Journal of Network and Computer Applications 203 (2022) 103354

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Context-aware distribution of fog applications using deep reinforcement learning

Nan Wang [a], Blesson Varghese [b,*]

[a] Mindtrace Ltd., UK
[b] University of St Andrews, UK

ARTICLE INFO

Keywords:
Fog computing
Decentralised cloud
Edge computing
Context-aware distribution

ABSTRACT

Fog computing is an emerging paradigm that aims to meet the increasing computation demands arising from the billions of devices connected to the Internet. Offloading services of an application from the Cloud to the edge of the network can improve the overall latency of the application since it can process data closer to user devices. Diverse Fog nodes ranging from Wi-Fi routers to mini-clouds with varying resource capabilities makes it challenging to determine which services of an application need to be offloaded. In this paper, a context-aware mechanism for distributing applications across the Cloud and the Fog is proposed. The mechanism dynamically generates (re)deployment plans for the application to maximise the performance efficiency of the application by taking operational conditions, such as hardware utilisation and network state, and running costs into account. The mechanism relies on deep Q-networks to generate a distribution plan without prior knowledge of the available resources on the Fog node, the network condition, and the application. The feasibility of the proposed context-aware distribution mechanism is demonstrated on two use-cases, namely a face detection application and a location-based mobile game. The benefits are increased utility of dynamic distribution by 50% and 20% for the two use-cases respectively when compared to a static distribution approach used in existing research.

1. Introduction

The next generation of Cloud applications is anticipated to leverage computing resources available at the edge of the network (Varghese and Buyya, 2018; Satyanarayanan, 2017; Dias de Assunção et al., 2018). Resources at the edge of the network will be more constrained in terms of processing capabilities when compared to the Cloud (Hong and Varghese, 2019; Varghese et al., 2019, 2016). The computing paradigm that makes use of resources both in the Cloud and along the Cloud–Edge continuum is referred to as Fog computing (Bonomi et al., 2012; Hu et al., 2017).

In Cloud computing, applications that service end-user devices reside in Cloud data centres. Fog applications, on the other hand, will be serviced by both distant Clouds and Fog resources near user devices. This is done to bring latency-sensitive computing to the Fog resource (Wang et al., 2017) and make the application more responsive for the end-user (Satyanarayanan, 2017; Wang et al., 2017; Chen et al., 2017). Similarly, the volume of data that is transferred to the Cloud for processing, which can be expensive in monetary terms, can be reduced if processed nearer to the source on Fog resources (Varghese et al., 2016; Hong and Varghese, 2019). The benefits include improving the overall Quality-of-Service (QoS) of the application that directly impacts a user's experience as well as making them cost-efficient.

Distributing applications across the Cloud and the Fog can be achieved if applications are designed as a composition of multiple services (McChesney et al., 2019; Donassolo et al., 2019; Bittencourt et al., 2017; Gupta et al., 2017). Either one or more application services in the Cloud can be brought to the edge of the network, which may be geographically closer to end-users. This would require the deployment of the appropriate services in three ways: (i) Cloud-only — all the services are hosted in the Cloud VM; (ii) Fog-based — some of the services are hosted in a Fog node while the rest in the Cloud VM; (iii) Fog-only — all services are hosted in a Fog node (McChesney et al., 2019). The number of services that will move to the Fog from the Cloud will be based on the availability of Fog resources and network conditions at any given time. Offloading application services from the Cloud to the Fog is a complex task because Fog resource availability and utilisation, and network conditions change over time.

It is challenging to determine how many and which services need to be moved to the Fog in the face of variable systems and network conditions. A key question to address in such a context is *which of the services of an application should be distributed across the Cloud and Fog?*

* Corresponding author.
E-mail address: bv6@st-andrews.ac.uk (B. Varghese).

https://doi.org/10.1016/j.jnca.2022.103354
Received 12 June 2021; Received in revised form 15 January 2022; Accepted 20 February 2022
Available online 14 April 2022

Ad hoc distribution across the Cloud and Fog can be detrimental to the performance of the application and in turn, affects the QoS. In this article, QoS is the combination of the computation and communication latencies of the distributed application. Therefore, to maximise the performance of an application it is necessary to determine the best combination of services that are distributed across the Cloud and the Fog. A variety of factors will need to be considered for determining this, including resource availability and utilisation on the Fog and network conditions. Also, the economic model used at the Fog (whether the Fog is more expensive than the Cloud) is important in arriving at a decision. This naturally results in a trade-off between the overall QoS and the monetary cost.

The research problem targeted in this article is '*the distribution of Fog applications across a multi-layer Fog computing system*'. The three research questions addressed to tackle the problem are:

Q1: When should a Cloud application make use of the Fog for improving performance?
Q2: How many services of an application should be placed on the Fog?
Q3: What is the trade-off between QoS and monetary costs when using Fog computing?

We propose a deep Reinforcement Learning (RL) based approach to address the research problem. RL is considered a good solution for this because: (i) it provides the capability to learn an optimal solution; (ii) it does not require a pre-trained model in contrast to other machine learning mechanisms. This means that no prior knowledge of Fog nodes is required. In this research, Deep Q-Network (DQN), a model-free RL approach is employed, which works well with continuous input and discrete output, to develop the context-aware distribution mechanism in a Fog system.

The contributions of this research are as follows:

(1) *The formulation of the problem and a mathematical model to capture the distribution of an application across Cloud and Fog resources, both in terms of the QoS of the application and the running costs on Cloud and Fog resources*. The distribution problem is long-term, and the goal is to maximise the overall benefits of a series of deployments of applications, which has not been considered in existing works.

(2) *The design and development of a context-aware mechanism to distribute modular applications in a three-tiered distributed system hierarchy*. Existing research for offloading modules from the Cloud to the Fog is based on static distribution or scheduling of the application by relying on initial deployment. The mechanism we propose on the other hand distributes the modular components across Cloud and Fog resources in a dynamic manner. The decision of which components needs to be distributed is made based on operational conditions, such as hardware utilisation of the Fog node and network state between the Cloud and the Fog resource whenever it is required once an application is deployed to improve the overall performance both in terms of QoS and running costs.

(3) *The application of deep RL for distributing modular Fog applications*. The DQN-based offloading algorithm proposed in this paper can effectively select a Fog-based deployment plan that leads to the highest utility of an application service. Configuring the distribution of applications at run-time through learning from the operational environment and feedback of each deployment is highly desirable in the Fog where the applications deployed and system and network conditions change rapidly.

The feasibility of the proposed context-aware distribution mechanism is validated using two fog-computing use-cases: a mobile game and a face detection system. These workloads are a natural fit for using the Fog since they are latency-critical — the response time is affected by the distance between the end device and the application server. The merit of the context-aware distribution mechanism is observed in that to select an appropriate deployment plan only has a sub-second overhead. Additionally, we observed that the proposed mechanism effectively outperforms the other static deployment approaches for both use-cases. Through the analysis of cost-efficient, QoS-aware and hybrid strategies, and the impact of Fog pricing, it is also found that the benefits of context-aware distribution mechanism are more significant when it is optimised towards the QoS of applications and Fog resources are priced less than Cloud resources.

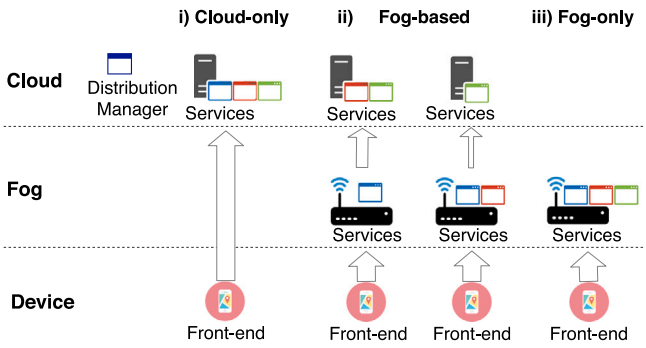


Fig. 1. Illustration of the distribution of a modular Fog application. Data movement is indicated by directed arrows, width of which represents the amount of data transmitted.

The remainder of this paper is organised as follows. Section 2 presents background to modular Fog applications and discusses related research. Section 3 presents the mathematical model that underpins the Fog system, and the problem tackled in this article. Section 4 presents a context-aware methodology for distributing a modular Fog application across the Cloud and the Fog using deep RL. Section 5 presents two real use-cases that can benefit from Fog computing and the experimental setup. Section 6 evaluates the proposed methodology and results obtained using the chosen use-cases. Section 7 concludes this article by considering future work.

2. Background and related work

This section presents the background to modular Fog applications and presents the related work on distribution of applications across the Cloud and the Fog.

2.1. Modular Fog applications

Many existing Cloud applications are service-based and modular, which are geo-distributed across clusters of systems (Thai et al., 2014, 2018). These applications are a good design fit for Fog computing because: (i) some of the application modules can be moved from the Cloud to the Fog; (ii) modular applications lend themselves to flexible usage of Fog resources.

Currently, Cloud applications are distributed horizontally across clustered systems. In a Fog system, the application will need to be distributed vertically across multiple tiers. Fig. 1 compares different types of distributions of a modularised Cloud application in a three-tier Fog system. The modularised Cloud application is either the original modular applications that consist of a collection of modules or applications that are partitioned to suit a Fog system. These modules (the blue, red and green modules in Fig. 1) work together to provide the overall functionality of the application.

Three distribution scenarios can be applied to a modularised Cloud application, namely the Cloud-only, the Fog-based and the Fog-only distribution. In the Cloud-only scenario, the server of the Cloud application is hosted in a Cloud VM and the front-end application is installed on an end device (e.g. smartphone). While the front-end application is active, the end device sends data to the server in the Cloud as indicated by the directed arrow in Fig. 1. The width of the arrow is representative of the amount of data sent. In the Fog-based scenario there could be several distribution plans depending on the number of modules to be deployed on a Fog node. In the Fog-only scenario, all the modules are

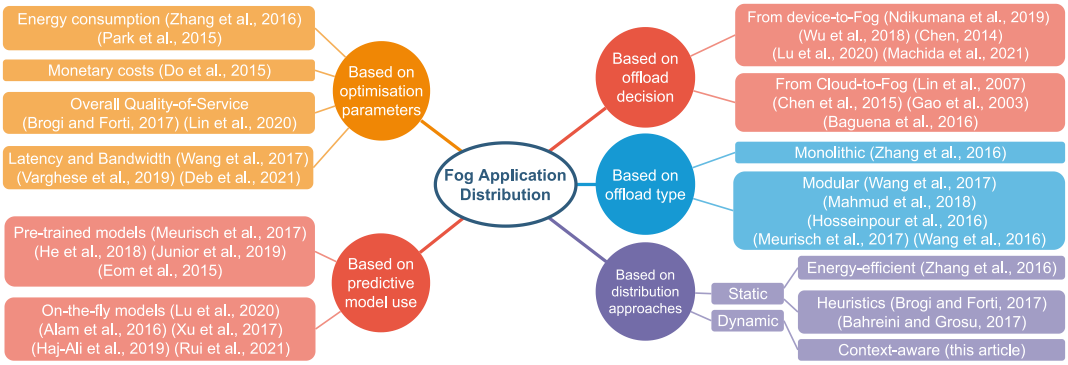


Fig. 2. A classification of existing research on Fog application distribution.

deployed in a Fog node such that all data received from an end device is processed in the Fog node.

Although employing Fog computing has been proven to be effective to improve the QoS of Cloud applications (Wang et al., 2017; Sarkar et al., 2018), using Fog computing services is challenging. This is because the Fog is restricted due to limited hardware resources and the availability of resources changes over time due to varying network and system conditions. The question of when and how to utilise a Fog node for elastic Fog applications is not well addressed in the literature. Therefore, in this paper, the impact of different distribution plans on the performance of elastic Fog applications is explored and a context-aware distribution mechanism that generates an optimal distribution plan is proposed.

2.2. Related work

The distribution of applications across the Cloud and the Fog (or resources located at the edge of the network) has been investigated in the context of related areas, namely Fog computing, Multi-access Edge Computing (MEC) and Mobile Cloud Computing (MCC). A classification of existing research is shown in Fig. 2. In this article, five classifications are highlighted based on (i) the direction of offload, (ii) the type of application, (iii) the parameters that are optimised in the problem space, (iv) the modelling techniques used, and (v) the approaches used for distributing computation workloads.

Based on the direction of offload, computational workloads can either be offloaded from end-user devices to the MEC/Fog platform (Ndikumana et al., 2019; Wu et al., 2018; Chen, 2014; Lu et al., 2020; Machida and Andrade, 2021), or from the Cloud to the MEC/Fog platform (Lin et al., 2007; Chen et al., 2015; Gao et al., 2003; Báguena et al., 2016). This paper focuses on offloading computations from the Cloud to the Fog platforms. In contrast to the replication of either data (Lin et al., 2007; Gao et al., 2003) or application (Chen et al., 2015) in the Fog, this work proposes a method to configure the offloaded computation by taking the Fog environment into account.

Based on the type of application, either a monolithic (single application whose computational tasks cannot be distributed or divided (Zhang et al., 2016)) or a modular (an application that is composed of different services that can be geo-distributed (Mahmud et al., 2018; Hosseinpour et al., 2016)) application can be executed in the Fog/Edge environment. When monolithic applications are offloaded there are fewer software related-challenges than resource challenges that need to be addressed. For example, monolithic application do not need to be partitioned, but mapped to suitable resources that can meet its requirements and objectives.

Modular applications, on the other hand, require partitioning to determine feasible Fog/Edge services that can be offloaded. Such offloading strategies may use manual techniques for partitioning an application into multiple services (Wang et al., 2017). For mapping the services that need to be mapped onto resources multiple techniques are proposed. For example, probing resources (Fog/Edge nodes) with micro-tasks to estimate the performance when more computationally intensive services are offloaded (Meurisch et al., 2017); the data sizes are different for the micro tasks and the computationally intensive service. The estimations are used for a decision support mechanism. Similarly, partitioning based on task-input data is proposed to determine whether to execute the tasks locally in end devices or MEC servers (Wang et al., 2016). This paper focuses on the partial offloading of modular Cloud applications. The workloads being considered are representative of both latency-sensitive mobile game and bandwidth-hungry image processing, which have not been thoroughly studied in other offloading research.

Based on the parameters that are optimised in the problem space, parameters such as energy consumption (Zhang et al., 2016; Park et al., 2015), monetary costs (Do et al., 2015), the overall Quality-of-Service (QoS) (Brogi and Forti, 2017; Lin et al., 2020), latency and bandwidth (Wang et al., 2017; Varghese et al., 2019; Deb et al., 2021) are used. This paper studies the impact of offloading on both the QoS of applications and the cost of Fog-based distribution. The aim of the proposed context-aware offloading algorithm is to select the best distribution plan in order to obtain the highest utility of application services.

Another classification of Fog application distribution is *based on predictive models* (Nayeri et al., 2021; Guevara et al., 2020). When developing offloading strategies, it is essential to know how well the applications would perform in a given Fog computing system. Predictive models are often employed to estimate parameters, such as QoS (Brogi and Forti, 2017), cost (Do et al., 2015), and energy consumption (Park et al., 2015) of applications given the context of Fog systems. In order to gather training data for modelling, benchmarking is carried out on Fog systems either offline (He et al., 2018; Junior et al., 2019) or online (Meurisch et al., 2017; Eom et al., 2015). A disadvantage of these methods is that they require training data to build predictive models that deal with unseen data in inference. However, it is impractical to benchmark a Fog application on every available Fog node in a real-world setting due to the heterogeneity and the volume of Fog nodes. Therefore, the system optimisation problems are tacked by using RL (Alam et al., 2016; Xu et al., 2017; Haj-Ali et al., 2019; Lu et al., 2020; Rui et al., 2021a), which provides an agent to learn on-the-fly how to behave in an environment by taking actions and seeing the results.

Model-free RL mechanisms are implemented when designing offloading (Dinh et al., 2018) and scheduling (Zhao et al., 2021) policies in edge computing. The Q-learning based algorithm in this work does not require that mobile users have prior knowledge of wireless channel information. The chosen Q-learning algorithm in this work is effective but also limited since the state variables are discretised from continuous values, which does not apply to the case when a large set of state variables is needed to account for the heterogeneity of Fog nodes. In

contrast, the DQN algorithm chosen in this paper is more suitable for the complex Fog environment.

DQN has been tested in recent works on MEC and Fog computing (Rui et al., 2021b). A DQN-based approach to dynamically orchestrate networking, caching and computing resources for smart cities applications is employed in He et al. (2017). Similarly, a DQN-based scheduling algorithm is adopted in Wang et al. (2019) to solve the optimal offloading problem. Both these works, however, assume that Fog applications are distributed statically and only focus on the scheduling of jobs. Our work, on the contrary, apply DQN-based solutions to a different problem of distributing Fog application across the multi-layer Fog computing environment.

Based on the approaches used for distribution, they may either be classified as static or dynamic (Islam et al., 2021). Static distribution refers to when the Cloud–Fog/MEC partition of an application remains the same (i.e. the same services or modules of an application are offloaded) and does not adapt to the varying context of the Fog systems over time. Energy-efficient (Zhang et al., 2016) and heuristic (Bahreini and Grosu, 2017) techniques are considered for static deployments.

In addition to the above static approaches, FogTorch (Brogi and Forti, 2017) is designed as an offloading framework that uses a greedy heuristic approach for deriving offloading plans. This framework helps to deploy multi-component IoT applications in multi-layer Fog computing systems by searching through all possible deployment plans. However, this work is more useful at the design time when the applications are tested with all deployment plans, and it only considers network conditions as the system state. On the contrary, the context-aware distribution of Fog applications proposed in this paper works at the run time when a real-time configuration of the deployment plan is needed and considers the state of Fog computing nodes. Moreover, the proposed offloading algorithm in this paper is evaluated on a test-bed comprising an embedded system representative of the Fog node, whereas many of the existing offloading solutions for Fog computing are evaluated using simulations.

3. Problem model

This section presents modularised applications that can be distributed in a Fog system. Subsequently, the mathematical model for the context-aware distribution of the modular components of the application in a Fog system is considered. The suitability of RL for the context-aware distribution problem is then discussed.

3.1. Problem model

Table 1 shows the mathematical notation employed in this research for a Device–Fog–Cloud system. Application server a is modular and comprises N different modules. a is hosted on a Cloud VM and the users of a install the client-side application on their devices. When a Fog node is available to provide computing services, Distribution Manager (DM) of a would need to decide whether a redistribution of a is beneficial. The redistribution decision is to find the optimal $k \in \{0 \dots N\}$ such that the first k modules of the application are deployed on a Fog node (we assume a sequential workflow with device input to the first module of the application).

Fog applications may have a short life cycle on a Fog node due to varying computing capabilities that will be available on the node (Tortonesi et al., 2019). When the resource availability on the Fog and network conditions change, the initial deployment may be less efficient, resulting in performance degradation. Therefore, the application will need to be redeployed with a new distribution configuration.

In this context, the distribution strategy S of a will need to combine a series of deployment plans. The problem addressed in this paper is to *determine and manage m successive distributions of a modular Fog application in the face of variable resource availability and network conditions of a Fog system.*

Table 1
Notation of parameters to model a Fog computing system.

Parameter	Description
a	A server of a modular application
k	The number of modules to be deployed on a Fog node in one deployment
S	A distribution strategy containing m deployment plans
U	Utility of a Fog application
m	Number of deployments in a distribution problem of a Fog application
T	Completion time of one deployment
R	Number of user requests being processed in one deployment
C	Overall cost of one deployment
C_C	Cost of using Cloud computing services
P_C	Price of a reserved Cloud VM
λ	Ratio of the unit price of Fog resource (CPU/memory/storage) to Cloud resources
$P_{cpu/mem/str}$	Price of a unit of resource (CPU/memory/storage) in a customised Cloud VM
$R_{cpu/mem/str}$	Average used units of resource (CPU/memory/storage) in a Fog node in one deployment

The objective is to maximise the utility of S for a. Utility functions are frequently employed in Fog computing research, for example, to measure revenue of Fog services (He et al., 2018; Zhang et al., 2017b) and performance of Fog networks (Zhang et al., 2017a). Eq. (1) defines the utility in this paper, which accounts for the trade-off between the QoS of a and its running cost given S.

$$U(S,a) = \sum_{i=1}^{m} (\alpha \frac{T_i}{R_i} + \beta C(k_i)) \tag{1}$$

T is the time taken to complete a job in a single deployment and R is the number of jobs processed in one deployment. The average time taken to service a single request ($\frac{T}{R}$) represents the QoS of a when adopting a particular deployment. The utility function takes the trade-off between the QoS and the running cost C into consideration. α and β represent the relative importance assigned to the QoS and cost factors. The running cost of one deployment is a function of k:

$$C(k) = \begin{cases} C_C, & \text{if } k = 0 \\ C_F, & \text{if } k = N \\ C_C + C_F, & \text{otherwise} \end{cases} \tag{2}$$

where C_F and C_C are the costs entailed by employing Fog and Cloud services, respectively. The cost of Cloud services is defined on a subscription basis, as adopted by popular public Cloud service providers such as Amazon Web Services (AWS):

$$C_C = P_C \cdot T \tag{3}$$

where P_C is the price of a reserved Cloud VM for hosting an application, which is in the unit of dollar per hour. The cost of Fog computing services is defined on a pay-for-resource-used basis:

$$C_F = \lambda P_{cpu/mem/str} \cdot R_{cpu/mem/str} \cdot T \tag{4}$$

where $P_{cpu/mem/str}$ is the unit price of Cloud resources including CPU, memory and storage. λ represents how expensive Fog resources are compared to the Cloud resources. In this model λ is chosen from $[0.001, 0.01, 0.1, 1]$. It is assumed that Fog resources are not more expensive than Cloud resources. This may be an incentive for Cloud-native application providers to make use of the Fog. The cost of Fog-based application services is in addition to the cost of the Cloud-based application services. This is because in this paper, an application is assumed to have one running server on the Cloud. When an application has multiple Cloud servers, the cost of Fog-based application services could

be less than that of the Cloud-based application services as the number of Cloud servers may decrease due to offloading. $R_{cpu/mem/str}$ is the average amount of Fog resources used in a single deployment.

The motivation for the above Fog pricing is as follows. Fog applications may have multiple deployments to respond to the system or network changes in the environment, and in each deployment, the modules deployed on a Fog node may vary. Therefore, it is not efficient to reserve the same amount of Fog resources for an application. For example, if resources are reserved for the maximum number of modules ($k = N$), then for deployments with fewer modules moving to the Fog ($k < N$) some of the already paid for Fog resources will not be utilised, and vice-versa. Hence, a fine-grained pricing model for Fog resources is required to support each deployment of modular Fog applications.

3.2. Machine learning for distributing Fog applications

A naive mechanism for distributing a Fog application may be always deploying the same number of services. Such a static strategy does not adapt to the varying Fog contexts and may result in applications under-performing. A dynamic distribution strategy would instead deploy a varying and appropriate number of services in a Fog node given the context. Such a dynamic strategy requires a decision-making process that adapts to the context — variable Fog nodes, Fog resource availability and network conditions.

To achieve this, the utility, a measure of system performance, defined in Eq. (1) for every possible deployment plan is used in this paper to compare the effectiveness of different deployment plans. By comparing the utility values of an application being deployed on the Cloud or the Fog computing platform, it is possible to identify the best deployment plan of the application. Furthermore, if the utility value can be accurately estimated, then it is possible to proactively select the best deployment plan in advance. This results in avoiding poor application performance due to selecting an ad hoc deployment plan.

Supervised learning techniques have been frequently used in Fog research to make decisions by estimating system performance (Osanaiye et al., 2017) and cost (Su et al., 2018). These techniques require a predictive model to be trained on benchmarking data that is obtained from heterogeneous Fog nodes and all potential applications that may use Fog nodes. This is impractical when there is a large volume of heterogeneous Fog resources and a variety of applications with different system requirements as seen in real-world computing environments.

Given that there are currently no large-scale Fog computing platforms in the real world, the data that can be collected for training a predictive model will mainly come from test-beds or even simulations. This limits the use of models trained with unrealistic data in real-word platforms. RL is an alternative to the above and is suitable to generate dynamic distribution strategies. It learns the quality of an offloading decision at run-time, thereby making adaptive decisions in the Fog infrastructure of varying scale and with different constraints.

Therefore, reinforcement learning is considered a suitable alternative to supervised learning. operate in an environment comprising heterogeneous Fog nodes and a variety of different Fog applications. In the following section, the distribution problem of Fog applications as a reinforcement-learning task is presented and then the context-aware distribution mechanism using Deep Q-Network (DQN) is proposed.

4. Context-aware distribution using reinforcement learning

RL is defined as a process to learn the best actions based on rewards or punishments. A RL problem comprises the following components: (i) agent — the RL algorithm; (ii) environment – a physical world in which the agent operates; (iii) state — the current situation of the environment; (iv) action — the operation the agent takes; (v) reward — the feedback from the environment based on the action the agent takes. The goal of the agent is to collect as much reward as possible through interacting with the environment by trial and error using feedback on its actions.

Table 2
Factors used in the state vector in the reinforcement-learning task.

Parameter	Description
CPU_u	Current system-wide CPU utilisation of a Fog node
CPU_n	Number of logical CPUs in a Fog node
CPU_f	Current CPU frequency of a Fog node
MEM_p	Total physical memory in a Fog node
MEM_{pu}	Current physical memory usage of a Fog node
MEM_s	Total swap memory of a Fog node
MEM_{su}	Current swap memory usage of a Fog node
STR_d	Total disk space in a Fog node
STR_{du}	Current disk usage of a Fog node
IO_r	System-wide number of reads from the disk in a Fog node
IO_w	System-wide number of writes to the disk in a Fog node
IO_{rb}	System-wide number of bytes read from the disk in a Fog node
IO_{wb}	System-wide number of bytes written to the disk in a Fog node
IO_{bs}	System-wide number of bytes sent from a Fog node
IO_{br}	System-wide number of bytes received by a Fog node
IO_{ps}	System-wide number of packets sent from a Fog node
IO_{pr}	System-wide number of packets received by a Fog node
D_{FC}	Network delay between a Fog node and a Cloud VM
D_{EC}	Network delay between an end device and a Cloud VM

In this paper, we transform the above concepts into a Fog context: (i) the agent in our problem is a DM located in a Cloud VM and responsible for distributing a modular Fog application; (ii) the environment is a Fog node for deployment (iii) the state is the current representation of the Fog environment; (iv) the action is to select k services of the Fog application to deploy into the Fog node; (v) the reward is the utility defined in Eq. (1). The aim of the DM is to select the k that results in the highest utility. This is in line with the theoretical design of a DQN agent, which is to learn through trial and errors to achieve the highest reward. RL is considered an appropriate approach for solving the problem of distribution Fog applications, because the reward for optimal allocation of resources on each deployment could be delayed, as the goal is to find an optimal deployment solution for application providers in the long term.

In the problem defined above, the state of a Fog node at a certain time consists of 19 factors (Table 2) related to the processor, the memory, the storage, the disk I/O, and the network I/O in a Fog node, and the network conditions. These factors are selected to be representatives of the specification and computing capability of a Fog node, the network condition and the relative distance between the Fog node and the Cloud VM. Context-awareness within the computing environment is achieved by considering these factors. Although in this work the Fog node is assumed to be owned by the Cloud service provider, these metrics could be obtained in a multi-party Fog infrastructure with a standard application programming interface (we assume given that multi-Cloud interfaces are available from Apache jclouds,[1] similar Cloud–Fog interfaces will be available). The set of actions is the possible values of k, which are $N + 1$ discrete values.

DQN-based Distribution Mechanism: DQN is an algorithm in RL that uses deep neural networks to represent the mapping between states and actions (Mnih et al., 2015). It learns an optimal function to maximise the total reward over several successive steps. This is suitable for the long-term distribution problem of Fog applications. Due

[1] https://jclouds.apache.org/

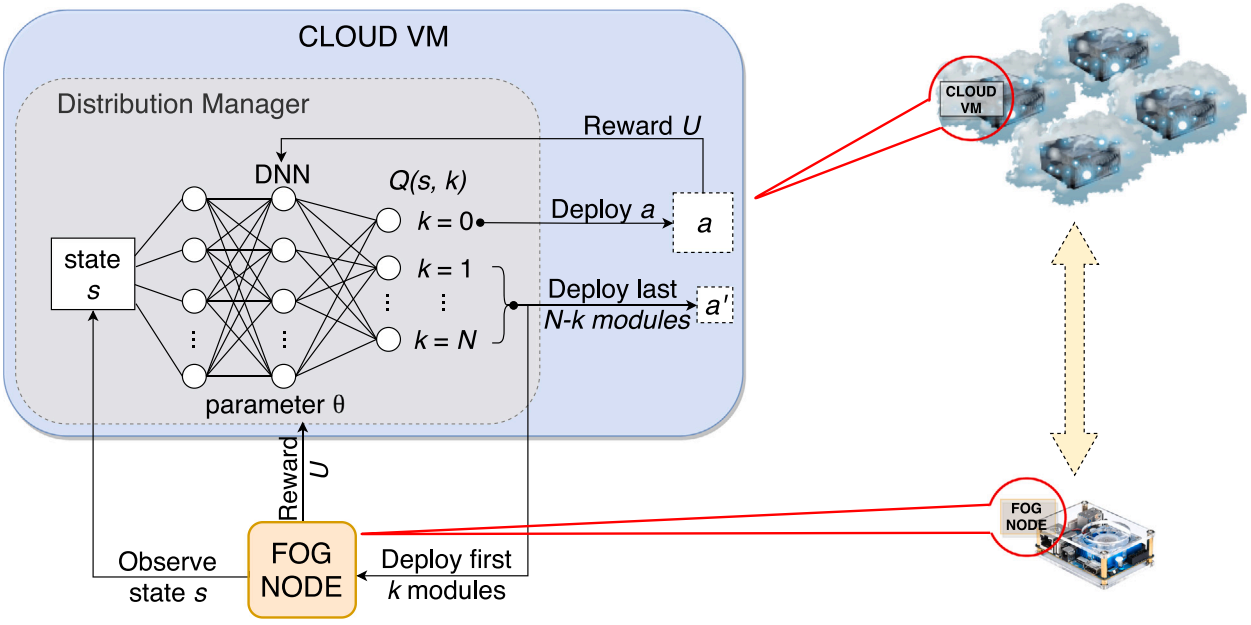


Fig. 3. A deep Q-network for a Fog computing system.

to the changing availability of Fog resources, the state of a Fog-based application is also expected to change over time. For example, moving the service to a different Fog node or changing the service that executes on the Fog. Therefore, the performance of the application may change over different redeployment rounds, which is different from the Cloud where the deployment of an application may not happen frequently over a short time. The DQN-based distribution mechanism takes the changing contexts into account when learning.

Fig. 3 describes the DQN-based context-aware distribution mechanism in a Fog environment. It involves a DM, a set of states, a set of actions per state (k values). The state s of a Fog node is requested before each deployment is made by the DM in a Cloud VM. When a Fog application is to be distributed, the current state is fed into the DQN.

The state of the Fog environment needs to be known for deploying k modules to the Fog node and $N - k$ modules on the Cloud VM. Selecting a particular k value for the distribution plan relative to a specific state provides the DM with a reward (the utility of a single deployment). The goal of the DM is to maximise the accumulated utility over m deployments. It does this by adding the maximum utility attainable from future states to the utility for achieving its current state, effectively influencing the current selection of k value by the potential future utility. This potential utility is a weighted sum of the expected values of the utility of all future steps starting from the current state.

The DQN approximates a Q function between s and k and updates the model during a series of deployments. Initially, the DM requests an observed state from a Fog node and randomly deploys the first k modules on the node. After this deployment has been completed (all user requests have been processed), the job completion time T and used Fog resources $R_{cpu/mem/str}$ are reported to calculate the utility of this deployment. Subsequently, the DM uses the utility as feedback to the neural network. In the following deployments, the DM starts to learn a predictive model to find the optimal k from the $N+1$ distribution plans such that the overall utility of m successive deployments is maximised.

Algorithm 1 describes the DQN-based distribution mechanism of a Fog application. The system continuously trains a DQN, the parameters of which are listed in Table 3. The DQN takes an input state s and outputs Q values over all possible distribution plans (k values). Episodic training is considered for the scenario of multiple deployments. An episode can be defined by different restrictions, for example a budget constraint or a maximum number of successive deployments. In this work one episode consists m successive deployments. The DQN first

Algorithm 1: DQN-based distribution mechanism

1 Initialise replay memory D with capacity d
2 Initialise k-value function Q with random weights θ
3 Initialise target k-value function $\hat{Q}$ with weights $\hat{\theta}$
4 **foreach** *episode* **do**
5 Request the Fog node for the initial state s_1
6 **for** $j \in \{1, \ldots, m\}$ **do**
7 With probability ϵ select a random value k, otherwise select $k_j = argmax_k Q(s_j, k; \theta)$
8 Deploy the first k modules of application a in the Fog node
9 Deploy the rest modules of a in the Cloud VM
10 Observe $T, R_{cpu}, R_{mem}, R_{str}, s_{j+1}$ and calculate utility U_j
11 Set $s_{j+1} = s_j$
12 Store (s_j, k_j, U_j, s_{j+1}) in replay memory D
13 **if** $len(D) > batchSize$ **then**
14 Sample random minibatch of $batchSize$ from D
15 **foreach** (s_q, k_q, U_q, s_{q+1}) **do**
16 **if** *episode terminates at step* $q+1$ **then**
17 $y_q = U_q$
18 **else**
19 $y_q = U_q + \gamma max_{k'} \hat{Q}(s_{q+1}, a'; \hat{\theta})$
20 **end**
21 Perform a gradient descent step on $(y_q - Q(s_q, k_q; \theta))^2$ with respect to θ
22 $Q = \hat{Q}$
23 **end**
24 **if** $\epsilon > \epsilon_{min}$ **then**
25 $\epsilon = v\epsilon$
26 **end**
27 **end**
28 **end**
29 **end**

initialises a replay memory specified capacity (Line 1). This is the experience replay mechanism in a DQN that reuses previous experience to improve model performance. Then the neural networks are initialised

Table 3
Parameter of the Deep Q Network.

Parameter	Description
ϵ	Exploration rate of the DNN in the Cloud agent
ϵ_{min}	Minimum exploration rate of the DNN in the Cloud agent
γ	Discount rate to calculate the future discounted reward in the DNN in the Cloud agent
υ	Decay rate to decrease the number of explorations as the DNN gets good at predictions
$batchSize$	Number of randomly sample experiences to replay in the DQN
d	Maximum number of experiences to store in the replay memory D in the DQN

with random weights (Lines 2–3). In each episode, the Fog node is requested for the initial state (Line 5), and then m deployment requests are processed (Line 6). The episode terminates when the mth deployment is completed. In each deployment, k is selected either randomly with an exploration rate ϵ or by the current Q function (Line 7). Consequently, the DM deploys the associated modules on the Fog node and the Cloud VM, respectively (Lines 8–9). After the jobs in this deployment are completed, the DM calculates the utility of this deployment. Replay memory is implemented so that past experiences (including s, k, U) is remembered and can be reused to train the model efficiently. The next state of the Fog node is also remembered (Lines 10–12). To learn from past experiences, the DM takes a random sample from its replay memory and train the current model to minimise the loss, which is the squared difference between the target and the predicted Q values (Lines 13–22). The exploration rate ϵ is continuously reduced as the model performance gets better (Lines 24–26). By defining a minimum exploration rate ϵ_{min} it is ensured that the DM explores for at least this amount of time.

Strategies: through tuning α and β, we are able to define different strategies of the context-aware distribution approach, including: (i) *Cost-effective* – when α is 0, the DQN is trained to minimise the overall cost of redistribution; (ii) *QoS-aware* – when β is 0, the DQN is trained to minimise the average application's QoS of the redistribution; (iii) *Hybrid* – when α and β are not 0, the DQN is trained to minimise the weighted sum of these two factors.

As comparisons to the context-aware approaches with the above strategies, $N + 1$ static distribution approaches of a Fog application are considered. When employing a static distribution approach, the DM keeps deploying the same k modules of a on a Fog node in the m-episode distribution.

5. Experimental evaluation

In this section, the context-aware distribution approaches presented in Section 4 are evaluated. The Fog use-cases and the hardware platform employed in this research are presented.

5.1. Fog use-cases

Two applications are employed for evaluating the context-aware distribution mechanism. The first is a real-time face detection application, and the second is a location-based mobile game. Both use-cases are server-based and a natural fit for Fog computing since they are latency critical. The chosen use-cases also represent different workloads that can benefit from Fog computing: the mobile game represents an online application whose Fog server responds to incoming user requests; the face detection workload is representative of a data-intensive streaming application.

5.1.1. Real-time face detection

The face detection application is Cloud server-based. An end device with an embedded video camera captures a continuous video stream and transmits it to the Cloud server. The goal of the application is to detect faces from each individual video frame by using Pillow.[2] and OpenCV[3] Typically, the application streams the video to the Cloud, and all processing is performed on the Cloud server. In the experiments, the images captured in real-time are recorded once and repeatedly used in order to ensure that our comparisons are even.

By employing Fog computing (data processing near where it is generated), the amount of data transferred to the Cloud can be reduced, thereby minimising communication latencies, while obtaining reasonable overall system performance. The application is a good fit for Fog computing since firstly, it is latency-critical and bandwidth consuming – response time is heavily affected by the distance between user devices and the Cloud server. Secondly, a subset of the services from the Cloud can be brought closer to devices to reduce the amount of data transferred to the Cloud.

The server application is modular and comprises three services ($N = 3$) for detecting faces from a frame of the video: (i) *Grey-scale converter* is a data pre-processing service. As a result of this component, only one-third of the size of data will be processed in the remaining procedures. (ii) *Motion detector* is a data filtering service to reduce computations spent on similar frames streamed to the server. (iii) *Face detector* is a computationally expensive service that identifies frontal human faces in a video frame using machine learning.

This application can be distributed in the following four ways: as shown in Fig. 4: (i) Cloud-only services($k = 0$) – all the services mentioned above are deployed in a Cloud VM; (ii) Fog-based pre-processing ($k = 1$) – the grey-scale converter is deployed in a Fog node and the other services deployed in a Cloud VM; (iii) Fog-based data filtering ($k = 2$) – the grey-scale converter and the motion detector are deployed in a Fog node and the face detector is deployed in a Cloud VM; (iv) Fog-only services ($k = 3$) – all the services are deployed in a Fog node. If the same distribution plan from the above four is employed for a series of deployments, then the approach is considered as static. The context-aware distribution mechanism proposed in this paper dynamically (re)selects one of the above four distribution plans in order to maximise performance by considering the varying states of a Fog node presented in Section 4.

5.1.2. Location-based mobile game

The application is an open-source mobile game similar to Pokémon Go, named iPokeMon. iPokeMon comprises a client[4] for the iOS platform and a server[5] that is hosted on a public Cloud. The iPokeMon game server was redesigned to be hosted on the Cloud and a Fog node. The Fog hosts a game server that handles requests from recognised users whose data exists in the game database. The Cloud hosts the original iPokeMon server that is able to handle requests from new users whose data does not exist in the game database.

The iPokeMon server is tested using Apache JMeter.[6] 100 HTTP requests during a connection (a user is playing the iPokeMon game) between the user device and the original Cloud server is recorded. During this time, the number and type of requests and the parameters sent through the requests are recorded. Subsequently, JMeter replays the user requests in the experiments.

This application could be distributed in three ways as illustrated in Fig. 5: (i) Cloud-only services ($k = 0$) – the original iPokeMon server is hosted in a Cloud VM; (ii) Fog-based services ($k = 1$) – the functionality

[2] https://pillow.readthedocs.io
[3] https://opencv.org
[4] https://github.com/Kjuly/iPokeMon
[5] https://github.com/Kjuly/iPokeMon-Server
[6] http://jmeter.apache.org/

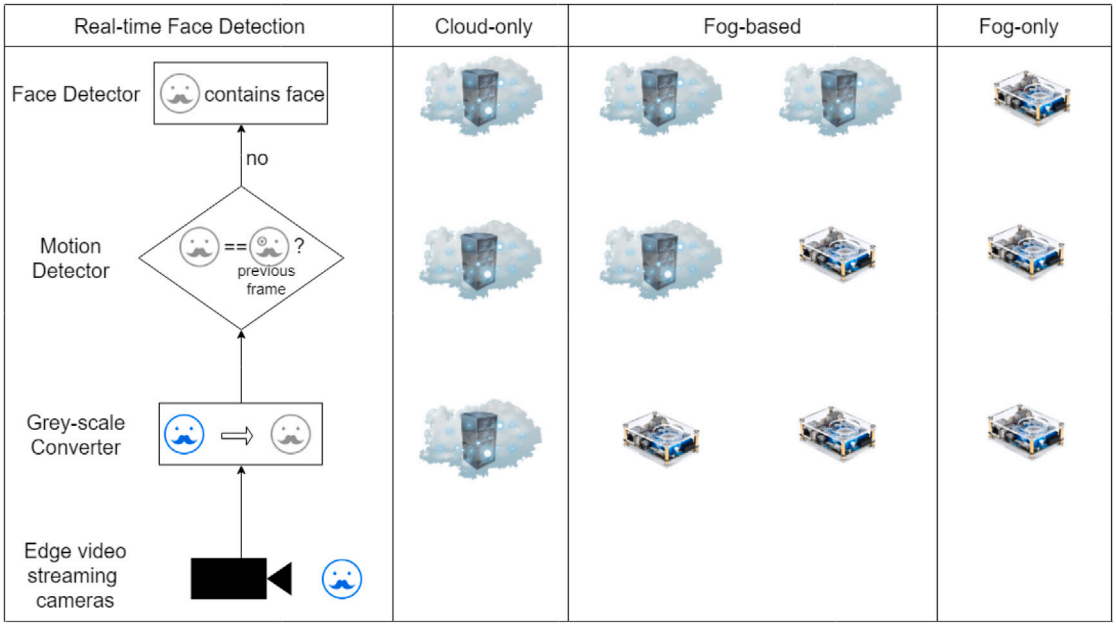


Fig. 4. The Cloud-only, Fog-based and Fog-only distribution options for the real-time face detection use-case.

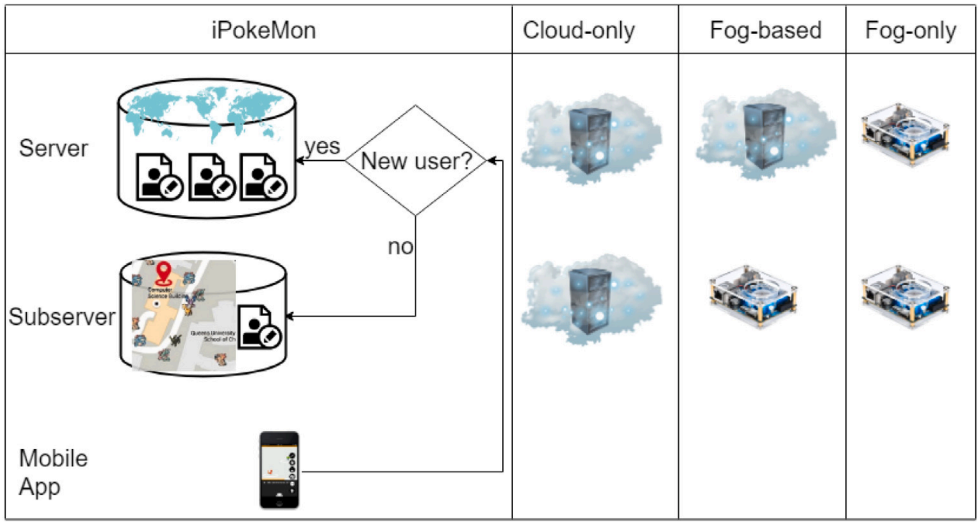


Fig. 5. The Cloud-only, Fog-based and Fog-only distribution options for the iPokeMon use-case.

and database to service existing users are hosted in a Fog node and the functionality and database to service new users are hosted in a Cloud VM; (iii) Fog-only services ($k = 2$) – the original iPokeMon server is hosted in a Fog node. The static and context-aware approaches for distributing iPokeMon is defined the same as for FD above.

5.2. Experimental setup and approaches evaluated

Setup: A Device–Fog–Cloud system is set up using a laptop connected to a router via Wi-Fi, an ODROID-XU4 board connected to the same router via Ethernet, and a t2.micro VM which is running Ubuntu 14.04 LTS provided by AWS Elastic Compute Cloud from its Dublin data centre. The Fog node is located in the Computer Science Building of Queen's University Belfast in Northern Ireland. The ODROID board has 2 GB of DRAM memory, and one ARM Big.LITTLE architecture Exynos 5 Octa processor running Ubuntu.

Table 4 displays the default values of parameters used in the experiments. As defined in Section 3.1, the distribution problem is considered as a series of m successive deployments and the objective is to maximise the overall utility defined in Eq. (1). The values of the parameters α and β in the utility function are by default -1, such that the utility value increases when the application latency and the service cost decreases. In other words, if an application takes a long time to respond a user request and/or the overall cost of deploying the application on the

Table 4
Default parameter values.

Parameter	Value	Parameter	Value	Parameter	Value
λ	0.01	γ	0.95	P_{str}	0.000032
m	20	ϵ	1	P_{mem}	0.005458
α	−1	ϵ_{min}	0.01	P_{cpu}	0.04073
β	−1	υ	0.99	$hiddenLayerNode$	24
$batchSize$	5	$hiddenLayer$	2	$learningRate$	0.001
P_C	0.0132				

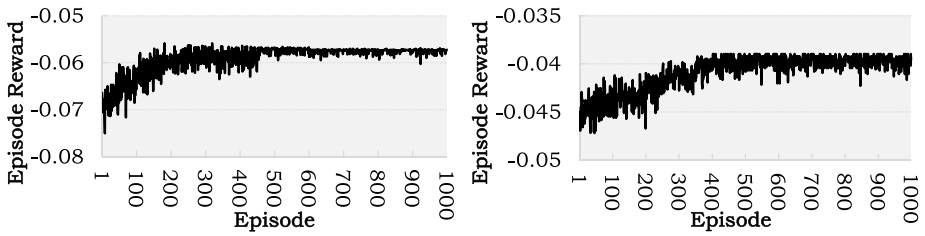


Fig. 6. Learning curve of the hybrid context-aware approach showing the training improves the episode reward.

Cloud/Fog, it will have low utility value and vice versa. A total of 20 deployments are considered in each experiment for the DM to learn for 1,000 episodes, which results in the experiment being run for over 33 h. The prices of Cloud computing resources ($P_C, P_{cpu/mem/str}$) are defined as US dollars per hour for a single unit of the resources (a VM, vCPU core, 1-GB memory, 1-GB storage). This is how current Cloud Infrastructure-as-a-Service such as Google Compute Engine[7] is priced. In the context-aware distribution mechanism, a DQN with two hidden layers and 24 nodes is learnt during the 20 deployments (Algorithm 1) to model the relationship between the system state of a Fog node and the k value. The DQN is implemented with the Keras[8] deep learning library, with the learning rate of 0.001 and 5 memorised experience to replay.

To create a realistic Fog computing node, which is likely to have variable resource availability over time, the Linux package *stress*,[9] is chosen to stress test the Fog node. The CPU and memory of the Fog node are divided into eight units, each unit comprising 1 CPU core and 256 megabytes of memory. For the experiments in Sections 6.2 and 6.3 the Fog node was stressed continuously — the cores execute a workload that consumes a random value of $[0, 7]$ and the memory is flooded. The random number is changed every 10 s. This ensures that the system state changes on the Fog node. The same sequence of the stress tests is applied to all distribution approaches to ensure that system states are similar in all experiments.

Approaches Evaluated: Since the application relies on three services, there are consequently four static approaches (the Cloud-only services, Fog-based pre-processing, Fog-based data filtering, and the Fog-only services) with $k \in \{0 \dots 3\}$. A static approach (*S-k*) means that for the 20 successive deployments in each distribution, no matter what the state of the Fog node is, the Distribution Manager always deploys the first k modules on the Fog node and the remaining $N-k$ modules on the Cloud VM. On the contrary, the context-aware approaches (*Context-aware*) employ Algorithm 1 and dynamically assign a k value using DQN in each deployment.

6. Results

The experiments provide insight into the benefits of using the context-aware distribution mechanism of elastic Fog applications. The system overhead of the online decision-making process is discussed before the three context-aware approaches and four static distribution approaches are compared by measuring the application performance. A further discussion on the impacts of varying parameters in the system is presented.

6.1. Training and overhead

Fig. 6 displays the episode reward (i.e. the accumulated utility of 20 successive deployments) when the DQNs in the Distribution Manager of the FD application 6(a) and iPokeMon 6(b) for up to 1,000 episodes. For both use-cases, the episode reward is observed to gradually increase with the number of episodes, which means the DQN is getting better performance with more experiences. The episode reward reaches and stays in its maximum point from around 600 and 400 episodes, at which point a total of 12,000 and 8,000 deployments have been carried out by the Distribution Manager for FD and iPokeMon respectively. This means the DQNs have learnt an optimal distribution plan for the 20 successive deployments and could not increase the utility anymore. Therefore, for the analysis of the context-aware distribution approaches (including QoS-aware, cost-effective and hybrid) in the following sections, we present the DQNs trained with 600 and 400 episodes for FD and iPokeMon respectively, except specified otherwise.

Table 5
Statistics of the time (in milliseconds) taken by each decision making of the hybrid DQNs trained for FD and iPokeMon.

Minimum	1st Quartile	Median	Mean	3rd Quartile	Maximum
85	124	136	139.5	149	248

As a multi-tenant computing environment, the resource availability of a Fog node is expected to change rapidly. For example, the system state acquired by the Distribution Manager may become invalid if the context-aware distribution mechanism takes a long time to choose a distribution plan (k value). Therefore, the shorter the time taken by each execution of the decision-making process (Lines 7–26 in Algorithm 1), the more real-time response is achieved. Each time the Distribution Manager generates a redistribution plan, it takes 85–250 ms (Table 5). This overhead may be ignored when compared to the time taken for processing a single video frame using the Cloud-only method, which is nearly 2 s from empirical results. In the iPokeMon use-case, this overhead translates into the time taken to process 1–2 iPokeMon requests using the Cloud-only distribution method. Since each deployment of the iPokeMon server is expected to process a large number of user requests, the overhead is negligible.

6.2. Performance

It is observed from empirical results that the DQNs converges after 400 episodes of training for the chosen use-cases. Therefore, the results presented in this section are obtained during the 400th–500th episode of the DQNs. The 100 episodes involves 2,000 (i.e. $100 * m$) deployments of the applications in total.

Figs. 7 and 8 illustrate the distribution of the utility of each experiment (including 20 deployments) when the static and context-aware approaches are employed for both use-cases. The utility values in the experiments are negative because as defined in Eq. (1), the value is negatively affected by both the application latency and the service cost. In other words, if in a particular deployment an application takes a long time to respond a user request and/or the overall cost of deploying the application on the Cloud/Fog is large, this deployment will have low utility value and vice versa.

[7] https://cloud.google.com/compute/pricing
[8] https://keras.io/
[9] http://manpages.ubuntu.com/manpages/trusty/man1/stress.1.html

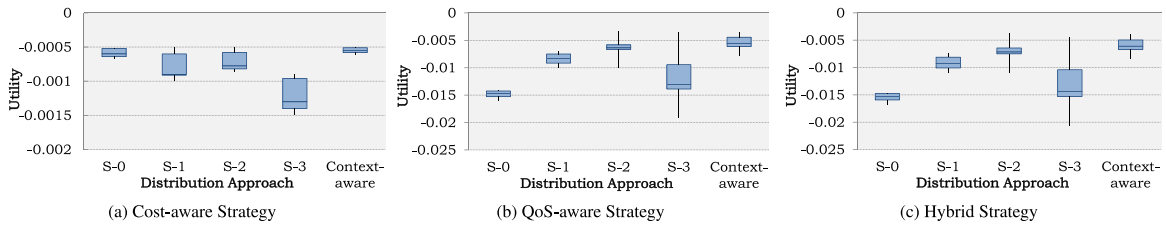


Fig. 7. Distribution of utilities over 100 experiments when applying different strategies to the FD application.

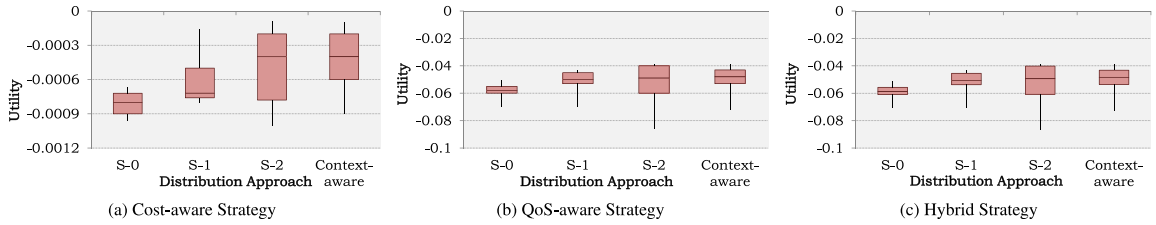


Fig. 8. Distribution of utilities over 100 experiments when applying different strategies to the iPokeMon application.

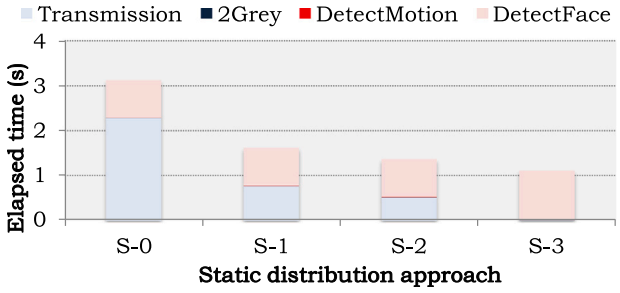


Fig. 9. Average time taken to process a single video frame in FD using static distribution approaches.

Three combinations of (α, β) are considered: (i) $(0, -1)$, which represent a cost-aware strategy (Fig. 7(a) and 8(a)); (ii) $(-1, 0)$, which represents a QoS-aware strategy (Fig. 7(b) and 8(b)); and (iii) the default setting, i.e. $(-1, -1)$, which represent a hybrid strategy (Fig. 7(c) and 8(c)).

It is observed for both use-cases that, for all strategies, context-aware approach outperforms the static approaches. For example, when the DQN is trained to minimise cost for FD (Fig. 7(a)), it achieves the same maximum utility as *S-1* and *S-2*. In the experiments, *S-n* refers to a static approach that always deploy the first n modules of an application onto the Fog. An improvement over *S-1, S-2* and *S-3* are indicated by the fact that its third quartile of the utility (−0.00058) is larger than the median of the utility obtained by the three static approaches (−0.0006, −0.0009 and −0.0008 respectively). The minimum utility achieves by the DQN is higher than the maximum utility achieves by *S-3*, which means that the cost-optimised DQN never deploy the entire application on the Fog node. This is because, with more modules deployed on the Fog node, there is more cost of using the Fog resources, even though the overall job completion time is reduced.

Similarly, the context-aware approach for iPokeMon (Fig. 8(a)) tends to benefit from the lessons learnt through *S-1* and *S-2*. It achieves the maximum (−0.00009) and median (−0.0004) utilities that are close to *S-2*, while successfully improves the third quartile (−0.0006) when compared to *S-2* (−0.00078). This is due to the Distribution Manager's selection of *S-1* in a number of deployments.

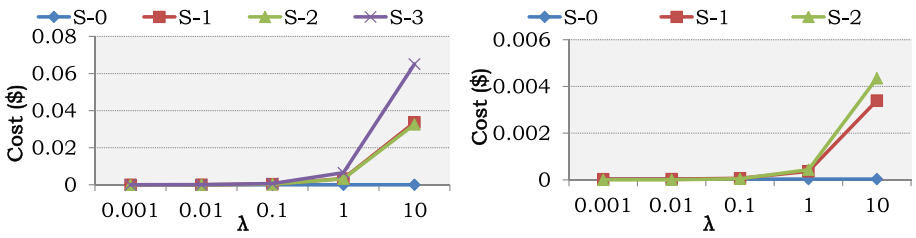
The benefit of applying the context-aware approach is more obvious for FD compared to iPokeMon when the DQN is trained to maximise the QoS (i.e. to minimise the job completion time, Fig. 7(b)). The majority of the utilities achieved by the context-aware approach is larger than that achieved by all four static approaches. To better understand the difference of the QoS when applying different static approaches, Fig. 9 provides a breakdown of the average time to process a single video frame when the Fog node system is not stress-tested (i.e. almost all CPU cores and memory are available). The overall time is divided into the data transmission time, the time taken by the three modules of the application — grey-scale conversion, motion detection, and face detection. It is inferred that the main difference of the QoS comes from the data transmission time. For example, by applying $S-1, S-2$ and $S-3$, the transmission time is reduced from 2.28 s (s) to 0.77s, 0.52s and 0.11s respectively. The first two modules of the application, namely grey-scale conversion and motion detection take a short time between 0.003 and 0.004, no matter where they are hosted. The face detection module causes the other main difference among these approaches as there is a 0.2s delay observed when it is hosted on the Fog node instead of the Cloud VM. Such differences of QoS among the static approaches are expected to be magnified in the experiment for Fig. 7(b) when the Fog resources are deliberately restricted. Therefore, by in the context-aware approach, the DQN tends to only choose the optimal deployment from *S-1*, *S-2* and *S-3* to avoid the long transmission time caused by *S-0*.

When applying the QoS-aware strategy to iPokeMon 8(b), the context-aware approach improves the overall application performance over the static approaches by achieving the highest median (−0.048) and third quartile (−0.053) values of utility. However, in the best and worst cases, the context-aware approach performs slightly worse than *S-3* by 0.3% and *S-1* by 2%. This difference is indicative that the context-aware approach has more benefits for FD 7(b) since there is a larger QoS gain over iPokeMon.

When we consider both the cost and the QoS (Fig. 7(c) and 8(c)), the distribution of all approaches for both use-cases is similar to the QoS-aware strategy. The reason is that with the default parameter setting, the difference of QoS among the approaches happens to have a larger influence than the difference in costs. Therefore, in the next section, we further investigate the impact of different parameter settings on the utility.

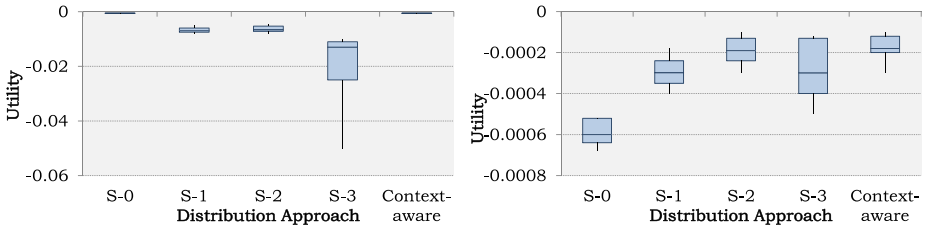
6.3. Impact of utility parameters

From empirical study, we found that when assigning different weights to α and β (i.e. the relative importance of QoS and cost), the performance of all approaches varies. Another factor that makes a difference is λ, which indicates how expensive the Fog resources are priced compared to the Cloud resources. Hence, several different values



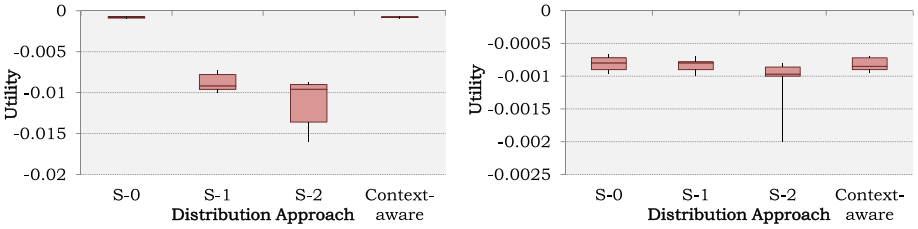
(a) FD (b) iPokeMon

Fig. 10. Average cost of a single deployment using static distribution approaches with varying λ.



(a) $\lambda = 0.1$ (b) $\lambda = 0.001$

Fig. 11. Distribution of utilities over 100 experiments when $\alpha/\beta = 0$ with varying λ for the FD application.



(a) $\lambda = 1$ (b) $\lambda = 0.1$

Fig. 12. Distribution of utilities over 100 experiments when $\alpha/\beta = 0$ with varying λ for iPokeMon.

for (α, β) and λ are applied to explore their impacts on the distribution approaches.

Fig. 10 presents the relationship between the average cost of a single deployment and λ for FD and iPokeMon when the Fog node system is not stress tested. It is noted that in the FD use case although the Fog computing services effectively improve the QoS (Fig. 9), its cost increases drastically when λ is larger than 0.1 10(a). For example, the cost of *S-3* is as 200 and 2,000 times large as the cost of *S-0*, when λ is 1 and 10 respectively. In such cases, when $\beta \neq 0$ the context-aware approach would always tend to choose *S-1* in order to minimise the cost factor in Eq. (1). This would not distinguish the context-aware approach from the static approaches. For the iPokeMon use case 10(b), the increase in cost is moderate. For instance, the cost of *S-2* is close to 0.1, 1.3, and 13 times of *S-0* when λ is 0.01, 0.1 and 1 respectively. Therefore, in the following experiments we consider $\lambda \in \{0.0001, 0.001, 0.1\}$ for FD and $\lambda \in \{0.001, 0.1, 1\}$ for iPokeMon to explore the impact of λ over the application performances. We also present the analysis of varying importance assigned to the QoS and cost factors over the application performances, by applying $\lambda/\beta \in \{0.1, 1, 10\}$ for both use-cases.

Fig. 11 displays the performance of different approaches when $\alpha/\beta = 0$ (i.e. the cost-efficient strategy) with the higher and lower values of λ in addition to the medium λ (Fig. 7(a)). The Fog resources are randomly restricted as in Section 6.2. When $\lambda = 0.1$, i.e. Fog resources are priced as one-tenth of Cloud resources, the context-aware approach acts the same as *S-0* since *S-0* has a clear advantage over the other static approaches. When $\lambda = 0.001$, i.e. Fog resources are priced as one-thousandth of Cloud resources, the context-aware approach can benefit from the Fog services. Note that under such an assumption the overall costs of Fog-based distribution approaches (*S-1, S-2* and *S-3*) become less than the Cloud-only distribution approach (*S-0*). When $\lambda = 0.01$ (Fig. 7(a)), i.e. Fog resources are priced as one hundredth of Cloud resources, the context-aware approach benefits from some of the Fog-based distribution approaches (*S-1* and *S-2*). In this case, the costs of Fog-based distribution approaches are slightly more than the Cloud-only distribution approach.

Fig. 12 displays the performance of iPokeMon when different approaches are applied with the cost-efficient strategy and with the higher and medium values of λ in addition to the lower λ (Fig. 8(a)). When $\lambda = 1$ (i.e. Fog resources are priced the same as Cloud resources), the context-aware approach acts the same as *S-0* since *S-1* has a significant advantage over the other static approaches. When $\lambda = 0.1$, the distribution of utility in the context-aware approach is close to *S-0*, with a variance in the median value caused by the occasional selection of *S-2* and *S-3*. When $\lambda = 0.01$ (Fig. 8(a)), the context-aware approach mainly benefits from the Fog-based, and Fog-only distribution approaches as the costs of these deployment plans are less than the Cloud-only distribution approach.

Fig. 13 presents the performance of FD when applying different λ values with the hybrid strategy when $\alpha/\beta = 0.1$, i.e. when the cost is considered as 10 times important as the QoS. It is found that when $\lambda = 0.1$, the gaps between the five approaches are similar to the gaps observed in the cost-aware strategy (Fig. 11(a)). When λ is 0.01 (Fig. 13(b)) or 0.001 (Fig. 13(c)), the context-aware approach is no longer dominated by the most cost-efficient deployment (i.e. *S-0*) and is able to benefit from the Fog-based and Fog-only deployment

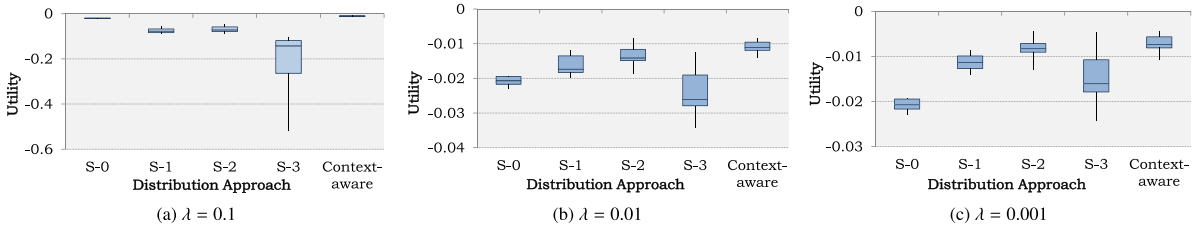


Fig. 13. Distribution of utilities over 100 experiments when $\alpha/\beta = 0.1$ with varying λ for the FD application.

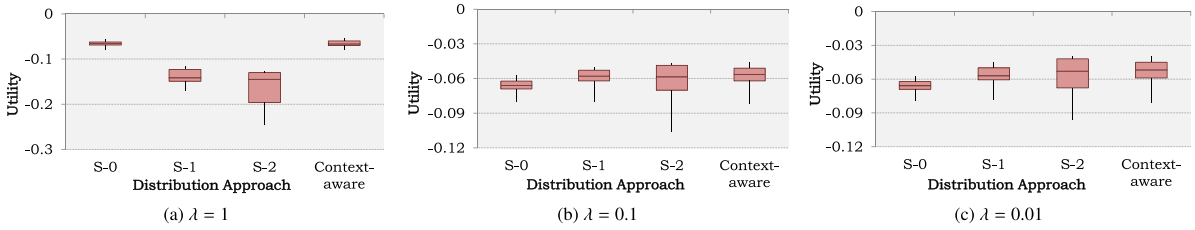


Fig. 14. Distribution of utilities over 100 experiments when $\alpha/\beta = 0.1$ with varying λ for iPokeMon.

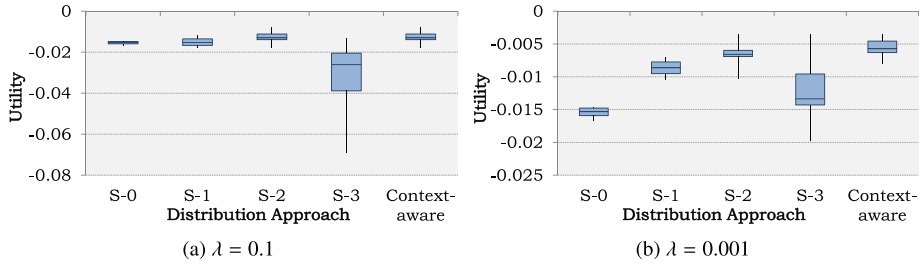


Fig. 15. Distribution of utilities over 100 experiments when $\alpha/\beta = 1$ with varying λ for the FD application.

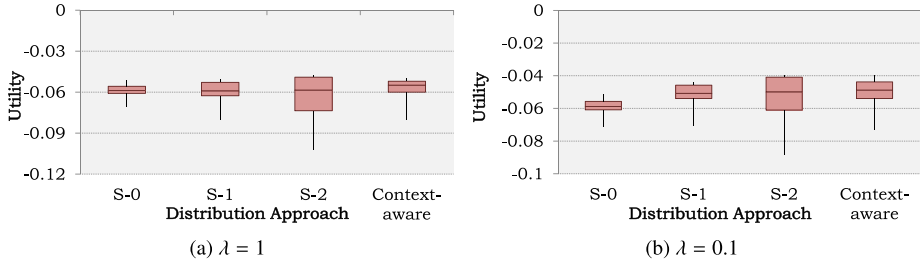


Fig. 16. Distribution of utilities over 100 experiments when $\alpha/\beta = 1$ with varying λ for iPokeMon.

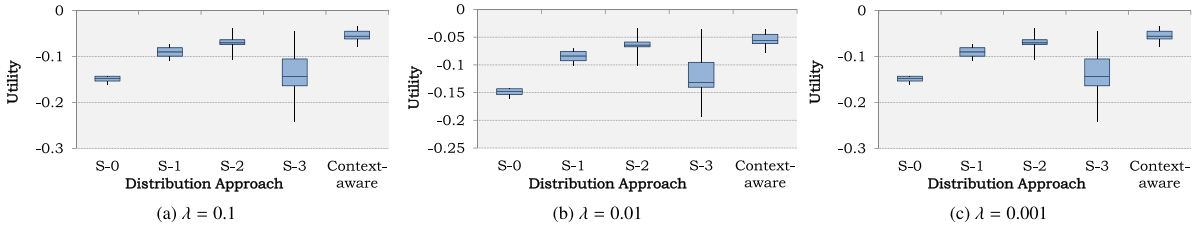
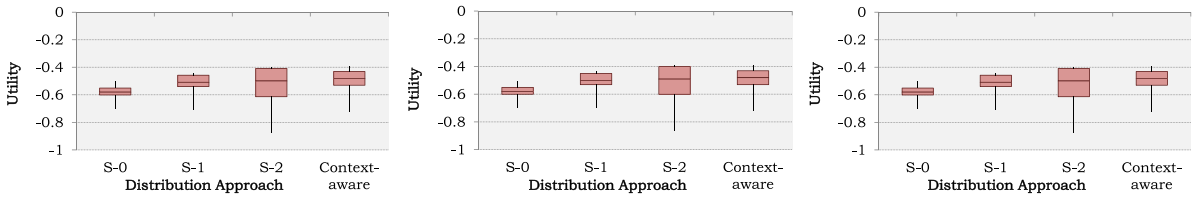


Fig. 17. Distribution of utilities over 100 experiments when $\alpha/\beta = 10$ with varying λ for the FD application.

plans. This is because by assigning importance to the QoS factor, the context-aware approach starts to acknowledge the large reductions of application latency introduced by *S-2* and *S-3*. The context-aware approach, regardless of λ values, outperforms the static approaches with the higher utility in the interquartile range.

Similarly, the iPokeMon use-case is tested with high, medium, and low λ values (1, 0.1, 0.01) with the hybrid strategy when $\alpha/\beta = 0.1$ (Fig. 14). It is found that when $\lambda = 1$, the context-aware approach is dominated by *S-0* (i.e. the Cloud-only deployment), as was seen in the cost-aware strategy (Fig. 12(a)). When λ is 0.1 (Fig. 14(b)) or 0.01 (Fig. 14(c)), the application performance achieved by applying static approaches is comparable. The context-aware approach can achieve the best performance with the highest median value of the utility, though the performance gain in this use-case is less significant compared to the FD use-case. We interpret this finding as a result of the fact that the QoS improvement in iPokeMon is less significant than that in FD.



(a) $\lambda = 1$ (b) $\lambda = 0.1$ (c) $\lambda = 0.01$

Fig. 18. Distribution of utilities over 100 experiments when $\alpha/\beta = 10$ with varying λ for iPokeMon.

Fig. 15 displays the default hybrid strategy applied to the FD use-case, with the higher and lower values of λ in addition to the medium value in Fig. 7(c). Since the cost and QoS factors are of equal importance in this strategy, the context-aware approach does not keep selecting the most cost-efficient *S-0* as seen in the previous strategies with a high λ value. With this hybrid strategy, regardless of λ values, the context-aware approach can benefit from Fog-based distribution approaches. The interquartile range of the utility when applying the context-aware approach is either close to or higher than the other static approaches, which means the context-aware approach is adaptable to varying pricing methods and outperforms the static approaches. Similar finding is observed when distributing iPokeMon with the hybrid strategy with high (Fig. 16(a)), medium (Fig. 16(b)), and low (Fig. 8(c)) λ values.

When QoS is considered 10 times more important than cost in Eq. (1), little difference is observed when tuning λ in both use-cases (Fig. 17 and 18. The impact of λ in this specific hybrid strategy is mitigated by the diminished importance of running cost. The benefits of the context-aware approach are more significant in FD than in the iPokeMon use-case. This is because the performance gain achieved on the Fog-based FD is larger than on the Fog-based iPokeMon.

7. Conclusions

Native Cloud applications exploit microservices architecture in which an application is composed of multiple services that are geographically distributed. Cloud applications can leverage edge resources (referred to as Fog nodes) to improve their overall QoS in a computing model referred to as Fog computing. Fog nodes are resource constrained when compared to the Cloud and may be available intermittently. Therefore, distributing the application across the Cloud and Fog is not trivial.

The key challenge addressed in this paper is the distribution of a modular application, comprising multiple services across the Cloud and Fog in a dynamic manner. To tackle the challenge, a context-aware mechanism was proposed that dynamically generates (re)deployment plans for the application to maximise the performance efficiency of the application by taking the overall QoS and running costs into account. The mechanism relies on deep RL to generate a distribution strategy without prior knowledge of the available resources on the Fog nodes, network conditions, and the Fog application.

The above context-aware distribution approach is validated on two use-cases, namely a real-time face detection application and a location-based mobile game. Both these are representative of real workloads. The experimental results obtained from different distribution approaches for the chosen use-cases highlight the following: (i) the context-aware distribution mechanism can increase the utility on average by 50% for the FD application and 20% for iPokeMon, when compared to static approaches; (ii) when cost is the dominant factor that affects the utility, if Fog resources are priced less than the Cloud the DM is motivated to select Fog-based deployments. Otherwise the DM sticks to Cloud-only deployments as it is the most economical solution; (iii) when QoS is the dominant factor that affects utility, the context-aware distribution mechanism outperforms the static approaches by up to 60% for the face detection use-case and 25% for iPokeMon.

Limitations and Future Work: The limitations of the current work are: (i) Multi-tenancy is not considered in this paper. When multiple Fog applications share the same Fog node, the context changes become the result of the actions taken by all applications' DMs. (ii) The impact of different Cloud pricing models is not investigated in this paper. Other Cloud services, such as container-based services and serverless computing, maybe more cost-efficient than the VM-based services used in this work. (iii) The context-aware offloading solution is evaluated on a small-scale Cloud–Fog–Device as a prototype. In the future, we aim to extend this work to address the above.

CRediT authorship contribution statement

Nan Wang: Conceptaulization, Methdology, Software, Validation, Formal analysis, Investigation, Writing – original draft. **Blesson Varghese:** Conceptualization, Methodology, Investigation, Resources, Writing – review & editing, Supervision, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Alam, M.G.R., Tun, Y.K., Hong, C.S., 2016. Multi-agent and reinforcement learning based code offloading in mobile fog. In: Int. Conf. on Information Networking. pp. 285–290.

Báguena, M., Samaras, G., Pamboris, A., Sichitiu, M.L., Pietzuch, P., Manzoni, P., 2016. Towards enabling hyper-responsive mobile apps through network edge assistance. In: IEEE Annual Consumer Communications and Networking Conference. IEEE, pp. 399–404.

Bahreini, T., Grosu, D., 2017. Efficient placement of multi-component applications in edge computing systems. In: ACM/IEEE Symp. on Edge Comput.. ACM, p. 5.

Bittencourt, L.F., Diaz-Montes, J., Buyya, R., Rana, O.F., Parashar, M., 2017. Mobility-aware application scheduling in fog computing. IEEE Cloud Comput. 4 (2), 26–35.

Bonomi, F., Milito, R., Zhu, J., Addepalli, S., 2012. Fog computing and its role in the IoT. In: Wksp. on Mobile Cloud Comp.. pp. 13–16.

Brogi, A., Forti, S., 2017. Qos-aware deployment of IoT applications through the fog. IEEE Internet Things J. 4 (5), 1185–1192.

Chen, X., 2014. Decentralized computation offloading game for mobile cloud computing. IEEE Trans. Parallel Distrib. Syst. 26 (4), 974–983.

Chen, Z., Hu, W., Wang, J., Zhao, S., Amos, B., Wu, G., Ha, K., Elgazzar, K., Pillai, P., Klatzky, R., et al., 2017. An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance. In: ACM/IEEE Symp. on Edge Comput.. ACM.

Chen, Z., Jiang, L., Hu, W., Ha, K., Amos, B., Pillai, P., Hauptmann, A., Satyanarayanan, M., 2015. Early implementation experience with wearable cognitive assistance applications. In: Workshop on Wearable Systems and Applications. ACM, pp. 33–38.

Deb, P.K., Misra, S., Mukherjee, A., 2021. Latency-aware horizontal computation offloading for parallel processing in fog-enabled IoT. IEEE Syst. J..

Dias de Assunção, M., da Silva Veith, A., Buyya, R., 2018. Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. J. Netw. Comput. Appl. 103, 1–17.

Dinh, T.Q., La, Q.D., Quek, T.Q.S., Shin, H., 2018. Learning for computation offloading in mobile edge computing. IEEE Trans. Commun. 66 (12), 6353–6367.

Do, C.T., Tran, N.H., Pham, C., Alam, M.G.R., Son, J.H., Hong, C.S., 2015. A proximal algorithm for joint resource allocation and minimizing carbon footprint in geo-distributed fog computing. In: Int. Conf. on Information Networking. IEEE, pp. 324–329.

Donassolo, B., Fajjari, I., Legrand, A., Mertikopoulos, P., 2019. Fog based framework for IoT service provisioning. In: Proc. of the IEEE Consumer Communications and Networking Conf..

Eom, H., Figueiredo, R., Cai, H., Zhang, Y., Huang, G., 2015. Malmos: Machine learning-based mobile offloading scheduler with online training. In: IEEE Int. Conf. on Mobile Cloud Comp., Services, and Eng..

Gao, L., Dahlin, M., Nayate, A., Zheng, J., Iyengar, A., 2003. Application specific data replication for edge services. In: Int. Conf. on World Wide Web. ACM, pp. 449–460.

Guevara, J.C., da S. Torres, R., da Fonseca, N.L., 2020. On the classification of fog computing applications: A machine learning perspective. J. Netw. Comput. Appl. 159, 102596.

Gupta, H., Vahid Dastjerdi, A., Ghosh, S.K., Buyya, R., 2017. IFogSim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. Softw. - Pract. Exp. 47 (9), 1275–1296.

Haj-Ali, A., Ahmed, N.K., Willke, T., Gonzalez, J., Asanovic, K., Stoica, I., 2019. A view on deep reinforcement learning in system optimization. arXiv:arXiv:1908.01275.

He, J., Wei, J., Chen, K., Tang, Z., Zhou, Y., Zhang, Y., 2018. Multitier fog computing with large-scale IoT data analytics for smart cities. IEEE Internet Things J. 5 (2), 677–686.

He, Y., Yu, F.R., Zhao, N., Leung, V.C., Yin, H., 2017. Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach. IEEE Commun. Mag. 55 (12), 31–37.

Hong, C.-H., Varghese, B., 2019. Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms. ACM Comput. Surv. 52 (5).

Hosseinpour, F., Plosila, J., Tenhunen, H., 2016. An approach for smart management of big data in the fog computing context. In: IEEE Int. Conf. on Cloud Computing Technology and Science. IEEE, pp. 468–471.

Hu, P., Dhelim, S., Ning, H., Qiu, T., 2017. Survey on fog computing. J. Netw. Comput. Appl. 98 (C), 27–42.

Islam, M.S.U., Kumar, A., Hu, Y.-C., 2021. Context-aware scheduling in fog computing: A survey, taxonomy, challenges and future directions. J. Netw. Comput. Appl. 180, 103008.

Junior, W., Oliveira, E., Santos, A., Dias, K., 2019. A context-sensitive offloading system using machine-learning classification algorithms for mobile cloud environment. Future Gener. Comput. Syst. 90, 503–520.

Lin, Y., Kemme, B., Patino-Martinez, M., Jimenez-Peris, R., 2007. Enhancing edge computing with database replication. In: IEEE Int. Symp. on Reliable Distributed Systems. IEEE, pp. 45–54.

Lin, H., Zeadally, S., Chen, Z., Labiod, H., Wang, L., 2020. A survey on computation offloading modeling for edge computing. J. Netw. Comput. Appl. 169, 102781.

Lu, H., Gu, C., Luo, F., Ding, W., Liu, X., 2020. Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning. Future Gener. Comput. Syst. 102, 847–861.

Machida, F., Andrade, E., 2021. PA-Offload: Performability-aware adaptive fog offloading for drone image processing. In: 5th IEEE International Conference on Fog and Edge Computing. pp. 66–73.

Mahmud, R., Ramamohanarao, K., Buyya, R., 2018. Latency-aware application module management for fog computing environments. ACM Trans. Internet Technol. 19 (1), 9.

McChesney, J., Wang, N., Tanwer, A., de Lara, E., Varghese, B., 2019. DeFog: Fog computing benchmarks. In: ACM/IEEE Symp. on Edge Comp.. pp. 47–58.

Meurisch, C., Gedeon, J., Nguyen, T.A.B., Kaup, F., Muhlhauser, M., 2017. Decision support for computational offloading by probing unknown services. In: IEEE Int. Conf. on Computer Comm. and Networks.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529.

Nayeri, Z.M., Ghafarian, T., Javadi, B., 2021. Application placement in fog computing with AI approach: Taxonomy and a state of the art survey. J. Netw. Comput. Appl. 185, 103078.

Ndikumana, A., Tran, N.H., Ho, T.M., Han, Z., Saad, W., Niyato, D., Hong, C.S., 2019. Joint communication, computation, caching, and control in big data multi-access edge computing. IEEE Trans. Mob. Comput..

Osanaiye, O., Chen, S., Yan, Z., Lu, R., Choo, K.-K.R., Dlodlo, M., 2017. From cloud to fog computing: A review and a conceptual live vm migration framework. IEEE Access 5, 8284–8300.

Park, S.-W., Park, J., Bong, K., Shin, D., Lee, J., Choi, S., Yoo, H.-J., 2015. An energy-efficient and scalable deep learning/inference processor with tetra-parallel MIMD architecture for big data applications. IEEE Trans. Biomed. Circuits Syst. 9 (6), 838–848.

Rui, L., Zhang, M., Gao, Z., Qiu, X., Wang, Z., Xiong, A., 2021a. Service migration in multi-access edge computing: A joint state adaptation and reinforcement learning mechanism. J. Netw. Comput. Appl. 183–184.

Rui, L., Zhang, M., Gao, Z., Qiu, X., Wang, Z., Xiong, A., 2021b. Service migration in multi-access edge computing: A joint state adaptation and reinforcement learning mechanism. J. Netw. Comput. Appl. 183–184, 103058.

Sarkar, S., Chatterjee, S., Misra, S., 2018. Assessment of the suitability of fog computing in the context of internet of things. IEEE Trans. Cloud Comput. 6 (1), 46–59.

Satyanarayanan, M., 2017. The emergence of edge computing. Computer 50 (1), 30–39.

Su, Z., Xu, Q., Luo, J., Pu, H., Peng, Y., Lu, R., 2018. A secure content caching scheme for disaster backup in fog computing enabled mobile social networks. IEEE Trans. Ind. Inf. 14 (10), 4579–4589.

Thai, L., Barker, A., Varghese, B., Akgun, O., Miguel, I., 2014. Optimal deployment of geographically distributed workflow engines on the cloud. In: IEEE Int. Conf. on Cloud Comp. Tech. and Science. pp. 811–816.

Thai, L., Varghese, B., Barker, A., 2018. A survey and taxonomy of resource optimisation for executing bag-of-task applications on public clouds. Future Gener. Comput. Syst. 82, 1–11.

Tortonesi, M., Govoni, M., Morelli, A., Riberto, G., Stefanelli, C., Suri, N., 2019. Taming the IoT data deluge: An innovative information-centric service model for fog computing applications. Future Gener. Comput. Syst. 93, 888–902.

Varghese, B., Buyya, R., 2018. Next generation cloud computing: New trends and research directions. Future Gener. Comput. Syst. 79, 849–861.

Varghese, B., Leitner, P., Ray, S., Chard, K., Barker, A., Elkhatib, Y., Herry, H., Hong, C., Singer, J., Tso, F.P., Yoneki, E., Zhani, M., 2019. Cloud futurology. Computer 52 (9), 68–77.

Varghese, B., Wang, N., Barbhuiya, S., Kilpatrick, P., Nikolopoulos, D.S., 2016. Challenges and opportunities in edge computing. In: IEEE Int. Conf. Smart Cloud. pp. 20–26.

Varghese, B., Wang, N., Nikolopoulos, D., Buyya, R., 2019. Feasibility of fog computing. In: Handbook of Integration of Cloud Computing, Cyber Physical Systems and Internet of Things. Springer.

Wang, Y., Sheng, M., Wang, X., Wang, L., Li, J., 2016. Mobile-edge computing: Partial computation offloading using dynamic voltage scaling. IEEE Trans. Commun. 64 (10), 4268–4282.

Wang, N., Varghese, B., Matthaiou, M., Nikolopoulos, D.S., 2017. ENORM: A framework for edge node resource management. IEEE Trans. Services Comput. 1.

Wang, Y., Wang, K., Huang, H., Miyazaki, T., Guo, S., 2019. Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications. IEEE Trans. Ind. Inf. 15, 976–986.

Wu, Y., Ni, K., Zhang, C., Qian, L.P., Tsang, D.H., 2018. NOMA-Assisted multi-access mobile edge computing: A joint optimization of computation offloading and time allocation. IEEE Trans. Veh. Technol. 67 (12), 12244–12258.

Xu, J., Chen, L., Ren, S., 2017. Online learning for offloading and autoscaling in energy harvesting mobile edge computing. IEEE Trans. Cogn. Commun. Netw. 3 (3), 361–373.

Zhang, K., Mao, Y., Leng, S., Zhao, Q., Li, L., Peng, X., Pan, L., Maharjan, S., Zhang, Y., 2016. Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. IEEE Access 4, 5896–5907.

Zhang, H., Qiu, Y., Chu, X., Long, K., Leung, V.C., 2017a. Fog radio access networks: Mobility management, interference mitigation, and resource optimization. IEEE Wirel. Commun. 24 (6), 120–127.

Zhang, H., Zhang, Y., Gu, Y., Niyato, D., Han, Z., 2017b. A hierarchical game framework for resource management in fog computing. IEEE Commun. Magazine 55 (8), 52–57.

Zhao, X., Huang, G., Gao, L., Li, M., Gao, Q., 2021. Low load DIDS task scheduling based on Q-learning in edge computing environment. J. Netw. Comput. Appl. 188, 103095.

Nan Wang received the Ph.D. degree in computer science from Queen's University Belfast, UK. She is a product lead at Mindtrace Ltd., UK. She obtained MRes in Web Science and Big Data Analytics from the University College London, UK, and M.Sc. in Management and Information Technology from the University of St Andrews, UK. She obtained her undergraduate degree from Beijing Jiaotong University, China. Her research interests include resource management for edge/fog computing systems and machine learning.

Blesson Varghese received the Ph.D. degree in computer science from the University of Reading, UK on international scholarships. He is a Reader (Associate Professor) in computer science at the University of St Andrews and an honorary faculty member at Queen's University Belfast. He is the Principal Investigator of the Edge Computing Hub and was a Royal Society Short Industry Fellow to British Telecommunications plc. His interests include developing and analysing novel parallel and distributed systems and applications that span the cloud–edge–device continuum. More information is available from http://www.blessonv.com