Self-Adaptive Healing for Containerized Cluster Architectures with Hidden Markov Models

Areeg Samir and Claus Pahl
Faculty of Computer Science
Free University of Bozen-Bolzano
Bolzano, Italy
Email: {areegsamir,Claus.Pahl}@unibz.it

***Abstract*—Edge cloud environments are often build as virtualized coordinated clusters of possibly heterogeneous devices. In the emerging of cluster platforms like Kubernetes or Docker Swarm, scalability is based on resource utilization. Resource utilization has been used for capacity planning and for forecasting resource demand. However, due to the large scale and complex structure of these architectures, analyzing large amount of monitoring data may cause a huge resource overhead that affects the performance of anomaly detection and the accuracy of anomaly location. To address such challenges, we propose a self-adaptive healing approach that detects, identifies, predicts and recovers anomalies in clustered architectures. The approach will be evaluated to assess the accuracy of the mechanism.**

Keywords*—Cloud, Anomaly detection, Anomaly identification, Prediction, Anomaly recovery, Fault, Containers, Cluster, HMMs, Performance monitoring.

I. Research Problem

Edge cloud computing requires applications to be deployed on distributed virtualized clusters of nodes that are possibly heterogeneous in nature. We target cluster architectures for edge cloud computing where applications are deployed in the form of lightweight containers on these possible mobile virtualized nodes [6], for instance using Docker Swarm or Kubernetes for cluster orchestration [1], [2] in a microservices style [3]. Here, not necessarily all infrastructure information is accessible and only application-level observations are possible.

In this context, resources that run deployed containers may cause performance anomalies that may lead to system performance degradation. Such performance degradation arises when a resource behaviour deviates from its expectation to cause something called anomaly or fault [4]. Typical anomalies in an edge cloud setting are response time degradations caused by overloaded low-capacity devices. Another typical anomaly situation arises if connected sensors produce irregular amounts of data, which in high volume cases cause spike demands for data processing and storage. Thus, anomaly detection can help in capturing unusual patterns, which do not conform to expected patterns, and anomaly identification can aid in locating the root cause of an anomaly. However, linking observations to underlying hidden infrastructure in an edge environment has not been sufficiently addressed.

Moreover, there are many metrics to monitor in a containerized cluster environment [5]. To determine the kind of proper metrics to be monitored is considered a difficult task. In practice, containerized cluster monitoring tools provide static rule-based auto-scaling approaches, but are not flexible enough to adjust to dynamic root cause mappings during runtime. For example, violating the response time threshold may be a consequence of an edge node that is overloaded due to an increasing workload [7].

Nevertheless, 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 [8], [9], [12].

In edge cloud computing, the workload can significantly change depending on various events in the runtime environment. Thus, offering desirable performance provided by edge cloud computing architecture has been the goal of several studies. In the literature, different techniques are proposed to mitigate those challenges. In [10] the author detected faults in real-time embedded systems using HMM through describing the healthy and faulty states of a system's hardware components. In [11], HMM is used to find which anomaly is part of the same anomaly injection scenarios. In [13], a workload algorithm in a hierarchical edge cloud network is provided. The algorithm allocates the computing resources to minimize the response time for all overloaded tasks. Jayapandian et al. [16] proposed clustering based task deployment approach for load balancing using the Bayes Theorem. However, the approach is mainly designed for data centers based cloud to optimize the computation delay of workload execution.

In contrast, our proposed approach analyzed the relations between the observed anomalous behaviours, and correlated them with their hidden container and node. The approach defined a set of recovery actions to be applied to the identified anomaly case to achieve fair workload among system levels. Our contribution is to detect, identify and heal anomalous behaviour of containers and nodes in a containerized cluster edge environment. Based on the observed response time and resource utilization variations of containers and nodes, an analysis is performed to correlate the variation observed to its hidden anomalous workload changes.

II. Challenges

Providing efficient anomaly detection, identification and recovery for containerized cluster environment in edge cloud computing face many challenges [17], [7], [18] such as: (1) The existence of anomaly in previous knowledge makes it difficult to detect and diagnose the root cause of the observed anomaly. Thus, uncovering performance problems require knowledge about different types of anomalies and their characteristics to be able to detect and identify anomaly, and suggest accurate recovery. (2) Addressing the relationships between different types of anomalies, and possible root causes in dynamic environment is considered a complex task as single anomaly often produces multiple symptoms at different system levels. (3) Certain workload and usage situations may miss lead the detection as not all the load fluctuations refer to anomalous behaviour. Thus, the success of measurement highly depends on understanding system behaviours under different workloads. (4) Another problem is performance testing as there are many measurements at different system levels, which make the selection of an accurate measurements a difficult task. (5) To identify the root cause of an anomaly, detailed information about the performance of corresponding system is required. This can be achieved either by conducting few performance tests, while capturing detailed performance data, or executing many performance tests. However, the former case results in a high measurement overhead impairing measurement accuracy, the last case is time consuming, and thus impractical.

To address the aforementioned challenge, and to improve the efficiency of edge resource utilization, our contribution is to address two questions: (1) "How to accurately detect and identify the anomalous behavior in a containerized cluster environment?". (2) "How can we enhance the autonomous behaviors of self-adaptive and healing systems whilst minimizing the operating costs of large-scale computing environments?".

III. The Approach to the Problem

We focus on analyzing performance anomalies within containers and nodes. We defined an anomaly as a deviation in a container or node workload from its normal behaviour. We defined different cases of anomaly based on observing our system behaviour and focused on the common anomaly cases reported in the literature [17], [19]. Each case represents the observed response time at container and node level. We related each case with the resource utilization of containers and nodes. Fig. 1 shows the system under observation with fully connected nodes and containers. At the container level, we may observe low response time due to a container entered into self-load loop because of it is heavily loaded, or because of the existence of communication with another overloaded container within the same node (internal dependency), or in another node (external dependency). At the node level, a node may show low response time because of infrastructure issues (e.g., service down), hardware issues (e.g., CPU, Memory or Disk down), kernel issues (e.g., kernel deadlock, corrupted file system), container runtime issues (e.g., unresponsive runtime), file configuration issues, or change in cluster management tool version. To simplify the anomaly identification, we focus on hardware and runtime issues due to overloaded resources. We assumed two anomaly cases at node level: node self-dependency, which happens when a node has limited resources. Here, a node enters into self load due to its limited capacity. This causes an overload at the node level and at the container level as well. A node may also show low response time if it communicates with another overloaded node at the same cluster. This result on low response time observed at the node level, which slow down the whole operation of the system.

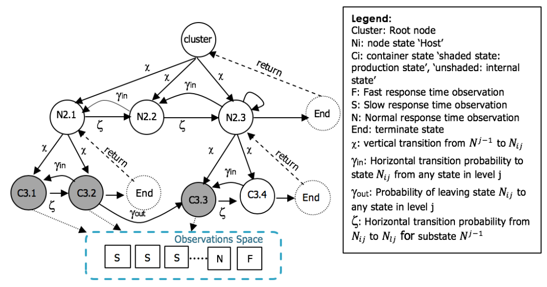
cluster
N2.1
N2.2
N2.3
End
C3.1
C3.2
C3.3
C3.4
return
χ
γin
γout
ζ
Observations Space
S
N
F
Legend:
Cluster: Root node
Ni: node state 'Host'
Ci: container state 'shaded state: production state', 'unshaded: internal state'
F: Fast response time observation
S: Slow response time observation
N: Normal response time observation
End: terminate state
χ: vertical transition from N^{j-1} to N_{ij}
γin: Horizontal transition probability to state N_{ij} from any state in level j
γout: Probability of leaving state N_{ij} to any state in level j
ζ: Horizontal transition probability from N_{ij} to N_{ij} for substate N^{j-1}

Figure 1. Nodes-Container Dependency

Accordingly, to detect deviations from expected behaviour (e.g., response time variations), to track the reason behind the workload, and to find the dependency between the affected nodes and their loads in a cluster, we differentiated between two types of observation in relation to workload and response time fluctuations: (1) System states (anomaly) that refer to anomalous or faulty behavior, which is hidden from the consumer. This indicates that the behavior of a system resource is significantly different from normal behavior. An anomaly in our case may point to an undesirable behavior of a resource such as overload, or to a desirable behavior like underload of a system resource, which can be used as a solution to reduce the load at overloaded resources. (2) Emission or Observation (observed failure from these states), which indicates the occurrence of failure resulting from a hidden state.

Hence, to achieve our objective, we utilized Hidden Markov Models (HMMs) [20] as HMMs depict significant outcome in terms of false alarm rate, anomaly detection, identification, and recovery [21], [11], [10]. A HMM [20] is a probabilistic model that is represented by distribution of probabilities for all the possible results. HMMs can specifically address the correct mapping between observable and hidden properties here.

From an implementation perspective, controllers following a MAPE-K architecture are often used in cloud and edge environments to manage elasticity, reliability or resource consumption [14], [15]. Our anomaly detection and management approach is meant to be seen as an integral component of these controllers.

A. Self-Adaptive Fault Management Model

The fault management model consists of detection and identification as shown in Fig. 2. Each phase is organised according to the MAPE-K control loop. The detection phase (Analysis) detects faults by collecting the monitored data from metrics and log files about the system container, nodes, and their resources. The monitoring is done through implementing a software agent that does not require prior knowledge about the containers or nodes. The detection phase is performed by using Hidden Markov Models (HMMs) that are trained on the response time to detect and predict the anomalous workload within resources at containers and nodes for each. After training, the observed data is compared with the predicted based on the data stored in ”Real-Time Historical Data” storage. If there is no detected anomaly, then the approach will declare the status of the system ”Anomaly Free”, and it will loop back to the (Monitor) for obtaining more observations. In the presence of an anomaly, the approach will verify if the detected performance variation at container/node level is a symptom of anomaly, the spearman’s rank correlation coefficient is measured to estimate the dissociation between container/node response time, and the collected parameters from the measured metrics. The value of correlation is compared with the parameters estimated by HMM to verify the accuracy of detection. We further estimated the correlation between number of user-transactions processed in a period at container level, and the collected parameters from the measured metrics to verify if such performance variation at container/node levels is associated with the number of user transactions. Once the anomaly is accurately detected, the approach will move to the identification phase.

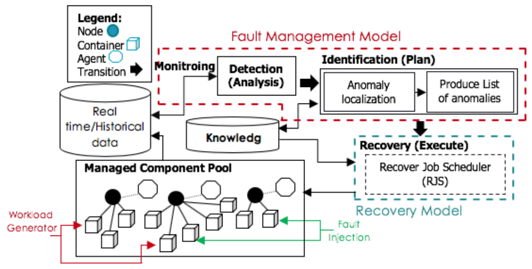

Figure 2. The Proposed Architecture

The identification phase (Plan) tracks the causes of anomaly through observing variations in response time of containers, nodes, or application. It establishes relationships between the observed variations, and the hidden workload of containers or nodes. The phase employs Hierarchical HMMs [22] to show the anomaly at different system levels as shown in Fig. 1. The model is trained on the same assumptions as the detection phase. Once the type of anomaly is identified, we compare the sequences similarity obtained by the detection and identification phases through utilizing Euclidean distance. Then we use the first-depth search algorithm to list the states by their activation time. Each state has time that indicates: the time of it’s activation, it’s last activated substate, probability, and the time at which the control returns to the calling state. Based on the algorithm, the state that is firstly emits low response time, and has high probability is mainly the one that causes overload at the node/container level. At the end, an order list of the anomalous components is produced to be healed based on the type of the identified anomaly, and to aid the recovery model determining how many containers or nodes will need to be recovered.

In both phases, we mapped the containers and nodes into states. The edges that connect nodes and containers represent the communication and thus dependencies between them. The observations from the models represent the response time variations that are emitted from the hidden (i.e., non-observable) states. Baum-Welch and Viterbi algorithms are employed to train and compute the models parameters iteratively. We designed further an algorithm to aid in identifying the case of anomaly at the container and node levels. The algorithm aims also to produce another list of the underloaded containers and nodes.

B. Self-Healing Recovery Model

Once an anomaly is identified, the recovery model (Execute) is activated. A recovery action is selected from a set of identified recovery actions stored in the knowledge storage, and applied to the managed component pool. This model consists of:

a) Recover job scheduler (RJS): A RJS checks the anomaly type, and path of the identified anomalous state by iterating through the produced anomaly list. Then it applies a suitable recovery action to achieve fair workload distribution. Once the action is selected and applied to the managed component pool, the RJS removes the recovered state from the list, stores it in the recovered list with ’Anomaly Free’ flag, and updates the rules to enhance the future prediction of the model. If the recovery process doesn’t success, another action will be applied based on the identified case, and the number of applied recovery actions. The number of applied recovery actions is stored in the knowledge storage to check the limit of applied actions for each container and node. If the recovery process fail, the RJS will apply another recovery action, and a feedback on its implementation will be received. Once the recovery succeeded, the applied action will get a higher recovery priority to indicate the success of the action for the recovered container or node. The result of that step is directly reflected in the pool to repeat the cycle of anomaly analysis all over again.

b) Check Rules: Each resource assigned to container and node has a defined limit. Thus, we define rules that should be checked for each container and node before applying the recovery action: Scope of recovery (container or node level), Node/Container size (available capacity), Dependency (relation between containers, nodes or among each other), the number of recovery actions applied to the anomalous component, Utilization of Resources(CPU, Memory), the number of available running nodes and containers, check the anomaly and normal lists stored in the knowledge store.

c) Recovery Actions: We identify a set of actions to be applied at the container or node level. These recovery actions are stored in the knowledge storage. Following represent the main actions:

Create: create a new node (allocate a new node through the cloud provider), or container (launch a container on node and allocating resources). For this option, a new container or node is created to mitigate the overload that occurred due to insufficient resources (CPU, Memory, etc) in container or node.

Migrate: move a container to another existed node, or move a node to another cluster. Here we have to consider the current status, and capacity of the acquired node and cluster before applying the move action. Usually, we can’t move a container from one node to another, we move only a container image onto another node, and then start a new container. One challenge that we can face here, is this doesn’t preserve data that is stored inside volumes as the data volumes need to be moved manually to a new node.

Distribute Load: distribute the load to another underloaded container or node. By pushing the jobs from the overloaded container(s) or node to the underloaded ones. At this option, the Recover Job Scheduler (RJS) checks both the anomaly list and normal list stored in the knowledge. The RJS chooses the anomalous component based on its order in the list. Then it searches in the normal list for underloaded components that has the same scope of the anomalous component. Also it checks the capacity of the underloaded components to avoid overloading the component. If there isn't underloaded component at the same scope of the anomalous one, another scope will be selected.

Pause: node stop accepting new workload while its remaining tasks keep executing. Once the node returns to its normal load, it is resumed again.

Split: separate a container to a new created node in the same cluster. At such situation, the container size, node size, cluster capacity, dependency, and resource utilization should be checked to apply the action.

Terminate: it ends the life of container or node to mitigate resource starvation, or if the containers or nodes are stopping. When a container or node is terminated, any changes to its status that are not stored in persistent storage disappear. Thus, before terminating a component life, a backup is applied.

Once the action is applied, we keep information in the knowledge store about: (1) the applied action to know which action is applied to which container or node. (2) flag the recovered component to check later the statue of recovery. (3) recorded number of recoveries (NR): each applied recovery action should be stored to check the limit of the recovery actions applied to the affected container or node. Further, to save the cost and time of applying multiple recovery actions, before applying any of the actions, a ”Restart” action will be applied. If the ”Restart” action does not enhance the situation, and the NR exceeds the specified limit, the container or node will be backed up and terminated.

d) The Recovery Model: To react to anomalous behaviour in workload, fair workload distribution should be adopted to improve the application QoS. For example, if edge resources allocated to process incoming requests are overloaded, and the node is not able to improve the application QoS due to it has a limited capacity, then additional arrived requests will be sent to the other underloaded nodes.

In this context, we have configured the models in the fault management model to have a specific number of nodes and containers because increasing the number of nodes and containers will lead to a large amount of different recovery options, which reduces model performance. We defined different recovery actions for each case. Consequently, for an identified anomaly case, we select the most appropriate action from the time and cost perspectives.

We propose a HMM Congestion Game to map the recovery actions (resources) into a set of states considering that the states can not be directly observed, but instead they can be estimated from the observations of the identified anomaly case (players). The payoff of each player depends on the resources it chooses and the number of players choosing the same resource. We consider that each player is associated with a number of resources. We assume that the transitions between the states and the observations are fixed to reduce the number of the applied actions and to save the time. We are mainly concerned with two 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 containers or nodes. We train a HMM, and choose the observable state with the greatest probability. We used the Forward algorithm and maximum likelihood to select the most probable action from every iteration, and order them based on their probability. To determine the precedence of the selected actions, the action with the highest probability will be applied first. A conditional rule if (anomaly == ¡Fault-Failure Case¿) is utilized to load the recovery actions for the identified anomaly case.

IV. Expected Results

The implementation of the proposed approach is being tested with TPC-W benchmark. TPC-W emulates an online bookstore that consists of 3 tiers: client application, web server, and database. The experiment environment consists of 3 nodes (VMs). Each node is equipped with different containers. Each tier is installed on VM. We assume that the database does not have anomaly, and we do not include it. The web server is emulated using client application, which simulates a number of user requests that is increased iteratively. To obtain historical data, TPC-W is run for 200 minutes. The model training lasts 100 minutes on a standard PC with results varying according to the number of states. The rest 50% is used to test the model. To simulate real anomalies of the system, scripts are written to inject different types of faults (CPU hog and Memory leak) into the target system. The fault injection varies slightly depending on the anomaly type, but all last between 4 and 8 minutes. The workload is generated by the TPC-W client application. We create a workload that runs for 50 min. In this workload the number of emulated-users, and duration of each period changes periodically.

A. The Detection Assessment

The detection model is evaluated by Root Mean Square Error (RMSE), Mean Absolute Percentage Error (MAPE), R^2, Precision, and Recall, which are the commonly used metrics for evaluating the quality of detection and prediction. The efficiency of the model is compared with a Dynamic Bayesian network (DBN), Random Forest (RF), and Support Vector Machine (SVM). The results in Table I show that both HMM, DBN, and RF achieved good results comparing to SVM. This may return to RF required many features due to RF descending from decision tree. The RF built multiple tress in which each node represented a certain feature while HMM required fewer less features to construct the model. The SVM achieved a dramatic performance. The reason behind the degradation of SVM performance returns to the SVM doesn't perform well on dataset with multiple classes. The results show that the HMM model detects anomalous behaviour with promised results for precision, MAPE, recall and RMSE comparing to other algorithms.

B. The Identification Assessment

The accuracy of identification is compared with DBN, and Hierarchical Temporal Memory (HTM), and it is evaluated based on different metrics:

TABLE I. DETECTION EVALUATION.

Metrics	HMM	DBN	SVM	RF
RMSE	0.144	0.148	0.157	0.163
MAPE	0.162	0.170	0.181	0.177
R^2	0.890	0.881	0.829	0.874
Precision	98%	96%	92%	95%
Recall	97.34%	94.95%	95.63%	95.99%

Accuracy of Identification (AI): 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. *Number of Correctly Identified Anomaly (CIA)*: measures 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. *Number of Incorrectly Identified Anomaly (IIA)*: measures the number of identified component, which represents an anomaly but misidentified as normal by the model. The lower value indicates that the model correctly identified anomaly component. *False Alarm Rate (FAR)*: measures the number of normal identified component, which has been misclassified as anomalous by the model.

As shown in Table II, 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 HHMM, and 6% than HTM. Both HHMM and HTM showed 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 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

C. 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) Mean Time to Recovery (MTTR), the average time that the approach takes to recover starting from the anomaly injection until recovering it. (3) 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 HMM model is configured properly, it can efficiently recover the detected anomalies with an accuracy of 99%.

TABLE III. RECOVERY EVALUATION.

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

V. DISCUSSION AND CONCLUSIONS

Edge computing architecture is designed with the aim of achieving high QoS operation. Thus, different aspects such as workload fluctuations and response time variations need to be taken into account when designing suitable edge architecture.

The workload can significantly change depending on various events in the runtime environment. This requires new mechanisms to be used at the edge of network. Thus, we modelled our system hierarchically to locate and recover anomalous behaviour across different system levels. We proposed HMMs to provide a self-adaptive healing approach for containerized cluster environments. We introduced different fault-failure cases at different system levels and provide a set of recovery actions for each anomaly case. The concepts of HMMs is accompanied with our algorithms to detect and identify anomalous behaviour that is emitted from container and nodes. Then, a recovery mechanism is applied to adaptively distribute workloads among different containers or nodes of cluster(s). The workload is fairly distributed based on the status of computational capacity that is assigned to nodes and cluster to minimize the workload in the edge architecture.

We conducted performance evaluation over small-scale computing clusters. We injected different anomalies, and we generated various workloads to measure the effects on the edge nodes at different system settings. This allows us to find the relationships among the system components at different levels, and design adaptive and healing approach to adjust the overall system goal. To this end, the results demonstrated that the proposed approach can detect and identify anomalous behaviour with accuracy 98% and 94% respectively. The approach recovered the identified anomalies with accuracy reached 99%.

Currently, we are working on providing more detailed experiments to fully confirm these conclusions. Furthermore, machine learning approaches [23] shall also be considered.

REFERENCES

[1] C. Mahmoudi, F. Mourlin, and A. Battou, "Formal definition of edge computing: An emphasis on mobile cloud and IoT composition," *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*, pp. 34–42, 2018.

[2] 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," IoTBDS, pp. 73–84, 2018.

[3] P. Jamshidi, C. Pahl, N.C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead, " *IEEE Software* 35 (3), 24-35, 2018.

[4] ISDW Group, "IEEE Standard Classification for Software Anomalies," pp. 1–23, 2010.

[5] H. Lawrence, G. H. Dinh, A. Williams, and B. Ball, "Container Monitoring and Management," 2018.

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

[7] S. Taherizadeh, V. Stankovski, and M. Grobelnik, "A capillary computing architecture for dynamic internet of things: Orchestration of microservices from edge devices to fog and cloud providers," *Sensors (Switzerland)*, vol. 18, no. 9, 2018.

[8] A. Samir and C. Pahl, "Anomaly Detection and Analysis for Clustered Cloud Computing Reliability," in *Intl Conf on Cloud Computing, Grids, and Virtualization*, 2019.

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

[10] N. Ge, S. Nakajima, and M. Pantel, "Online diagnosis of accidental faults for real-time embedded systems using a hidden markov model," *Simulation*, vol. 91, no. 19, pp. 851–868, 2015.

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

[12] A. Samir and C. Pahl, "Detecting and Predicting Anomalies for Edge Cluster Environments using Hidden Markov Models,", *Intl Conf Fog & Mobile Edge Comp*, 2019.

[13] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Proceedings - IEEE INFOCOM*, vol. 2016-July, 2016.

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

[15] P. Jamshidi, A. Sharifloo, C. Pahl, H. Arabnejad, A. Metzger, G. Estrada, "Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures," *Intl Conf on Quality of Software Architectures*, 2016.

[16] N. Jayapandian, R. Menagadevi, S. Abinaya, O. S. Sampoorani, and A. M. J. Z. Rahman, "Dynamic Load Balancing Cluster and Fault-Tolerant In Cloud Environment," May, 2017.

[17] N. G. Bachiega, P. S. Souza, S. M. Bruschi, and S. D. R. De Souza, "Container-based performance evaluation: A survey and challenges," in *Proceedings - 2018 IEEE International Conference on Cloud Engineering, IC2E 2018*. IEEE, apr 2018, pp. 398–403.

[18] C. Pahl and B. Lee, "Containers and Clusters for Edge Cloud Architectures – A Technology Review," *2015 3rd International Conference on Future Internet of Things and Cloud*, pp. 379–386, 2015.

[19] A. La Rosa, "Docker monitoring: challenges introduced by containers and microservices," 2018.

[20] D. Jurafsky and J. H. Martin, *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*, 2nd ed. Prentice Hall, 2008.

[21] A. Y. Nikravesh, S. A. Ajila, and C.-H. Lung, "Cloud Resource Auto-scaling System Based on Hidden Markov Model (HMM)," *2014 IEEE International Conference on Semantic Computing*, no. Section III, pp. 124–127, 2014.

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

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