# 0 Abstract

In this PhD work, we aim to extend a horizon of knowledge in the context of adaptive approaches in run-time prediction of issues and localisation of their root causes in complex, cloud based, evolving software systems with non-stationary behaviour and dynamic operating context, decentralised and / or distributed architecture, high dimensional metric set, and unknown fault types.

Complex systems often behave in unexpected ways that are not easily predictable from the behaviour of their components. This is known as an emergent behaviour [9]. The nature of complex software systems makes failures unavoidable and the intrinsic characteristics of cloud systems amplify the problem and reduce the effectiveness of classic testing approaches (conventional white / grey box testing, pure statistical models, testing using models pre-trained with seeded faults and / or labelled data, etc). At the same time, cost of missing an anomalous, unexpected, behavior is often very high.

According to our ﬁndings while surveying the literature, the most promising approaches for failure prediction and fault localisation exploit statistical analysis and machine learning to reveal anomalies and their correlation with possible failures. However, both approaches have their specific limitations, hence, majority of solutions exploits different combinations of techniques and strategies to make the prediction and localization more adaptive, robust and efficient.

During the ﬁrst year we implemented the anomaly detection and fault localisation solution based on machine learning and statistical techniques. We used Deep Autoencoders and One Class SVM for failure prediction; graph theory and granger causality concept for tracing the propagation of the fault; and PageRank network centrality algorithm to rank the suspicious system resources. The scope of work done includes (1) setting up of the static virtualised environment; (2) conﬁguration and deployment of the testbed based on a widely used in industry, open-source decentralised distributed solution; (3) simulation of the normal behaviour of the system during a long period to gather data for the baseline models training; (4) execution of the fault injection experiments; (5) data collection and pre-processing; (6) evaluation of the prototype’s models and selection of an optimal hyper-parameter combination. The experimental analysis showed that our approach is effective (high rate of accuracy, low false alarm rate, early enough prediction, very accurate fault localisation) and efﬁcient (negligible level of overhead) in most of the cases (fault injection types) in static virtualised environment. We also evaluated EmBeD (RBM based failure prediction approach [43]) on the data collected and compared the results with the results of experimental analysis of our Deep Autoencoder based solution. The comparative analysis did not reveal significant differences in an efficiency of these approaches.

During the next years we plan to work on improvement of our approach to overcome the observed limitations; evaluation and adapting of the approaches to dynamic cloud container-based environment; generalization of the approaches to make them more domain and fault type agnostic; extension of the ﬁeld of application of the approach to conceptually different kind of fault types.

# 1 Introduction

Unexpected behaviour has a signiﬁcant impact on smooth operation of systems and the costly penalties due to the failures. Predicting issues before their occurrence by detecting and analyzing anomalies (deviations from normal or expected behavior) in system metrics is a viable solution to enable failure preventing or mitigating actions.

Many of modern industrial-strength software solutions are complex: it has a large number of parts that have many interactions [44]. They can be composed of heterogeneous software components distributed across separate and often geographically dispersed physical servers. Its components can be deployed on virtual machines or containers that can be migrated from one physical node to another within and across data centres. Applications can be distributed and / or multi-tier systems deployed on dedicated or shared server environments, doing computations across distributed nodes. Performance anomaly detection and fault localization in such complex systems is a difficult task since the issues may have many sources, may be hidden in any one or more of the components, may emerge from the interactions among them or system-environment interaction. Moreover, interdependencies between system resources enables faults to propagate throughout the system in a cascade manner.

Thus, the **major challenges** in failure prediction in complex systems, **that impact the accuracy of the approaches** are: (1) **dynamic dependency** - multiple interdependent application components deployed in data centre servers with heterogeneous and interdependent resources; (2) **dynamic anomaly characteristics** - deﬁning a priori all possible normal or anomalous behaviour types is not always feasible. The notion of normality, anomalies and their characteristics vary widely across applications, execution environments, and load contexts; (3) **nature of data** - different formats and semantics causes presence of noise when the normal values may be similar to anomalies resulting in high false-positive detections; (4) **high-dimensional metric space** - in high dimensional space, the data are sparse and concepts using the notion of proximity fail to retain their effectiveness (also known as a ‘curse of dimensionality’ problem) [27].

On top of that, some of inherent characteristics of cloud environment, such as multi-tenancy, resource sharing, dynamic resource allocation, even more aggravate the problem of **dynamic dependency** of complex systems. Frequent performance variations exhibited by cloud-based applications can often be caused by the (5) **fluctuations in the execution context** of the underlying environment. Hence, failure prediction approaches for cloud-based complex systems should consider not only the current state of the system resources and dynamics of the target application, but also must take into account the elasticity of the underlying system.

There are different strategies, techniques and solutions exist in failure prediction and fault identification domain. The most promising approaches exploit statistical analysis or machine learning methods. Majority of modern solutions use profile-based learning strategy, where the key idea is to observe the system behaviour over time to make sense of underlying system dynamics and build a baseline model (can be probability density function, neural network with adjusted wights, clustered representation of dominant behavior patterns of the system, etc.) and then define the deviation of new observations from the baseline. This deviation usually compared to certain threshold to reveal anomalous states. The baseline models can be built/trained statically in ofﬂine phase using data collected while normal execution (also sometimes ML models need extensive training with seeded faults) or dynamically, in online phase basing on observed historical data.

The main **problems related to the implementation** of the failure prediction solutions are: (1) **accuracy** of anomaly detection (often conditioned by the high false positive rate of prediction); (2) need for **extensive training** which is often unacceptable in runtime context adaptive solutions (no time to retrain the model); (3) need for training with **labelled data** (which requires signiﬁcant human efforts and can only handle previously known anomalies), (4) high **runtime overhead** (which affects the normal execution of the systems under observation), horizontal scaling issues (handle amounts of KPIs); (5) application and/or fault type dependency (the problem of **generalisation** of the approach).

The problem with accuracy in implementation essentially caused by relying of many statistical and ML profile-based solutions on the premise that:

(1) **Statistical distribution** of metric measurements is either **known or static. This is no**t a case in typical modern cloud-based scenarios considering the **dynamic dependency** and dynamic execution context of the underlying system. Static anomaly detection baseline models build using profile-based learning strategy in offline mode exploiting a priory available data set (for example, a data collected during normal, non-faulty, execution of the target system) becomes soon obsolete and not valid for explanation of the changed behavior of the underlaying system (in many cases, the characteristics of the target application also are being changed, adapting to the changes in the operating context as a result of **dynamic dependency in complex systems**).

(2) **Monitoring data is available** a priori for model training. Considering dynamic nature of behavior of evolving complex systems and unavoidable fluctuations of an operating context (caused by the elasticity of cloud environment), a priory availability of a data set, that would be valid for training of the baseline model is not possible in majority of use cases.

(3) What constitutes **normal and abnormal behaviour is well delineated**. Many solutions rely on thresholding with normality assumption which is not always relevant in case of complex systems with its inherit **dynamic anomaly characteristics**. Additionally, existence of noise in data makes false alarms even more frequent.

These problems limits applicability of failure prediction approaches in real-time scenarios and there are many solutions existed that address these challenges in virtualized cloud environments often by use of combinations of different techniques. One of such works is [1], where the presented solution for fault detection in virtualised clouds does not rely on the premises of static or known statistical distribution of metric values, and availability of the data for training. The proposed solution adopts an online model generation approach which makes the solution more robust to the dynamism of the operating context and changing workload, and also makes this approach more generalisable (applicable to different problem domains). While online monitoring of each metric series in isolation the solution captures the deviations of a metric values from the behaviour learned by the model (kNN algorithm is used) on historical data. Once the anomaly is detected in one of the metric time series, the correlation of this metric with other metrics within the VM (operating context) and with the same metrics among peer VMs is analysed. Based on the knowledge of typical correlation values under normal behaviour, any signiﬁcant deviations from normal correlation values are noted. If the deviations are larger than an empirically determined threshold, a fault event is generated and forwarded for problem diagnosis. The model of the normal behaviour is continuously generated online (using the historical data portions free of anomalies) to cope with dynamic changes in the workload and operating environment. Correlation analysis is being performed on only events generated in the Event Generation Engine stage, and only for the VM(s) and resource(s) tagged in this phase. This improves the scalability signiﬁcantly.

In our work we aim to cover a specter of the aspects related to the problems caused by (1) evolving nature of modern complex systems; (2) unavailability of data a priory in unknown or partially known dynamic environments for building of baseline models; (3) dynamism of operating context in clouds, focusing on container-based environments.

During the ﬁrst year we implemented an automated mechanism for node/resource -level failure prediction and fault localization in large-scale virtualized systems. A set of techniques is presented for monitoring, data retrieval, data transformation (to construct a uniform data format for data analysis), unsupervised learning (Deep Autoencoders and One Class SVM ML approaches used to detect deviations in acting of the nodes/resources), localization of faults (graph theory, Granger causality concept, centrality index algorithms were used for the root cause analysis of the propagation of faulty behavior). We evaluate our prototype implementation by injecting a variety of faults into a production system. The results show that our mechanism can eﬀectively identify faulty nodes with high accuracy and no computation overhead. Evaluation of EmBeD (RBM based failure prediction approach [43]) solution on the same data and comparing the results with the results of our experimental analysis did not reveal significant differences in an efficiency of these approaches.

On the next steps we plan to work on improvement of our approach to overcome the observed limitations (more accurate modeling the dynamicity of the application workload, working on the mechanism of adaptive normality deviation threshold adjustment, working on runtime modeling of dynamically changing system dependencies (causality graph), etc.); evaluation and adapting of the approaches to dynamic container-based cloud environment; generalization of the approaches to make them more domain and fault type agnostic and extension of the ﬁeld of application of the approach to conceptually different kind of fault types (related to a system healthiness concept).

# 2 State-of-the-Art

The general logic of failure prediction solutions consists in: (1) continuous observation of the system measurements in order to automaticaly detect unexpected behaviour before failure; (2) identification of the root cause of an observed anomaly; (3) identification of the resources or components responsible for an observed violation. The key expectation, is to balance high detection rates with low false-alarm rates while adapting rapidly to changes in service performance.

Failure prediction solutions typically work with data presented by time series (a sequence of values represented in order of the time the events occur) of performance metrics values, which systematically sampled over a regular interval and deﬁne a state of the system. Three major aspects of data collection are the (1) System Observability (impacts the method of detection – black / grey / white -box approaches); (2) Sampling interval (impacts the level of information resolution, compute and storage overheads); (3) The method of collection – proﬁling (studying the resource consumption behaviour in black-box manner) or tracing (white- or grey-box methods of tracking ﬁne-grained source-level events via source code instrumentation; may reveal the execution ﬂows, caller-threads, and time spent in speciﬁc code blocks).

There are different strategies and methods of anomaly detection and fault localization. The choice of strategy is greatly inﬂuenced by system observability, detection mode (real-time or post-mortem), availability of labeled data. In most cases, thresholding is used to limit the range of metrics values beyond which an event is raised.

**1. Profile-based and signature-based detection** strategy implies generation of the baseline profiles (density distributions, learning models) of prevailing system behaviour and using them to ﬁlter new observations for unexpected behaviour. Proﬁle-based approaches typically create a representation of normal behaviour, and anomalies are detected from deviations with respect to this representation. Proﬁle-based techniques often have higher false alarm rates, however they are more ﬂexible due to ability to detect unknown anomalies. In contrast, signature-based approaches use a prior knowledge about the characteristics of each kind of anomaly to identify potential previously known incidents. The limitation of such approaches is that they can detect only known fault types.

Some methods of prediction of an unexpected behaviour imply (1) defining of the probability of the new observation to belong to the density distribution of the data corresponding to the normal behaviour and comparing this probability with certain threshold; others imply (2) making prediction using the model trained on data corresponding to the normal behaviour and comparing the deviation between the predicted and observed values with certain threshold.

In [4] a time-series of each observed metrics in a particular epoch (time interval) used to compute the medians of the measured values. Then, basing on the past values of each median, the current values of the medians of each metric characterized as high, low, or normal and these categories combined into the Epoch Summary Vector. Then, using ML feature selection technique, the relevant metrics are selected for building the Epoch Fingerprint (subspace of the Epoch Summary Vector). Since most anomalies span multiple epochs, the consecutive Epoch Fingerprints combined into Anomaly Fingerprint by averaging the corresponding epoch ﬁngerprints, thus summarising them across time. After that, the anomaly ﬁngerprint is compared to already labelled by the administrator anomaly ﬁngerprints to identify the underlying problem. The identicality of anomaly ﬁngerprints are defined by thresholding of the Euclidean distance between them.

In [39] a proﬁle-based approach for network anomaly detection is presented. A modiﬁcation of Ant Colony Optimization metaheuristic (based on the principles of swarm intelligence, inspired by the foraging behavior of bio ant colony with its ability to ﬁnd the shortest path between the nest and food source) [40], [41] is used as a clustering approach (to create a representation of normal behavior) which seeks an optimal solutions to grouping data through self-organized agents. Ants, using statistics and probabilities, travel through the search space represented by a graph. These agents are attracted to more favorable locations to optimize an objective function, in other words, those in which the concentration of pheromone deposited by ants which previously went through the same path is higher. In this paper, we assume that the paths are formed between the center of a cluster (centroid). The algorithm aims to optimize the efﬁciency of clustering, minimizing the objective function value (ﬁnding the. shortest baths to the clusters’ centroids), ensuring that each ﬂow will be grouped to the best cluster (with known number of clusters). Steps on each iteration: Build solutions (movement of ants by the states of the problem); Local Search (evaluation of solutions created through a local search); Pheromone Update.

**2. Observational Detection strategy** implies that applications are observed through direct experimentation (in contrast to exploiting baseline profiles), followed by in-depth analysis of observed anomalies and root-cause identiﬁcation. Because this approach depends mostly on experiential knowledge, it yields high accuracy in detecting known and unknown anomalies.

In [5] the run-time fault root-cause solution is presented. It makes use of an online proﬁling and triggered when some of the transactions start presenting symptoms of performance anomaly. The ﬁrst step is to detect if a performance variation is due to a workload change or it is anomaly. This detection is done by measuring the correlation (Pearson correlation coefﬁcient [6] is used) between the response time and the number of user transactions processed. Considering that the workload and response time might not be fully aligned (synchronized), an additional analysis based on the Dynamic Time Warping algorithm [7] is performed (Measures similarity between two sequences which may vary in time or speed. Keeps track of the distance necessary to keep them aligned). Both a sudden decrease of correlation and an increase in the distance to keep the workload and response time aligned is interpreted as a performance anomaly. Then, the solution checks whether the anomaly is caused by the application or the server change. This time the correlations between the aggregated workload and the application and server metrics are measured. The third step is to determine if the extra response time of a given transaction is caused by the application components or some remote service changes. The signiﬁcance of the relation between the different metrics (estimators) and the total response time is estimated by use of ANOVA (calculates the F statistics and then the p-value or difference between means of response times obtained by use of different estimators) and the effect-size of estimators on the response time is measured by the coefﬁcient of determination (R2).

**3. Knowledge-based detection strategy** implies identification of performance issues and their causes based on historical records of previously observed anomalies – a dynamic store (knowledge base) where deﬁnitions of known anomalies, their possible root-causes are maintained. The detection of new issues often triggers an update of the knowledge base. Although there exists some similarity between knowledge-based and signature-based detections, generation of rules and deﬁnitions does not necessarily have to be entirely at run-time in the former.

In [8] the framework for automated performance bottleneck detection is presented. The rules for detection of different bottlenecks are stored (added/modified) in the framework’s DB by its user. A bottleneck in DB is deﬁned as a condition, called rule, on a set of metrics. Each bottleneck is classiﬁed into one (or more) of the ﬁve dimensions (CPU, Memory, I/O, Communication, Threads). The BDE receives the metrics collected during observation, evaluates the rules and composes a bottleneck description for all bottlenecks whose rules evaluate to be true (name, the area of code where it was evaluated).

**4. Flow and Dependency Analysis** considers studying the ﬂow of communication across components in distributed applications to identify anomalies. This approach typically involves real-time collection and analysis of trafﬁc data (such as SNMP and TCP packets). To detect anomalies and their causes, frequency, correlation, and causal path analysis are usually performed.

In [10], the presented debugging tool (Spectroscope) aims to identify performance problems in distributed systems and analyze their root causes. It searches for the causes of changes in request execution performance by comparing execution ﬂows of those requests. It categorizes the flows to (1) Precursors (request ﬂows corresponding to the executions of non-problem period) and Mutations (request ﬂows of the executions of problem periods). Identifying mutations and comparing them to their precursors helps localise sources of change and gives insight into their effects. Mutations are divided into two types: response time mutations (differ from the corresponding precursor only by response time), and structural mutations (requests that take different paths through the system in the problem period). Spectroscope groups identically structured requests into unique categories and uses them as the basic unit for comparing request ﬂows. To identify response-time mutations, for each category distributions of response times for the non-problem period and the problem period are extracted and input into the Kolmogorov-Smirnov two sample non-parametric hypothesis test [11]. The category is marked as containing response-time mutations if the test rejects the null hypothesis. To identify the components or interactions responsible for the mutation, Spectroscope extracts the critical path (the path of the request on which response time depends), which is the same for both precursors and mutations in case of response time mutations, and runs the same hypothesis test on the edge latency distributions. Edges for which the null hypothesis is rejected are marked as fault locations. To identify structural mutations, Spectroscope assumes a similar workload was run in both the non-problem period and the problem period. As such, it is reasonable to expect that an increase in the number of requests that take one path in the problem period should correspond to a decrease in the number of requests that take other paths (i.e. the increase in one category should be balanced by the decrease in another category). Thus, if that or another category contains more requests from the problem period than requests from the non-problem one (difference compared with pre-defined threshold), then this category is labelled as one which contains mutations. Otherwise, this category is labelled as one which contains precursors.

**Statistical methods:**

Statistical methods can be categorized on parametric and non-parametric based on the assumptions they make regarding the underlying data. Parametric statistical techniques (ANOVA tests, Pearson correlation, t-tests) assume that some characteristics of the data are known a priori or can be inferred. For example, assuming that the probability density of a performance metric follows a Gaussian distribution. Nonparametric methods (CUSUM, Spearman correlation, Kruskal-Wallis, and Wilcoxon’s test) require little or no assumptions about the underlying nature of the data.

1. Gaussian-based detection (exploit the assumption that underlying data distribution is normal): detects anomalous data points based on the distance from the distribution mean. The density distribution may also be exploited for detecting anomalous data points in GMM -parametric models of the probability distribution of continuous random variables estimated using the Expectation-Maximization (EM) algorithm [13, 14].

2. Regression analysis investigates relationships between performance metrics and quantiﬁes the statistical signiﬁcance of such relationships. The goal is to estimate the set of model parameters that minimises the absolute or the squared error. The magnitude of the residuals used to determine an anomaly score of new observations. New observations with deviation falling outside the conﬁdence interval produced by the model may be classiﬁed as anomalous. Commonly used algorithms for estimating model parameters include Ordinary Least Squares (OLS), Least Angle (LA) and Recursive Least Square (RLS) [15].

In [38], the performance diagnostic framework (DAPA) is presented, which aims to help system administrators to analyse the application performance anomalies and identify potential causes of SLA violations in virtualised environments. The systems metrics considered as estimation variables and the observed application response time as response variable. DAPA constructs a series of models using non-overlapping time windows that span across time between 2 hours before potential SLA violation and 1 hour after real SLA violation. The constructed models may exhibit different characteristics, however, those from the potential SLA violation phase may demonstrate characteristics of both SLA compliance and SLA violation. These models are then classiﬁed into different categories representing distinct system states (by use of k-means clustering with 2 centroids). Each model is represented by the vector of its regression coefﬁcients, Euclidean distance is used as the distance measure. The ﬁnal stage involves aggregation of all the sample data belonging to the SLA violation cluster and creating a parsimonious regression model (least angle regression algorithm – LARS - used) selecting the estimators with highest explanatory predictive power. The top selected metrics from this model are then presented to the system administrator as suspicious system attributes.

3. Correlation Analysis: quantiﬁes the degree of association between performance metrics. Commonly used techniques to estimate the correlation are the Pearson, the Kendal rank and the Spearman algorithms [16].

The framework presented in [16] is used to distinguish the natural cause of unexpected behavior (workload variations, upgrades) from internal anomaly that may end up in a failure. In its implementation the Pearson’s r is used (1) by the performance analysis module to describe the degree of association between the response time of a given transaction and number of processed transactions of the same type. A lower degree is a symptom of performance anomaly, and (2) by the anomaly detection module to describe the degree of association between the amount of concurrent users per period and the collected parameters. A lower degree explains which parameter is mostly correlated with a detected performance anomaly.

4. Statistical process control (SPC) [17], is a quality control method widely used to monitor production processes for early detection of undesirable variation in process output. SPC provides a set of control charts, such as CUSUM, Shewhart charts, for monitoring process stability and variation.

In [18] two unsupervised incremental adapting (evolving with the data using limited parameter settings or ofﬂine retraining) learning techniques are proposed to address the problem of black-box real-time contextual anomalies detection in performance metric streams in Internet services domain. This work addresses the issue of speciﬁcity of the failure prediction solutions due to their (1) rigid statistical assumptions and (2) targeting ofﬂine scenarios of model training using priorly collected data. Such approaches may result in high false-alarm rates when applied in online scenarios as the models become obsolete.These issues raise the need for the techniques that can adapt to the evolving metric stream behaviour. Both proposed techniques (BAD - Behaviourbased Anomaly Detection, and PAD - Prediction-based Anomaly Detection) follow two-step strategy - ﬁrst, the underlying temporal property of the most recent behaviour of the stream is estimated, second, the statistically robust control charts applied to recognise deviation of new observation from the baseline.  
BAD scenario: (1) tracking the underlying statistical behaviour of the most recent period and generation of the PDF using KDE; (2) estimation of the probability density of current observation within this PDF; (3) construction of Shewhart Control chart (from the densities of observations) using the mean and the standard deviation of the PDF; (4) deriving of an adaptive threshold of probability density based on the lower side of the control chart; (5) comparing of the probability density of current observation with the threshold obtained.  
PAD scenario: (1) building of an adaptive Cubic Spline model (computationally fast and capable to pick up complex non-linear shapes) from observation of the most recent period. Coefﬁcients of a spline model estimated via ordinary least squares (OLS) algorithm. (2) making a prediction for the current step using the built cubic spline (3) estimation of the prediction deviation (4) building an Exponential Weighted Moving Average control chart from the residuals of the most recent period; (5) alarm an anomaly in case the prediction deviation is out of the control limits of the EWMA.

Unlike statistical detection, learning techniques do not make assumptions about the underlying distribution of data.

Commonly used learning techniques in literature: (1) Classiﬁcation-based Techniques (SVM, ANN, Decision trees, Bayesian networks); (2) Neighbour-based Techniques (KNN [24]; KNNW [25]; LOF [26]); (3) Clustering-based Techniques (K-means, Expectation Maximization (EM), SOM [34]); (4) Subspace-Based Detection (PCA, ICA, Autoencoders); (5) Ensemble-Based Detection (integration of results of various detection techniques and / or data subsets to achieve a consensus).

**ML-based methods**

(1) Supervised learning - each data instance in a training dataset is assumed to belong to one of several classes. The goal is to build a generalised model that captures the relationship between the feature set and each class during the training phase. Well suited for recognising known anomalies. Usage in dynamic environments is hampered by the cost of retraining due to dynamic reconﬁguration of application components and change in underlying execution environments.

In [29] LSTM RNN [30] is used for real-time detection of collective anomalies in network intrusion analysis domain. LSTM model is trained with normal time-series data, adjusting its weights to obtain the ability to remember the context of the points in the training time-series in order to predict a coherent output in agreement with the context of the test sample. In run time model also must decide whether a set of inputs within a number of the latest time steps forms a collective anomaly. The prediction deviations are measured within a certain period and collected in circular array used to represent the level of anomaly of the latest time steps (to consider the ensemble of points simultaneously). By analyzing the circular array at every time step, the possibility of facing a collective anomaly is evaluated (based on predeﬁned during the evaluation step threshold).

In [36] LSTM is used in a framework (DeepLog) for online anomaly detection from a log key sequence. DeepLog learns log patterns from normal execution during the learning phase (small training data set that consists of a sequence of “normal log entries”). After the training it can recognise normal log sequences and can be used for online anomaly detection over incoming log entries in a streaming fashion. The key intuition behind the design of DeepLog is from natural language processing: log entries interpreted as an elements of a sequence that follows certain patterns and grammar rules. DeepLog also provides a mechanism for a user to provide a feedback to use a false positive to adjust its weights. DeepLog incrementally updates the model’s weights to incorporate and adapt to new log patterns.

(2) Unsupervised learning - requires no labelled training data (in some cases no training data at all). In unsupervised learning, in contrast to supervised one, no speciﬁc output value is provided. Instead, one tries to infer some underlying structure from the inputs. For instance, in unsupervised clustering, the goal is to infer a mapping from the given inputs (e.g. vectors of real numbers) to groups such that similar inputs are mapped to the same group [21]. The objective is to discover hidden patterns or regularities in the data. The algorithms cluster the input data into classes based solely on their statistical properties, no assumption is made of the distribution of the underlying data. For improved accuracy, it expected that normal data instances are more frequent in the dataset than abnormal instances. Particularly suitable for detecting unknown anomalies in cloud data centres where precise deﬁnition of anomaly characteristics may not always exist.

Examples of unsupervised learning based algorithms: (1) Clustering algorithms (Hierarchical clustering, k-means, GMM (EM), SOM; (2) Density based algorithms - KNN, LOF, Cell based algorithm; (3) Blind signal separation algorithms - PCA, ICA; (4) ANNs - AE (particularly RBM), Deep Belief Nets, VAE [45], GAN [46] paradigm based approaches.

**In [33],** HTM (Hierarchical Temporal Memory) network is used in anomaly detection in streaming data domain. HTM network, ML algorithm derived from neuroscience, continuously learns and models the spatiotemporal characteristics of the time series data to predict the values of the next time step. Due to the continuous learning nature of HTMs, changes in the underlying system are handled gracefully making the resulting anomaly detection solution tolerant to noisy data and continuously adapting to changes in its statistics. After receiving an input vector of streaming data, the HTM component generates 2 vectors of an internal state representations of the current step – (1) the sparse binary representation of the current input vector and (2) the sparse binary representation of the prediction of the input for the next step. Then, a raw anomaly score of the current step computed as a deviation between the predicted input and the actual one. After that, the likelihood of anomality of the current state is computed as a probability density of the current raw anomaly score in a rolling normal distribution of the last prediction errors (where the sample mean and variance are continuously updated from error values of latest steps). This approach is useful to handle a problem of the dynamic context of the timeseries where instantaneous predictions are often incorrect and thresholding the prediction error directly would lead to many false positives.

**In [20],** the authors present a black-box unsupervised behavior learning (UBL) solution for virtualized cloud environments. UBL leverages a Self Organizing Map (SOM) [34] which is capable of capturing complex system behavior while being computationally less expensive than comparable approaches such as kNN [35]. During training phase, SOM maps a high dimensional input space into a 2-dimensional map which represents a generalization of the whole measurement vector space and can capture the normal system behaviors under different workloads. After learning, frequently trained neurons (centroids) will have modiﬁed the weight vector values of their neighbor neurons with the same input measurement vectors. As a result, the weight vectors of the neurons that are frequently trained will look similar to the weight vectors of their neighbour neurons. Since systems are usually in the normal state, neurons representing a normal state will be more frequently trained than the neurons representing a pre-failure or failure states. Thus, there will be dominance of clusters representing different normal system behaviors. Then, in order to decide which of the system state represented by each cluster (normal, pre-failure, or failure), UBL calculates a neighborhood area size for each neuron. If the neighborhood area size is small, we know that the neuron is in a tight cluster of neurons, meaning the neuron is normal. During runtime, each measurement vector is mapped to that or another neuron using the same Euclidean distance and the input’s state is characterized by the state of the neuron it is mapped to. UBL also supports incremental updates which can continuously adjust the SOM with new measurement vectors. The basic idea of anomaly component localization in this work is to look at the difference between anomalous and normal neurons and output the metrics that differ most as faulty ones. Once a set of normal neurons has been found, the difference between the individual metric values of each normal neuron and those of the anomalous neuron is computed. After that, the metric differences are sorted from the highest to the lowest to determine a ranking order. After this process completes, there will be Q metric ranking lists which are then examined to determine a ﬁnal order. UBL achieves scalable behavior learning by virtualizing and distributing the learning tasks among distributed hosts. For this purpose, UBL monitors the residual resources on each host by aggregating the resource consumption of all the VMs running on the host. If the available residual resources are insufﬁcient, live migration is employed to move the learning VM to a host with sufﬁcient residual resources.

**In [37],** the light-weight black-box runtime IaaS cloud-oriented performance anomaly detection / localization tool presented (PerfCompass). It traces kernel-level system calls (low overhead) and performs real-time analysis to determine if the anomaly caused by an internal (application’s software bugs) or external fault (interference from other co-located applications, improper resource allocations). Thus, it is capable to diagnose previously unseen anomalies in black-box fashion. PerfCompass ﬁrst extracts different groups of closely related system calls (execution units), from the continuous raw system call traces. An execution unit is said to be affected by the fault if the solution detects any outlier in either the execution time or the frequency (based on the deviation from the mean) for any of the unit’s system calls. Then, for fault localisation purpose, the fault onset time (interval between the unit start and anomaly detection time) is calculated for each affected execution unit. If the fault onset time is lower than a predeﬁned threshold, the unit is considered as affected directly. Then, the total share of directly affected units is calculated (fault impact factor). If the value of the fault impact factor is close to 100 percent, the source of the fault is considered as external. Otherwise, the fault onset time dispersion is calculated (the standard deviation of the fault onset time among all the affected units). If large fault onset time dispersion is observed, the system infers that the fault is an internal one. The idea is that internal fault is likely to directly affect a subset of threads executing the buggy code and then indirectly affect other threads that communicate with the directly affected threads.

(3) Semi-supervised learning implies harnessing the large amounts of unlabelled data (which is available in many cases) in combination with typically smaller sets of labelled data [21]. Supervised learning methods can be applied (with certain assumptions about its distribution) to scenarios where a lack of labelled data exists or if the unlabelled data can provide additional information to improve the prediction / classification model.

An example of semisupervised learning algorithms are wrapper methods. They utilize one or more supervised base learners and iteratively train these with the original labelled data as well as previously unlabelled data that is augmented with predictions from earlier iterations of the learners (pseudo-labelled data). The procedure usually consists of two alternating steps of training and pseudo-labelling. In the training step, one or more supervised classifiers are trained (using original labelled data plus, pseudo-labelled data from previous iterations). In the pseudo-labelling step, the resulting classifiers are used to infer labels for the previously unlabelled objects; the data points for which the learners were most confident of their predictions are pseudo-labelled for use in the next iteration.

**In [19],** the self-evolving anomaly detector (SEAD) for cloud is presented. It does not require an extensive training on previously labelled data and is capable to catch unknown anomalies. It selfevolves by learning from newly generated and veriﬁed (by the system administrator) detection results. SEAD includes two components – anomaly detector and the working dataset. Anomaly detector is based on the continuously re-trained classiﬁcation models (SVM, OCSVM). For a new data record, the detector calculates an abnormality score. If the score is above a threshold, a warning is triggered with the type of abnormality and the cloud operators verify and label the event. This labelled data point is selectively included into the working dataset and the anomaly detector’s model is being retrained.

Recent failure prediction methods have mainly been focused on ﬁnding latent representations of the input data using deep neural networks. The most prominent example of such method are autoencoders - a class of neural networks with one or more hidden layers that has the objective of reconstructing its input. Autoencoders attempt to ﬁnd a lower-dimensional representation of the input space without sacriﬁcing substantial amounts of information. Thus, they inherently act on the assumption that the input space contains lower-dimensional substructures on which the data lie. [21].

Autoencoders were traditionally used for the purposes of denoising data – training on noisy versions of the input data, penalizing the reconstruction error of the reconstructions against the noiseless originals [47] or for feature space dimensionality reduction (in contrast to PCA and ICA algorithms, autoencoders are capable to catch a non-linear dependencies). Feature space dimensionality reduction techniques are highly important in ML based failure prediction since they accelerate data analysis and improve its accuracy (particularly, by amplifying the difference between normal and abnormal behaviours). **In [28],** the framework for node level anomaly detection / localization for large-scale systems presented. Its main components: (1) feature extraction, which generates a matrix with lower dimensionality, and (2) outlier detection, determining the nodes lying far away from the majority. After feature extraction phase, a cell-based algorithm [31] is used to quickly identify the outliers. It determines outliers on a cell by cell basis (number of objects in a cell) rather than on an object by object basis and has a complexity that is linear with respect to the number of the data points. The key idea is that an object is considered an outlier if certain fraction of the data lies greater than certain distance (Euclidean) away from it. Evaluation of the PCA and ICA-based feature extraction mechanisms showed that both approaches are time efﬁcient, but PCA based mechanism is less efﬁcient when there are multiple types of faults coexisted.

One of the recently proposed anomaly detection approaches based on autoencoders is EmBeD - runtime anomaly detection framework for cloud systems based on unsupervised model training techniques [43]. The anomaly detector is based on the hypotheses that (1) erroneous states of the system reﬂect in collective anomalies and (2) the anomalous behaviours reﬂect in anomalous values of the Gibbs free energy (intuitive meaning - an approximation of the status of the system). EmBeD detects anomalies at runtime using the baseline model (trend line) which represents the distribution of normal behaviour as a function of time with a standard deviation. This baseline model built using the values of the Gibbs free energy function computed while training the Restricted Boltzmann Machine (RBM) with KPI values monitored during normal execution. The anomaly detector signals incoming failures as anomalies in the free energy, that is, deviations from the baseline model produced during training. While training an RBM computes the joint probability distribution of the input data with respect to the set of parameters (the weights W of the edges and the biases) by associating a scalar energy function to visible units combination after each backward propagation (log-likelihood function based on the free energy function value). The most likely combination of the visible units corresponds to the combination that best approximates the input vector and is obtained by minimising of the free energy during the weights adjusting iterations.

# 3 Work done

Our work is dominantly based on the LOUD approach, presented in [48]. LOUD is an online lightweight metric-driven fault localization technique, which analyses the dependencies among anomalous KPIs to pinpoint the faulty resources that are likely responsible of future failures. The key differences between the Loud and our solution are: (1) our solution exploits failure prediction mechanism, implemented as a complementary combination of models trained with deep autoencoder ANN, for anomaly detection and One Class Support Vector Machine (OCSVM) with RBF kernel for binary classiﬁcation of anomalies; (2) in our solution we used our own implementation of the causality graph constructor. In contrast, LOUD in both cases relies on IBM ITOA-PI solution [49].

The implemented solution consists of 2 subsystems: (1) Failure predictor and (2) Fault localizer.

The Failure Prediction subsystem includes the following components / sub-processes: Data Collection, Anomaly Detection, and Anomaly Classiﬁcation. The Fault Localisation subsystem includes Granger Causality Graph generation, Propagation Graph derivation, and Ranking sub-processes.

The Training phase produces a set of Baseline Models (Deep Autoencoder model for anomaly detection, One Class SVM (OCSVM) Classiﬁer model for the anomaly set classiﬁcation) which captures the legal behavior of the KPIs, and a Granger Causality Graph, which captures causal relations between KPIs (key performance indicators representing a certain resource on certain node). The prototype computes the baseline models and the Granger causality graph using the data collected when monitoring normal execution.

During the test phase the failure prediction subsystem regularly monitors the target system and veriﬁes the new collected KPIs against the baseline models, and in case of a failure alert (failure prediction) returns the set of anomalous KPIs to the Fault Localiser, which exploits the Granger Causality graph for ranking the anomalous KPIs and resource-level fault localization.

Data collection / preprocessing is implemented by means of ElasticSearch toolset (monitoring, collection) and additional python scripts for data retrieval / preprocessing. The component elaborates on the data about KPIs provided in the form of time-ordered sequences of data points. Data collector monitors and regularly collects 85 KPIs per cluster node (1700 KPIs in total ) with an interval of 1 minute. Data collection is executed on an independent machine to avoid side eﬀects on the target system, which is monitored with lightweight monitoring probes (metricbeat agents). After collection, the data is retrieved and preprocessed. The metrics capture measurable aspects of the behaviour of the monitored system, for example, memory consumption, a number of requests served per minute, and so on. The resource can be any element of a system, for instance, a host, a virtual machine, and a speciﬁc application.

The anomaly detection is implemented by Anomaly Detector component, which analyses numerical data in the form of KPIs collected in a given time window from the target environment. It exploits a model trained with a deep autoencoder ANN to detect a KPIs with anomalous values and returns a list of the KPIs ﬂagged as anomalous in the given time interval.

The Classiﬁer implements a One Class Support Vector Machine (OCSVM) with RBF kernel for binary classiﬁcation of sets of KPIs, ﬂagged by the Anomaly Detector as anomalous in the given time interval. The OCSVM classiﬁer model is trained using sets of anomalous KPIs produced by the Anomaly Detector after processing the data collected during normal execution, thus the component can discriminate spurious anomaly sets (for example system overload provoked by the intensiﬁed user activity) from the ones that can lead to failure.

When triggered by a failure prediction event, the Ranker identiﬁes the likely faulty resource by ﬁrst (1) derivating of the Propagation Graph, which is a subgraph of the Granger Causality Graph with only the anomalous KPIs of the current timestamp, then (2) ranking the anomalous KPIs (represented by the nodes of the propagation graph), and after that (3) identifying the faulty resources according to the rankings of the corresponding KPIs.

A Granger Causality Graph is a directed weighted graph where nodes correspond to KPIs and edges represent granger-causal relations among KPIs. A directed edge with a weight W between nodes A and B that correspond to the KPIs A and B, respectively, indicates that node A granger-causes node B and the strength of that causality is equal to W, where W is a Coeﬃcient of Determination (or R squared), representing the percentage of the variation of B within a restricted linear regression model (where independent variables are represented by the past values of B) reduced by the unrestricted linear regression model (where independent variables are represented by the past values of B and past values of A). In our implementation, the statistical signiﬁcance of the granger-causality is estimated by F-test.

Our fault localisation approach is grounded on the observation that, once activated, faults cause increasingly many anomalies that spread both within the faulty resource and to resources either directly or indirectly interconnected with the faulty one. The problem of identifying the fault location can be formulated as the problem of locating the nodes that originated the spreading of the anomalies represented in the series of propagation graphs built at runtime.

Fault Localiser identiﬁes the nodes in the Propagation Graph that correspond to the faulty resource, assigning scores to the nodes through centrality indices, which are commonly used to identify the relevant nodes in weighted graphs concerning the connectivity among nodes. The nodes in the propagation graph with the highest centrality scores identify the anomalous KPIs most likely related to the faulty resource. Thus we have a series of scored KPIs, one for each timestamp.

We focused on the centrality indices, that take into account the global inﬂuence of a single node on the whole graph. We used one of the generalisations of the eigenvector centrality index - PageRank. It assigns the highest score to the node with the highest information ﬂow, and take into account the directions and weights of the links, and the presence of noise in the spread of the information within the graph. PageRank scores nodes according to both the number of incoming edges and the probability of anomalies to randomly spread through the graph (teleportation). The PageRank score of node A is computed iteratively from the scores of the nodes that are sources of edges directed to the node A.

Ranker identiﬁes the likely faulty resource as the resource, whose KPIs are most frequently presented in the list of the top-ranked KPIs (sublist of the list of ranked KPIs) according to the selected centrality index. The size of the sublist of the top ranked KPIs is set by the conﬁguration (”Top Ranked KPIs List Size” parameter). The intuition is that a faulty resource is likely to show an anomalous behaviour for multiple KPIs, which in turn are likely to aﬀect several other resources, and thus their KPIs, of the system. As a consequence, a faulty resource is likely to be present with multiple KPIs in the top part of the ranking.

We executed the experiments on a cloud environment running Redis, an open-source, BSD licensed, advanced key-value cache and store. It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets, sorted sets, bitmaps, and hyperloglogs. Redis is widely used in the industry.

We executed our testbed with a workload implemented with our traﬃc generator (deployed on 2 separated VMs). We shaped the number of users and calls in the workload based on week and day patterns: within each week, the number of users in working days is higher than in weekends, with each a day, the number of users grows at daytime and decreases at nighttime, with peaks at 9 am and 7 pm.

We simulated faulty scenarios using fault injection techniques. We injected the faults of the following types: packet loss, memory leak, CPU hog. For each fault we considered diﬀerent severity growth patterns: (i) linear pattern, the fault is triggered with the same frequency over time, (ii) exponential pattern, the fault is activated with a frequency that increases exponentially, resulting in a shorter time to failure, (iii) random pattern, the fault is activated randomly over time. In all experiments, the faults were injected into one of the master/slave pairs. The same fault was injected both in the master node and in assigned to this node slave one.

The evaluation results show that our mechanism can eﬀectively identify faulty nodes with high accuracy and no computation overhead. Evaluation of EmBeD (RBM based failure prediction approach [43]) solution on the same data and comparing the results with the results of our experimental analysis did not reveal significant differences in an efficiency of these approaches.

# 4 Work Plan

During the next years we plan to work on:

1. Searching the ways to improve our approach to overcome the observed limitations:
   1. Considering temporal relations in time series (some measurements that are anomalous at one point in time may be completely normal at another);
   2. Adaptive thresholding and hyper-parameterization;
   3. Dealing with high dimensional metric sets while using light-weight real-time prediction approaches;
2. Dynamic context-aware detection techniques - evaluation and adaptation of the approaches to a dynamic environment. The dynamism is a unique feature of cloud environments and based on its elasticity, which allows to provision/deprovision or reconfigure resources (accomplished by horizontal or vertical scaling), i.e., VMs and containers [42]. Hence, the operating context in clouds changes more frequently, compared to traditional distributed systems. High dynamism and sharing of resources in the cloud often lead to frequent changes in an application’s resource model, obviating the applicability of problem determination techniques that create a stationary model for a system and identify deviations from the model to ﬂag errors. The challenge is identifying and characterising execution contexts as they evolve over time. Context-aware solutions capable of achieving this in addition to adapting to non-stationary cloud behaviours will greatly improve application performance.

Particularly, we plan to evaluate and adapt our deep autoencoder and RMB based solutions to the container-based cloud environments. One of the possible scenarios is using a cloud managed by Kubertnetes (an open-source system for automating deployment, scaling, and management of containerized applications).

1. Generalisability – searching for domain independent approaches - working with different fault types and fault injection patterns, conducting experiments and evaluate the approach on different system architectures;
2. Extension of the ﬁeld of application of the approach to conceptually different kind of fault types, related to the system healthiness.

# References

1. CloudPD: Problem Determination and Diagnosis in Shared Dynamic Clouds
2. A Wavelet-inspired Anomaly Detection Framework for Cloud Platforms
3. Adaptive proﬁling for root-cause analysis of performance anomalies in web-based applications
4. Fingerprinting the Datacenter: Automated Classification of Performance Crises
5. Adaptive Profiling for Root-cause Analysis of Performance Anomalies in Web-based Applications
6. Thirteen ways to look at the correlation coefficient
7. Computing and Visualizing Dynamic Time Warping Alignments in R: The dtw Package
8. A Framework for Automated Performance Bottleneck Detection
9. Emergent (Mis)behavior vs. Complex Software Systems
10. Diagnosing performance changes by comparing request flows
11. The Kolmogorov-Smirnov Test for Goodness of Fit
12. Missing values, outliers, robust statistics & non-parametric methods
13. Novelty detection: A review part 1: Statistical approaches
14. Gaussian mixture models
15. Applied Regression Analysis and Other Multivariable Methods
16. Detection of Performance Anomalies in Web-based Applications
17. Statistical Process Control
18. Adaptive Anomaly Detection in Performance Metric Streams
19. A self-evolving anomaly detection framework for developing highly dependable utility clouds
20. UBL: Unsupervised Behavior Learning for Predicting Performance Anomalies in Virtualized Cloud Systems
21. A survey on semi-supervised learning
22. Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study
23. Estimating the support of a High-Dimensional Distribution
24. Efﬁcient algorithms for mining outliers from large data sets
25. Fast outlier detection in high dimensional spaces
26. Lof: identifying density-based local outliers
27. A survey on unsupervised outlier detection in high-dimensional numerical data
28. Toward automated anomaly identiﬁcation in large-scale systems
29. Collective Anomaly Detection based on Long Short Term Memory Recurrent Neural Network
30. Long short-term memory
31. Distance-Based Outliers: Algorithms and Applications
32. Deep learning for anomaly detection: A survey
33. Unsupervised real-time anomaly detection for streaming data
34. Self-Organizing Maps
35. Introduction to Data Mining
36. DeepLog: Anomaaly Detection and Diagnosis from System Logs through Deep Learning
37. PerfCompass: Toward Runtime Performance Anomaly Fault Localization for Infrastructure-as-a-Service Clouds
38. DAPA: Diagnosing Application Performance Anomalies for Virtualized Infrastructures
39. Network anomaly detection using IP flows with Principal Component Analysis and Ant Colony Optimization
40. Ant Colony Optimization
41. Ant Colony Optimization and Swarm Intelligence
42. Coordinating Vertical Elasticity of both Containers and Virtual Machines
43. An RBM Anomaly Detector for the Cloud
44. AN AGENT-BASED APPROACH FOR BUILDING COMPLEX SOFTWARE SYSTEMS
45. Tutorial on Variational Autoencoders
46. Generative Adversarial Nets
47. Extracting and composing robust features with denoising autoencoders
48. Localizing faults in cloud systems
49. SmartCloud Analytics—Predictive Insights
50. A deep learning framework for financial time series using stacked autoencoders and longshort term memory
51. Anomaly Detection using Deep Autoencoders for in-situ Wastewater Systems Monitoring Data
52. Online Failure Prediction for Microservice Architectures
53. Performance Anomaly Detection in Microservice Architectures Under Continuous Change
54. Predicting Failures in Multi-Tier Distributed Systems
55. Architecting Principles for Systems-of-Systems
56. Large-scale complex IT systems
57. IBM Corporation, “IBM IT Operation Analytics - Predictive Insights,” https://www.ibm.com/cloud-computing/products/ devops/it-operations-analytics/, last access: jan 2017.
58. A Survey of Eigenvector Methods for Web Information Retrieval