Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Deep Q Learning for Self Adaptive Distributed Microservices Architecture

BASEL MAGABLEH[1], MUDER ALMIANI[2], AND LUCA LONGO[3]
[1]Technological University, Dublin, Ireland (e-mail: basel.magableh@dit.ie)
[2]Al-Hussein Bin Tala University, Ma'an, Jordan (e-mail: malmiani@my.bridgeport.edu)
[3]Technological University, Dublin, Ireland (e-mail: luca.longo@dit.ie)

Corresponding author: Magableh B. (e-mail: basel.magableh@dit.ie).

ABSTRACT One desired aspect of a self-healing Microservices architecture is the ability to continuously monitor the operational environment, detect and observe anomalous behaviour, and provide a reasonable policy for self-scaling, self-healing, and self-tuning the computational resources to adapt a sudden changes in its operational environment dynamically at rune-time. The behaviour of Microservices architecture is continuously changing overtime, which makes it a challenging task to use a statistical model to identify normal and abnormal behaviour of the running services. The performance of the cluster nodes could fluctuate around the demand to accommodate scalability, orchestration and load balancing demands. To achieve highly level of self adaptability, this research implements Microservices architectures a MAPE-K model that employ Markov Decision Process (MDP) in identifying the transition model from one cluster state to another. Also, it employees a Deep Q-learning network (DQN) for dynamically selecting the adaptation action that fits in the current architecture state. Also this paper evaluates the effectiveness of using DQN and MDP agent to achieve high level of self-adaptability of Microservices architecture. We argue in this paper that such integration between DQN and MDP in MAPE-K model offers Microservices architecture with self-adaptability against the contextual changes in the operational environment. The self-adaptation property is achieved by means of parameter tuning and dynamic adjustment of the architecture configuration. We believe integrating DQN in the dynamic decision-making process improves the effectiveness of the adaptation and reduces the adaptation risk including resources over-provisioning and thrashing. Also, it preserving the cluster state by preventing multiple adaptation to take place at the same time. It also guarantees that the executed adaptation action returns the highest reward and achieve the adaptation goals.

INDEX TERMS Anomaly detection, Micoroservices Architecture, Q Learning, Runtime configuration, Self Healing, Reinforcement learning, Policy Approximation

I. INTRODUCTION

MICROSERVICES architecture could be defined in the context of a service-oriented architecture as a composition of tiny fine-grained distributed loosely coupled building blocks of software components [1]. Microservices improve software modularity and make the application easy to develop and maintain. With the rapid development of cloud infrastructures and virtualisation techniques, a high demand for building Microservices architectures in a complete virtualised environment has emerged. This need was met by introducing containers engine like Docker [1] as well as cluster management framework such as Docker swarm [2]. The performance of Microservices running in cluster mode could fluctuate around the demand to accommodate scalability, orchestration and load balancing offered by the cluster leader [1]. It is essential for Microservice architecture to be able to reason about its own state and its surrounding environment in a closed control loop and act dynamically at runtime to achieve high level of adaptability [2]. Such level of self-adaptation requires the Microservices architecture to be able to observe its current state and provide a suitable adaptation action so it can adjust itself to reason about various contextual changes.

Nowadays, Microservices architecture does not have components that can guarantee continuous monitoring and adaptation of the operational environment. Also Microservices architecture can not offer the architecture with dynamic

[1]https://www.docker.com
[2]https://docs.docker.com/engine/swarm/

capability to reason about context changes at run-time.

To achieve such a high level of adaptability, a Microservices cluster, for instance, should have a component for continuously monitoring the cluster and a component for adaptation that can implement a reasonable reaction/scaling policy to accommodate the changes in the operating environment. This presents a challenge to build a self-adapting microservices architecture that can dynamically adjust its own behaviour and heal itself against anomalous behaviour detected at real-time.

The proposed model in this paper offers Microservices architecture a self-adaptation property by following MAPE-K (Monitor-Analyse-Plan-Execute over a shared Knowledge) model. Our model provides a mechanism for continuous monitoring, context detecting of anomalous behaviour , dynamic decision making using reinforcement learning, enabling dynamic adaptation horizontally or vertically based on the demand and the changes of the operational environment, and runtime verification and validation of the fitness of proposed adaptation strategy.

To achieve highly level of self adaptability, this research implements a MAPE-K model that employ Markov Decision Process in identifying the transition model from one cluster state to another. Also, it employees a Deep Q-learning network (DQN) that able to select the optimal adaptation action that returns the highest reward gained from executing the adaptation actions. At the same time, the use of Deep Q-learning guarantees that the knowledge from each pair of action-state is used in selecting future adaptation action to avoid adaptation failure and provide the maximum level of availability, reliability and scalability.

This paper is structured as follows: Section II presents a model that can continuously observe Microservice architectures with Self-healing capabilities. Adaptation planing and execution is discussed in Section II-B. Section III proposes a strategy for analysing and evaluating the capability of the model to detect anomalous behaviours and to trigger suitable adaptation actions. The implementation of this model is discussed in Section III-A. Section III-B is focused on results found followed a by a critical discussion of the effectiveness of this model. Section IV provides an overview of self-healing architectures and surveys the approaches for anomaly detection, and run-time configurations. Section V summarises this research, highlighting its contribution and setting future work.

II. DESIGN AND METHODOLOGY

A. SELF-HEALING MICROSERVICES ARCHITECTURE

One important aspect of a self-healing Microservices architecture is the ability to continuously monitor the operational environment, detect and observe anomalous behaviour, and provide a reasonable policy for self-scaling, self-healing, and self-tuning the computational resources to adapt a sudden changes in its operational environment dynamically at runetime.

To validate the ideas presented in this paper, we design and develop a working prototype of Microservice architecture in Docker swarm [3] as shown in Figure 1. The cluster consisted of one leader and many manager and worker nodes. To meet scalability and availability, the cluster leader distributed the work load between the workers based on Raft Consensus Algorithm [3]. This means that each service could be executed by assigning multiple containers across the cluster.

The Microservices Architecture is shown in Figure II. The architecture was designed according to the MAPE-K (Monitor-Analyse-Plan-Execute over a shared Knowledge) model [4]. So the architecture implements a service for monitoring the environment, a service for analysing the metric values, a service for planing and executing the adaptation, and a service for calculating the reward gained from executing a specific adaptation action. This model offering the Microservices architecture with the following services:

1) **Monitoring Service:** This service provides continuous collection of fine-grained metrics about cluster nodes, services, and containers including (CPU usage, Memory, Disk Reads Bytes/sec, Network Read/s, network write/s and Disk Writes Bytes/sec). The data are streamed into anomaly detection service at real-time.
2) **Analysing Service:** this service is responsible for reading the collected observations and calculate how anomalous the current observation are comparing to the architecture historical behaviour. This achieved by implementing an anomaly detection service based on NUPIC framework [5]. The NUPIC anomaly detection service [5] is continuously running over the streamed matrices collected in the matrices database, which enables the generation of the training model for the collected metric. The collected real-time data is feed on the fly to NUPIC anomaly detection service [5], which provides two features: First, continuous detection of anomalous behaviour with high accuracy. Second, it also provides predictions about the architecture performance based on the collected historic data. This can alert the architecture about incoming spike on resources demand which can be used by the adaptation manager to schedule a proactive adaptation strategy ahead of time. In addition, the anomaly detection service is able to detect anomalies as early as possible before the anomalous behaviour is interrupting the functionality of the running services in the cluster as demonstrated by Ahmad et al. [5].
3) **Adaptation Planing:** Once their is an anomalous behaviour detected with high anomaly score and likelihood, both values are calculated by the Anomaly Detection Service as shown in Figure 1. The Alert manager notifies the adaptation manager about the anomalous detected. The adaptation manager selects the adaptation action(s) after calculating the utility value for all actions as explained in the following

[3] https://docs.docker.com/engine/swarm/

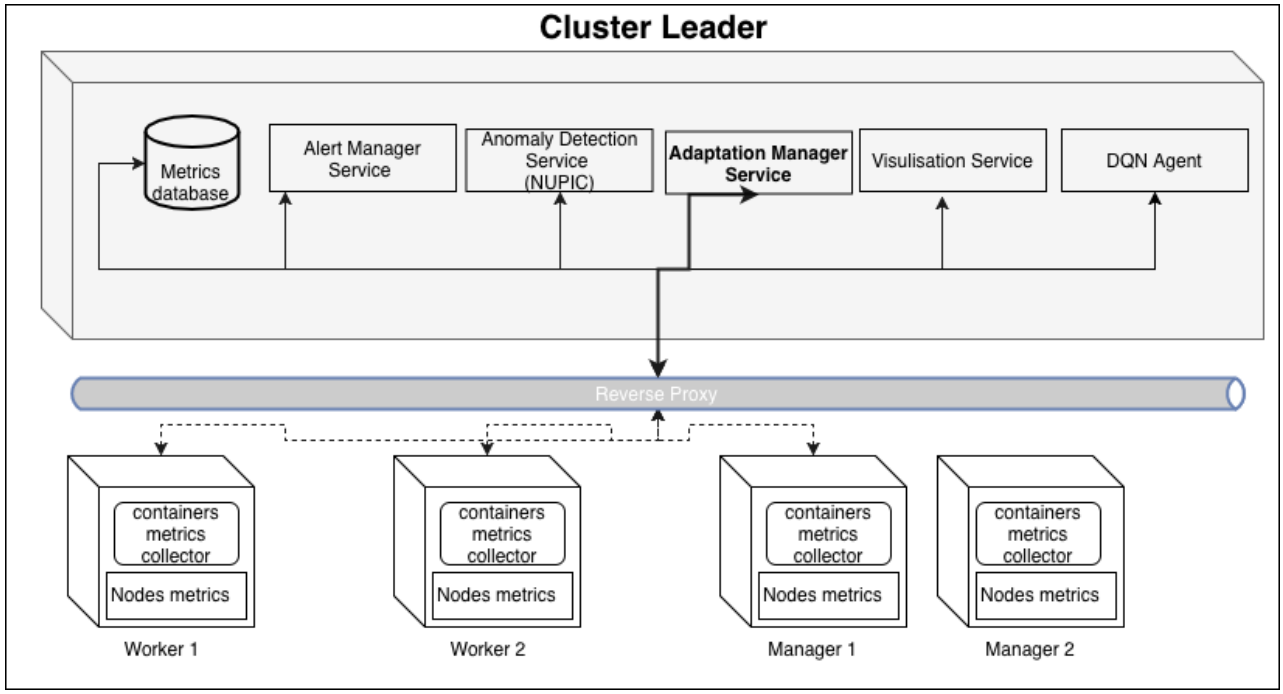

FIGURE 1. Microservices Architecture implemented in Docker Swarm

Section II-B. Then, the adaptation manager uses the input of the anomaly likelihood, architecture constraints (specified by the DevOp during deployment), and desired/predicted QoS to calculate the best variation of the adaptation that has the highest utility using a deep reinforcement learning algorithm as we will explain in Section II-B.

4) **Adaptation Election:** The adaptation manager executes the action based on the aggregated value of the Q-value returned by the DQN. Once the adaptation action is completed by the adaptation manager, a set of adaptation actions are deployed in the architecture. To avoid, conflicts between multiple adaptation polices, the adapter allow the adaptation actions to be fully completed and verified by the cluster leader according to the consensus performed by RAFT, then it will put a cool off timer before initiating new adaptation actions. This technique is used to avoid resources thrashing and preserving the cluster state for auto-recovery. The adaptation manager sends the cluster leader a set of instructions that might involve tuning of cluster parameters, (horizontal scaling), adding/removing nodes, or vertical scaling of Microservices' containers like scaling a service in/out.
5) **Adaptation Verification:** The cluster leader and all managers in the cluster will vote on the adaptation action based on the consensus algorithm [3]. The vote results is used to validate and verifies the adaptation action. If the adaptation action won the votes, the adaptation action will be executed by the cluster leader, the adaptation manager records the adaptation attempt as successful. If the adaptation action lost the voting process, then the adaptation manager keeps the current state of the cluster and records the adaptation attempt as failed. In both cases, the adaptation manger records the number of attempts used to complete the adaptation actions.

B. ADAPTATION STRATEGY

This research focuses on proposing a model that can continuously observe and monitor the Microservices architecture and be able to provide an adaptation action that can maintain the cluster state with high availability. At the same time, the architecture should be able to respond to True positive alarms by suggesting a set of adaptation actions (adaptation strategy), that can be deployed in the cluster to achieve high level of self-adaptation in response to changes in its operating environment. The problem of self-healing architecture requires an algorithm to learn how to choose an adaptation action from discrete action space and to optimise the adaptation action to guarantees that the architecture will reach the adaptation objectives [6]. This objective can be achieved using reinforcement learning algorithm, which can guarantee high accuracy of selecting the best adaptation action that fits in the current execution context.

Our adaptation strategy follows Markov Decision Process (MDP). MDP defined as set of states $s \in S$ and actions $a \in A$. The transition model from state s to state $\tilde{s}$ is defined as a function $T(s, a, \tilde{s})$ and the reward of this action in the new state $\tilde{s}$ is defined by $R(s, a, \tilde{s})$, which return a real

value every time the system moves from one state to another. In Microservices architecture the set of possible adaptation actions are identified based on the observation of the current state of the cluster. This require identifying a set of sequential actions to adapt the contextual changes by executing an adaptation action to reach the adaptation objectives or quality of services. This means, that the adaptation manager will be rewarded with positive value once it reach the adaptation objectives (i.e. service convergence). Also the adaptation manager will be rewarded negative value for every failed adaptation action.

However, the adaptation manager with discrete adaptation actions has no idea what the transition probabilities are! It does not know $T(s, a, \tilde{s})$, and it does not know what the rewards are going to be either (it does not know $R(s, a, \tilde{s})$) once it moves from one state to another. It must experience each state and each transition at least once to know the rewards, and it must experience them multiple times if it is to have a reasonable estimate of the transition probabilities. Knowing the optimal state value is very useful in identify the best adaptation action. Bellman [7] found an algorithm to estimate the optimal state-action values called Q-values. The Q-value of a state-action pair is noted by $Q(s, a)$. The $Q(s, a)$ refers to the sum of discounted future rewards that the adaptation action expect to reach in a state s after selecting the adaptation action α. Q-value estimation is not applicable in environment with large set of states and actions. Alternatively, neural network could be used to estimate the Q-value by defining approximation function and train the model in deep Q-network, this approach is called Deep Approximate learning (Deep Q-learning). Deep Q-learning is a multi-layered neural network that for a given state s outputs a vector of action values using Markov Decision Process [7] and it uses approximation function to estimate the Q-value $Q(s, a)$.

So, our adaptation manager will be using a deep Q-learning approach for identifying the best adaptation action that can return the highest reward once it reaches the desired adaptation objective. For this aim, we need to defined the reward function i.e. Q-value $Q(s, a)$ using function approximation technique. To achieve this, we need to look back at the state s of the Microservices architecture (see Figure 2. At each state $s0$ in Figure 2 there is a set of context values $c \in C$ measuring the matrices of operating environment such as: CPU, Memory, Disk I/O and Network as shown in Figure 2. The Anomaly detection services provides us with useful information about how anomalous the current cluster start comparing to the distribution of the previous learned state of the Microservice architecture (see Figure 2). Our target is to provide the Deep Q-Learning algorithm with a scalable value that can be used to assign a weight $W(s, c)$ for all context values c found in state s, this value is calculated using equation 1.

$$W(al_m, C_m) = \sum_{i=1}^{m} al_i \cdot c_i \cdot as_i \tag{1}$$

$$R(s, \alpha, \tilde{s}) = 1 - softmax(W(al_m, C_m)) \tag{2}$$

At each state $s0$ in Figure 2, the Anomaly detection service calculates the Anomaly Score (as) and Anomaly Likelihood (al) of all current context values c. The anomaly likelihood is accurately defining how anomalous the current context value comparing to the distribution of previously learned values about that specific context c. This enables the adaptation manager to scale the weight of each metric value over the distribution value calculated and aggregated in the anomaly likelihood value. The anomaly likelihood is a scaler value between 0 to 1, meaning if the context $c1$ is referring to the value of CPU usage of 70% and the Anomaly Likelihood $al(cpu)$ value is 1 then this might give the Q-learning algorithm higher probability in selecting an adaptation action that would add new node to the cluster to reduce the CPU load and keeping the cluster in the desired state. In this case the reward will be probability of $y = 1 - W(al, c)$ and it is calculated using equation 2, Which take the Softmax of all values calculated for all metrics. This will return the reward from executing an action until the cluster reaches a optimal state. By optimal state we mean the state that would return the highest reward to the DQN.

In this paper, we assuming that both the anomaly detection and Deep Q-learning algorithms are fully trained at the initial state $s0$, which normally the state following the deployment of the Microservices cluster, So at each new state (see Figure 2), the adaptation manager perform the following functions:

1) Get the current observation from the metric database service $GetObservation(s, c)$, which return all metric values from the metrics database as in Figure 2.
2) Get the current anomaly score, anomaly likelihood from the anomaly detection service *GetAnomalyScore(s,c)* as in Figure 2. This function call the Anomaly detection service to return the values of Anomaly Score and Anomaly Likelihood for each context value $c \in C$, this will return a vector of the calculated values.
3) Get the possible adaptation action $a \in A$ defined in the action space in Figure 2. Those actions could be adding/removing node, scale a services in/out, trigger auto recovery and roll-back to previous sate or stay at the current state .
4) Finally, the DQN will run the adaptation policy for several times and at each step it will calculate the reward from executing the chosen action but it will not apply the action yet. After running several episodes, it will compute each action reward, which is the action that return less anomaly likelihood. This will allows the DQN to balance the adaptation action by calculating the highest probability as in equation 2, that achieve the lowest value of the Anomaly likelihood.

In this paper. We argue that the use of anomaly likelihood to weight the collected metrics provides an accurate calculation of the weight of metrics and provides the model

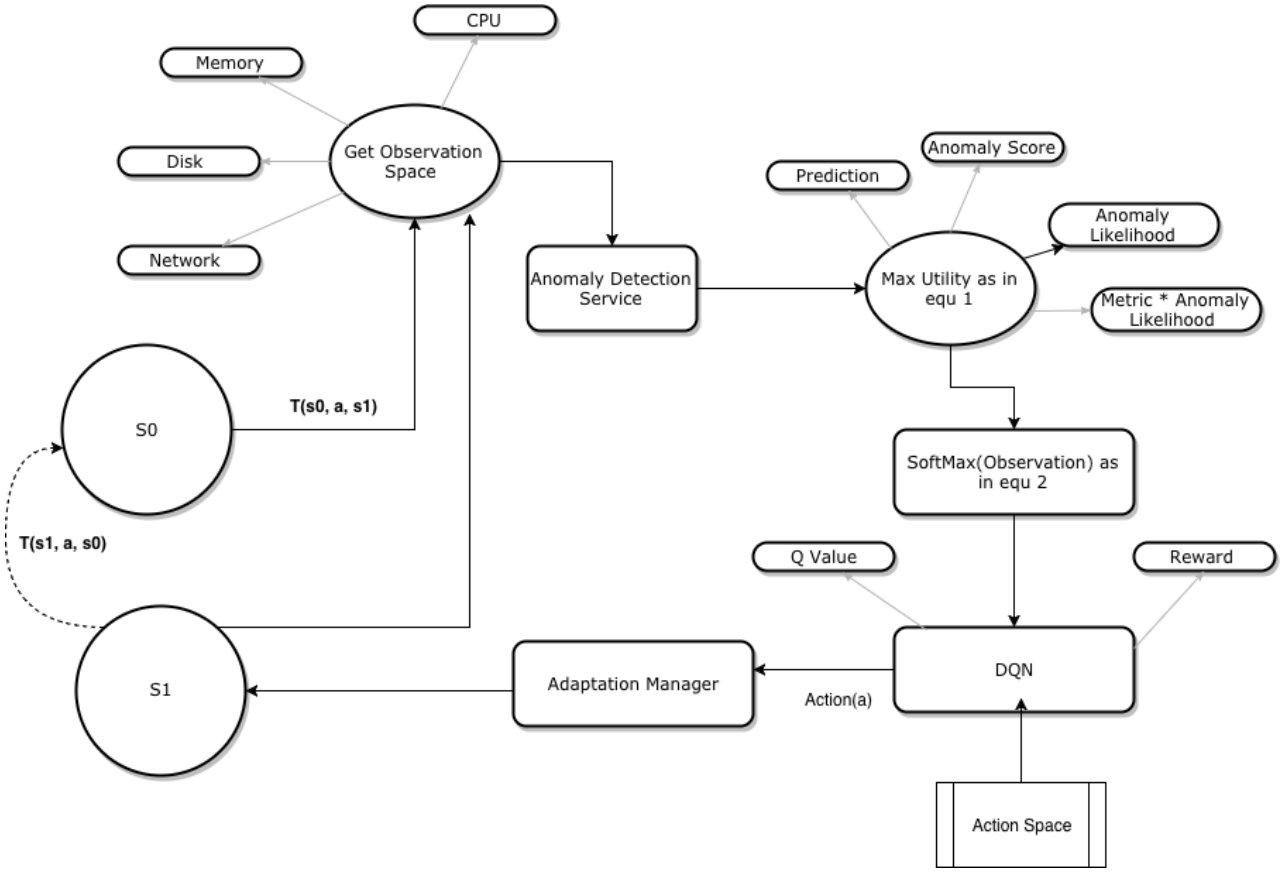

FIGURE 2. Transition Model from S0 to S1

with better estimation of the adaptation action. So to achieve highly level of self adaptability, this research implements a MAPE-K model that employ Markov Decision Process in identifying the transition model from one cluster state to another. Also, it employees a Deep Q-learning algorithm that able to select the optimal adaptation action that returns the highest reward from adaptation execution. At the same time, the use of reinforcement learning guarantees that the learned knowledge from each pair of action-state is learned by the Deep Q-learning, which will prevent the adaptation manager from executing adaptation action that would fail and return negative reward. In the following section, we will describe the experiment setup and the tools used to construct the Microservices architecture. Also, we will describe the structure of the neural network used in this experiment.

III. RESULTS

A. EXPERIMENT SETUP

To validate the ideas presented in this paper, we design and develop a working prototype of Microservice architecture in Docker swarm [4] as shown in Figure 1. Docker swarm enables the architecture to add nodes manager and worker nodes. Each cluster has one leader, which maintains the cluster state and preserves the cluster logs. Also, the leader node initializes the vote of Raft Consensus Algorithm [3] to agree/disagree on specific value based on the consensus by all nodes in the cluster. Only the leader node is allowed to commit and maintain the cluster state and initiate load balancing and orchestration. The leader node distributed the work load between the workers based on Raft Consensus Algorithm [3].

The main services implemented in this architecture are: Time series metrics database for context collection, Nodes metrics used to collect metrics from all nodes in the cluster, Alert and notification manager used to notify the adaptation manager about contextual changes offered by Prometheus framework [5]. Docker containers metrics collector for collecting fine-grained metrics about all running containers in all nodes [6]. Reverse proxy for routing traffic between all services in the cluster [7]. Unsupervised Real-time Anomaly Detection based on NUPIC [8]. Time series analytic and visualisation dashboard for observing the behaviour of the Microservices

[4]https://docs.docker.com/engine/swarm/

[5]https://prometheus.io
[6]https://github.com/google/cadvisor
[7]https://caddyserver.com/docs/proxy
[8]http://nupic.docs.numenta.org/stable/index.html

cluster [9]. Adaptation manager for executing, validating the adaptation actions developed as a prototype of this research. The adaptation manager implements a MDP agent that can observes the Microservices architecture and executes adaptation actions. The adaptation planning and execution is implemented using Deep Q-learning Network (DQN) using Tensorflow framework [10]. The DQN collects the observation of the Microservices architecture and proposes the adaptation action that return the highest reward. In this experiment, we implement policy exploration using Boltzmann Q Policy [8]. Boltzmann Q Policy is a stochastic exploration policy, where the probability of performing an action is related to the distribution of the associated Q-values. It worth mentioning that the reward from executing a specific action is calculated using equation 2. We assume that the anomaly likelihood provides a good measurement of how anomalous the current state comparing to the distribution of previous state. This would provide the DQN with accurate measurement of the probability to move from state s to $\tilde{s}$. So calculating how anomalous the action is in the new state after transition will be used by the DQN to decide to stay in this state or move to another state.

$$Cost(s,c) = \frac{(Current(c_m) - Predicted(c_m)) \cdot al(c_m)}{UsageTime * Cost(instanceType)} \quad (3)$$

Finally, to provide the DQN with proper regularisation about how much nodes/containers to add/remove at certain sate it is important to provide an architecture constraints that can be used to prevent over provisioning, allocating or thrashing of the computational resources. The DQN uses equation 3 to calculate the required number of node/replicas based on the current demand in the current state.

The $Current(c_m)$ is the current value of the metric value. The $Predicted(c_m)$ refers to the Predicted value of the utility dimension. The $al(c_m)$ is the anomaly likelihood value calculated using anomaly detection service (see Figure 2). The $UsageTime$ refers to the total number of hours the node is expected to be used per/day, this value is the man of action duration returned from the DQN. The $Cost(instanceType)$ is the cost in $ for provisioning an instance per/day, normally this is a constant value specified by the cloud infrastructure provider based on the instance type. Finally, the value of $Cost(s,c)$ is calculated against the constraint of $budget$ as $Cost(u_m) \leq budget$, The $budget$ is assigned by the Dev-Op to reflect the value of the available budget, so the adaptation manager will not exceed this value at any case. A negative value returned by $Cost(u_i)$ function means the number of nodes/replicas in the cluster should be reduced by the adaptation action.

The $Cost(s,c)$ value is used to dynamically adjust the required number of nodes/replicas to reach an optimal state, which guarantees high availability. Once there is a change in the cluster state the DQN repeat the process of collecting the observation, proposing actions, calculating the highest reward until it reaches a terminal/optimal state.

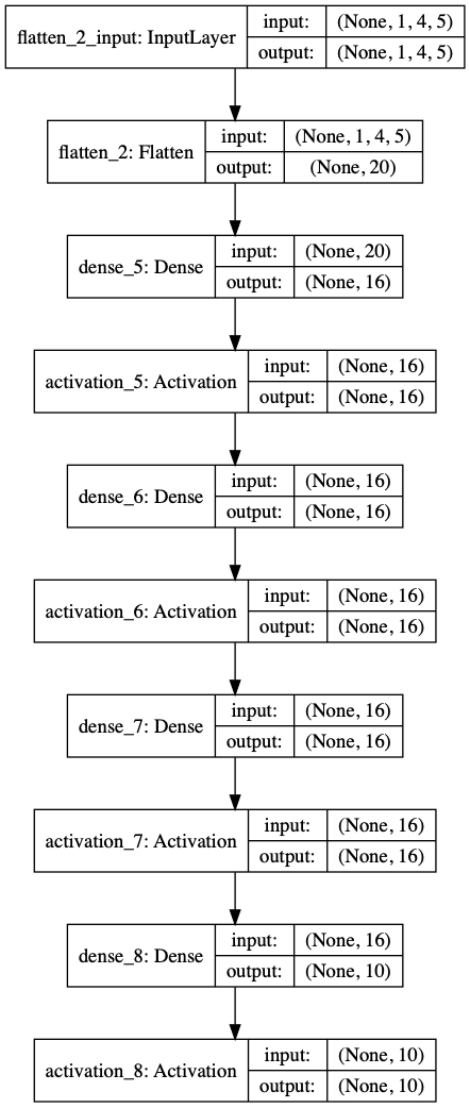

FIGURE 3. Deep Neural Network

The neural network architecture shown in Figure 3 consisted from ten layers. The input layer is the size of the observation space. In this experiment, we collect CPU usage, Memory usage, Disk space, Network I/O. for each metric we get the value from Prometheus and feed it to Anomaly detection service which will calculate the prediction, anomaly score, anomaly likelihood. The observation at each state is the input for the DQN first layer (see Figure 3). The second layer is dense hidden layer of 20 units. The RELU activation function is used in the remaining layers. The last layer in the DQN as in Figure 3 is the output layer of the action space of total of 10 actions. The action space contains the following:

- **Action 0:** Create a leader node and initiate the swarm cluster

[9]https://grafana.com
[10]https://www.tensorflow.org

- **Action 1:** Add new node
- **Action 2:** Join a node to the cluster
- **Action 3:** Remove a node form the cluster
- **Action 4:** Add a manager node to the cluster.
- **Action 5:** Scale a service vertically (add/remove replicas).
- **Action 6:** Free disk space by deleting unneeded docker images and volumes.
- **Action 7:** Free allocated memory by removing dangling docker containers.
- **Action 8:** Create a new cluster and delete excising one, then all available nodes to the new cluster.
- **Action 9:** Maintain current state as long as it returns the highest reward.

The output layer uses linear activation function, which output an action number from the above action space, which will be passed to the adaptation manager to execute it. Finally, the DQN is implemented using Adam Optimizer, which is used to calculate the Q-value at each sate action pair and return the highest reward. A full code of the adaptation manager and services stack used in this experiment can be found in [11]. This live snapshot [9] provides a full virtualisation of all services running in the cluster.

The evaluation of the effectiveness of this model will be based on calculating the reward at state action pair and the adaptation time needed to execute an action. Also we will calculate the number of adaptation attempts, successful convergence of services/nodes, or errors which leads to unstable state of the cluster.

The evaluation of this model will come in three folds: First, evaluating the consistent behaviour of the cluster by evaluating the state of the swarm after allowing the DQN to play and run the cluster. The idea is to start with no nodes and the DQN should be able to create a new cluster and add the required number of nodes/replicas until the cluster reach an optimal state. The decision will be left for the DQN to scale the cluster horizontally or vertically until the cluster reach a stable state as shown in the following section III-B. Second, evaluating the accuracy of the model in electing the correct adaptation action by identifying the highest metric value that need to be consider in the adaptation and the cumulated reward function. So, the evaluation objectives are:

1) **Obj. 1:** The ability of the DQN to take to swarm cluster and to scale it horizontally or vertically until it reach an optimal state.
2) **Obj. 2:** The ability of DQN in handling dynamic changes in the cluster and to dynamically adapt a sudden changes like a simulated stress test or Distributed Denial of Service attack (DDOS).
3) **Obj. 3:** The ability of the architecture to meet the demand dynamically and maintaining the cluster state.

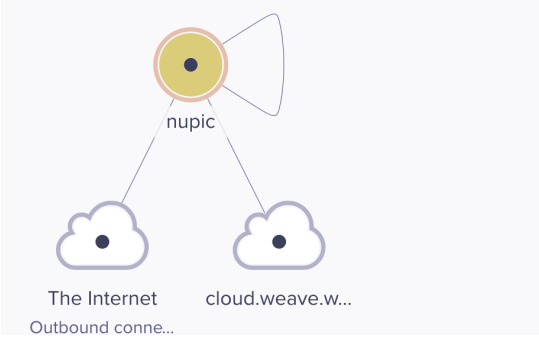

FIGURE 4. Microservice architecture initialised with a leader.

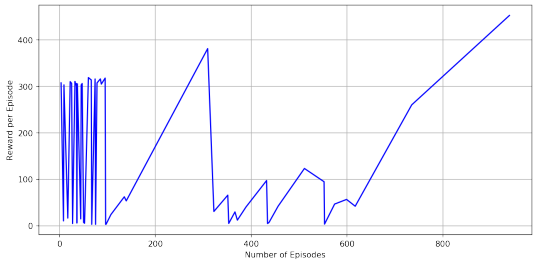

FIGURE 5. episode reward value

B. DISCUSSION

In the first experiment, the DQN is executed until it manages to find the state of creating a leader node and initiate the cluster with the services mentioned above as shown in Figure 4.

Then, the DQN will start to observe and listen for the observation collected by the adaptation manager and try to explore the optimal policy using the Boltzmann Q Policy. After running the DQN for 5000 episodes, we collect the mean of the Q value per episode as shown in Figure 5. The Figure shows that the DQN is indeed reaching a highest reward after trying different types of action until it reaches an optimal value. It is good indicator that the DQN needs to run for many episodes before it can bring the architecture to a optimal state. Also we try to evaluate the adaptation time of performing each action as this could provide could indicator about how long it take the DQN to bring the architecture to optimal state.

Figure 6 shows that the action duration in seconds and it clearly shows that DQN needs about 600 seconds to reach an optimal state as shown in Figure 7. This results is confirmed by measuring the mean Q value against time as shown in Figure 9. After finishing the training we run a test of the DQN for 10 episodes. The result of this is the Q mean absolute error shown in Figure 9. Also the loss of the model is shown in Figure 8. The final architecture of the cluster is shown in Figure 4, which achieve the first objective of the evaluation (obj. 1)

[11] https://github.com/baselm/mgr-selfhealing.git

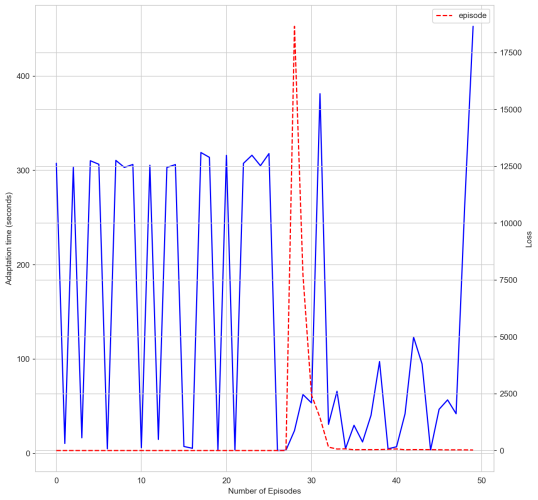

FIGURE 6. Action duration value

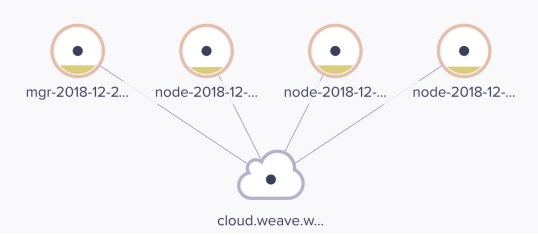

FIGURE 7. Optimal State

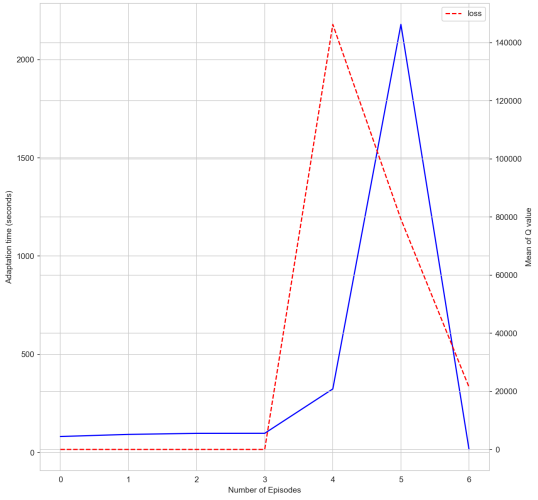

FIGURE 8. AdaptationTime Seconds

Second we run a stress test in the cluster manager until its CPU usage reaches 70%, which triggers an alert to the adaptation manager. The adaptation manager collects the current reading of all metrics, the anomaly service calculates the anomaly score, and anomaly likelihood of the current state (see figure 1). Then, the DQN calculates the Q value for the possible action to take then select an action to add

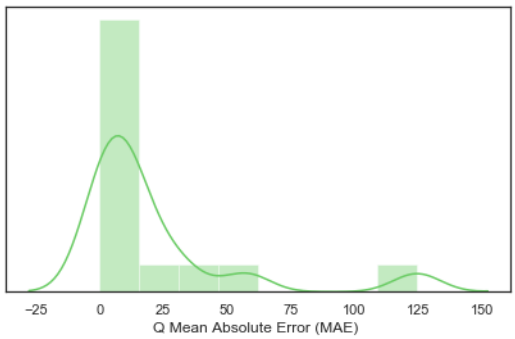

FIGURE 9. mean absolute error (MAE)

new node to the cluster. As example, Figure 10 is showing the CPU usage, Memory usage, Disk reads (bytes/s), Disk writes (bytes/s), Docker Network (sent/received bytes/s) . The CPU usage has the Maximum weight as confirmed by the Indifference indicator shown in Figure 10, remember that the Maximum weight is calculated by taken the Softmax of all matrices weight as in equation 2. Also, the memory usage of the service shows slow rate of changes over time, which make it optional to be considered in the adaptation action by the DQN. With regard to the utility of Disk Read/Write, the Figure 10 shows no divergent above the moving average (i.e. utility indifference curve) so it will not be considered in the next adaptation action. The docker network shows no changes over the time of the experiment as the load balancer and the reverse proxy manage to divert the traffic to many containers distributed across the cluster, which achieve the second objective of the evaluation (obj. 2) .

As the $W_(CPU)$ has the highest value of changes as shown in Figure 10. This will trigger an adaptation action to reason about the high demand of CPU usage, so the DQN select an action that will return the highest reward from decreasing the CPU value by adding additional node to the cluster. The number of nodes is equals to the cost calculated as in equ. 3. This results in adding new nodes to the swarm as shown in the snapshot [10] (A full visualised and analysis dashboard of the swarm after the adaptation) and confirmed in Figure 7. Once the CPU demand is reduced, the DQN will calculate the variations of the weight and remove number of nodes equals to value returned by the utility cost function in equ. 3. A snapshot of the system after executing the adaptation action to reason about the low level of the cpu usage can be found in [11], which achieve the first objective of the evaluation (obj. 2) .

In another scenario, we simulated Distributed Denial of Service attack to a web service running in the cluster, to verify that the DQN will be able to accommodate the DDOS attack by adding more replicas to the service. In this case, we wish to verify the ability of the proposed model to dynamical adjust the number of service's replicas against the variations of the CPU usage and to maintain an acceptable response time of the web service. At the same time, it is very im-

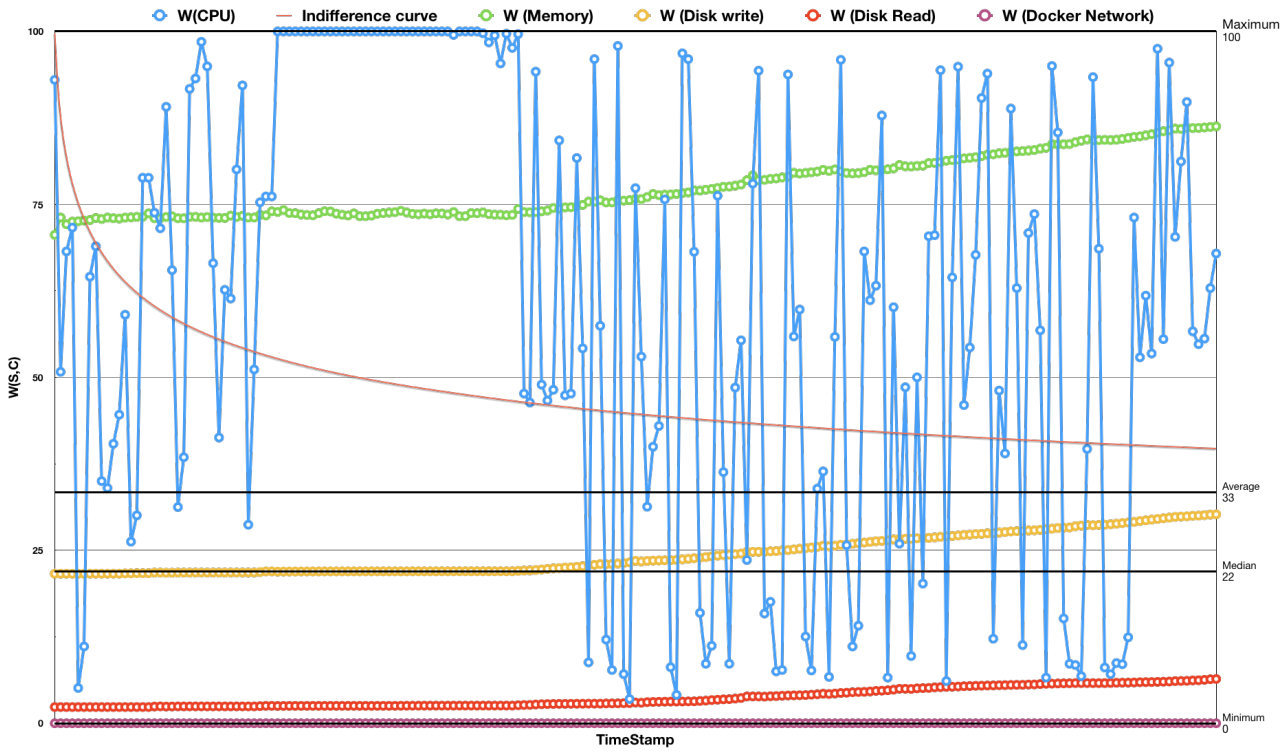

FIGURE 10. Dimensional analysis of the variations of the observation space as in 2

portant that the adaptation action would not scale the service endlessly. So the cost is calculated to count the number of replicas needed. The outcome of this experiments is shown in Figure 12. The figure show how the number of scaled replicas are tuned linearly against the CPU usage. Also, it shows the number of steps taken by the adaptation manager to execute the scaling policy in/out. The adaptation manager calculates the weight of the CPU metric and the cost to define the number of added/removed replicas. The adaptation manager waits for receiving a high value by sending heartbeat signal to get the latest value of the observation and cost every 20s for a window of 300s. Once the weight of the CPU get the highest value, the adaptation manager calculates the number of replicas to be added/removed to the service. Also, the number of steps needed to achieve the desired state are counted as shown in Figure 12. The number of steps needed to perform the adaptation varies based on the severity and variation of the cost over time. Once the adaptation is applied and verified by winning the consensus algorithm votes, the service will be scaled and the adaptation manager puts a cool off timer of 300s before initiating new adaptation action. Also, it reset the steps timer, which achieve the first objective of the evaluation (obj. 2) .

The accuracy of the cost, rate of changes, and the maximum weight are vital for the success of the adaptation process. So, Figure 11 depicts the calculation of the rate of changes and the cost to reach the desired number of nodes/replicas needed. We find the calculation accurately satisfies the adaptation objectives and provides the architecture with accurate calculation of the needed number of nodes/replicas. As shown in Figure 11, the number of nodes increases at the right time when the CPU demand spikes, then the number of nodes/replicas reduced just before the CPU demand is declined significantly as shown in Figure 11. The rate of changes in CPU usage declined so the cost return negative value for the required number of nodes/replicas as long they are above the minimum number of nodes/replicas specified by the Dev-Ops. Also, as shown in Figure 11 The cost normalizes and tunes the CPU demand. This provides a great evidence that the employment of the weight as a reward function provides the DQN with dynamic variability over the needed/allocated resources. Instead off scaling the architecture in/out according to a static value, which achieve the first objective of the evaluation (obj. 3) .

IV. RELATED WORK

A. SELF-ADAPTIVE SOFTWARE

Self-adaptive software is characterised by a number of properties best referred to as autonomic [12]. These the 'self-* properties' include Self-organisation, Self-healing, Self-optimisation and Self-protection [13]. Self-healing architecture refers to the capability of discovering, diagnosing and reacting to disruptions. It can also anticipate potential problems and, accordingly, take suitable actions to prevent a failure [13]. Self-healing aspects of Microservices architectures requires a decision-making strategy that can work in

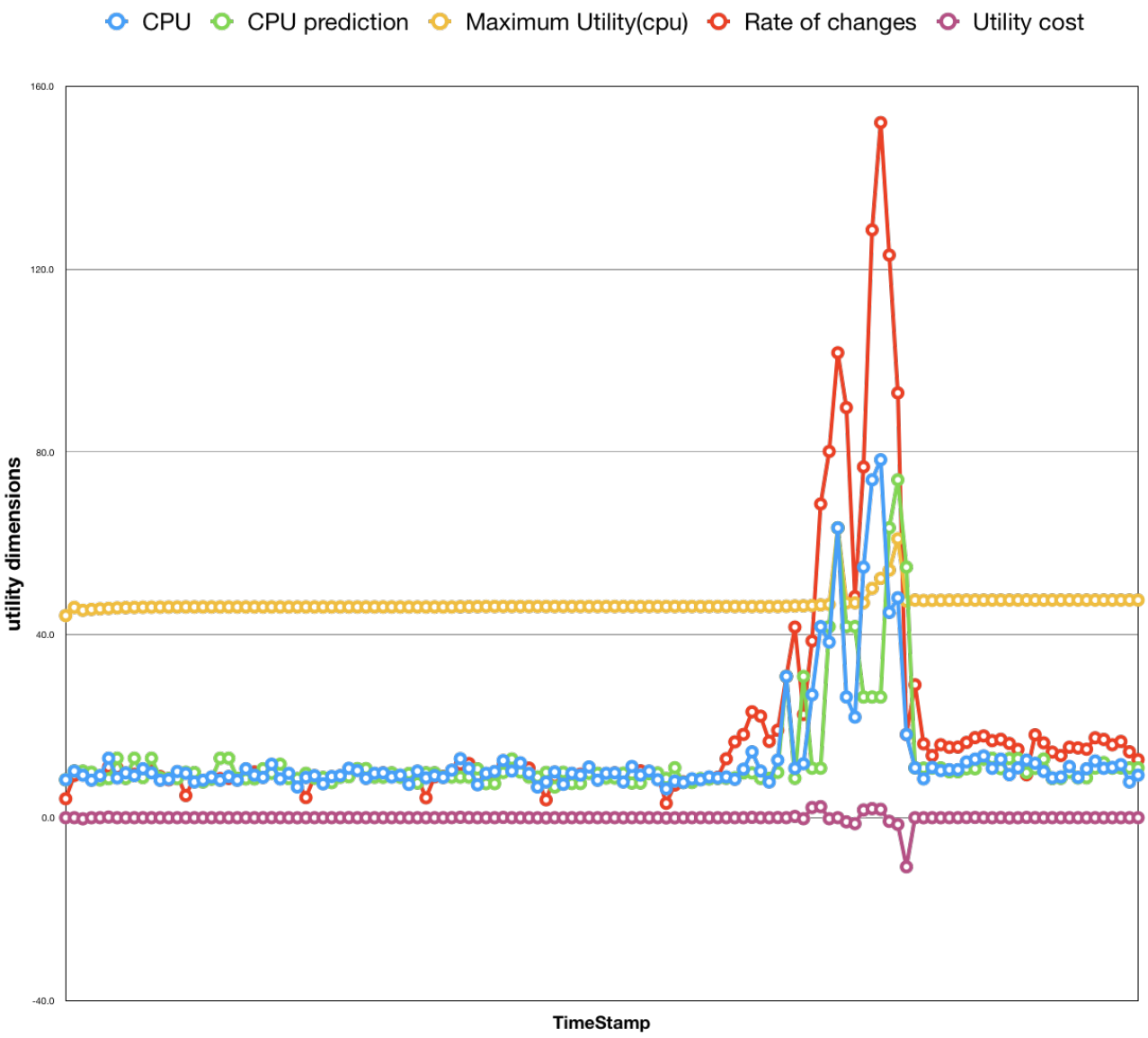

FIGURE 11. Weight(s,cpu) rate of changes and cost calculated based in equ. 3

real-time. This is essential for Microservice to reason about its own state and its surrounding environment in a closed control loop model and act accordingly [2]. Typically, a self-adapting system follows MAPE-K model (Monitor-Analyse-Plan-Execute over a shared Knowledge). a self-healing system should implements MAPE-K model including: Gathering of data related to the surrounding context (Context Sensing); Context observation and detection; Dynamic decision making; Adaptation execution to achieve the adaptation objectives defined as QoS; Verification and validation of the applied adaptation actions in terms of its ability to meet the adaptation objectives and meet the desired QoS.

B. CONTEXT SENSING

However, there is many approaches are used for achieving high level of self-adaptability though Context sensing involving context collection, observation and detection of contextual changes in the operational environment [14]. Also, the ability of the system to dynamically adjust its behaviour can be achieved using parameter-tuning [15], component-based composition [16], or Middleware-based approaches [17]. Another important aspect of self-adaptive system is related to its ability to validate and verify the adaptation action at runtime based on Game theory [18], Utility theory as in [19], [20], or Model driven approach as in [21].

Context information (1) refers to any information that is computationally accessible and upon which behavioural variations depend [22]. Context observation and detection approaches (2) are used to detect abnormal behaviour within the microservices architecture at run-time. Related work in context modelling, context detection and engineering self-adaptive software system are discussed in [2], [14], [23],

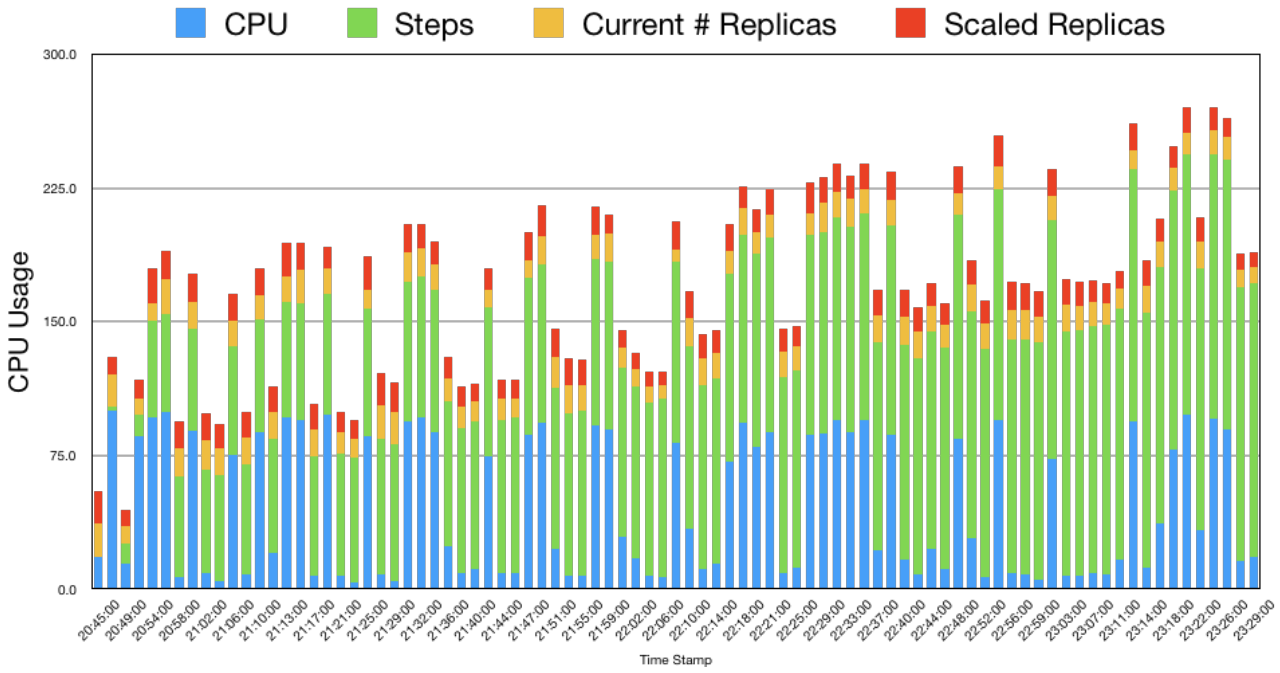

FIGURE 12. Dynamic Scaling of Web Service

[24]. In dynamic decision making and context reasoning (3), the architecture should be able to monitor and detect normal/abnormal behaviour by continuously monitoring the contextual information found in the Microservices cluster.

C. ADAPTATION PLANING AND EXECUTION

In Microservices cluster, the performance of the cluster nodes could fluctuate around the demand to accommodate scalability, orchestration and load balancing issued by cluster leader. This requires a model that is able to detect anomalies in real-time and generate a high rate of accuracy in detecting any anomalies and a low rate of false alarms. In addition, there will be a set of variations that can be used by the system to adapt the changes in its operational environment. This requires a dynamic decision making that can calculate the utility of all possible adaptation actions based on the architecture constraints (i.e. number of replicas, number of nodes, desired objectives, metrics thresholds), anomaly score of the detected conditions (CPU, Memory, DISK I/O, Network I/O), and the confidence and accuracy of the anomaly score of the detected abnormal behaviour, and the desired/predicted cluster state. Then, the adaptation manager will execute the adaptation action and verifies its successfulness over the cluster architecture. Also, the adaptation manager will be able to self-tune and self-adjust the architecture parameters to meet high/low demand for services. Finally, the architecture will preserve the cluster state through the adaptation cycle (monitoring, observing, detecting, reacting, and verifying). This research focuses on finding a method to continuously observe and monitor the swarm cluster and be able to detect anomalous behaviour with a high accuracy and generate a low rate of false alarms. Then provide the architecture with adaptation strategies with high utility to reason about the detected anomalies and be able to self-adjust the architecture parameters and verifies the adaptation actions at runtime without human intervention.

D. ANOMALY DETECTION

There are two phases for detecting anomalies in a software system: a training phase which involves profiling the normal behaviour of the system; a second phase aimed at testing the learned profile of the system with new data and employing it to detect normal/abnormal behaviours [25].

Three major techniques for anomaly detection have emerged from the literature: statistical anomaly detection, data-mining and machine-learning based techniques.

Within the statistical methods, a system observes the activity of the system and generates profiles of system metrics to represent its behaviour. The system profile includes performance measures of the system resources such as CPU and Memory. For each measure, a separate profile is stored. Then, the current readings of the system are profiled and compared against the memorised past profile to calculate an anomaly score. This score is calculated by comparing all measures within the profile against a threshold specified by the developer. Once the system detects that the current readings of the system are higher than this threshold, then these will be automatically categorised as intrusions thus triggering an alert [26].

Various statistical anomaly detection systems have been proposed and they have some advantages [27], [28]. One of this is that they can detect an anomaly without prior

knowledge of the system. This can mitigate the common problem of a cold start found in machine learning techniques. Additionally, statistical anomaly detection provides accurate notifications of malicious attacks that occurred over long periods of times and it performs better in detecting denial-of-service attacks [25]. However, a disadvantage is that a skilled attacker might train a statistical anomaly detection system to accept the abnormal behaviour as normal. It is difficult to determine the thresholds that make a balance between the likelihood of a false negative (the system fails to identify an activity as an abnormal behaviour) and the likelihood of a false positive (false alarms). Statistical methods need an accurate model with a precise distribution of all measures. In practice, the behaviour of virtual machines/computers cannot be entirely be modelled using solely statistical methods.

With regard to data-mining approaches, data-mining is about finding insights which are statistically reliable, unknown previously, and actionable from data [29]. The dataset must be available, relevant, adequate, and clean. The data mining process involves discovering a novel, distinguished and useful data pattern in large datasets to extract hidden relationships and information about the data. In general, there are two issues involved in the use of data mining in an anomaly detection system. First, there is a lack of a large dataset to be used by the algorithm containing lots of information about the architecture. Second, few approaches were targeting the anomaly detection system in Microservices architecture [29]. Data mining based anomaly detection systems have three major difficulties which prevent them from being widely adopted in Microservices architecture [25]. Firstly, the low accuracy of detecting anomalous behaviour [25], [30], as the data mining process would require large dataset with longer time interval to be able to improve the accuracy of detection. Most data mining techniques are heavily on computational resources, this negatively influences their adoption in a Microservices architecture [25]. Additionally, usually a data mining method used to classify an attack within a specific system cannot be successfully employed within another system for the same purpose. This because the process of training, testing the model and performing classification of anomalies needs to be repeated with different data or architecture [31].

Machine learning, in the context of anomaly detection, can allow the creation of software system able to learn and improve its detection accuracy over time [32]. Machine learning-based anomaly detection models aims to detect anomalies similar to statistical and data mining approaches. However, unlike them latter which tend to focus on understanding the process that generated the data, the former are data-driven and are mainly focus on training a model based exclusively on past data [25]. This means that, when additional and new data is provided they can intrinsically change their detection strategy and classify significant deviations from the normal behaviour of an underlying software program. An application of Machine Learning which enables the Microservices cluster to distinguish between normal and abnormal behaviour in the data can be found in [31]. In general, anomaly detection systems uses a combination of clustering and classification algorithms to detect anomalies. The clustering algorithm is used to cluster the dataset and label them. Then, a decision tree algorithm can be used to distinguish between normal and abnormal behaviour. Golmah [33] suggested the use of an effective classification model to identify normal and abnormal behaviour in network-based anomaly detection. The usage of Machine Mearning algorithm in this context can be found in [31], [33], [34]. Due to the opening deployment and limited resources found in a Microservices cluster, it is very important to use a lightweight approach of data clustering and classification. Due to this issue, this research focuses on proposing an anomaly detection mechanism that is more suitable for the Microservices architecture and can be easily deployed with less footprints on the limited resources found in the tiny containers running in Microservices cluster .

Numenta Platform for Intelligent Computing (NUPIC) is based on the Hierarchical Temporal Memory (HTM) model proposed in [35]. HTM has been experimentally applied in real-time anomaly detection of streaming data in [36], [37]. The proposed system based on the HTM model claimed to be efficient and tolerant to noisy data. Most importantly it offers continuous monitoring of real-time data and adapts the changes of the data statistics. It also detects very subtle anomalies with a very minimum rate of false positives. In a recent study, Ahmad et al. [5] proposed an updated version of the anomaly detection algorithm with the introduction of the anomaly likelihood concept. The anomaly score calculated by the NUPIC anomaly detection algorithm represents an immediate calculation of the predictability of the current input stream. This approach works very well with predictable scenarios in many practical applications. As there is no noisy and unpredictable data found, the raw anomaly score gives an accurate prediction of false negatives. However, the changes in predictions would lead to revealing anomalies in the system's behaviour. Instead of using the raw anomaly score, Ahmad et al. [5] proposed a method for calculating the anomaly likelihood by modelling the distribution of anomaly scores and using the distribution to check the likelihood of the current state of the system to identify anomalous behaviour. The anomaly likelihood refers to a metric which defines how anomalous the current state is based on the prediction history calculated by the HTM model. So, the anomaly likelihood is calculated by maintaining a window of the last raw anomaly scores and then calculating the normal distribution over the last obtained/trained values, then the most recent average of anomalies is calculated using the Gaussian tail probability function (Q-function) [38].

E. REINFORCMENT LEARNING

The goal of reinforcement learning (RL) is to provide the software agent with a possibility to learn a specific policy that can be used to take decision among a set of actions by maximizing the cumulative rewards [39]. Several efforts were

made to employ Neural Networks in the implementation of RL algorithms. The idea is to use the DQN to identify the mathematical relation between the input data and to identify the maximum reward function of finding the output. Such effort can be found in [40] where RL and NN were used to play Atari games or Go games as in [41]. There are two popular approaches in deep RL algorithms: Deep Q Networks (DQN) and policy gradient. DQN is a form of Q-learning with function approximation using deep neural networks. The goal of DQN is to learn a state-action value function (Q), which is given by the deep networks, by minimizing temporal-difference errors [40]. Based on the DQN algorithm, various network architectures such as Double DQN [42] and DDQN [43] were proposed to improve performance and keep stability. Policy gradient methods directly learn the policy by optimizing the deep policy networks with respect to the expected future reward using gradient descent. Williams et al. [44] proposed REINFORCE algorithm simply using the immediate reward to estimate the value of the policy. Silver et al. [45] proposed a deterministic algorithm to improve the performance and effectiveness of the policy gradient in high-dimensional action space. In the work of Silver et al. [27], it is shown that pre-training the policy networks with supervised learning before employing policy gradient can improve the performance.

V. CONCLUSIONS AND FUTURE WORK

This model manages to offer the Microservices architecture with continuous monitoring, continuous detection of anomalous behaviour, and provides the architecture with dynamic decision making based on the employment of Deep Q-learning. The results in above, shows high accuracy in detecting the anomaly and an accurate selection of the adaptation action. It Also shows high success rate in performing horizontal and vertical adaptation in response to various contextual changes. The uses of DQN enables the architecture to dynamically elect a reasoning approach based on the highest reward gained from each action state pair. The self-healing property is achieved by parameter tuning of the running services and dynamic adjustment of the swarm cluster. We believe integrating reinforcement learning in the decision making process improves the effectiveness of the adaptation and reduces the adaptation risk including resources over-provisioning and thrashing. Also, it preserving the cluster state by preventing multiple adaptation to take place at the same time and it eliminates the actions that would return the lowest reward. Currently, this model supports both vertical and horizontal scaling of the Microservices architecture, but it can be extended by adding new actions to the action space and the DQN will manage to select the action once it will return the highest demand and be suitable in the current architecture state.

Currently, Docker swarm enables the cluster to have one leader, which prevents us from testing this model in multi leaders environment. This enforces us to implement the adaptation manager as a central component in the leader node. Also, the current implementation of NUPIC anomaly detection requires multiple implementations for each contextual changes. NUPIC has no support for training its model over multiple variables. Finally, we believe the ability of the Microservices to self-adapt itself is achievable task by integrating MDP and DQN in MAPE-K architecture.

REFERENCES

[1] J. Stubbs, W. Moreira, and R. Dooley, "Distributed systems of microservices using docker and serfnode," in Science Gateways (IWSG), 2015 7th International Workshop on. IEEE, 2015, pp. 34–39.
[2] B. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, R. M. Malek, H. Müller, S. Park, M. Shaw, and M. Tichy, "Software Engineering for Self-Adaptive Systems: A Research Road Map (Draft Version)," Dagstuhl Seminar Proc. 08031, 2008.
[3] D. Ongaro and J. K. Ousterhout, "In search of an understandable consensus algorithm." in USENIX Annual Technical Conference, 2014, pp. 305–319.
[4] A. Computing et al., "An architectural blueprint for autonomic computing," IBM White Paper, vol. 31, pp. 1–6, 2006.
[5] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," Neurocomputing, vol. 262, no. Supplement C, pp. 134–147, 2017.
[6] H. Van Hasselt, "Reinforcement learning in continuous state and action spaces," in Reinforcement learning. Springer, 2012, pp. 207–251.
[7] R. Bellman, "A Markovian decision process," Journal of Mathematics and Mechanics, pp. 679–684, 1957.
[8] R. Dearden, N. Friedman, and S. Russell, "Bayesian Q-learning," in AAAI/IAAI, 1998, pp. 761–768.
[9] B. Magableh. (2018) Docker swarm services. [Online]. Available: https://snapshot.raintank.io/dashboard/snapshot/SyKQ96o2JWfuVyc43hcGgAI9YLcj
[10] ——. (2018) Docker swarm nodes. [Online]. Available: https://snapshot.raintank.io/dashboard/snapshot/sstuT2tuYkob8zjIbh1YXzBYxSJDFd
[11] ——. (2018) Docker swarm services. [Online]. Available: https://snapshot.raintank.io/dashboard/snapshot/UJlrTzwubrRQjDwM1YFJle5zvdK3A
[12] O. B. M. Jelasity, A. M. C. Fetzer, S. L. A. van Moorsel, and M. van Steen, "Self-star properties in complex information systems," 2005.
[13] P. Horn, "Autonomic computing: IBM's Perspective on the State of Information Technology," Tech. Rep., 2001.
[14] T. Strang and C. Linnhoff-Popien, "A context modeling survey," in Workshop on advanced context modelling, reasoning and management, UbiComp, vol. 4, 2004, pp. 34–41.
[15] S.-W. Cheng, D. Garlan, and B. Schmerl, "Evaluating the effectiveness of the rainbow self-adaptive system," in Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS'09. ICSE Workshop on. IEEE, 2009, pp. 132–141.
[16] M. Mikalsen, N. Paspallis, J. Floch, E. Stav, G. A. Papadopoulos, and A. Chimaris, "Distributed context management in a mobility and adaptation enabling middleware (madam)," in Proceedings of the 2006 ACM symposium on Applied computing. ACM, 2006, pp. 733–734.
[17] D. Cheung-Foo-Wo, J. Y. Tigli, S. Lavirotte, and M. Riveill, "Self-adaptation of event-driven component-oriented middleware using aspects of assembly," Proceedings of the 5th international workshop on Middleware for pervasive and ad-hoc computing: held at the ACM/IFIP/USENIX 8th International Middleware Conference, pp. 31–36, 2007.
[18] W. Wei, X. Fan, H. Song, X. Fan, and J. Yang, "Imperfect Information Dynamic Stackelberg Game Based Resource Allocation Using Hidden Markov for Cloud Computing," IEEE Transactions on Services Computing, vol. 11, no. 1, pp. 78–89, Feb. 2016.
[19] D. A. Menascé and V. Dubey, "Utility-based qos brokering in service oriented architectures," in Web Services, 2007. ICWS 2007. IEEE International Conference on. IEEE, 2007, pp. 422–430.
[20] K. Kakousis, N. Paspallis, and G. A. Papadopoulos, "Optimizing the utility function-based self-adaptive behavior of context-aware systems using user feedback," in OTM Confederated International Conferences" On the Move to Meaningful Internet Systems". Springer, 2008, pp. 657–674.
[21] M. Sama, D. S. Rosenblum, Z. Wang, and S. Elbaum, "Model-based fault detection in context-aware adaptive applications," in Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering. ACM, 2008, pp. 261–271.
[22] R. Hirschfeld, P. Costanza, and O. M. Nierstrasz, "Context-oriented programming," Journal of Object technology, vol. 7, no. 3, pp. 125–151, 2008.

[23] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," Transactions on Autonomous and Adaptive Systems (TAAS, vol. 4, no. 2, May 2009.
[24] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel et al., "Software engineering for self-adaptive systems: A second research roadmap," in Software Engineering for Self-Adaptive Systems II. Springer, 2013, pp. 1–32.
[25] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," COMPUTER NETWORKS, vol. 51, no. 12, pp. 3448–3470, Aug. 2007.
[26] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," in Proceedings of the 10th ACM conference on Computer and communications security. ACM, 2003, pp. 251–261.
[27] D. Anderson, T. Frivold, and A. Valdes, "Next-generation intrusion detection expert system (NIDES): A summary," 1995.
[28] M. Roesch, "Snort: Lightweight intrusion detection for networks," in Lisa, 1999, pp. 229–238.
[29] C. Phua, V. Lee, K. Smith, and R. Gayler, "A comprehensive survey of data mining-based fraud detection research," arXiv preprint arXiv:1009.6119, 2010.
[30] D. Gupta, S. Singhal, S. Malik, and A. Singh, "Network intrusion detection system using various data mining techniques," in Research Advances in Integrated Navigation Systems (RAINS), International Conference on. IEEE, 2016, pp. 1–6.
[31] A. L. Buczak and E. Guven, "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection," IEEE Communications Surveys & Tutorials, vol. 18, no. 2, pp. 1153–1176, May 2016.
[32] T. Bujlow, T. Riaz, and J. M. Pedersen, "A method for classification of network traffic based on C5. 0 Machine Learning Algorithm," in Computing, Networking and Communications (ICNC), 2012 International Conference on. IEEE, 2012, pp. 237–241.
[33] V. Golmah, "An efficient hybrid intrusion detection system based on C5. 0 and SVM," International Journal of Database Theory and Application, vol. 7, no. 2, pp. 59–70, 2014.
[34] N. F. Haq, A. R. Onik, M. A. K. Hridoy, M. Rafni, F. M. Shah, and D. M. Farid, "Application of machine learning approaches in intrusion detection system: a survey," IJARAI-International Journal of Advanced Research in Artificial Intelligence, vol. 4, no. 3, pp. 9–18, 2015.
[35] J. Hawkins and S. Blakeslee, On Intelligence. Macmillan, Apr. 2007.
[36] S. Ahmad and S. Purdy, "Real-Time Anomaly Detection for Streaming Analytics," arXiv preprint arXiv:1607.02480, vol. abs/1607.02480, 2016.
[37] A. Lavin and S. Ahmad, "Evaluating Real-time Anomaly Detection Algorithms - the Numenta Anomaly Benchmark," CoRR, vol. abs/1510.03336, pp. 38–44, 2015.
[38] J. W. Craig, "A new, simple and exact result for calculating the probability of error for two-dimensional signal constellations," in Military Communications Conference, 1991. MILCOM'91, Conference Record, Military Communications in a Changing World., IEEE. IEEE, 1991, pp. 571–575.
[39] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, and others, "Mastering the game of Go with deep neural networks and tree search," Nature, vol. 529, no. 7587, p. 484, 2016.
[40] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," arXiv preprint arXiv:1312.5602, 2013.
[41] R. S. Sutton and A. G. Barto, Introduction to reinforcement learning. MIT press Cambridge, 1998, vol. 135.
[42] H. Van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning." in AAAI. Phoenix, AZ, 2016, p. 5.
[43] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," arXiv preprint arXiv:1511.06581, 2015.
[44] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," Machine learning, vol. 8, no. 3-4, pp. 229–256, 1992.
[45] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in ICML, 2014.

BASEL MAGABLEH Dr. Basel Magableh is Lecturer in the School of Computer Science, Technological University Dublin. Basel has a BSc from the University of Yarmouk, Jordan, MSc from New York Institute of technology, USA and a PhD from Trinity College Dublin, Ireland. Basel has previously worked as a Chief Scientist in Mobile/Terminal Context Awareness in Huawei Technologies, Finland. He has also completed two post doctorate positions in UCD, and the Knowledge and Data Engineering Group (KDEG), Trinity College.

MUDER ALMANI Dr. Muder Almani is a Chair, Computer Information Systems Department / College of Information Technology Chair, Management Information Systems Department / College of Business Administration and Economics, Al-Hussein Bin Talal University, Ma'an - Jordan. Muder is actively researching in document handling, modems, peer-to-peer computing, power amplifiers, redundancy, security of data, storage management

LUCA LONGO Dr. Luca Longo is Lecturer in the School of Computer Science, Technological University Dublin. Luca interests have always turned around Artificial Intelligence and innovative applications, especially applied to formal reasoning and the World Wide Web. Luca have completed a BSc with honours and a MSc awarded distinction both in Computer Science at the University of Insubria (Varese, Italy). Luca have obtained also a Postgraduate Diploma in statistics and a MSc in Health Informatics awarded distinction at Trinity College Dublin (Ireland). Luca successfully defended PhD thesis in Artificial Intelligence at Trinity college Dublin. Additionally, Luca have recently obtained a Postgraduate diploma in Learning and Teaching at Dublin Institute of Technology, where Luca currently am tenured lecturer, covering both MSc and PhD courses.

Luca vision is to formalise the ill-defined construct of human Mental Workload as a computational concept through deductive knowledge representation and reasoning techniques (Defeasible reasoning, Argumentation Theory) and inductive modelling techniques (Machine Learning). Field of applications includes Human-Computer Interaction, Education, Neuro-Science, Universal Design.