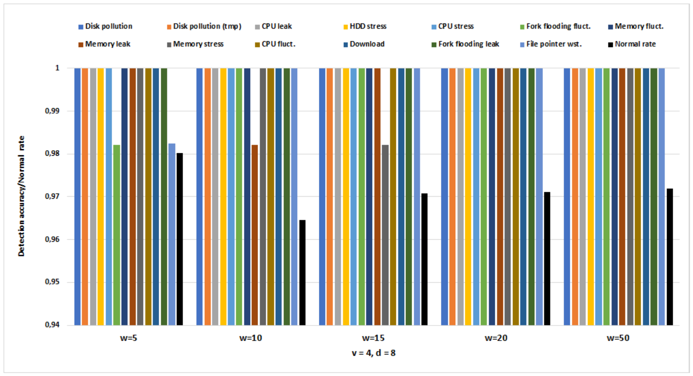

Fig. 4. Detection accuracy/Normal rate, $v = 4$, $d = 8$

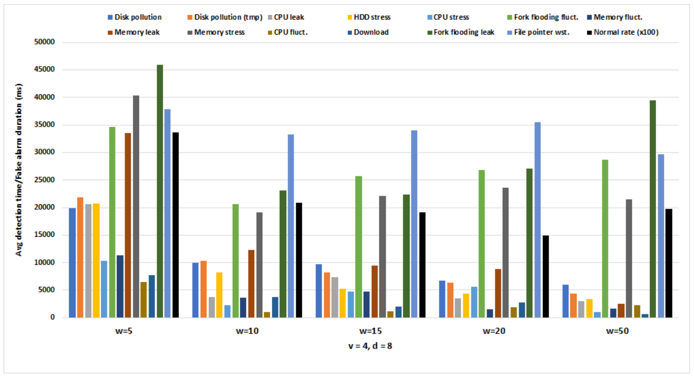

Fig. 5. Detection time/False alarm duration, $v = 4$, $d = 8$

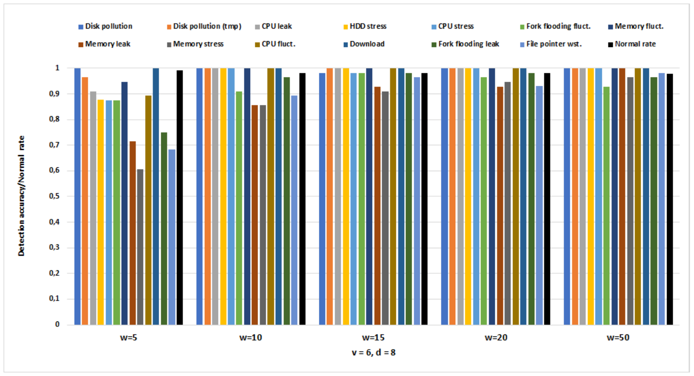

Fig. 6. Detection accuracy/Normal rate, $v = 6$, $d = 8$

alarms (smaller than 3%) and with respect to the average time in order to detect an abnormal event (<40s).

VII. Conclusion

This work presented an unsupervised anomaly event detection approach for service monitoring. We showed how the Online Arima forecasting algorithm can be used for anomaly detection. This approach was evaluated using a virtual network function use case applying several different anomalies into the service infrastructure. The results showed, that the approach achieves high rate to detect anomalies while giving a small number of false alarms, showing the applicability of this approach for anomaly detection for service monitoring.

In future, we like to evaluate this approach on further VNF services and extend the evaluation on the private cloud provider hosts in order to monitor the physical machines. Furthermore, we like to investigate in more detail seasonal behaviours of load usages as they appear often in real-world scenarios. In future, we hope that such automatic methods should provide the possibility for zero touch administration.

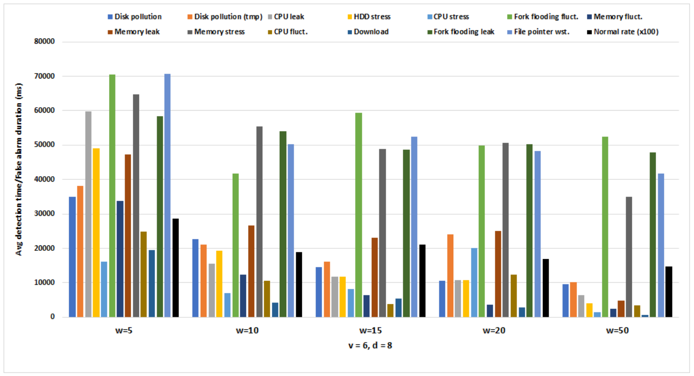

Fig. 7. Detection time/False alarm duration, $v = 6$, $d = 8$

References

[1] E. Bauer, X. Zhang, and D. A. Kimber, *Practical system reliability*. John Wiley & Sons, 2009.

[2] M. H. Ghahramani, M. Zhou, and C. T. Hon, "Toward cloud computing qos architecture: Analysis of cloud systems and cloud services," *IEEE/CAA Journal of Automatica Sinica*, vol. 4, no. 1, pp. 6–18, 2017.

[3] A. Gulenko, M. Wallschläger, F. Schmidt, O. Kao, and F. Liu, "A system architecture for real-time anomaly detection in large-scale nfv systems," *Procedia Computer Science*, vol. 94, pp. 491–496, 2016.

[4] C. Liu, S. C. H. Hoi, P. Zhao, and J. Sun, "Online arima algorithms for time series prediction," in *Proceedings of AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, pp. 1867–1873.

[5] C. Sauvanaud, K. Lazri, M. Kaâniche, and K. Kanoun, "Anomaly detection and root cause localization in virtual network functions," in *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on*. IEEE, 2016, pp. 196–206.

[6] J. Liu, S. Chen, Z. Zhou, and T. Wu, "An anomaly detection algorithm of cloud platform based on self-organizing maps," *Mathematical Problems in Engineering*, vol. 2016, 2016.

[7] D. J. Dean, H. Nguyen, and X. Gu, "Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems," in *Proceedings of the 9th international conference on Autonomic computing*. ACM, 2012, pp. 191–200.

[8] A. Bhattacharyya, S. A. J. Jandaghi, S. Sotiriadis, and C. Amza, "Semantic aware online detection of resource anomalies on the cloud," in *Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on*. IEEE, 2016, pp. 134–143.

[9] D. Cotroneo, R. Natella, and S. Rosiello, "A fault correlation approach to detect performance anomalies in virtual network function chains," in *Software Reliability Engineering (ISSRE), 2017 IEEE 28th International Symposium on*. IEEE, 2017, pp. 90–100.

[10] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[11] J. D. Hamilton, *Time series analysis*. Princeton University Press, 1994, vol. 2.

[12] E. J. Hannan, *Multiple time series*. John Wiley & Sons, 2009, vol. 38.

[13] S. Bubeck, "Introduction to online optimization," *Lecture Notes*, pp. 1–86, 2011.

[14] E. Hazan, A. Agarwal, and S. Kale, "Logarithmic regret algorithms for online convex optimization," *Machine Learning*, vol. 69, no. 2-3, pp. 169–192, 2007.

[15] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proceedings of International Conference on Machine Learning (ICML-03)*, 2003, pp. 928–936.

[16] J. Sherman and W. J. Morrison, "Adjustment of an inverse matrix corresponding to a change in one element of a given matrix," *The Annals of Mathematical Statistics*, vol. 21, no. 1, pp. 124–127, 1950.