A Controller for Anomaly Detection, Analysis and Management for Self-Adaptive Container Clusters

Areeg Samir, Nabil El Ioini, Ilenia Fronza, Hamid R. Barzegar, Van Thanh Le and Claus Pahl

Faculty of Computer Science
Free University of Bozen-Bolzano
39100 Bolzano, Italy
Email: firstname.surname@unibz.it

***Abstract*—Service computing in the cloud allows applications to be deployed remotely. These are managed by third-party service providers that make virtualised resources available for these services. Self-adaptive features for load-balancing and auto-scaling are available here, but generally there is no direct access to the infrastructure or platform-level execution environment. Some quality parameters of a provided service can be directly observed while others remain hidden from the service consumer. Our solution is an autonomous self-adaptive controller for anomaly remediation in this semi-hidden setting. The objective of the controller is to, firstly, determine possible root causes of consumer-observed anomalies and, secondly, take appropriate action. This needs to happen in an underlying provider-controlled infrastructure. We use Hidden Markov Models to map observed performance anomalies into hidden resources, and to identify the root causes of the observed anomalies. We apply the model to a clustered computing resource environment that is based on three layers of aggregated resources. We discuss use cases to illustrate the utility of the proposed solution.**

***Index Terms*—Cloud Computing; Container Technology, Distributed Clusters; Markov Model; Anomaly Detection; Anomaly Analysis; Workload; Performance.**

I. Introduction

Due to the dynamic nature of loads in a distributed cloud or edge computing setting, consumers may experience anomalies (e.g., variation in a resource performance) due to distribution, heterogeneity, or scale of computing that may lead to performance degradation and potential application failures. Furthermore, loads might vary over time:

- changes of the load on individual resources,
- changing workload demand and prioritisation,
- reallocation or removal of resources in dynamic environments.

These may affect the workload of current system components (container, node, cluster), and may require rebalancing their workloads. These cloud and edge computing settings allow services to be provided by allowing applications to be deployed and managed by third-party providers. These make shared virtualised resources accompanied by dynamic management facilities [1],[2],[3],[4] available.

Recent works such as [5],[6],[7] have looked at resource usage, rejuvenation, or analysing the correlation between resource consumption and abnormal behaviour of applications. Less attention has been given to the possibly hidden reason behind the occurrence of an observable performance degradation (root cause) [8], and how to deal with the degradation in a hierarchically organised cluster setting.

In order to address these challenges in a shared virtualised environment, third party providers provide some factors that can be directly observed (e.g., the response time of service activations) while others remain hidden from the consumer (e.g., the reason behind the workload, the possibility to predict the future load behaviour, the dependency between the affected nodes and their loads in a cluster).

The core solution presented in this paper is a controller [9],[10] that automatically detects the anomalous behaviour within a cluster of containers running on cluster nodes, where a sequence of observations is emitted by the system resource. The controller remedies the detected anomalies that occur at the container, node or cluster level. To achieve that this paper: (i) analyse the possible causes of observable anomalies in an underlying provider-controlled infrastructure; (ii) define an anomaly detection, and analysis controller for a self-adaptive cluster environment, that automatically manages the resource workload fluctuations. Our objective is to introduce this controller in terms of its architecture and processing activities. We expand our earlier work in [1],[18] here by discussing context and the application of the architecture in use cases in more detail.

A specific feature of our solution is the differentiation between two types of observations:

- System states (anomaly/fault) that refer to anomalous or faulty behaviour, which is hidden from the consumer. This indicates that the behaviour of a system resource is significantly different from normal behaviour. An anomaly in our case may point to an undesirable behaviour of a resource such as overload, or to a desirable behaviour like underload of a system resource, which can be used as a solution to reduce the load at overloaded resources.
- Emission or Observation (observed failure from these states), which indicates the occurrence of failure resulting from a hidden state.

We focus on technical concerns in relation to workload and response time fluctuations.

To address this problem, we use so-called Hierarchical Hidden Markov Models (HHMMs) [11] as a stochastic model to map the observed failure behaviour of a system resource to its hidden anomaly causes (e.g., overload) through tracking the detected anomaly to locate its root cause. We implement the proposed controller for a clustered computing resource environment targeting specifically container technologies.

The paper is structured as follows. Section II reviews related work extensively. Section III discusses use cases to clarify the problem context and how an anomaly solution can address the problems. Section IV explains the motivation behind our work in terms of the proposed architecture. Section V gives a brief introduction of HHMM as the central formal construct. Section VI explains the mapping of failure and fault. Section VII explains the controller architecture with its analysis and recovery components. Section VIII evaluates the proposed architecture. As an outlook into the transferability of the architecture, we discuss trust anomalies in Section IX. Section X concludes the paper.

II. Related Work

There are a number of studies that have addressed workload analysis in dynamic environments [5],[6],[7]. They proposed various methods for analysing and modelling workload.

Dullmann [17] provides an online performance anomaly detection approach that detects anomalies in performance data based on discrete time series analysis. Peiris et al. [21] analyse root causes of performance anomalies by combining the correlation and comparative analysis techniques in distributed environments. Sorkunlu et al. [22] identify system performance anomalies through analysing the correlations in the resource usage data. Wang et al. [7] propose to model the correlation between workload and the resource utilization of applications to characterize the system status. Maurya and Ahmad [24] propose an algorithm that dynamically estimates the load of each node and migrates the task if necessary. The algorithm migrates the jobs from overloaded nodes to underloaded ones through working on pair of nodes, it uses a server node as a hub to transfer the load information in the network, which may result in overhead at the node.

Moreover, many works have used the HMM and its derivations to detect anomaly. In [25], the author proposes various techniques implemented for the detection of anomalies and intrusions in the network using the HMM. In [27] the author detects faults in real-time embedded systems using the HMM through describing the healthy and faulty states of a system's hardware components. In [29], HMM is used to find, which anomaly is part of the same anomaly injection scenarios.

A fault tolerance management solution for physical and virtual machines level is presented in [34]. It uses Redundant Array of Independent Disks technology in order to optimize storage space and to recover data in case of failure. More concretely, the author divides a set of VM and PM into subsets of the same size. Then, two services are used to collect information about a resource status and to manage resources through adding and removing resources in order to mitigate resource failure. Here, only two aspects of recovery handling, namely handling storage disk crash and system crashes are considered. In [33], the detection of anomalous behaviours (such as CPU overload or Denial of Service Attacks) are the topic. The authors provide an adaptation policy based on a multi-dimensional utility-based model. The score and likelihood for the anomaly detected to select adaptation policy is provided in order to scale compute resources. Here, a node leader for each microservice cluster is selected and each node maintains the cluster state and preserves the cluster records. This node leader also decides on the adaptation policy action. In this work, however, only two types of anomalies are handled. It also limits actions to mitigate the anomalous behaviour to horizontal and vertical auto-scaling actions. Furthermore, prediction is not included.

In [35], performance and prediction are the key concerns. The performance of several machine learning models is investigated in order to predict attacks on the IoT systems accurately. A random forest technique is shown to achieve good anomaly prediction in comparison to other machine learning solutions. Nonetheless, the focus is on predicting network anomaly only. In [36], a solution to estimate the capacity of a microservice is presented. Here measuring the maximal number of successfully processed user requests per second for a given service such that no SLO is violated is the key idea. The authors carry out a number of load tests and then fit an appropriate regression model to the acquired performance data. This work investigates the impact of workload on measures CPU and memory usage. Changing the number of requests affects the number of virtual CPU cores does not affect the memory utilization significantly. What is somewhat neglected is the dependency between nodes and services. Predicting future workload is not covered.

Localizing faulty resources in cloud environments through modelling correlations among anomalous resources is addressed in [65]. Graph theory is employed in order to locate the correlation between pairs of resources. analysing the amount of occupied memory in a physical server, the CPU consumption of a virtual machine and the number of connections accepted by an application is the focus. This work does not cover anomalies specifically in microservice or container environments. In [39] the authors focus on detecting anomalous behaviour of services deployed on VM in cloud environment. Like our architecture, different anomaly injection scenarios are created and workload is generated to test the impact of anomaly on the cloud services. The authors emulated different anomalies at the CPU, memory, disk, and network. However, their work does not track the cause of anomalous behaviour in containerized cluster environment, and it neglects the dependency between nodes.

More dedicated to microservices and containers are:

- In [69], the authors investigate the mutual impact of microservices on the same host. This study looks at the

consequences of these side effects have on failure prediction. For this, the authors measure the CPU, memory and network usage metrics of the containers and nodes. The work evaluates the current failure prediction methods in a microservice architecture, but does not locate or detect anomalous behaviour. The work focus is CPU-bound workload.

- Kratzke [43] looks at the impact of network performance on containers deployed on VMs. He carries out a number of experiments in order to analyse the network performance of containers by using horizontal scaling and considering the network data transfer rate. Nonetheless, the focus is on network aspects and their impact on container performance.

Another couple of studies focus on performance:

- Wert [47] presents a specification language, called the Performance Problem Diagnostics Description Model, in order to specify information needed for an automatic performance problem diagnostics. Here, workload is analysed to detect performance faults and categorize performance faults into three layers: (i) symptoms, which are externally visible indicators of a performance problem, (ii) manifestation, which are internal performance indicators, and (iii) finally root-causes, which are the physical factors whose removal eliminates the manifestations and symptoms of a performance problem. The author's approach does not consider dependencies between faults nor avoids human interaction. For example, there should be heuristics to be able to detect performance problems. The approach is designed to apply for a specific application domain. A recovery mechanism for the detected faults is not covered. Also, dependencies between anomalies are not addressed. The solution is based on predefined heuristics (rules) in order to detect performance problems. Consequently, applying the approach on a different domain or changing the fault model requires heuristics update.
- Ibidunmoye et al. [72] look at the detection the anomalous behaviour in performance using forecasting model to estimate the bandwidth, detect performance changes and decompose time series into components. In this work, hard thresholds are used in all datasets, which might not reflect the actual workloads in system accurately. They only look at the detection of anomalies without a further analysis. Labelled-time is used there, which is often not suitable in order to detect all anomalies as some anomalies could not be discovered during the detection process and time complexity in terms of data size may occur.

Prediction is another important concern:

- Predicting the impact of processor cache interference within consolidated workloads is the focus of [62]. In order to predict the performance degradation of these consolidated applications, the proposed prediction solution is linear in terms of the number of cores sharing the last-level cache. The authors limit their discussion to cache contention issues, ignoring other resource types and resulting faults.
- Guan and co-authors have implemented a probabilistic prediction model using a supervised learning method in [48]. This model serves to detect anomalous behaviour in cloud-based environments by analysing the correlation between different selected metrics (including CPU, memory, disk, and network concerns) in order to determine essential metrics that characterize the correlation between performance and an anomaly event. Here, directed acyclic graphs DAGs are used in order to analyse the correlation of the different performance metrics with failure events in both virtual and physical machines. The authors determine in the paper the conditional probability of each metric for the anomaly occurrences. Those metrics where conditional probabilities are greater than a predefined threshold are then selected. Nevertheless, their results show that their model suffers from poor prediction efficiency when it is used to predict cloud anomalies.
- In [67], the authors introduce a general-purpose prediction model that aims at preventing anomalies in cloud environment. Their supervised learning-based model utilised as in our case Markov models. There they combine two dependent Markov chains with a tree augmented the Bayesian network. Statistical learning algorithms are applied based on system-level metrics (CPU, memory, network I/O statistics) aiming to predict anomalous behaviour. A limitation is that the authors do not discuss prediction efficiency.

Nathuji et al. [78] look at a control theoretic consolidation solution that aims at mitigating effects of anomalies in the context of cache, memory and hardware contention of co-existing workloads. Their solution manages the interference between consolidated virtual machines by dynamically adapting resource allocations to applications based on workload SLAs. The focus in that paper is on CPU-bound workload and compute-intensive applications. Monitoring is at the centre of [77]. They discuss a technique for localizing anomalies at runtime using the Kieker monitoring approach. For anomaly localization, an anomaly score is calculated for each operation using a specified threshold. A set of rules is given to detect performance anomaly, which are continuously evaluated based on the anomaly score by utilising forecasting techniques to predict future values in time series. Experimentally observed measurement values such as response times are analysed with the forecasted values to detect anomalies. Different types of performance anomalies and anomaly dependencies are however not considered.

The objective of this paper here is to detect and locate the anomalous behaviour in containerized cluster environment [28] through considering the influence of dynamic workloads on their anomaly detection solutions. The proposed controller consists of:

- (1) *Monitoring*, that collects the performance data of

(services, containers, nodes 'VM') such as CPU, memory, and network metrics;
- (2) *Detection*, that detects anomalous behaviour, which is observed in response time of a component;
- (3) *Identification*, which tracks the cause of the detected anomaly.
- (4) *Recovery*, that heals the identified anomalous components.
- (5) *Anomaly injection*, which simulates different anomalies, and gathers dataset of performance data representing normal and abnormal conditions.

III. A Discussion of Use Cases

In order to better motivate our solution, we look at two use cases now. One looks at a widely used cloud-based scenario. Here, we assume a cluster of containers that are managed by an orchestrator such as Kubernetes or Docker Swarm. The other use case considers an edge cloud setting, where again a cluster architecture, but this time deployed on constrained hardware devices to host the container cluster is considered.

A. Use Case 1: Cloud-centric Container Orchestration

Container technology is increasingly popular recently. It is now widely used as the mechanism for software deployment. Containers as a more lightweight form of virtualisation compared to virtual machines (VMs) consume less resources. They compare favourably to VMs in terms of startup time to memory/storage needs. This applies also to cloud environments. Many cloud infrastructure (IaaS) providers and platform service (PaaS) providers provide different container deployment solutions. In many of these cases, an orchestration tool like Kubernetes[1], see Figure 1, or Docker Swarm, is used to support the automated deployment, scaling and management of containerized applications are used by the providers, see Figures 2 and 3. These are typically homogeneous cloud container cluster in terms of the underlying infrastructure.

This setting, however, creates problems regarding monitoring and problem detection. A concrete problem that becomes obvious here is that a service consumer generally have access to monitoring data at an (application or platform) service level, but not necessarily at the underlying physical infrastructure level [44], which is hidden by the service provider. Nonetheless, service consumer are often given access to controllers that can for instance auto-scale the application deployed.

In this case, our solution can be applied. The user could be provided with a anomaly management architecture (essentially a trained Hidden Markov Model HMM, as we will see later). This model then reflects possible underlying (and unobservable) faults for the failures that have been observed by the service consumer.

[1]https://kubernetes.io/

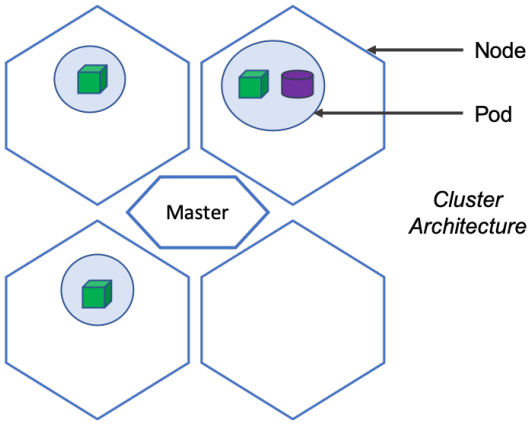

Figure 1. A Container Cluster Architecture based on Kubernetes.

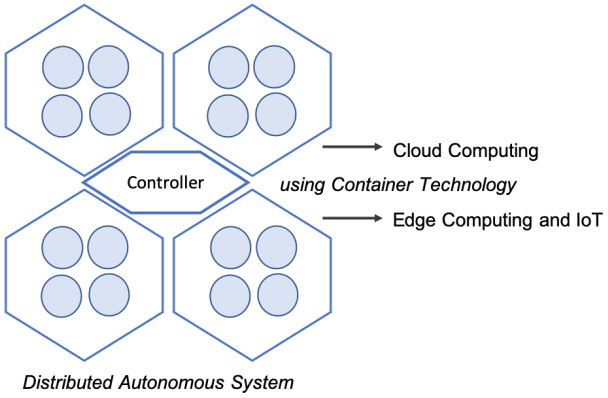

Figure 2. A Distributed Container System suitable for Cloud and Edge Computing.

B. Use Case 2: Edge Cloud Orchestration – Connected Cars

The lightweightness of containers as introduced above makes them very suitable to be utilised outside a classical centralised cloud setting. Here, edge cloud infrastructures that provide computational capabilities for IoT or other remote application can benefit from the containers' lightweightness. This is in particular useful if the edge infrastructure is limited in terms of its capabilities. For this type of situation, we assume now a cluster on single-board devices as the

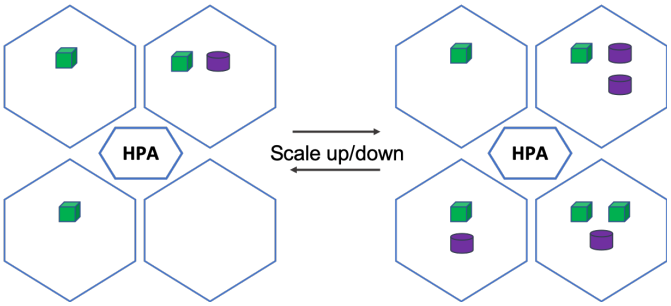

Figure 3. Auto-Scaling with Kubernetes – the Horizontal Pod Autoscaler (HPA).

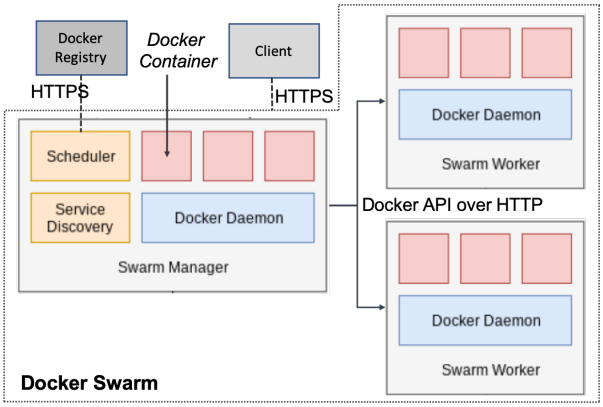

FIGURE 4. CONTAINER ORCHESTRATION WITH DOCKER SWARM.

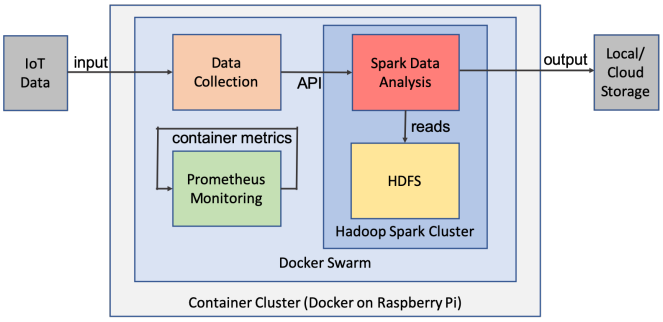

FIGURE 5. A DOCKER ARCHITECTURE FOR DATA STREAM PROCESSING HOSTED ON RASPBERRY PI (RPI) DEVICES.

physical infrastructure to host the container cluster platform. Specifically, we use single-board Raspberry Pi[2] devices. In our experiments, we use Docker Swarm[3] as the container orchestration tool, see Figure 4.

We first introduce the general edge cluster architecture and then illustrate this through a connected car use case.

Our proposed solution is based on a categorisation of hidden fault and observable failure cases, in which observable failures (to meet QoS requirements of the consumer) are mapped to their root causes, i.e., the underlying hidden faults that have caused them. Examples are an overloaded container that simply slows down or a neighbouring container on the same node on which a concrete container depends (e.g., is waiting for an answer for a request) [18],[19]. We use Markov models in our formalisation that reflect the possibility of several causes and the likelihood of each of these. Typical (hidden) fault types are CPU hog, network latency or workload contention, while only response time failures are observable. For each of these mapping cases, we have associated suitable remedial actions, such as workload distribution, container migration or resource rescaling.

A technical term for the connected cars scenario is CCAM. CCAM stands for connected Cooperative, Connected and Automated Mobility. We specifically look at connected cars in this context to make the problem more concrete. CCAM brings in this context the world of 5G telecommunications, automotive solutions and cloud and edge computing together.

Concrete use cases are car manoeuvre support, for instance for lane changing, or video streaming. For lane changing, several cars might need to be coordinated in their actions using mobile edge clouds. Here, latency is a critical issue and needs to be constantly monitored.

There are a number of identifiable problems:

- node overload: both on-board units as well as road-side units in a connected car situation are very limited in terms of their computational and storage capacities.
- connection dependency: 5G is normally considered as the minimum required standard in order to meet the required latency needs. Even with 5G enabled, a high density of cars combined with the need to support while the cars are moving creates capacity challenges also for the network.

Solutions that are applicable as remedial actions are:

- allocate more capacity, e.g., for video buffering
- migrate resources to avoid network problems (migrate container and/or data/state)
- repurpose nodes (redeploy other containers)

Our work in [20],[46] demonstrates how a container cluster solution implemented on Raspberry Pis can support this type of scenario. There, a Docker Swarm based management supports containers for data stream processing (Apache Spark), supported by Prometheus as a monitoring tool, Grafana for analyse/visualised data and databases like InfluxDB to store data, see Figures 5 and 6.

A concrete technology that involves mobile edge clouds and additional on-board and road-side assistance is CCAM – Cooperative, Connected and Automated Mobility.

Both node and connection problem can trigger an anomaly management system. Once an anomaly is identified, actions such as moving containers to avoid the node/connection problem or repurpose nodes to meet changing needs, indicated for instance by underload, are pursued.

In this context, the decision model can be a Hidden Markov Model HMM. The probabilities reflected in the HMM represent the following aspects:

- the adequacy of the failure/fault mapping (identify anomaly),
- the suitability of the recovery action (recover anomaly).

The HMM identifies different anomaly states [1]. These are dependent on the monitored performance and workload/utilisation metrics. In other works [9],[10], we have used fuzzy logic to map monitored data to so-called membership functions that represent these different states. We refer the reader to these works for more detail. Here, we focus on the anomaly processing.

[2]https://www.raspberrypi.org/
[3]https://docs.docker.com/engine/swarm/

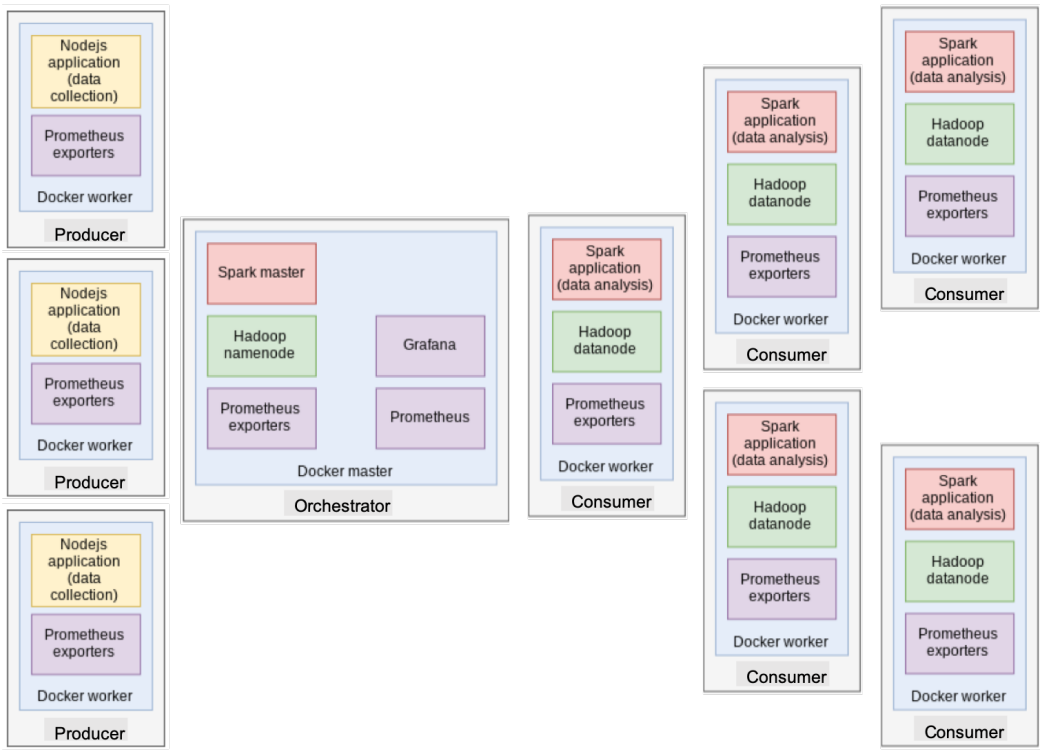

FIGURE 6. A DOCKER CONTAINER DISTRIBUTION FOR THE DATA STREAM PROCESSING APPLICATION ON RPIS.

IV. MOTIVATING EXAMPLE FOR THE CONTROLLER ARCHITECTURE

A failure is the inability of a component to perform its functions with respect to a specified (e.g., performance) requirements [26]. Faults (also called anomalies) are system properties that describe an exceptional condition occurring in the system operation that may cause one or more failures [31].

We assume that a failure is a kind of unexpected response time observed during system component runtime (i.e., observation), while fluctuations occurring during a resource execution of a component are considered as faults or anomalies (state of a hidden component). For example, fluctuations in workload such as overload faults may cause delay in a system response time (observed failure).

Generally, the observed metrics do not provide enough information to identify the cause of an observed failure. For example, the CPU utilization of a containerized application is about 30% with 400 users, and it increases to about 70% with 800 users in the normal situation. Obviously, the system is normal with 800 users. But probably the system shows anomalous behaviour with 400 users, when the CPU utilization is about 70%. Thus, it is hard to identify whether the system is normal or anomaly just based on the CPU utilization. Thus, specifying a threshold for the utilization of resource without considering the number of users, will raise anomalous behaviours. Consequently, it is important to integrate the data of workload into anomaly detection and identification solutions.

Once provided with a link between faults (workloads), and failures (response time) emitted from components, we can also apply a suitable recovery strategy depending on the type of identified fault.

Thus, a self-adaptation controller will be introduced later in this paper to automatically manage faults through identifying the degradation of performance, determining the dependency between faults and failures, and applying recovery strategies. We can align the steps of the fault management with the Monitoring, Analysis, Planning, Execution, and Knowledge (MAPE-K) control loop as a conceptual framework.

V. HIERARCHICAL HIDDEN MARKOV MODEL (HHMM)

The Hierarchical Hidden Markov Model (HHMM) [11] is a generalization of the Hidden Markov Model (HMM) that is used to model domains with hierarchical structure (e.g., intrusion detection, plan recognition, visual action recognition). The HHMM can characterize the dependency of the workload (e.g., when at least one of the states is heavy loaded). The states (cluster, node, container) in the HHMM are hidden from the observer and only the observation space is visible (response time). The states of HHMM emit sequences rather than a single observation by a recursive activation of one of the substates (nodes) of a state (cluster). This substate might also be hierarchically composed of substates (containers). Each container has an application that runs on it. In case a node or a container emits observation, it will be considered a production state. The states that do not emit observations directly are called internal states. The activation of a substate

by an internal state is a vertical transition that reflects the dependency between states. The states at the same level have horizontal transitions. Once the transition reaches to the End state, the control returns to the root state of the chain as shown in Figure 7. The edge direction indicates the dependency between states.

We choose the HHMM as every state can be represented as a multi-level HMM in order to: (1) show communication between nodes and containers, (2) demonstrate the impact of workloads on the resources, (3) track the anomaly cause, and (4) represent the response time variations that emit from nodes or containers.

VI. Failure-to-Fault Mapping

Based on analysing the log file and monitored metrics from existing systems, we can obtain knowledge regarding (1) the dependencies between containers, nodes and clusters; (2) response time fluctuations emitted from containers or nodes; (3) workload fluctuations that cause changes in response time. We need a mechanism that automatically maps a type of anomaly to its causes. We can identify different failure-fault cases that may occur at container, node or cluster level as illustrated in Figure 8. We focus on addressing the correlation between workload (overload) and the response time at container, node, and cluster.

A. Low Response Time Observed at Container Level

There are different reasons that may cause this:

- *Case 1.1. Container overload (self-dependency)*: means that a container is busy, causing low response times, e.g., c_1 in N_1 has entered into load loop as it tries to execute its processes while N_1 keeps sending requests to it, ignoring its limited capacity.
- *Case 1.2. Container sibling overloaded (internal container dependency)*: this indicates another container c_2 in N_1 is overloaded. This overloaded container indirectly affects the other container c_1 as there is a communication between them. For example, c_2 has an application that almost consumes all resources. The container has a communication with c_1. At such situation, when c_2 is overloaded, c_1 will go into underload. The reason is that c_2 and c_1 share the resources of the same node.
- *Case 1.3. Container neighbour overload (external container dependency)*: this happens when a container c_3 in N_2 is linked to another container c_2 in another node N_1. In another case, some containers c_3 and c_4 in N_2 dependent on each other, and container c_2 in N_1 depends on c_3. In both cases c_2 in N_1 is badly affected once c_3 or c_4 in N_2 are heavily loaded. This results in low response time observed from those containers.

B. Low Response Time Observed at Node Level

There are different reasons that cause such observations:

- *Case 2.1. Node overload (self-dependency)*: generally, node overload happens when a node has low capacity, many jobs waited to be processed, or problem in network. Example, N_2 has entered into self-load due to its limited capacity, which causes an overload at the container level as well c_3 and c_4.
- *Case 2.2. External node dependency*: occurs when low response time is observed at node neighbour level, e.g., when N_2 is overloaded due to low capacity or network problem, and N_1 depends on N_2. Such overload may cause low response time observed at the node level, which slow the whole operation of a cluster because of the communication between the two nodes. The reason behind that is N_1 and N_2 share the resources of the same cluster. Thus, when N_1 shows a heavier load, it would affect the performance of N_2.

C. Low Response Time Observed at Cluster Level

If a cluster coordinates between all nodes and containers, we may observe low response time at container and node levels that cause difficulty at the whole cluster level, e.g., nodes disconnected or insufficient resources.

- *Case 3.1. Communication disconnection* may happen due to problem in the node configuration, e.g., when a node in the cluster is stopped or disconnected due to failure or a user disconnect.
- *Case 3.2. Resource limitation* happens if we create a cluster with too low capacity which causing low response time observed at the system level.

The mapping between faults and failures needs to be formalised in a model that distinguishes observations and hidden states. Thus, HHMM is used to reflect the system topology.

VII. Self-Adaptive Controller Architecture

This section explains the controller architecture (Figure 9).

A. Managed Component Pool

The system under observation consists of a cluster that is composed of a set of nodes that host containers as the application components. A node could be a virtual machine that has a given capacity. The main job of the node is to assign requests to its containers. Containers are stand-alone, executable packages of software. Multiple containers can run on the same node, and share the operating environment with other containers. Each component either cluster, node, or container may emit observations. Observations are emissions of failure from a component resource.

We install an agent on each node to collect metrics from the pool, and to expose log files of containers and nodes to Real-Time/Historical Data storage. The agent adds data interval function to determine the time interval at which the data collected belongs. The data interval function specifies the lower and upper limits for the data arrivals. The response time,

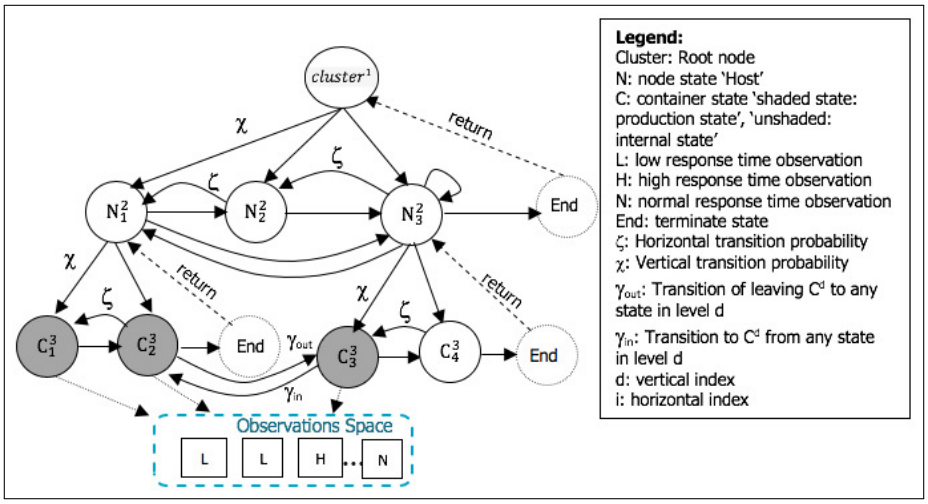

FIGURE 7. THE HHMM FOR WORKLOAD.

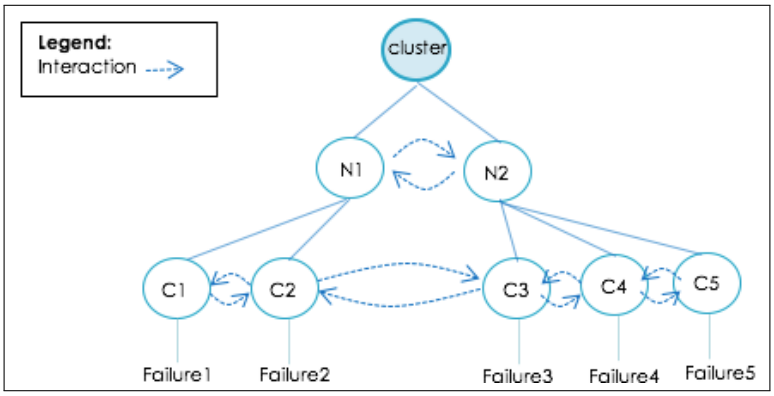

FIGURE 8. DEPENDENCIES BETWEEN CLUSTER, NODES AND CONTAINERS.

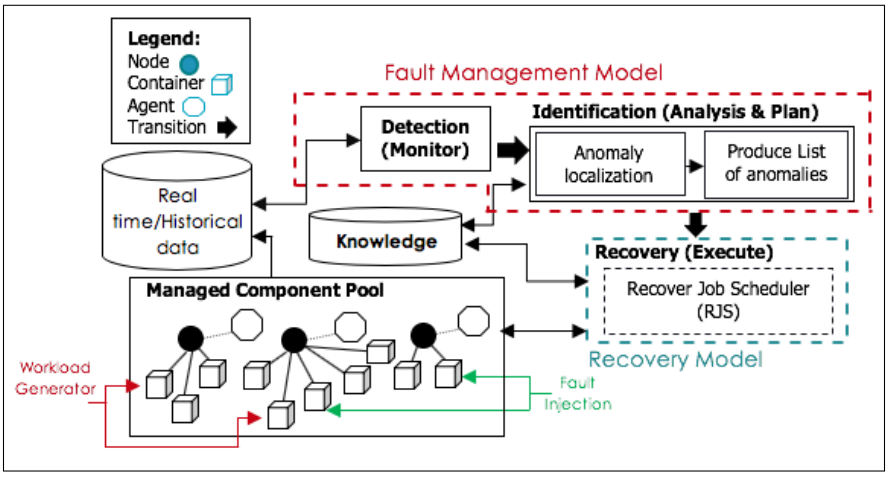

FIGURE 9. THE ANOMALY MANAGEMENT CONTROLLER ARCHITECTURE.

and the state of the component are assigned to each interval. Moreover, the agent gathers data regarding the workload (i.e., no. of requests issued to component), and monitored metrics (i.e., CPU, Memory) to characterize the workload of components processed in an interval. The agent pushes the data to be stored in the Real time/Historical storage to be used by the Fault Management Model.

B. Fault Management Model

The model is based on the history of the overall system performance. This can be used to compare the predicted status with the currently observed one to detect anomalous behaviour. The fault management model consists of:

a) Detection (Monitor): To detect anomaly, the monitor collects system data from the Real time/Historical storage. Then, it checks if there is anomalous behaviour at the managed components through utilizing spearmans rank correlation coefficient to estimate the dissociation between the response time and the number of requests (workload). If there is a decrease in the correlation degree, then the metric is not associated with the increasing workload, which means the observed variation in performance is not an anomaly. In case the correlation degree increase, this refers to the existence of anomaly occurred as the impact of dissociation between the workload and the response time exceeds a certain value. To achieve that we wrote an algorithm to be used as a general threshold to highlight the occurrence of anomaly in the managed pool under different workloads. We added a unique workload identifier to the group of workloads in the same period to achieve traceability through the entire system. We specified that the degree of dissociation (DD = 15) can be used as an indicator for performance degradation considering different response time, and different workloads. The value of DD will be compared against the monitored metrics (i.e., CPU, Memory utilization) to detect anomalous behaviour within the system. In case an anomaly is detected, the controller moves to the fault management to track the cause of anomaly in the system.

b) Identification (Analysis and Plan): Once there is appearance of anomaly, we build HHMMs to identify anomalies in system components as shown in Figure 7.

The HHMM vertically calls one of its substates $N_1^2 = \{C_1^3, C_2^3\}$, N_2^2, $N_3^2 = \{C_3^3, C_4^3\}$ with vertical transition χ and d index (superscript), where $d = \{1, 2, 3\}$. Since N_1^2 is abstract state, it enters its child HMM substates C_1^3 and C_2^3. Since C_2^3 is a production state, it emits observations, and may make horizontal transition γ, with i horizontal index (subscript), where $i = \{1, 2, 3, 4\}$, from C_1^3 to C_4^3. Once there is no another transition, C_2^3 transits to the end state End, which ends the transition for this substate, to return the control to the calling state N_1^2. Once the control returns to the state N_1^2, it makes a horizontal transition (if exist) to state N_2^2, which horizontally transits to state N_3^2. State N_3^2 has substates C_3^3 that transits to C_4^3 which may transit back to C_3^3 or transits to the End state. Once all the transitions under this node are achieved, the control returns to N_3^2. State N_3^2 may loop around, transits back to N_2^2, or enters its End state, which ends the whole process and returns control to the cluster. The model cannot horizontally do transition unless it vertically transited. Further, the internal states do not need to have the same number of substates. It can be seen that N_1^2 calls containers C_1^3 and C_2^3, while N_2^2 has no substates. The horizontal transition between containers reflect the request/reply between the client/server in our system under test, and the vertical transition refers to child/parent relationship between containers/node.

The observation O is denoted by $F_i = \{f_1, f_2, ..., f_n\}$ to refer to the response time observations sequence (failures). An observed low response time might reflect workload fluctuation. This fluctuation in workload is associated with a probability that reflects the state transition status from OL to NL ($PF_{OL \to NL}$) at a failure rate $\Re$, which indicates the number of failures for a N, C or $cluster$ over a period of time.

We use the generalized Baum-Welch algorithm [11] to train the model by calculating the probabilities of the model parameters: (1) the horizontal transitions from a state to another. (2) probability that the O is started to be emitted for $state_i^d$ at t. $state_i^d$ refers to container, node, or cluster. (3) the O of $state_i^d$ are emitted and finished at t. (4) the probability that $state^{d-1}$ is entered at t before O_t to activate state $state_i^d$. (5) the forward and backward transition from bottom-up.

The output of algorithm is used to train Viterbi algorithm to find the anomalous hierarchy of the detected anomalous states. As shown in "(1)-(3)", we recursively calculate $\Im$ which is the ψ for a time set $(\bar{t} = \psi(t, t+k, C_i^d, C^{d-1}))$, where ψ is a state list, which is the index of the most probable production state to be activated by C^{d-1} before activating C_i^d. $\bar{t}$ is the time when C_i^d is activated by C^{d-1}. The δ is the likelihood of the most probable state sequence generating $(O_t, \cdots, O_{(t+k)})$ by a recursive activation. The τ is the transition time at which C_i^d is called by C^{d-1}. Once all the recursive transitions are finished and returned to $cluster$, we get the most probable hierarchies starting from $cluster$ to the production states at T period through scanning the state list ψ, the states likelihood δ, and transition time τ.

$$L = \max_{(1 \le r \le N_i^d)} \left\{ \delta(\bar{t}, t+k, N_r^{d+1}, N_i^d)\, a_{End}^{N_i^d} \right\} \quad (1)$$

$$\Im = \max_{(1 \le y \le N^{j-1})} \left\{ \delta(t, \bar{t}-1, N_i^d, N^{d-1}) a_{End}^{N^{d-1}}\, L \right\} \quad (2)$$

$$stSeq = \max_{cluster} \left\{ \delta(T, cluster), \tau(T, cluster), \psi(T, cluster) \right\} \quad (3)$$

Once we have trained the model, we compare the detected hierarchies against the observed one to identify the type of workload. The hierarchies with the lowest probabilities are considered anomaly. Once we detect and identify the workload type (e.g., OL), a hierarchy of faulty states (e.g., $cluster$, N_1^2, C_1^3 and C_2^3) that is affected by the anomalous component (C_1^3) is obtained that reflects observed anomalous behaviour. We repeat these steps until the probability of the model states become fixed. Each state is correlated with time that indicates: the time of it's activation, it's activated substates, and the time at which the control returns to the calling state. The result of the fault management model (anomalous components) is

stored in Knowledge storage. This aid us in the recovery procedure as the anomalous state is recovered as first come-first heal.

C. Fault-Failure Recovery Cases

Based on the fault type, we apply a recovery mechanism that considers the dependencies between components, and the current component status. The recovery mechanism is specified based on historic and current observations of a response time for a container or node and the hidden states (containers or nodes). The following steps and concerns are considered by the recovery mechanism:

- Analysis: relies on current and historic observation.
- Observation (failure): indicates the type of observed failure (e.g., low response time).
- Anomaly (fault): reflects the fault type (e.g., overload).
- Reason: explains the causes of the problem.
- Remedial Action: explains different solutions that can be applied to solve the problem.
- Requirements: constraint that might apply.

We look at two anomaly cases and suitable recovery strategies, which exemplify recovery strategies for the fault-failure mapping cases 1.3 and 2.1. These strategies can be applied based on the observed response time (current and historic observations) and related faults (hidden states).

1) Container neighbour overload (external container dependency) **Analysis**: current/historic observations, hidden states
Observation (failure): low response time at the anomalous container and the dependent one.
Anomaly: overload in one or more containers results in underload for another container at different node.
Reason: heavily loaded container with external dependent one (communication)
Remedial Actions:

Option 1: Separate the overloaded container and the external one depending on it from their nodes. Then, create a new node containing the separated containers considering the cluster capacity. Redirect other containers that in communication to these 2 containers in the new node. Connect current nodes with the new one and calculate the probability of the whole model to know the number of transitions (to avoid the occurrence of overload) and to predict the future behaviour.

Option 2: For the anomalous container, add a new one to the node that has the anomalous container to provide fair workload distribution among containers considering the node resource limits. Or, if the node does not yet reach the resource limits available, move the overloaded container to another node with free resource limits. At the end, update the node.

Option 3: create another (MM) node within the node with anomalous container behaviour. Next, direct the communication of current containers to (MM). We need to redetermine the probability of the whole model to redistribute the load between containers. Finally, update the cluster and the nodes.

Option 4: distribute load.

Option 5: rescale node.

Option 6: do nothing, this means that the observed failure relates to regular system maintenance or update happened to the system. Thus, no recovery option is applied.
Requirements: need to consider node capacity.

2) Node overload (self-dependency) **Analysis**: current and historic observations
Observation (failure): low response time at node level.
Anomaly: overloaded node.
Reason: limited node capacity.
Remedial Actions:

Option 1: distribute load.

Option 2: rescale node.

Option 3: do nothing.
Requirements: collect information regarding containers and nodes, consider node capacity and rescale node(s).

D. Recovery Model

The recovery model (Execute stage in MAPE-K) receives an ordered list of faulty states from the identification step. It applies a recovery mechanism considering the type of the identified anomaly and the resource capacity. We have configured the fault management model to have a specific number of nodes and containers because increasing the number of nodes and containers leads to a large amount of different recovery actions (Load balancing rules), which reduces model performance.

We are mainly concerned with two workload anomalies: (1) overload as it reflects anomalous behaviour, (2) underload category, as it is considered anomaly but it represents a solution to migrate load from heavy loaded component. We define different recovery actions for each fault-failure case. Consequently, for an identified anomaly case, we need to select the most appropriate action from the time and cost perspectives.

The Recover Job Scheduler (RJS) heals the identified anomaly based on first identified-first heal. It mitigates the anomalous state, by distributing the load to the underloaded components considering their status.

The recovery actions are stored in the Knowledge storage to keep track of the number of applied actions to the identified anomalous component. Before applying any of the recovery option, "Restart" option is applied to save the cost of trying multiple recovery options if the component does not reach its restart action number limit. In case a restart option does not enhance the situation, RJS checks the existence of underloaded component identified by the fault management model and stored in the knowledge storage. If there is underloaded component, the HMM is trained using the Forward-Backward algorithm to select the most probable action for the anomalous component as shown in Figure 10. The states A_i in the model refer to the hidden recovery actions. The $rejuvenation$ hidden state refers to the restart action, and $P_i(r)$, is the probability of

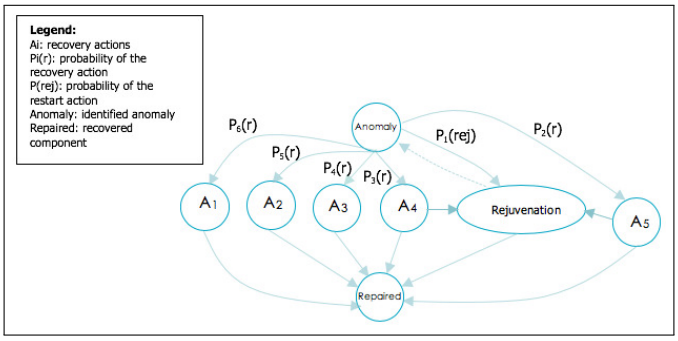

FIGURE 10. HMM FOR RECOVERY ACTIONS.

the recovery actions. We estimate $P_i(r)$ based on computing the maximum likelihood. The result of the HMM will be, for instance, the most probable action for anomalous state C_1^3 is 'distribute load'. The RJS apply the selected action to the fault component in the "Managed Component Pool". In case, RJS could not find underloaded components, the "pause" action is applied. If the number of applied recovery actions for the anomalous component exceeds a predefined threshold, a terminate action is applied after backing up the component. For each component, we further keep a profile of the type of applied action to enhance the recovery procedure in the future.

a) Metrics for Recovery Plan Determination: In order to better capture the accuracy of the proposed fault identification, we estimate the Fault Rate to capture (1) the number of faults during system execution $\Re(FN)$, and (2) the overall length of failure occurrence $\Re(FL)$ as depicted in "(4)" and "(5)". This aids us later in reducing the fault/failure occurrence through providing the best suited recovery mechanism, for instance for frequent or long-lasting failures. The observed behaviour is analysed in terms of failure rates for each state – e.g., low response times may result from overload states or normal load states – in order to determine the number of failures observed for each state and to estimate the total failure numbers for all the states. We define $\Re$ as follows:

$$\Re(FN) = \frac{No\ of\ Detected\ Faults}{Total\ No\ of\ Faults\ of\ Resource} \quad (4)$$

$$\Re(FL) = \frac{Total\ Time\ of\ Observed\ Failures}{Total\ Time\ of\ Execution\ of\ Resource} \quad (5)$$

The Average Failure Length (AFL), as in "(6)", might also be relevant to judge the relative urgency of recovery. Other relevant metrics that impact on the decision which strategy to use, but which we do not detail here, are resilience metrics addressing recovery times.

$$AFL = \frac{\sum Time\ of\ Failure\ Occurrence}{Number\ of\ Observed\ Failures} \quad (6)$$

VIII. EVALUATION

The proposed architecture runs on Kubernetes and Docker containers. We deploy the TPC-W[4] benchmark on the containers to validate the architecture. We focus on three types of faults the CPU hog, Network packet loss/latency, and performance anomaly caused by workload congestion.

A. Environment Set-Up

To evaluate the accuracy of the proposed architecture, the experiment environment consists of three VMs. Each VM is equipped with Linux OS, 3 VCPU, 2 GB VRAM, Xen 4.11 [5], and an agent. Agents are installed on each VM to collect the monitoring data from the system (e.g., host metrics, container, performance metrics, and workloads), and send them to the Real-Time/Historical storage to be processed by the Monitor. The VMs are connected through a 100 Mbps network. For each VM, we deploy two containers, and we run into them the TPC-W benchmark.

The TPC-W benchmark is used for resource provisioning, scalability, and capacity planning for e-commerce websites. The TPC-W emulates an online bookstore that consists of 3 tiers: client application, web server, and database. Each tier is installed on VM. We do not consider the database tier in the anomaly detection and identification, as a powerful VM should be dedicated to the database. The CPU and Memory utilization are gathered from the web server, while the Response time is measured from clients end. We ran the TPC-W for 300 minutes. The number of records that we obtain from the TPC-W is 2000 records.

We further use docker $stats$ command to obtain a live data stream for running containers. The SignalFX Smart Agent[6] monitoring tool is used and configured to observe the runtime performance of components and their resources. We also use the Heapster[7] to group the collected data, and store them in a time series database using the InfluxDB[8]. The gathered data from the monitoring tool, and from datasets are stored in the Real-Time/Historical Data storage to enhance the future anomaly detection and identification. The gathered dataset is classified into training and testing datasets 50% for each. The model training last 150 minutes.

To simulate real anomaly scenarios, script is written to inject different types of anomalies. The anomaly injection for each component last 5 minutes. The anomaly scenarios are: (1) CPU Hog, consumes all CPU cycles by employing infinite loops. (2) Memory Leak, exhausts the component memory. The stress[9] tool is used to create pressure on the CPU and Memory.

Furthermore, workload contention is generated to test the controller under different workloads. To generate workload,

[4]http://www.tpc.org/tpcw/
[5]https://xenproject.org/
[6]https://www.signalfx.com/
[7]https://github.com/kubernetes-retired/heapster
[8]https://www.influxdata.com/
[9]https://linux.die.net/man/1/stress

TABLE I. DETECTION EVALUATION.

Metrics	HHMM	DBN	HTM
RMSE	0.23	0.31	0.26
MAPE	0.14	0.27	0.16
CDA	96.12%	91.38%	94.64%
AD	0.94	0.84	0.91
FAR	0.27	0.46	0.31

TABLE II. ASSESSMENT OF IDENTIFICATION.

Metrics	HHMM	DBN	HTM
AI	0.94	0.84	0.94
CIA	94.73%	87.67%	93.94%
IIA	4.56%	12.33%	6.07%
FAR	0.12	0.26	0.17

the TPC-W web server is emulated using client application, which generates workload (using Remote Browser Emulator) by simulating a number of user requests that is increased iteratively. Since the workload is always described by the access behaviour, we consider the container is gradually loaded within [30-2000] emulated users requests, and the number of requests is changed periodically. To measure the number of requests and response (latency), the HTTPing[10] is installed on each node. Also, the AWS X-Ray[11] is used to trace of the request through the system.

B. The Detection Assessment

The detection model is evaluated by the Root Mean Square Error (RMSE), Mean Absolute Percentage Error (MAPE), and False Alarm Rate (FAR), which are the commonly used metrics [32] for evaluating the quality of detection. We further measure the Number of Correctly Detected Anomaly (CDA) and Accuracy of Detection (AD).

a) Root Mean Square Error (RMSE): It measures the differences between the detected value and the observed one by the model. A smaller RMSE value indicates a more effective detection scheme.

b) Mean Absolute Percentage Error (MAPE): It measures the detection accuracy of a model. Both the RMSE and MAPE are negatively-oriented scores, which means lower values are better.

c) Number of Correctly Detected Anomaly (CDA): It measures the percentage of the correctly detected anomalies to the total number of detected anomalies in a given dataset. High CDA indicates the model is correctly detected anomalous behaviour.

d) Accuracy of Detection (AD): It measures the completeness of the correctly detected anomalies to the total number of anomalies in a given dataset. Higher AD means that fewer anomaly cases are undetected.

e) False Alarm Rate (FAR): The number of the normal detected component, which has been misclassified as anomalous by the model.

The efficiency of the model is compared with the Dynamic Bayesian network (DBN), see Table I. The results show that the HHMM and HTM model detects anomalous behaviour with promised results comparing to the DBN.

[10]https://www.vanheusden.com/httping/

[11]https://aws.amazon.com/xray/

C. The Identification Assessment

The accuracy of the results is compared with the Dynamic Bayesian Network (DBN), and Hierarchical Temporal Memory (HTM), and it is evaluated based on different metrics such as: the Accuracy of Identification (AI), Number of Correctly Identified Anomaly (CIA), Number of Incorrectly Identified Anomaly (IIA), and FAR.

a) Accuracy of Identification (AI): It measures the completeness of the correctly identified anomalies to the total number of anomalies in a given dataset. Higher AI means that fewer anomaly cases are un-identified.

b) Number of Correctly Identified Anomaly (CIA): It is the number of correct identified anomaly (NCIA) out of the total set of identification, which is the number of correct Identification (NCIA) + the number of incorrect Identification (NICI)). The higher value indicates the model is correctly identified anomalous component.

$$CIA = \frac{NCIA}{NCIA + NICI} \tag{7}$$

c) Number of Incorrectly Identified Anomaly (IIA): The IIA is the number of the identified component, which represents an anomaly but misidentified as normal by the model. The lower value indicates that the model correctly identified anomaly component.

$$IIA = \frac{FN}{FN + TP} \tag{8}$$

d) False Alarm Rate (FAR): The number of the normal identified component, which has been misclassified as anomalous by the model.

$$FAR = \frac{FP}{TN + FP} \tag{9}$$

The false positive (FP) means the detection/identification of anomaly is incorrect as the model detects/identifies the normal behaviour as anomaly. True negative (TN) means the model can correctly detect and identify normal behaviour as normal.

As shown in Table II, the HHMM and HTM achieved promising results for the identification of anomaly. While the results of the DBN a little bit decayed for the CIA with approximately 7% than the HHMM, and 6% than the HTM. Both the HHMM and HTM show higher identification accuracy as they are able to identify temporal anomalies in the dataset. The result interferes that the HHMM is able to link the observed failure to its hidden workload.

TABLE III. RECOVERY EVALUATION.

Evaluation Metrics	*Results*
RA	99%
MTTR	60 seconds
OA	97%

D. The Recovery Assessment

To assess the recovery decisions of the model, we measure: (1) the Recovery Accuracy (RA) to be the number of successfully recovered anomalies to the total number of identified anomalies, (2) the Mean Time to Recovery (MTTR), the average time that the architecture takes to recover starting from the anomaly injection until recovering it. (3) The Overall Accuracy (OA) to be the number of correct recovered anomalies to the total number of anomalies. The results in Table III show that once the HMM model is configured properly, it can efficiently recover the anomalies with an accuracy of 99%.

IX. BEYOND PERFORMANCE: TRUST ANOMALIES

Traditionally, anomaly detection and analysis is addressing performance and resource management if applied to software systems management. Another wide area is security and trust. An open system has security vulnerabilities. Checking continuously for anomalies showing unusual behaviour that might indicate attacks or loss of information in some form is here also an important anomaly detection objective. Trust is here a related concern that covers security as well as performance and other technical factors.

Trust problems occur if different providers and consumers of services meet in a context without any prior trust relationship. A trust anomaly here is any situation in which the delivery of a previously guaranteed service (or its quality) is in doubt. An anomaly detection solution can help here to proactively invoke a remedial action or to record more information (in a tamper-proof way) to allow for later analysis and resolution of disputes.

So, we briefly discuss here the handling of trust regarding QoS compliance using a trust anomaly detection architecture. Since trust does not exist, it is important to capture and store relevant information in a tamper-proof way. The use of blockchains as a reaction to anomalies is here an option, if a consumer needs trustworthy documentation in failure cases, but blockchains maybe also always be used if a provider need assurance about having provided as planned/promised in contract. A blockchain is a distributed data store for digital transactions, resembling a ledger [76]. Blockchains have been used for various applications [50],[52],[53],[54],[55]. The blocks are linked and secured using cryptography. Each block typically contains a cryptographic hash of the previous block, a timestamp and transaction data. By design, a blockchain is inherently tamper-proof, i.e., resistant to modification of stored data.

In more concrete terms, an anomaly detection solution as we discuss here could, if QoS compliance is under threat, then switch on blockchains [45],[49]. This could act as remedial (support) action for later analysis and providing tamper-proof information for recovery.

We have not implemented this solution yet as part of our anomaly management solution, but we want to point out with this discussion that the solution presented is not limited to performance concerns and immediate remedial actions only.

X. CONCLUSIONS

Service management in virtualised, third-party environments has both benefits and limitations. Virtualisation allows resources assigned to application services to be adjusted dynamically to meet changing need. However, the reason for changes is often hidden from the service consumer. We present a controller architecture for the detection, and recovery of anomalies in hierarchically organised clustered computing environments. We pay specific attention to recent container cluster orchestration tools like Kubernetes or Docker Swarm that are now widely used to deploy software [30].

Our key objective here is to provide an analysis feature, that maps observable quality concerns onto hidden resources in a hierarchical clustered environment, and their operation in order to identify the reason for performance degradations and other anomalies [1],[68]. From this, a recovery strategy that removes the workload anomaly, thus removing the observed performance failure is the second step. We have proposed to use the Hidden Markov Models (HMMs) to reflect the hierarchical nature of the unobservable resources, and to support the detection, identification, and recovery of anomalous behaviours. We have further analysed mappings between observations and resource usage based on a clustered container scenario. The objective of this paper is to introduce the complete controller architecture with its key processing steps. Specifically, we try here to motivate the context of container-based deployment, illustrating different use case scenarios in more detail.

As part of our future, we will continue to complete the current controller implementation. Here, further simulations and experimental evaluations are planned [44],[75],[76].

With the focus on containers, we will also address practical concerns such as the relevance for microservice architectures as an architectural style [12]. Microservices are often linked to their container-based deployment. Here it would be worth investigating to which extend common microservice architectural patterns cloud influence the occurrence of anomalies in systems in which microservices are deployed as containers [13],[15],[64]. A stronger emphasis shall also be given to self-adaptive systems [14],[16],[66],[73].

Another direction is to link observed failures more into the context of the user. Here, adding semantics [71],[74] would help to link an observed anomaly to the processes and changing circumstances a user is involved in. We are thinking here of looking at educational technology systems [57],[58],[59],[61],[63],[60], where anomalies and failure are not just technical aspects, but might impact on cognitive processes as well.

References

[1] A. Samir and C. Pahl, "A Controller Architecture for Anomaly Detection, Root Cause Analysis and Self-Adaptation for Cluster Architectures," in *The Eleventh International Conference on Adaptive and Self-Adaptive Systems and Applications, ADAPTIVE*, 75–83. 2019.

[2] C. Pahl, P. Jamshidi, and O. Zimmermann, "Architectural principles for cloud software," in *ACM Transactions on Internet Technology (TOIT)*, 18 (2), 17. 2018.

[3] D. von Leon, L. Miori, J. Sanin, N. El Ioini, S. Helmer, and C. Pahl, "A Lightweight Container Middleware for Edge Cloud Architectures," in *Fog and Edge Computing: Principles and Paradigms*, 145-170. 2019.

[4] C. Pahl, I. Fronza, N. El Ioini, and H. Barzegar, "A Review of Architectural Principles and Patterns for Distributed Mobile Information Systems," in *14th Intl Conf on Web Information Systems and Technologies*. 2019.

[5] X. Chen, C.-D. Lu, and K. Pattabiraman, "Failure prediction of jobs in compute clouds: A google cluster case study," in *International Symposium on Software Reliability Engineering, ISSRE*, 167–177. 2014.

[6] G. C. Durelli, M. D. Santambrogio, D. Sciuto, and A. Bonarini, "On the design of autonomic techniques for runtime resource management in heterogeneous systems," Doctoral dissertation, Politecnico di Milano. 2016.

[7] T. Wang, J. Xu, W. Zhang, Z. Gu, and H. Zhong, "Self-adaptive cloud monitoring with online anomaly detection," in *Future Gen Comp Syst*, 80, 89-101. 2018.

[8] A. Samir and C. Pahl, "Anomaly Detection and Analysis for Clustered Cloud Computing Reliability," in *The Tenth International Conference on Cloud Computing, GRIDs, and Virtualization*, 110–119. 2019.

[9] P. Jamshidi, A. Sharifloo, C. Pahl, H. Arabnejad, A. Metzger, and G. Estrada, "Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures," in *12th Intl ACM SIGSOFT Conference on Quality of Software Architectures, QoSA*, 70–79. 2016.

[10] P. Jamshidi, A. Sharifloo, C. Pahl, A. Metzger, and G. Estrada, "Self-learning cloud controllers: Fuzzy q-learning for knowledge evolution," in *The International Conference on Cloud and Autonomic Computing*, September. 208-211. 2015.

[11] S. Fine, Y. Singer, and N. Tishby, "The hierarchical hidden markov model: analysis and applications," in *Machine Learning*, vol. 32, no. 1, 41–62, 1998.

[12] D. Taibi, V. Lenarduzzi, and C. Pahl, "Microservices Anti-Patterns: A Taxonomy," in *Microservices – Science and Engineering*, Springer. 2019.

[13] D. Taibi, V. Lenarduzzi, and C. Pahl, "Architecture Patterns for a Microservice Architectural Style," in *Communications in Comp and Inf Science*, Springer. 2019.

[14] H. Arabnejad, C. Pahl, G. Estrada, A. Samir, and F. Fowley, "A fuzzy load balancer for adaptive fault tolerance management in cloud platforms," in *European Conference on Service-Oriented and Cloud Computing*, September, 109-124. 2017.

[15] D. Taibi, V. Lenarduzzi, and C. Pahl, "Architectural Patterns for Microservices: A Systematic Mapping Study," in *Proceedings CLOSER Conference*, 221–232. 2018.

[16] H. Arabnejad, C. Pahl, P. Jamshidi, G. Estrada, "A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling," in *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. 2017.

[17] T. F. Düllmann, "Performance anomaly detection in microservice architectures under continuous change," Master, University of Stuttgart, 2016.

[18] C. Pahl, N. El Ioini, and S. Helmer, "A Decision Framework for Blockchain Platforms for IoT and Edge Computing," in *3rd International Conference on Internet of Things, Big Data and Security* , 105-113. 2018.

[19] A. Samir and C. Pahl, "Detecting and Predicting Anomalies for Edge Cluster Environments using Hidden Markov Models," in *The Fourth IEEE International Conference on Fog and Mobile Edge Computing*, 21–28. 2019.

[20] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation," in *IEEE Cloud Computing*, 4 (5), 22-32. 2017.

[21] M. Peiris, J. H. Hill, J. Thelin, S. Bykov, G. Kliot, and C. Konig, "PAD: Performance anomaly detection in multi-server distributed systems," in *International Conf on Cloud Computing, CLOUD*, June, 769–776. 2014.

[22] N. Sorkunlu, V. Chandola, and A. Patra, "Tracking system behaviour from resource usage data," in *International Conference on Cluster Computing*, 410–418. 2017.

[23] C. Pahl, "An ontology for software component matching," in *International Conference on Fundamental Approaches to Software Engineering*, 6–21. 2003.

[24] S. Maurya and K. Ahmad, "Load Balancing in Distributed System using Genetic Algorithm," in *Intl Journal of Engineering and Technology*, 5(2), 139–142. 2013.

[25] H. Sukhwani, "A survey of anomaly detection techniques and hidden markov model," in *International Journal of Computer Applications*, vol. 93, no. 18, 26–31. 2014.

[26] R. Heinrich, A. van Hoorn, H. Knoche, F. Li, L. E. Lwakatare, C. Pahl, S. Schulte, and J. Wettinger, "Performance engineering for microservices: research challenges and directions," in *ACM/SPEC International Conference on Performance Engineering Companion*, 223–226. 2017.

[27] N. Ge, S. Nakajima, and M. Pantel, "Online diagnosis of accidental faults for real-time embedded systems using a hidden Markov model," in *Simulation*, 91(19):851-868. 2016.

[28] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud container technologies: a state-of-the-art review," in *IEEE Transactions on Cloud Computing*. 2018.

[29] G. Brogi, "Real-time detection of advanced persistent threats using information flow tracking and hidden markov," Doctoral dissertation. 2018.

[30] D. von Leon, L. Miori, J. Sanin, N. El Ioini, S. Helmer,

and C. Pahl, "A performance exploration of architectural options for a middleware for decentralised lightweight edge cloud architectures," in *CLOSER Conference*. 2018.

[31] IEEE, "IEEE standard classification for software anomalies (IEEE 1044 - 2009)," 1–4. 2009.

[32] K. Markham, "Simple guide to confusion matrix terminology," 2014.

[33] B. Magableh and M. Almiani, "A Self Healing Microservices Architecture: A Case Study in Docker Swarm Cluster," in *International Conference on Advanced Information Networking and Applications*, 846–858. 2019.

[34] A. Khiat, "Cloud-RAIR: A Cloud Redundant Array of Independent Resources," in *The Tenth International Conference on Cloud Computing, GRIDs, and Virtualization CLOUD COMPUTING*, May, 133–137. 2019.

[35] M. Hasan, M. Milon Islam, I. Islam, and M. Hashem, "Attack and Anomaly Detection in IoT Sensors in IoT Sites Using Machine Learning Approaches," in *Internet of Things*, vol. 7, 1–14. 2019.

[36] A. Jindal, V. Podolskiy, and M. Gerndt, "Performance modelling for Cloud Microservice Applications," in *Intl Conf on Performance Engineering*, 25–32. 2019.

[37] P. Jamshidi, C. Pahl, and N. C. Mendonca, "Pattern-based multi-cloud architecture migration," *Software: Practice and Experience*, 47 (9), 1159-1184. 2017.

[38] P. Jamshidi, C. Pahl, N. C. Mendonca, J. Lewis, and S. Tilkov, "Microservices: The Journey So Far and Challenges Ahead," in *IEEE Software*, 35 (3), 24-35. 2018.

[39] C. Sauvanaud, M. Kaâniche, K. Kanoun, K. Lazri, and G. Da Silva Silvestre, "Anomaly detection and diagnosis for cloud services: Practical experiments and lessons learned," in *Journal of Syst and Softw*, 139, 84-106. 2018.

[40] P. Jamshidi, C. Pahl, and N. C. Mendonca, "Managing uncertainty in autonomic cloud elasticity controllers," in *IEEE Cloud Computing*, 50–60. 2016.

[41] C. Pahl and B. Lee, "Containers and clusters for edge cloud architectures - A technology review," in *IEEE International Conference on Future Internet of Things and Cloud*, 379–386. 2015.

[42] C. Pahl., N. El Ioini, S. Helmer, and B. Lee, "An architecture pattern for trusted orchestration in IoT edge clouds," in *The Third International Conference on Fog and Mobile Edge Computing, FMEC*, April, 63–70. 2018.

[43] N. Kratzke, "About Microservices, Containers and their Underestimated Impact on Network Performance," in *CoRR*, vol. abs/1710.0. 2017.

[44] F. Ghirardini, A. Samir, I. Fronza, and C. Pahl, "Performance Engineering for Cloud Cluster Architectures using Model-Driven Simulation," in *ESOCC Workshops - CloudWays'2018*. 2019.

[45] C. Pahl, N. El Ioini, S. Helmer, and B. Lee, "A Semantic Pattern for Trusted Orchestration in IoT Edge Clouds," in *Internet Technology Letters*. 2019.

[46] R. Scolati, I. Fronza, N. El Ioini, A. Samir, and C. Pahl, "A Containerized Big Data Streaming Architecture for Edge Cloud Computing on Clustered Single-Board Devices," in *10th International Conference on Cloud Computing and Services Science*, 68-80. 2019.

[47] A. Wert, "Performance problem diagnostics by systematic experimentation," PhD, KIT. 2015.

[48] Q. Guan, C. C. Chiu, and S. Fu, "CDA: A cloud dependability analysis framework for characterizing system dependability in cloud computing infrastructures," in *Proceedings of IEEE Pacific Rim International Symposium on Dependable Computing, PRDC*, vol. 18, 11–20. 2012.

[49] N. El Ioini and C. Pahl, "Trustworthy Orchestration of Container Based Edge Computing Using Permissioned Blockchain," in *The Fifth International Conference on Internet of Things: Systems, Management and Security, IoTSMS*, October, 147-154, IEEE Press. 2018.

[50] G. D'Atri, V. T. Le, C. Pahl, and N. El Ioini, "Towards Trustworthy Financial Reports Using Blockchain," in *Proceedings Tenth International Conference on Cloud Computing, GRIDs, and Virtualization*. 2019.

[51] N. El Ioini and C. Pahl, "A Review of Distributed Ledger Technologies," in *On the Move to Meaningful Internet Systems. OTM 2018 Conferences*, 227-288. 2018.

[52] C. A. Ardagna, R. Asal, E. Damiani, N. El Ioini, and C. Pahl, "Trustworthy IoT: An Evidence Collection Approach Based on Smart Contracts," in *2019 IEEE International Conference on Services Computing (SCC)*, 46–50. 2019.

[53] V. T. Le, C. Pahl, and N. El Ioini, "Blockchain Based Service Continuity in Mobile Edge Computing," in *6th International Conference on Internet of Things: Systems, Management and Security*, 2019.

[54] C. A. Ardagna, R. Asal, E. Damiani, T. Dimitrakos, N. El Ioini, and C. Pahl, "Certification-based cloud adaptation," in *IEEE Transactions on Services Computing*. 2018.

[55] S. Helmer, M. Roggia, N. El Ioini, and C. Pahl, "EthernityDB - Integrating Database Functionality into a Blockchain," in *European Conference on Advances in Databases and Information Systems*, 37–44. 2019.

[56] S. Helmer, C. Pahl, J. Sanin, L. Miori, S. Brocanelli, F. Cardano, D. Gadler, D. Morandini, A. Piccoli, S. Salam, A. M. Sharear, A. Ventura, P. Abrahamsson, and D. T. Oyetoyan, "Bringing the cloud to rural and remote areas via cloudlets," in *Proceedings of the 7th Annual Symposium on Computing for Development*, 14. 2016.

[57] C. Pahl, "Layered ontological modelling for web service-oriented model-driven architecture," in *Europ Conf on Model Driven Architecture – Foundations and Applications*. 2005.

[58] S. Murray, J. Ryan, and C. Pahl, "A tool-mediated cognitive apprenticeship approach for a computer engineering course," in *Proceedings 3rd IEEE International Conference on Advanced Technologies*, 2-6. 2003.

[59] C. Pahl, R. Barrett, and C. Kenny, "Supporting active database learning and training through interactive multimedia," in *ACM SIGCSE Bulletin* 36 (3), 27-31. 2004.

[60] C. Kenny and C. Pahl, "Automated tutoring for a database

skills training environment," in *ACM SIGCSE Symposium 2005*, 58-64. 2003.

[61] X. Lei, C. Pahl, and D. Donnellan, "An evaluation technique for content interaction in web-based teaching and learning environments," in *3rd IEEE International Conference on Advanced Technologies*, 294-295. 2003.

[62] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam, "Cuanta: Quantifying Effects of Shared On-chip Resource Interference for Consolidated Virtual Machines," in *ACM Symposium on Cloud Computing*, 1–14. 2011.

[63] M. Melia and C. Pahl, "Constraint-based validation of adaptive e-learning courseware," in *IEEE Transactions on Learning Technologies* 2(1), 37-49. 2009.

[64] D. Taibi, V. Lenarduzzi, C. Pahl, and A. Janes, "Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages," in *XP2017 Scientific Workshops*, 2017.

[65] L. Mariani, C. Monni, M. Pezze, O. Riganelli, and R. Xin, "Localizing Faults in Cloud Systems," in *11th International Conference on Software Testing, Verification and Validation*, 262–273. 2018.

[66] N. C. Mendonca, P. Jamshidi, D. Garlan, and C. Pahl, "Developing Self-Adaptive Microservice Systems: Challenges and Directions," in *IEEE Software*. 2020.

[67] Y. Tan, H. Nguyen, Z. Shen, and X. Gu, "PREPARE: Predictive Performance Anomaly Prevention for Virtualized Cloud Systems," in *IEEE International Conference on Distributed Computing Systems*, 285–294. 2012.

[68] A. Samir and C. Pahl, "Anomaly Detection and Analysis for Clustered Cloud Computing Reliability," in *International Conference on Cloud Computing, GRIDs, and Virtualization*. 2019.

[69] T. Zwietasch, "Online Failure Prediction for Microservice Architectures," Master Thesis, U Stuttgart. 2017.

[70] A. Samir and C. Pahl, "A Controller Architecture for Anomaly Detection, Root Cause Analysis and Self-Adaptation for Cluster Architectures," in *Intl Conf on Adaptive and Self-Adaptive Systems and Applications*. 2019.

[71] M. Javed, Y. M. Abgaz, and C. Pahl, "Ontology change management and identification of change patterns," in *Journal on Data Semantics* 2(2-3), 119-143. 2013.

[72] O. Ibidunmoye, T. Metsch, and E. Elmroth, "Real-time detection of performance anomalies for cloud services," in *IEEE/ACM 24th International Symposium on Quality of Service*. 2016.

[73] P. Jamshidi, C. Pahl, S. Chinenyeze, and X. Liu, "Cloud migration patterns: a multi-cloud service architecture perspective," in *Service-Oriented Computing ICSOC2014 Workshops*, 6-19. 2015.

[74] D. Fang, X. Liu, I. Romdhani, P. Jamshidi, and C. Pahl, "An agility-oriented and fuzziness-embedded semantic model for collaborative cloud service search, retrieval and recommendation," in *Future Generation Computer Systems*, 56, 11-26. 2016.

[75] F. Ghirardini, A. Samir, I. Fronza, and C. Pahl, "Model-Driven Simulation for Performance Engineering of Kubernetes-style Cloud Cluster Architectures," in *ES-OCC 2018 Workshops, PhD Symposium, EU-Projects*, 2019.

[76] N. El Ioini, C. Pahl, and S. Helmer, "A decision framework for blockchain platforms for IoT and edge computing," in *Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security*. 2018.

[77] J. Ehlers, A. van Hoorn, J. Waller, and W. Hasselbring, "Self-adaptive software system monitoring for performance anomaly localization," in *IEEE International Conference on Autonomic Computing*, 197–200. 2011.

[78] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: Managing performance interference effects for QoS-aware clouds," in *5th European Conference on Computer Systems*, 237–250, 2010.