An Architecture and Stochastic Method for Database Container Placement in the Edge-Fog-Cloud Continuum

Petar Kochovski *Faculty of Computer and Information Science*
University of Ljubljana
Ljubljana, Slovenia
Rizos Sakellariou *School of Computer Science*
University of Manchester
Manchester, UK
Marko Bajec *Faculty of Computer and Information Science*
University of Ljubljana
Ljubljana, Slovenia
Pavel Drobintsev *Institute of Computer Science and Technology*
Peter the Great St.Petersburg Polytechnic University
St.Petersburg, Russia
Vlado Stankovski *Faculty of Civil and Geodetic Engineering*
University of Ljubljana
Ljubljana, Slovenia
vlado.stankovski@fgg.uni-lj.si

Abstract

Databases as software components may be used to serve a variety of smart applications. Currently, the Internet of Things (IoT), Artificial Intelligence (AI) and Cloud technologies are used in the course of projects such as the Horizon 2020 EU-Korea DECENTER project in order to implement four smart applications in the domains of Smart Homes, Smart Cities, Smart Construction and Robot Logistics. In these smart applications the Big Data pipeline starts from various sensor and video streams on which AI and feature extraction methods are applied. The resulting information is stored in database containers, which have to be placed on Edge, Fog or Cloud infrastructures. The placement decision depends on complex application requirements, including Quality of Service (QoS) requirements. Information that must be considered when making placement decisions includes the expected workload, the list of candidate infrastructures, geolocation, connectivity and similar. Software engineers currently perform such decisions manually, which usually leads to QoS threshold violations. This paper aims to automate the process of making such decisions.

Therefore, the goals of this paper are to: (1) develop a decision making method for database container placement; (2) formally verify each placement decision and provide probability assurances to the software engineer for high QoS; and (3) design and implement a new architecture that automates the whole process.

A new optimisation method is introduced, which is based on the theory and practice of stochastic Markov Decision Processes (MDP). It uses as input monitoring data from the container runtime, the expected workload and user-related metrics in order to automatically construct a probabilistic finite automaton. The generated automaton is used for both automated decision making and placement success verification. The method is implemented in Java. It also uses the PRISM model-checking tool. Kubernetes is used in order to automate the whole process when orchestrating database containers across Edge, Fog and Cloud infrastructures.

Experiments are performed for NoSQL Cassandra database containers for three representative workloads of 50000 (workload 1), 200000 (workload 2) and 500000 (workload 3) CRUD database operations. Five computing infrastructures served as candidates for database container placement. The new MDP-based method is compared with the widely used Analytic Hierarchy Process (AHP) method. The obtained results are used to analyse container placement decisions. When using the new MDP based method there were no QoS violations in any of the placement cases, while when using the AHP based method the placement results in some QoS threshold violations in the cases of workloads 2 and 3. Due to its properties, the new MDP method is particularly suitable for implementation.

The paper also describes a multi-tier distributed computing system that uses multi-level (infrastructure, container, application) monitoring metrics and Kubernetes in order to orchestrate database containers across Edge, Fog and Cloud nodes. This architecture demonstrates fully automated decision making and high QoS container operation.

Index Terms

Cloud, Fog, Edge, storage, probabilistic decision-making, containers.

I. Introduction

Today, the Internet of Things (IoT) facilitates a vast number of smart applications in practically all domains [1]–[3]. IoTs continuously generate data that may raise multidimensional concerns (e.g., volume, velocity, veracity, variety), which are typical characteristics of Big Data applications. Various distributed database techniques and data management methods and technologies are currently being developed in order to address Big Data related issues and concerns [4].

The newly started Horizon 2020 European Union-Korea DECENTER[1] project is inspired by four smart applications in domains such as Smart Homes, Smart Cities, Smart Construction and Robot Logistics. The intended smart applications require the integration of data arriving from sensors and video-streams. Such data is then processed by using various Artificial Intelligence (AI) methods in order to extract useful information. Based on different application scenarios the data is processed, filtered and integrated at various stages of the Big Data pipeline, starting from Edge nodes via Fog nodes up to Cloud data centres. The whole smart application design process requires addressing various requirements including Quality of Service (QoS) requirements.

In order to facilitate efficient and effective development of smart IoT-based applications, distributed databases nowadays come hands-in-hands with component-based software engineering approaches and tools [5]. The latter promise radical improvements of the software lifecycle. One important improvement is the ability to flexibly use container images [6] within workbenches, i.e., Interactive Development Environments (IDEs), such as the recently developed SWITCH IDE [7], [8].

As a result, databases with various functionalities and properties can be implemented in containers and attached to a range of smart applications in various stages of the Big Data pipeline. Moreover, the software engineering process is completely platform-agnostic, in the sense that a software component can be designed and developed before an actual deployment decision is taken, that is, before an appropriate infrastructure is selected [9].

Existing Open Source solutions and technologies for interoperability, such as Docker[2] and Kubernetes[3], make it possible to orchestrate containers across the Edge-Fog-Cloud computing continuum [10], [11].

However, the decision on where to place (i.e., deploy) a software component, such as an instance of a database in the Edge-Fog-Cloud continuum, is particularly complex. Before making an appropriate decision, the software engineer has to consider various Quality of Service (QoS) related requirements including the great variability of infrastructures, different virtualisation technologies, geographical distribution including network related QoS (e.g., latency, throughput) and other application and user-related requirements. Therefore, new approaches, methods and technologies are needed to help improve the decision making process and make adequate and optimal or close-to-optimal deployment decisions.

Commonly, software engineers have to deal with multiple Non-Functional Requirements (NFRs), which require various trade-offs to be considered. For example, one such trade-off could be to achieve a low operational cost for data management operations versus fast data delivery. Thus, the deployment decision making task represents a complex multi-criteria decision making problem, which is difficult to be addressed manually by the software engineer.

This paper addresses the distributed systems aspect of such decision making with the development of a new probabilistic method for QoS assurance when orchestrating containers across multiple computing infrastructures (multi-tier). The method is also complemented with an architecture that helps orchestrate software components across the Edge-Fog-Cloud computing continuum. It is used for automated decision making for the placement of database containers and their orchestration as required by various smart applications. The new method relies on the theory and practice of stochastic Markov models [12], [13].

The specific contributions of this paper include:

- a decision making method for database container placement;
- formal verification of the decision and generation of QoS probability assurances for the software engineer;
- a new architecture that automates the process.

The new architecture and method are used to automatically select an optimal infrastructure where database containers should be deployed. Before an infrastructure for deployment is selected, the software engineer may also receive assurances in terms of QoS scores, that the particular deployment will not result in under- or over-provisioning of resources, additional operational costs, or specific QoS threshold violations.

This paper presents in detail an orchestration approach that uses a new probabilistic method for database placement. First, we discuss related work and then we describe the Fog-Edge-Cloud continuum concept, which is motivated by the DECENTER's smart applications. We provide a thorough description of the database placement method, and the description of selected QoS and NFR attributes for the study. A NoSQL Cassandra database is implemented in a container and serves as a use case for the study. Three workloads and five infrastructures are used to evaluate the whole decision making and orchestration process. An example of an actually generated probabilistic decision model (automaton) is also presented.

[1]https://www.decenter-project.eu/
[2]https://www.docker.com/
[3]https://kubernetes.io/

II. Related Work

This paper focuses on decision making processes in the deployment phase of databases implemented as software components. Databases are implemented in containers so that they can be deployed dynamically on a variety of infrastructures. This work focuses on the resulting QoS and non-functional properties of deployed databases.

The selection of an optimal IaaS provider with respect to the expected QoS has been addressed in various studies related to load balancing [14]–[17], resource management and allocation [18]–[21], resource provisioning [22]–[24] or service deployment and management systems [10], [25].

Many studies have proposed multi-objective optimisation solutions for various Cloud deployment scenarios. For instance, Karim et al. [26], Garg et al. [27] and Gonçalves et al. [28] take into account user's QoS requirements when selecting an appropriate Cloud provider. These studies implement a method called Analytic Hierarchy Process (AHP), which is used to calculate and rank the Cloud services according to various QoS properties. The AHP method is deterministic, based on user priorities. However, the AHP method does not provide probabilistic evaluation for QoS success, which is required for business critical Big Data applications. In addition, AHP can be computationally impractical if it is used for comparing large amounts of parameters due to the large number of embedded decisions that would be generated. For instance, if it is used to make decisions from 100 possible solutions, by comparing 20 attributes, the AHP will generate a decision tree with 2000 nodes.

Zheng et al. [29] developed a framework for optimal Cloud service selection, which ranks the services according to the provided QoS. The method, however, has been implemented for two network-level metrics (throughput and response time). The ranking system uses past usage data, while our new method takes into account the current deployment context for the application (e.g., geographic availability). Guerrero et al. [30] proposed an approach for container resource allocation. Their four-objective method for optimal resource allocation is based on the Non-dominated Sorting Genetic Algorithm (NSGA-II). This approach uses predefined parameters for which only thresholds can be dynamically managed by the software engineer. The inclusion or exclusion of QoS attributes is therefore not straightforward.

Various studies use Markov Decision Processes (MDP) to deliver decision making methods in non-deterministic and probabilistic situations. These seem highly relevant to our database container placement case. MDP has been used for context-specific decisions, which is the case when using databases in various smart Big Data applications. For example, MDP has been used to facilitate optimal medical treatment decisions (which must be tailored to individual patients) [31]. In addition, MDP has also been highly applicable in two related fields, probabilistic planning and reinforcement learning within the domain of artificial intelligence for modelling scenarios with probabilistic dynamics [32].

MDP may therefore be suitable for the Cloud computing context, to address its non-deterministic properties. In this context, Yang et al. [33] present a MDP Cloud service selection method with the purpose of choosing an environment that provides optimal performance to an application.

Nevertheless, to the best of our knowledge, the use of probabilities in order to guarantee QoS of a deployed software component in the context of containers has not been considered before. This represents a basis for formal verification of the correctness of the placement decision.

Two recent research studies [34], [35] have applied such techniques to the problem of horizontal scaling of VMs. An interesting part of the two studies is that they make use of very large MDP models composed of many VMs. However, their computational complexity is an obstacle that prevents the inclusion of such methods in mainstream software engineering practices; this has not been adequately addressed in existing studies.

The present paper complements the above efforts by focusing on the dependability, formal assurances and compatibility aspects required to select an optimal deployment infrastructure for database software components. The proposed architecture aims at providing formal guarantees to the software engineer using a stochastic approach, which may be considered as a novelty in the context of component-based software within the Edge-Fog-Cloud computing continuum.

III. Container Deployment Use-case

Typical IoT-based services are commonly built by using a layered architecture composed of sensors/actuators, such as DHT11 [36], Remote Terminal Units, such as SAITEL DR[4], IP Gateways, such as Cisco-ASA [37], Database Servers, such as Apache Cassandra[5] and IoT application layer components, such as Web servers, notification components, call centres, and so on. Usually, such IoT-based services generate and process large amounts of unstructured data. However, the amount of data differs between IoT use cases. For instance, an IoT-based construction site [3] may use various smart devices (e.g., sensors, actuators) that could be used to facilitate a smart, automated, secure and sustainable construction environment by generating and processing large amounts of data. Due to the different amount of smart devices per construction site, the data processing workloads differ between sites. For example, a smaller IoT construction site may execute 50000 data operations per minute, whilst a bigger IoT construction site may operate with more than 500000 data operations per minute. Thus, the performance of database containers in such multi-tier computing infrastructures plays an important role and usually affects the overall performance of the system. For example, significant fluctuations of the database workload may cause a low response rate and

[4]https://www.schneider-electric.com/

[5]http://cassandra.apache.org/

service unavailability. Therefore, it is important to assure high QoS in the context of large quantities of concurrent database operations, which may be placed at any stage of the Big Data pipeline.

The terms Edge, Fog, Cloud differentiate among properties such as: computing performance, network performance and geographic distribution [38]. Cloud computing is a centralised computing approach that operates with high computing power and high throughput in large data centres. However, to improve network performance, Cloud computing is being replaced by emerging computing paradigms that extend the computing capacity to the network edge, such as Fog and Edge computing. Fog computing can be considered as Cloud resources that exist between the Edge devices and the traditional Cloud computing data centres [39]. In contrast to Cloud computing it offers high computing performance with improved network performance. Edge computing is a highly distributed approach that allows processing data on various multiprocessor devices that operate in close proximity to sensors, such as Raspberry Pi, BeagleBoard or PCDuino [40]. This approach may have high network performance, but it is not recommended for intensive computing operations.

In the investigated case, the client to our database is located in Ljubljana, Slovenia. As presented later, in Table I, infrastructures located in Europe achieved high network performance. Therefore, they resemble Fog resources. Furthermore, the infrastructures g1-small and n1-standard-1 are characterised by computing power that is similar to those used by multiprocessor Edge devices. Infrastructures such as n1-standard-4 and n1-standard-8 use more powerful computing resources, thus, they can represent typical Cloud computing resources.

The middle layer between the sensors and the data centre, which is composed of various smart computing devices, significantly reduces the amount of data that is required to be sent to the centralised data centre. It also helps address privacy and security concerns and other high-level application requirements. Thus, it allows efficient data processing, analysis and storage of large quantities of data. Although the Fog infrastructure theoretically assures higher performance in such use cases, the overall performance significantly depends on the location within that Fog infrastructure, where a certain database container is deployed.

We consider databases implemented as software components. Such databases are designed to satisfy certain Big Data QoS requirements, such as: handling large volumes of unstructured data, supporting high incoming data velocity, scaling out and similar. A representative technology in this context is Cassandra Apache, a highly-scalable, open-source database technology, which can be used to store and process large amounts of unstructured data on multiple nodes. Cassandra supports nodes distribution across multiple infrastructures addressing low latency requirements for all clients. In order to provide reliable usage of Cassandra on heterogeneous hosts (e.g., data centres, micro-data centres, routers, IoT devices) and simplify its creation, deployment and running, we implemented an instance of Cassandra in a Docker container, which is used for the purposes of this study.

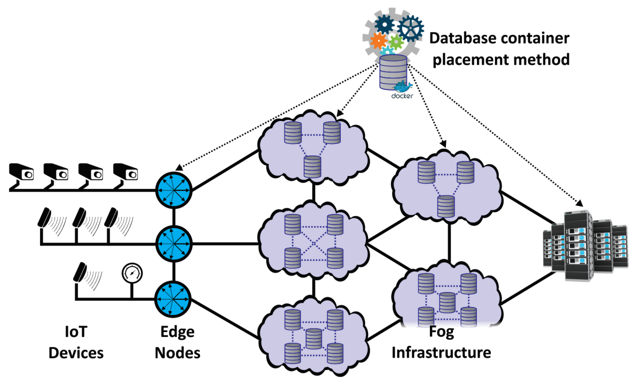


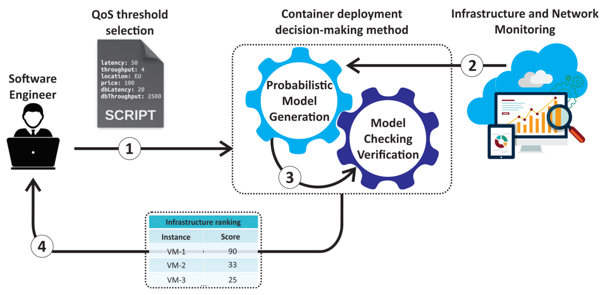
Fig. 1. Container database placement concept the Edge-Fog-Cloud continuum

IV. Deployment-optimisation method

The goal of our new method and architecture is to facilitate automated decision making and operation for the placement of databases. Databases are implemented as containers and deployed in the Edge-Fog-Cloud computing continuum. The new method helps automatically rank the available infrastructures according to the current workload context and the QoS requirements (see Fig. 2 for an illustration of the process). The ranking of deployment infrastructures is performed according to precisely defined quality constraints that the software engineer sets up at the beginning of the process. Such QoS requirements can be unique for every containerised database. Furthermore, in order to guarantee that the selected quality requirements are satisfied, the method generates probabilities for QoS assurance.

In order to achieve the above goals, we designed a new method that has the following distinctive steps:

- Quality of Service attributes selection;
- Infrastructure and network monitoring;
- Probabilistic model generation; and


QoS threshold selection
SCRIPT
Container deployment decision-making method
Infrastructure and Network Monitoring
Software Engineer
Probabilistic Model Generation
Model Checking Verification
1
2
3
4

```
latency: 50
throughput: 4
location: EU
price: 100
dbLatency: 20
dbThroughput: 2500
```

Infrastructure ranking	
Instance	Score
VM-1	90
VM-2	33
VM-3	25

Fig. 2. Database placement decision-making method

- Probabilistic output verification and infrastructure ranking.

These steps are elaborated in the following sub-sections.

A. *Quality of Service attributes selection*

The use of databases as software components applies to a great variety of applications. Hence, there is no universal set of QoS requirements that must be satisfied. In the course of our work, we studied a variety of non-functional properties that can be used by an automated decision making method and are associated to different containerised software components. A recent study [41] concluded that attributes such as: geographical location, CPU utilisation, availability and response time are important for achieving high QoS with present-day Big Data applications. Another study [42] describes that network-level metrics, such as bandwidth, throughput, jitter, power efficiency and cost are important quality attributes for their IoT applications. It is therefore necessary to design a method that allows the software engineer to choose among QoS attributes and thresholds that matter most to the specific application. Another study [43] thoroughly describes how the values of QoS attributes, such as latency, delay, jitter, data rate, error rate, loss rate and throughput, differ between application types; these values strongly depend on the specific use case, such as Web browsing, network communication, audio broadcasting, video rendering and so on.

In order to develop our proof-of-concept probabilistic QoS models, we use the following attributes:

1) *Network throughput (Gb/s)* - the rate at which data is transferred between two endpoints, without losses. It measures the quantity of data (TCP/UDP traffic) that a given software component can transfer successfully by unit of time.
2) *Network latency (ms)* - represents the time required for a packet to be transferred across the network. In our case it was measured as the round trip time for the package to reach some point and return to us.
3) *Packet loss (%)* - percentage of lost packets in the connection between the client and the running software service (component).
4) *CPU utilisation (%)* - a performance metric, which represents the sum of work handled by the CPU. The CPU utilisation varies according to the system workload of the deployed application.
5) *Database throughput (requests/s)* - the rate at which the database processes the read/write requests.
6) *Database Read/Write latency (ms)* - represents the time required to fulfil read/write operations, which begins when the database instance receives a client request and ends when it provides a response.
7) *Cost ($/month)* - cost for using the Cloud infrastructure per month.

Of course, our goal is to design a method, which is not limited to a specific set of QoS metrics and Non-Functional Requirements (NFRs). The proposed new probabilistic method can be used to support any quantified attribute that may be of interest to the software engineer.

B. *Decision making method*

The decision making method relies on MDP (see references in Section II). MDP is a stochastic process, where the transitions between states depend only upon the present state of the process, not on a preceding sequence of events. MDP is a powerful mathematical framework that can be used to support decision making in dynamic environments, where the results are partly random and partly under the control of a decision maker [44]. Its non-deterministic nature allows us to incorporate multiple potential system behaviours in one single model that takes the form of an automaton.

In this study, MDP is used to: (1) dynamically improve the automaton by using new QoS measurements; (2) implement utility functions that make it possible to express the success score from the QoS requirements; and (3) rank the possible deployment infrastructures according to QoS success score.

The output automaton of the MDP is hard to accurately predict, because it depends on variable input parameters, such as network throughput, latency, resource utilisation and other QoS metrics and NFRs.

MDP is defined as a tuple *M=(S, A, P, R,* γ*)*, where:

- $S = \{S_0, ..., S_n\}$ is a finite set of states, where, each state represents a different deployment infrastructure;

- A ={$a_0, ..., a_n$} is a finite set of actions, which, represent the deployment actions to the set of configurations;
- P ={s_{t+1} = s'|s_t = s, a_t = a} is the transition probability from state *s* at step *t* to state *s'* at the next step due to an action *a*;
- $R(s, s')$ is the expected reward received after transitioning from state *s* to state *s'*, due to action *a*.
- γ is called a discount factor. It represents the difference in importance between current and future rewards. Its value is in the range 0-1.

The probabilistic model was prepared by following the MDP tuple from the definition above. Its design and implementation are thoroughly described in the following subsections.

1) Probabilistic model: The probabilistic model is a finite automaton, which is a necessary component that is used to generate infrastructure ranking results. Probabilistic models are built specifically for every software component, and can dynamically change due to the variability of the input QoS/NFRs of the deployment infrastructures.

Each transition in the probabilistic model is a probabilistic choice over multiple next states. Each state of the probabilistic model represents a different placement (i.e., deployment) configuration. The external context can also influence the calculated probabilities. For example, on one occasion one deployment infrastructure may yield a 60% probability of reaching adequate QoS, while on another occasion this probability may be completely different.

Transitions between states happen due to different actions in the model, which give the non-deterministic behaviour of the model. The current model implements two actions: *deployment action*, which is responsible for selecting a deployment infrastructure and *idle state*, which is activated if the current deployment infrastructure has the optimal QoS requested by the software engineer. In the probabilistic model, every action has a corresponding transition. Every action results in a compatible transition between the states of the probabilistic model. For instance, a state that represents a deployment infrastructure with lower QoS will have valid transitions to all the states that represent configurations with higher QoS. At the same time, the same state may also have a transition to itself and to the states with lower QoS. Although there are multiple transitions from one state, they can be all mapped to a different probability value. In addition, whenever a transition from one state to another state takes place, a reward is gained, because every state in the model is associated to a reward value. The reward values are received from utility functions, which implement current measurements from the monitoring and threshold values set by the software engineer.

2) Calculating the model probabilities and rewards: When designing the probabilistic model, the main task is to calculate the probabilities for each transition and the rewards for each state. Essentially the transition probabilities must satisfy the Markov rule, which states that the probability to reach any future state of the process depends only upon the present state, and not on the states that preceded in the past. Hence, the probabilities that predict the future behaviour of the model are only dependent of the current state of the model. For instance, when calculating the transition probability between deployment infrastructures A and B, the transition probability does not depend on the probability values that were necessary to reach deployment infrastructure A.

In order to develop this probabilistic model, it is first necessary to select an initial state of the probabilistic model. Second, it is necessary to calculate the transition probabilities between the deployment infrastructures. Third, it is necessary to calculate the reward values, which are associated to each state in the model.

1) The first stage provides a probability estimation, which provides an initial state for the MDP. The transitions of the initial action are equally probable, and they allow the probabilistic model to commence the MDP from any available deployment infrastructure that is available at the moment. Hence, this is calculated using the following equation $P_1 = \frac{1}{N_{tran}}$, where N_{tran} is the amount of transitions of the same action from one state.
2) The second stage provides the transition probabilities between the states within the model. These probability values are used to compare the configurations QoS between each other. Therefore, the probability values that are acquired within this stage can determine if the software component would achieve better quality within one transition to another state. To calculate the probabilities during this stage, the method utilises the output from the preceding probability estimations, which allows better results within time. The probabilities are calculated by using the following equation $P_2 = \frac{N_{chosen}}{N_{listed}}$, where N_{chosen} is the number of times a deployment infrastructure has been chosen for that specific type of software component and the N_{listed} is the number of times the deployment infrastructure has been listed in the set of states to build a model.
3) The third stage provides the reward values for reaching each state of the probabilistic model and the decision results that are provided by a utility function. The reward values are scalar values that are estimated by an algorithm whose output depends on the data acquired from real-time monitoring measurements and knowledge from prior infrastructure utilisation. The reward values are calculated according to Algorithm 1. The algorithm inspects if the states of the model satisfy the thresholds, which were chosen by the software engineer. If all thresholds are satisfied by a specific deployment infrastructure, the algorithm will assign the highest reward that equals 1. In contrast, if the deployment infrastructure violates four threshold values, then according to the algorithm the reward will be calculated as $rewards_i = \frac{1}{5}$ and will equal 0.2. The algorithm uses the following input parameters: network throughput, network latency, cost, CPU utilisation, database throughput and database latency.

Algorithm 1 Algorithm for calculating the reward values in the probabilistic model

```
1: Input parameters: Network Throughput (NT), Network Latency (NL), Packet Loss (PL), CPU Utilisation (CPU), Database Throughput (DT), Database latency (DL), Network Throughput Threshold (NTT), Network Latency Threshold (NLT), Packet Loss Threshold (PLT), CPU Utilisation Threshold (CPUT), Database Throughput Threshold (DTT), Database Latency Threshold (DLT), Cost (C), Cost Threshold (CT), size - amount of deployment infrastructures
2: Output parameters: rewards
3: $count \leftarrow 1$
4: for each $i$ in $size$ do
5:     if $NT < NTT$ AND $NL > NLT$ AND $PL > PLT$ AND $CPU > CPUT$ AND $DT < DTT$ AND $DL > DLT$ AND $C > CT$ then
6:         $rewards_i \leftarrow 0$
7:     else
8:         if NT < NTT then $count \leftarrow count + 1$
9:         if NL > NLT then $count \leftarrow count + 1$
10:        if PL > PLT then $count \leftarrow count + 1$
11:        if CPU > CPUT then $count \leftarrow count + 1$
12:        if DT < DTT then $count \leftarrow count + 1$
13:        if DL > DLT then $count \leftarrow count + 1$
14:        if C > CT then $count \leftarrow count + 1$
15:        $rewards_i \leftarrow \frac{1}{count}$
16:    end if
17:    $count \leftarrow 1$
18: end for
```

Finally, a utility function is used to provide optimal solutions for the decision making process. To provide optimal results, it utilises the transition probabilities and reward values as input parameters. The utility function in this model is a reinforcement learning function and has the following recursive form: $u(S) = r(S) + \gamma \max_a \sum_{S'} P(S'|a,S)u(S')$, where $r(S)$ represents the reward value for reaching a state, $P(S'|a,S)$ is the transition probability value for reaching state S' from state S due to action a, and $P(S'|a,S)u(S')$ represents the future, discounted rewards.

Although the current model uses a utility function whose rewards are based on specific NFRs, the presented method is independent of the number and type of NFR attributes that could be used to prepare the rewards for each state. Thus, it could be extended with more attributes that could be implemented in the future.

C. Probabilistic model-checking

The proposed method builds an automaton, which has a finite number of states. It is therefore possible to employ a model-checking approach in order to verify that the system's output conforms to the requirements chosen by the software engineer, in other words, to evaluate its correctness.

Probabilistic Computation Tree Logic (PCTL) [45] is a temporal logic that allows probabilistic quantification of system's specification and was used to verify the generated automaton. The primary function of PCTL in the current methodology is to verify that the necessary QoS requirements are satisfied. It is considered beneficial that model-checking can be used to verify the system correctness against various criteria.

In the following, we present three examples to indicate the wide scope of verification criteria that could be used by the software engineer to assure that the system satisfies the required QoS and NFR standards:

- *What is the probability that database latency with a value less than 30 ms would be achieved, given that the database was redeployed from place (configuration) A to place (configuration) B?*
- *What is the probability of the service cost to be less than 100 \$/month if it requires a network latency less than 50ms and a CPU utilisation less than 55%?*

The above examples are presented as the following queries in PCTL, respectively:

- $P_{=?}$[F(databaseLatency < 30) & (chosenA U chosenB)];
- $P_{=?}$[F(cost < 100 & networkLatency < 50 & CPUutilisation < 55)];

However, PCTL is generic and can be used to assess a range of different criteria, when used to provide quality assurances.

V. Architecture

Computing infrastructures for IoT data processing may be composed of Edge/Fog infrastructures, such as: Raspberry PIs, routers and other computing devices that run in proximity of the smart environment; and data-centres, such as private or

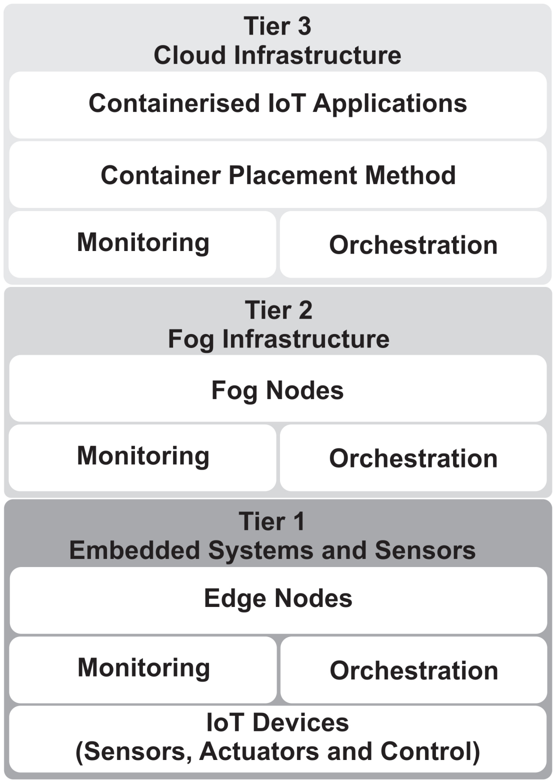


Fig. 3. Multi-tier architecture

public Clouds. Figure 3 depicts a multi-tier architecture, which follows the interoperability standards set by organisations such as the Cloud Native Computing Foundation (CNCF)[6] and OpenFog Consortium[7]. Since IoT applications can be built from reusable software components, the proposed architecture allows containerised databases (and other software components) to be deployed in three tier types. Each of the tiers is used at different stage of the Big Data pipeline. The first tier allows executing software components on various on-field IoT devices at the Edge of the network, the second tier allows running the software components on infrastructures in the Fog and the third tier runs the software components continuously on the most suitable Cloud infrastructure.

Each tier offers different properties and fulfils different operations in the data pipeline. The first tier is generally used for data collection from sensors, processing and storing for short time periods. The Edge nodes in this tier are responsible for rapid data acquisition and data normalisation, which requires the time interval between sensing and actuation to be just few milliseconds. Through various network gateways, the data can be also available to computing devices in the second tier. The distributed Fog infrastructure is focused on data filtering, compression and transformation of the data for further analysis in the higher tier. This tier constantly monitors the database containers, by a set of resource monitoring components and gathers knowledge on their QoS. If various Internet connected devices such as smartphones, robots and cars change locations it is necessary to move containers from one Fog node to another in geographic proximity of the device.

Whenever a QoS threshold violation is predicted, the monitoring component issues an alarm and appropriate action is taken by the orchestrator. The third tier represents data centres, which are centralised infrastructures, capable of performing data analytics and visualisation. In addition, the orchestration in the third tier allows the system to autonomously overcome the necessity of constant manual administration. Similarly, to the second tier, this tier also monitors the database containers and uses the measurements for the development of MPD based automatons as models. These are then used by the Kubernetes based orchestrator, which was developed and tested as part of a very recent study [10].

However, our new orchestration capability is based on the method described in the previous sections of this paper. It can be used to deploy containerised databases across the whole computing spectrum, for instance, in the data centre, the computing devices in the Fog or the Edge and even in the embedded systems and IoT devices. The orchestrator is based on Kubernetes and allows rapid start and stop of containers and their movement from one Fog node to another. In cases when the utilisation increases, it can autonomously scale containers horizontally and thus offers possibility for higher service availability. Because database reliability is closely related to the number of service replicas, the orchestrator can increase or decrease their number

[6]https://www.cncf.io/
[7]https://www.openfogconsortium.org/

based on the smart application requirements.

VI. Experimental evaluation

The goal of the experimental evaluation is to show that the decision making method and architecture are fully functional and can be used to achieve high quality database placement results. The experimental evaluation is divided into two parts: in the first part we perform a feasibility study, in the second part, we compare the new MDP based method with an AHP based method, which is considered as a baseline multicriteria decision making method for the type of problem we investigate.

TABLE I
Experimental testbed Cloud infrastructures

Infrastructure	id	vCPU	RAM	Location	Cost [$/month]
				Asia	16.5
g1-small	0	1	1.7	Europe	16.6
				US	13.1
				Asia	31.2
n1-standard 1	1	1	3.75	Europe	31.3
				US	24.3
				Asia	62.3
n1-standard 2	2	2	7	Europe	62.5
				US	48.5
				Asia	124.7
n1-standard 4	3	4	15	Europe	125.0
				US	97.0
				Asia	249.4
n1-standard 8	4	8	30	Europe	250.2
				US	194.2

Our new method was tested by three software engineers that are based in Ljubljana, Slovenia, who used this method to deploy their containerised Apache Cassandra databases on a suitable infrastructure for their purposes. They had substantial workload requirements for their databases: 50000 (Workload 1), 200000 (Workload 2) and 500000 (Workload 3) read/write operations, respectively. Essentially, one deployment infrastructure which is optimal for one of the engineers may not be optimal for the other two.

The engineers could choose one out of 20 possible Cloud infrastructures without additional performance optimisation, hosted on Google Cloud Platform[8]. There were 5 different deployment infrastructures at each of 4 different geographical locations: Frankfurt, Iowa, Taiwan and Tokyo. The properties of the deployment infrastructures are listed in Table I. All measurements, including network-based metrics from clients towards remote services were conducted from Ljubljana, Slovenia.

TABLE II
Average network performance on different geolocations

Location	Latency	Packet loss
Frankfurt	30.22	0.00
Iowa	333.56	0.00
Taiwan	621.79	0.00
Tokyo	556.39	0.00

The probabilistic model generation was done by using historical multi-level monitoring measurements, collected during the week before the actual database placement. From the network-level metrics, which are presented in Table II, it can be concluded that the network latency is heavily affected by the clients (Ljubljana) to infrastructure geographical distance.

To generate the reward values for the probabilistic model, the software engineers selected the following QoS thresholds: CPU utilisation by the container less than 30%, network latency less than 40 ms, network throughput greater than 4 Mbps, database latency less than 20 ms, database throughput greater than 2500 operations per second and price less than 100 $/month.

TABLE III
Decision-Making with the New MDP Based Method

	Workload 1		Workload 2		Workload 3	
id	Rank	Score	Rank	Score	Rank	Score
0	1	24.47	4	22.84	3	22.13
1	2	20.5	1	26.94	2	18.38
2	4	18.73	2	22.99	1	18.09
3	5	19.10	5	21.48	4	16.68
4	3	19.03	3	22.89	5	16.11

[8]https://cloud.google.com/

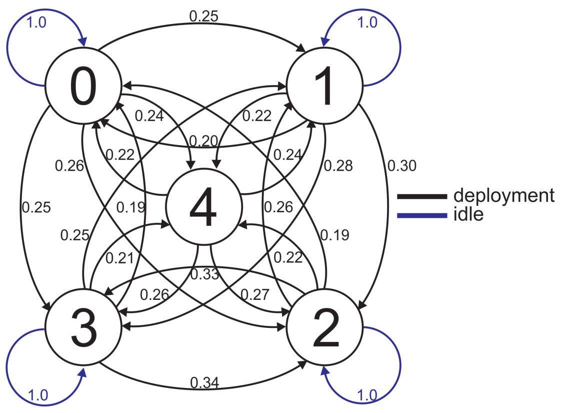


Fig. 4. Experimentally derived probabilistic model

Having prepared the probabilistic model, our new method provides ranking of deployment infrastructures (see Table III). As can be seen, the results strongly rely on the reward system: thus, the higher the reward is, the better the ranking score is. In addition, the results show that a different workload has direct impact on network performance, database performance and resource utilisation. As a result, the proposed decision making method suggests different optimal Cloud infrastructures under different workloads. An example of a probabilistic model generated for the requirement of 200000 database read operations is shown in Figure 4.

The results, presented in Table III show that the new method provides a ranking mechanism for infrastructures according to the generated score for high QoS success.

TABLE IV
DECISION-MAKING WITH THE AHP METHOD

	Workload 1		Workload 2		Workload 3	
id	Rank	Score	Rank	Score	Rank	Score
0	1	43.2	1	35.7	1	36.4
1	2	25.0	2	21.6	3	18.4
2	4	10.3	4	14.2	2	19.6
3	5	8.8	3	15.5	5	10.2
4	3	12.8	5	13.0	4	15.4

In the second stage of our evaluation, we compare the newly designed MDP based method with the AHP baseline method (see Section II). Table IV presents the results obtained by using AHP.

The obtained ranked lists for the first workload of 50000 read/write database operations are the same for both methods, while they differ significantly for the workloads of 200000 and 500000 read/write operations (see Table V). The MDP based method suggests that for 200000 read/write operations it is necessary to use the n1-standard-1 deployment infrastructure, while the AHP method suggests the g1-small configuration. For the use case of 500000 read/write operations the MDP method suggests to use the n1-standard-2 deployment infrastructure, while the AHP suggests the g1-small configuration.

In order to compare the quality of both methods, we define the metric N as the number of QoS metrics for which their thresholds have been violated by a deployment infrastructure. The results are presented in Table V. In contrast to the AHP method, our new MDP based method always arrives to a deployment infrastructure that does not violate the QoS metrics thresholds. Going down the ranked list, in the case of the MDP-ranked list, the values of the N metric increase monotonously, while this is not the case with the AHP method. This suggests that the newly designed MDP methods are more suitable for the given purpose.

The AHP method derives its ranked lists by applying pairwise comparison between deployment infrastructures' attributes, while the MDP method strongly relies on a reward value. The MDP method derives its ranked list by multiple simulations from random initial states of the probabilistic model (automaton), thus exploring all possible alternatives before arriving to a ranked list. It iterates through the model multiple times and checks if the current state provides optimal QoS and what action must be selected to reach an optimal QoS from any other state of the model.

VII. CONCLUSION

With the emergence of IoT, Edge, Fog and Cloud computing the concept of distributed systems evolves. It also poses some new optimisation problems when loosely coupled container based applications have to be orchestrated across geographic regions. Various smart applications, such as those developed in the course of the newly started DECENTER Research and

TABLE V
METHODS WITH RESPECT TO QOS THRESHOLD VIOLATIONS

	Workload 1			Workload 2			Workload 3		
id	MDP	AHP	N*	MDP	AHP	N	MDP	AHP	N
0	1	1	0	4	1	1	3	1	1
1	2	2	1	1	2	0	2	3	1
2	4	4	4	2	4	1	1	2	0
3	5	5	2	5	3	2	4	5	2
4	3	3	3	3	5	2	5	4	3

**N = Number of QoS metrics with threshold violations by the deployment infrastructure.*

Innovation Project require the deployment of software components in multiple computing tiers, starting close to the actual sensors and video-cameras that produce data streams up to data centres, where complex data analyses must be performed.

The purpose of the present work was to address the decision making problem when specific database containers must be deployed in any of the available infrastructures, i.e. the Edge, Fog or the Cloud. Bearing in mind the plethora of IoT and Big Data approaches on one hand, and Edge-Fog-Cloud infrastructures on another, the database container placement problem is an important research problem which was investigated in this study.

This study presented a new decision making method for database container placement with optimal QoS. It also provides QoS assurances to the software engineer. Finally, a multi-tier orchestration approach is presented, which is used to automate the whole process of using smart Big Data applications. Our new MDP based method takes as input QoS measurements collected from a distributed monitoring system. It also uses constraints (thresholds) on the collected QoS metrics. The obtained data are then used to derive models for specific workloads and database deployment cases. The generated models take the form of automata.

Our experiments are based on 25 infrastructures and 8 QoS metrics, however, the new method is suitable for potentially large quantity of available infrastructures and QoS metrics, which may be in the range of hundreds. Our results show that the new method can be used effectively and helps avoid any QoS threshold violations in the investigated experimentation cases. It can be used in real-world software engineering practice and the generated probabilities can be used to gain confidence (i.e., QoS assurances) about the database container placement process. The new method and the orchestration architecture is designed to be included in the new DECENTER's Fog Computing Platform.

Our future work will concentrate on the evaluation of the new MDP based method for other software components, such as compute and memory intensive Microservices implemented in containers. In the course of our DECENTER project, we also plan to use Blockchain and Smart Contracts to facilitate automated changes of Fog nodes by using a Fog Exchange Broker.

ACKNOWLEDGEMENTS

The research and development reported in this paper has received support from the European Union's Horizon 2020 Research and Innovation Programme under grant agreements no. 644179 (I-BiDaaS: Industrial-Driven Big Data as a Self-Service Solution) and no. 815141 (DECENTER: Decentralised technologies for orchestrated cloud-to-edge intelligence).

REFERENCES

[1] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things journal*, vol. 1, no. 1, pp. 22–32, 2014.

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[3] P. Kochovski and V. Stankovski, "Supporting smart construction with dependable edge computing infrastructures and applications," *Automation in Construction*, vol. 85, pp. 182–192, 2018.

[4] V. Stankovski and R. Prodan, "Guest editors' introduction: Special issue on storage for the big data era," *Journal of Grid Computing*, vol. 16, no. 2, pp. 161–163, Jun 2018. [Online]. Available: https://doi.org/10.1007/s10723-018-9439-1

[5] G. Casale, C. Chesta, P. Deussen, E. D. Nitto, P. Gouvas, S. Koussouris, V. Stankovski, A. Symeonidis, V. Vlassiou, A. Zafeiropoulos, and Z. Zhao, "Current and future challenges of software engineering for services and applications," *Procedia Computer Science*, vol. 97, pp. 34 – 42, 2016, 2nd International Conference on Cloud Forward: From Distributed to Complete Computing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050916320944

[6] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On*. IEEE, 2015, pp. 171–172.

[7] Z. Zhao, A. Taal, A. Jones, I. Taylor, V. Stankovski, I. G. Vega, F. J. Hidalgo, G. Suciu, A. Ulisses, P. Ferreira *et al.*, "A software workbench for interactive, time critical and highly self-adaptive cloud applications (switch)," in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*. IEEE, 2015, pp. 1181–1184.

[8] Z. Zhao, P. Martin, J. Wang, A. Taal, A. Jones, I. Taylor, V. Stankovski, I. G. Vega, G. Suciu, A. Ulisses *et al.*, "Developing and operating time critical applications in clouds: the state of the art and the switch approach," *Procedia Computer Science*, vol. 68, pp. 17–28, 2015.

[9] V. Stankovski and D. Petcu, "Developing a model driven approach for engineering applications based on mosaic," *Cluster computing*, vol. 17, no. 1, pp. 101–110, 2014.

[10] U. Paščinski, J. Trnkoczy, V. Stankovski, M. Cigale, and S. Gec, "Qos-aware orchestration of network intensive software utilities within software defined data centres," *Journal of Grid Computing*, pp. 1–28, 2017.

[11] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, and O. Rana, "The internet of things, fog and cloud continuum: Integration and challenges," *Internet of Things*, vol. 3-4, pp. 134 – 155, 2018.

[12] L. Rabiner and B. Juang, "An introduction to hidden markov models," *ieee assp magazine*, vol. 3, no. 1, pp. 4–16, 1986.
[13] H. M. Taylor and S. Karlin, *An introduction to stochastic modeling*. Academic press, 2014.
[14] J. Hu, J. Gu, G. Sun, and T. Zhao, "A scheduling strategy on load balancing of virtual machine resources in cloud computing environment," in *Parallel Architectures, Algorithms and Programming (PAAP), 2010 Third International Symposium on*. IEEE, 2010, pp. 89–96.
[15] L. E. Li and T. Woo, "Dynamic load balancing and scaling of allocated cloud resources in an enterprise network," Mar. 31 2011, uS Patent App. 12/571,271.
[16] M. Randles, D. Lamb, and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," in *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*. IEEE, 2010, pp. 551–556.
[17] Z. Chaczko, V. Mahadevan, S. Aslanzadeh, and C. Mcdermid, "Availability and load balancing in cloud computing," in *International Conference on Computer and Software Modeling, Singapore*, vol. 14, 2011.
[18] S. S. Manvi and G. K. Shyam, "Resource management for infrastructure as a service (iaas) in cloud computing: A survey," *Journal of Network and Computer Applications*, vol. 41, pp. 424–440, 2014.
[19] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, vol. 23, no. 3, pp. 567–619, 2015.
[20] N. C. Luong, P. Wang, D. Niyato, Y. Wen, and Z. Han, "Resource management in cloud networking using economic analysis and pricing models: A survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 954–1001, 2017.
[21] N. Jain and I. Menache, "Resource management for cloud computing platforms," Mar. 14 2017, uS Patent 9,595,054.
[22] S. Singh and I. Chana, "Q-aware: Quality of service based cloud resource provisioning," *Computers & Electrical Engineering*, vol. 47, pp. 138–160, 2015.
[23] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimization of resource provisioning cost in cloud computing," *IEEE transactions on services Computing*, vol. 5, no. 2, pp. 164–177, 2012.
[24] L. Zhang, Z. Li, and C. Wu, "Dynamic resource provisioning in cloud computing: A randomized auction approach," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 433–441.
[25] R. Mijumbi, J. Serrat, J.-L. Gorricho, S. Latre, M. Charalambides, and D. Lopez, "Management and orchestration challenges in network functions virtualization," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 98–105, 2016.
[26] R. Karim, C. Ding, and A. Miri, "An end-to-end qos mapping approach for cloud service selection," in *Services (SERVICES), 2013 IEEE Ninth World Congress on*. IEEE, 2013, pp. 341–348.
[27] S. K. Garg, S. Versteeg, and R. Buyya, "A framework for ranking of cloud computing services," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 1012–1023, 2013.
[28] R. Gonçalves Junior, T. Rolim, A. Sampaio, and N. C. Mendonça, "A multi-criteria approach for assessing cloud deployment options based on non-functional requirements," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 1383–1389.
[29] Z. Zheng, X. Wu, Y. Zhang, M. R. Lyu, and J. Wang, "Qos ranking prediction for cloud services," *IEEE transactions on parallel and distributed systems*, vol. 24, no. 6, pp. 1213–1222, 2013.
[30] C. Guerrero, I. Lera, and C. Juiz, "Genetic algorithm for multi-objective optimization of container allocation in cloud architecture," *Journal of Grid Computing*, vol. 16, no. 1, pp. 113–135, 2018.
[31] C. C. Bennett and K. Hauser, "Artificial intelligence framework for simulating clinical decision-making: A markov decision process approach," *Artificial intelligence in medicine*, vol. 57, no. 1, pp. 9–19, 2013.
[32] A. Kolobov, "Planning with markov decision processes: An ai perspective," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 6, no. 1, pp. 1–210, 2012.
[33] J. Yang, W. Lin, and W. Dou, "An adaptive service selection method for cross-cloud service composition," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 18, pp. 2435–2454, 2013.
[34] D. Tsoumakos, I. Konstantinou, C. Boumpouka, S. Sioutas, and N. Koziris, "Automated, elastic resource provisioning for nosql clusters using tiramola," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE, 2013, pp. 34–41.
[35] A. Naskos, E. Stachtiari, A. Gounaris, P. Katsaros, D. Tsoumakos, I. Konstantinou, and S. Sioutas, "Dependable horizontal scaling based on probabilistic model checking," in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*. IEEE, 2015, pp. 31–40.
[36] D. Robotics, "Dht11 humidity & temperature sensor," 2010.
[37] A. Cisco, "5500 series adaptive security appliances," 2007.
[38] 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*, vol. 18, no. 9, p. 2938, 2018.
[39] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.
[40] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.
[41] R. Sandhu and S. K. Sood, "Scheduling of big data applications on distributed cloud based on qos parameters," *Cluster Computing*, vol. 18, no. 2, pp. 817–828, 2015.
[42] L. Li, S. Li, and S. Zhao, "Qos-aware scheduling of services-oriented internet of things," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1497–1505, 2014.
[43] Y. Chen, T. Farley, and N. Ye, "Qos requirements of network applications on the internet," *Information Knowledge Systems Management*, vol. 4, no. 1, pp. 55–76, 2004.
[44] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.
[45] F. Ciesinski and M. Größer, "On probabilistic computation tree logic," in *Validation of Stochastic Systems*. Springer, 2004, pp. 147–188.