Efficient Data as a Service in Fog Computing: An Adaptive Multi-Agent Based Approach

Giulia Mangiaracina, Pierluigi Plebani, Mattia Salnitri, and Monica Vitali

Abstract—Data as a Service (DaaS) offers an effective provisioning model able to exploit the advantages of cloud computing in terms of accessibility and scalability when data providers need to make their data available to different data consumers. Nevertheless, in settings where data are generated at the edge and they need to be propagated (e.g., Industry 4.0, Smart Cities), DaaS model suffers of some limitations: data transfer from the edge to the cloud – and viceversa – could require a significant time and privacy issues could hamper the possibility to move the data. Goal of this article is to propose a DaaS model based on the Fog Computing paradigm, which combines the advantages of both cloud and edge computing. The proposed solution implements an adaptive multi-agent system where each agent autonomously manages the placement of data in the most convenient location considering the quality of service requirements of the user that it is serving. To guarantee the collaboration of the agents without imposing a centralized control, a reinforcement learning algorithm will be enacted to balance between the local optimum for the single data consumers and the satisfaction of the global requirements of all consumers.

Index Terms—Fog computing, adaptive information systems, distributed decision system, data management, Data as a Service

1 Introduction

Data as a Service (DaaS) provisioning model combines data management functions with the advantages of cloud computing especially in terms of accessibility, scalability, and capabilities. Instead of taking care of the whole data management life cycle, data providers can leave to the DaaS the burden of optimizing the storage even when the amount of data becomes significant, making the data accessible regardless of the location, and enacting redundancy policies to minimize security risks. As in many scenarios data are usually generated at the edge of the network (e.g., Industry 4.0, Smart Cities), the ability to manage the data according to the DaaS model poses new challenges: the movement from the edge to the cloud of massive data could require a long time which also delays the data processing, privacy policies could affect the possibility to move to the cloud sensitive data if the destination is outside the country borders, storage technologies adopted on the cloud could require complex data transformation which results in significant overhead.

To deal with these challenges, Fog Computing is emerging as a paradigm able to extend the Cloud Computing towards the edge of the network. In this way, the main characteristics of the Cloud, i.e, on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service [23], are offered considering resources distributed along the whole continuum including managed data centers, on premises infrastructures, as well as networking facilities [12]. One of the main advantages of adopting such a paradigm relies on the possibility to balance between the high availability and scalability of resources offered by the cloud and the reduced latency when accessing to the resources on the edge. From a data perspective, data and applications should be managed to place the application closer to where the interested data is generated.

The problem of adopting the Fog Computing paradigm when it comes to the DaaS context [5] relies on the discrepancy between a full knowledge about the data and a limited knowledge about the application which will process those data. Data providers expose the data without any prior knowledge of the type of analysis that the different data consumers will perform through a set of custom applications[1]. Some applications run on the cloud, some other applications are required to run on the edge. Some applications could migrate to a place closer to the data, some others could be constrained in a given environment so that data are the element that must migrate.

This paper proposes a novel approach to DaaS in a fog environment where the placement of data is not driven by pre-defined data provisioning models set by the data provider, but by the specific, customizable, and sometimes conflicting requirements of the data consumers. Goal of this paper is to propose an approach able to intertwine computation and storage resources in a fog environment and to react in case the consumers' requirements are not satisfied taking into account the needs of a dynamic set of consumers where actors can join and leave, and their needs could change over time. More specifically, the main contributions of the paper are:

- *The authors are with DEIB Department, Politecnico di Milano, 20133 Milan, Italy. E-mail: giulia.mangiaracina@mail.polimi.it, {pierluigi.plebani, mattia.salnitri, monica.vitali}@polimi.it.*

Manuscript received 5 November 2021; revised 2 November 2022; accepted 4 November 2022. Date of publication 9 November 2022; date of current version 6 September 2023.
(Corresponding author: Mattia Salnitri.)
Recommended for acceptance by S. Guo.
Digital Object Identifier no. 10.1109/TCC.2022.3220811

[1] Security and privacy implications concerning the data access are not directly considered in this paper, albeit meaningful, because protection mechanisms can be added to the proposed solution without affecting the relevance of the results.

- The definition of a framework enabling the adoption of Fog Computing to provide efficient data access through the DaaS provisioning model based on a balanced distribution of data storage and data processing capabilities between cloud and edge resources.
- A reactive system based on reinforcement learning that is able to adapt when data are placed on the fog environment. The adaptation is required whenever the data consumers' requirements are not satisfied due to a change in the consumers population or the network influence.
- A container-based platform implementing the proposed system used to demonstrate the feasibility of the solution and the convergence of the proposed algorithm.

The rest of the paper is organized as follows. Section 2 introduces the motivation of the proposed work, defining the application context. Section 3 introduces the main components of the proposed Data as a Service approach. Section 4 formalizes the main concepts on which the framework is built, while Section 5 defines in details the distributed decision system. A validation of the approach is reported in Section 6. Finally, a review of the related work is provided in Section 7 and conclusions are given in Section 8.

2 Motivating Scenario

The recent pandemic has demonstrated the importance of data sharing among organizations. For example, hospitals need to efficiently track who goes to the emergency room, what symptoms they have, how many people have been hospitalized, and how many sent home. These data are required by decision makers, which could belong to external organizations (e.g., governments), that need to have access to fine-grained data about the numbers of contagions and hospitalization, and the severity of the patients. Another class of interested users are medical researchers that need to have additional information about the vital signs of the patients and their symptoms before and after applying some treatments. On this basis, to adopt the so-called evidence based medicine [33], the more organizations are involved, the more data are produced and the more analysis can be done. From a technical perspective, whether they produce or consume clinical data, organizations can be placed in different areas of different countries and managed data could have different formats (e.g., structured databases, files, images) and could be differently updated (e.g., hourly, daily, weekly).

In this scenario, the adoption of a DaaS provisioning model can be beneficial for both data providers and consumers. In particular, as shown in Fig. 1, we consider an Italian hospital named *IT-H* (black pinpoint) as a data provider whose data are requested by many consumers (white pinpoints) located in different research centers around the world: i.e., ARG-R, AU-R, CN-R, EU-R, US-R.

A typical solution could imply the usage of a cloud platform, for instance based in a European Region (see black data center icon), where the data are transferred once they are generated and where DaaS is running to offer an access to these data from the other locations. In fact, cloud storage is theoretically unlimited and the needed storage capacity is available on-demand. This ensures the accessibility of the data stored on the cloud by every organization that has an internet connection. In this way, supporting the evidence-based medicine approach, *IT-H* allows a potential unlimited set of other hospitals and organizations around the world to query the offered data. As the query requires some computation, *IT-H* could also buy cloud computational resources, which can be easily scaled according to the workload, supporting the DaaS with the needed infrastructure to efficiently analyse the data.

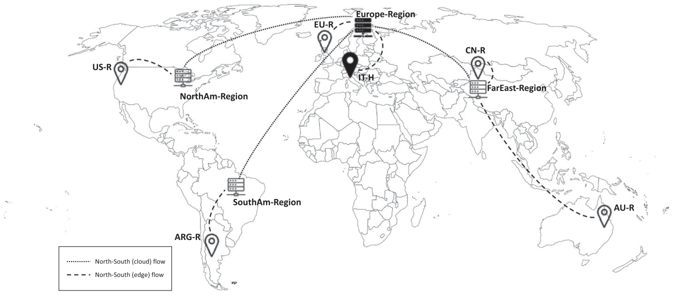


Fig. 1. A DaaS for sharing COVID-19 related data.

Albeit relying on a cloud platform can be efficient in terms of maintainability, scalability, and cost-reduction with respect to a solution in which data are stored on the premises of *IT-H*, there are some drawbacks mainly related to the effects of the latency [32] that a researcher (i.e., *ARG-R*, *AU-R*, *CN-R*, *EU-R*, *US-R*) may experience when accessing to the exposed data. The effect of latency depends on a combination of two main aspects:[2]:

- Cloud-to-Edge traffic between cloud sites where the DaaS is running and (i) the on-premise resources of *IT-H* which populate the DaaS (link from *IT-H* and Europe-Region), (ii) the resources used by *EU-R* connected to the same cloud provider (link from Europe-Region and EU node), and (iii) the resources used by a researcher connected to another cloud provider (e.g., link from *CN-R* and *AU-R* nodes with the Far-East region). Here the latency depends on the technology used to store the data (e.g., having data on SSD or HDD could reduce the latency of a couple of order of magnitude) and the characteristics of the network link (e.g., FTTH rather than FTTC) to the respective cloud provider.
- Cloud-to-Cloud network latency affecting the traffic between cloud sites. Concerning this aspect, if both the data centers are managed by the same cloud provider the latency is usually lower than when data centers are managed by different cloud providers [37].

It is possible to assume that performances can increase if data are processed closer to where they are stored [30]. While the effects of the latency can be controlled when all the resources are managed by the same authority, the scenario becomes more complex when considering a DaaS context. According to the service oriented architecture principles, a

[2] For the sake of completeness, when it comes to latency in data centers, the literature also consider the so-called East-West traffic, i.e., the data movement between resources belonging to the same data center. As this case considers the internal organization of a single node, it is out our scope of investigation.

data provider designs the service with a limited knowledge about the data consumers. As a consequence, from a data provider perspective, data latency can be reduced by selecting a cloud provider with resources closer to the *IT-H* premises and ensuring a good backbone infrastructure with the other regions managed around the world (to mitigate the latency of Cloud-to-Cloud traffic). Nevertheless, the effect of selecting a cloud provider able to mitigate the latency from the provider's perspective can be vanished by the infrastructure at the consumer side which could have outdated on-premise resources or rely on a cloud provider which does not have good performances when communicating with the provider's one.

While in some scenarios solutions based on Content Delivery Networks (CDNs) [27] can mitigate the effects of the latency, the amount of data, the different types of requirements, and the different computation which can be performed by the users require a high degree of flexibility that typical approaches are not able to support. In fact, in addition to the just mentioned infrastructural aspects, application-related aspects need to be taken into account when considering a DaaS. A DaaS exposes a set of methods which are not specifically designed for a given user but, rather, for a class of users. This means that each user, according to his/her needs, is more interested on some methods rather than other. Thus, the data to be delivered depends on the final user so that data distribution to the edge of the network needs to be user-specific and could change during the time. Moreover, latency becomes one of the requirements and it must be balanced with other aspects such as the quality of the data exposed and the availability of the nodes where the data are published.

In this context, Fog computing offers a paradigm exploiting all the resources available on the cloud and on the edge at both the provider and consumer side. Applied to our DaaS scenario and based on the location in which the final users want to perform the analysis of the data exposed by the DaaS, the main challenge is to find the right data distribution that can satisfy the different needs of all the final users, both in terms of performance and data quality.

3 Fog Enabled DaaS Framework

To achieve the aforementioned objectives, in this paper we aim to improve the data management in a DaaS environment by exploiting the Fog Computing paradigm. The adopted paradigm allows to balance between the advantages offered by an on-premise infrastructure (e.g., in term of latency) and the advantages offered by a cloud deployment (e.g., in terms of scalability and availability). The environment will adapt by deciding if and where to move or (partially) copy data among the fog resources to which the data consumers are connected.

Fig. 2 shows the logical view of the proposed framework. The data provider is in charge of offering a data set that will be processed by the applications at the consumers' side[3]. In the middle between the data provider and the data consumers, the proposed framework, that implements an innovative DaaS, can leverage fog resources that may be owned by the provider or consumers and that may be located on their premises or rented on some cloud sites.

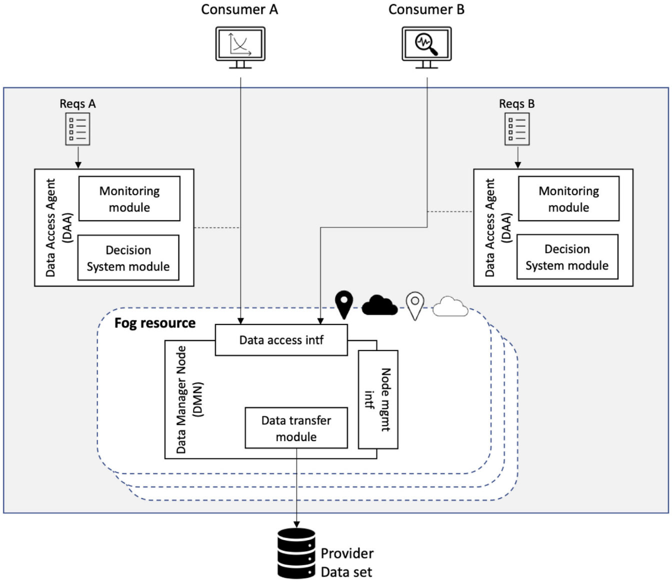


Fig. 2. Logical view of the proposed framework.

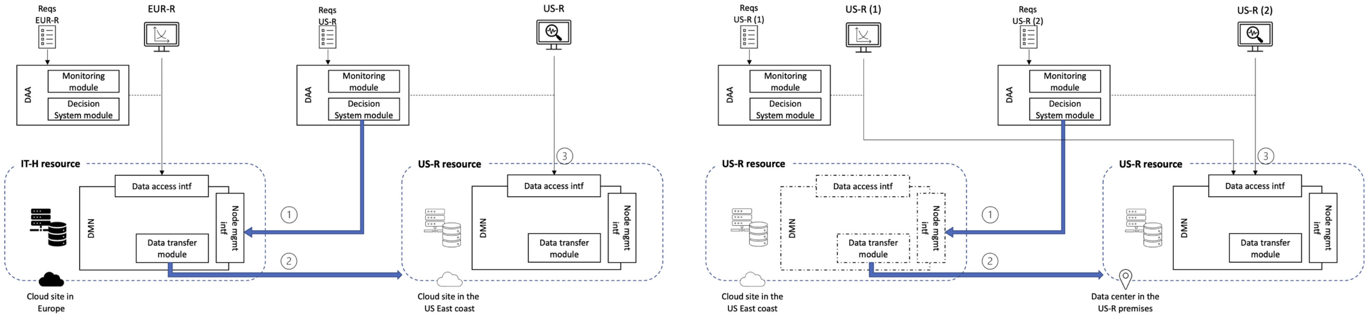
Depending on the configuration, the DaaS can rely on one or more fog resources where Data Manager Nodes (DMNs) and Data Access Agents (DAAs) will be deployed to enable the proposed approach. To this aim, we assume that the DaaS offers a configuration function which allows the provider and the consumers to specify the fog resources they made available[4]. As DMNs and DAAs are container-based modules, their deployment only requires the presence of a container runtime (e.g., containerd or dockerd) in the available fog resources.

3.1 Data Manager Node

The DMN is the component which hosts a full or partial replica of the data set offered by the cloud provider. The DaaS environment could have one or more DMNs. We assume that in the initial setting, a single DMN is present, then depending on the location of the data consumers, additional DMNs could be deployed in fog resources which locations are the more convenient to satisfy the consumers' requirements. Similarly, when the number of data consumers decreases, the numer of DMN could be requested to decrease accordingly. Having each DMN a replica of the provider's data set, such replicas are kept aligned exploiting the facilities offered by the data source technology. Thus, we assume that the adopted data source technology to host the data set supports this feature. While a full replica management is supported by many DBMS (e.g., MySQL, MongoDB), the management of partial replicas could require specific middleware (e.g., SymmetricDS) or dedicated solutions [6].

[3] It is worth noticing that in this paper it is not relevant what a consumer does with the data set. For this reason, all the discussion concerns how to improve the access to those data and to respect the requirements which could be customized by all the consumers.

[4] Albeit this resource sharing requires a proper credential and isolation management to avoid improper data leak or resource misuse by the DaaS, the analysis of this aspect is out of the scope of this paper. Anyway, solutions based on containers are good candidate to address all these aspects.



(a) (b)

Fig. 3. Reconfiguration in case of node duplication (a) and movement (b).

Internally, a DMN is composed of a *Data access interface* (e.g., REST API), which hides the technological aspects about the data set with a service layer offering to the data consumers a standard interface as typically done in DaaS settings. The DMN is completed with a *Node management interface* which offers to the DAAs – as discussed in the next paragaph – a set of functions to redeploy a DMN on a different fog resource or to duplicate the DMN in another fog resource. The actual implementation of these functions are provided by the *Data transfer module* which creates a new DMN and manages the data copy or movement from the old DMN to the new one.

3.2 Data Access Agent

A DAA exists for each consumer and it is placed between the application which needs the data set and the assigned DMN, a.k.a. *reference DMN*. As at least one DMN is created in the initial setting, all the DAA will have that as reference DMN. Based on the evolution of the system, which is governed by the approach defined in this paper, the number of DMN could increase (or decrease) and the reference DMN for the DAA could be changed. In this paper, we are not going to discuss in details how the match between a DAA and a DMN is performed and how the DAA is deployed. More details on this can be found in [29]. Generally speaking, each DAA is connected to one DMN and a DMN can be shared among different DAAs (e.g., as shown in Fig. 2 where the applications of two consumers share the same DMN). The DAA is configured with the specific *requirements* of a data consumer concerning the performance of the data flow which could include, for instance, constraints on latency, minimum availability, amount of free memory. In case the consumer is not interested in the whole data set offered by the data provider, the requirements also specify which is the relevant subset of data to drive the creation of partial replicas among different DMNs. Based on these requirements, the *Monitoring module* inspects the data flow to detect possible violations and to decide, in case of violation, whether to enact an adaptation action based on evaluation done by the *Decision System module*. These actions aim to improve the consumer experience when accessing to the data set managed by the connected DMN and they are:

- *Data Duplication*: the DAA asks the connected DMN to create a copy of itself in another fog resource available to the DaaS and to place there a replica of the stored data set. An example of the first adaptation action is shown in Fig. 3a where the DAA, currently connected to a DMN deployed in a fog resource controller by the data provider, decides, based on the monitored performance, to create a new DMN in a fog node located under the realm of the data consumer to which the DAA refers. This could be the case of a researcher located in the US West coast, i.e., *US-R* that has some resources on the cloud (on a site in the US East coast) and wants to massively access to the data offered by the *IT-H* which is located in Europe. To achieve this goal, three main steps are required: (1) the decision system module informs the node to which it is currently connected about the need to create a replica along with the endpoint and the credential to access to the facilities where to deploy the new replica; (2) the data transfer module creates the new node and the replica of the data set by leveraging on the functions offered by the data source technology; (3) the traffic initially directed to the node deployed at the provider side is re-routed to the new node exploiting, for instance, the URL redirection supported by the HTTP protocol.
- *Data Movement*: the DAA asks the DMN to move itself, and the data it manages, to another fog resource available to the DaaS. For instance, considering the configuration shown in Fig. 3b where two applications of *US-R*, located in the West Coast, are connected to the same node which is in a cloud site located in the East Coast. Assuming that the distance between the application and where the data are stored affects the performance of the second application, the related DAA decides to move the connected DMN to the fog resources made available on the premises of the *US-R*. To achieve this goal, the same steps composing the node duplication are executed considering that the URL redirection in the third step is applied to all the connections to the node that has been moved.
- *Change Reference Copy*: a DAA decides to change the reference DMN without changing any of the existing nodes. In this case, assuming that more than one DMN exists, the DAA selects the most convenient DMN and operates an URL redirection to call the selected one.

A fourth adaptation action called *null* is also considered. This applies when a violation of the user's requirements is detected, but the DAA's decision system doesn't take any action since none of the available ones is considered beneficial.

At this stage, we assume that an adaptation action can be enacted before or after the completion of the data transfer. This is particularly relevant in case of a massive data transfer which cannot be interrupted to change, for instance, the location of the data set currently accessed.

3.3 Motivations for a Distributed Approach

These adaptation actions, whose technical details are discussed in Section 4, allow our framework to implement a dynamic DaaS. From a logical view, the DaaS remains under the control of the data provider as it is in charge of creating the first DMN and of defining the policies to move/duplicate the node in other locations. At the same time, the ability of the DMN to move or to duplicate itself and the ability of the DAA to decide where to place the data set enable the definition of an adaptive DaaS able to re-configure itself based on the requirements expressed by the consumers and the available fog resources. The core of the framework concerns a distributed decision approach where the DMN has the capability to move or duplicate the offered data set, while the decision about the adaptation action to enact is locally made by a DAA placed at the consumer side.

The proposed approach is therefore based on a two-level decision-making process: (i) a *local decision*, taking into account the interests of the single data consumer (captured by the related DAA), and (ii) a *global decision* that involves also the other consumers sharing the same DMN. The creation of a mechanism which is able to integrate and balance these two components is necessary to improve the outcome and performance of the method and to take into account the side effects that a decision generated by an agent might have on others. The solution is based on two main pillars:

- the fully automated selection of an adaptation action according to the metrics status and to the impact that the action produces on the single user requirements (*internal impacts*);
- a distributed coordination mechanism, based on an *external feedback*, which allows to indirectly take into account the requirements of other actors in the action selection process.

The proposed solution is based on decentralized control and distributed monitoring exploiting fog resources. The approach offers several advantages [28]:

- *Fault-tolerance:* if any decision-maker fails, others are not affected. This avoids the dependence on any single point of failure.
- *Performance:* the parallelis m derived from multiple decision-making nodes can increase the throughput of the system, avoiding bottlenecks. Moreover, having decision-makers located close to where their decisions are needed increases responsiveness.
- *Scalability:* since the system is already designed to support multiple decision-making nodes, the addition of more entities does not require a significant change. Furthermore, the complexity of storing and managing monitoring data is distributed among the different nodes instead of requiring a significant set of centralized resources.

As opposed to a centralized decision approach, a distributed decision system has not complete observability of the global state, due to the fact that the state information is distributed among the various nodes (*state-information uncertainty problem*). This limitation might introduce sub-optimal decisions, negatively affecting other agents in the environment. For instance, as in case of Fig. 3b, the decision made by the DAA of the second application to move the DMN could have an impact to the first application. Yet, deciding to have duplicated data in all the nodes could be (i) infeasible in case the nodes are not properly equipped with enough storage resources, (ii) detrimental in term of performances because copying the data could require a significant amount of time.

To mediate between the lack of observability of the adopted decentralized solution and the ability to find an equilibrium able to satisfy the requirements of all the data consumers, the decision system is based on a reinforcement learning algorithm.

4 Framework Formalization

To properly define the envisioned DaaS environment, a more formal definition of the proposed framework follows.

Fog Resources. The fog environment on which the DaaS is built, is composed by a set of resources which can be made available by the data provider or data consumer. Without considering all the technical details related to the way in which the resource can be accessed and managed, which are not relevant to present the overall approach, given $\mathcal{R}$ the set of fog resources, a fog resource $r \in \mathcal{R}$ is defined as:

$$r = \langle url, owner, credentials, resources \rangle \tag{1}$$

where url uniquely identifies the resources on the network, $owner$ specifies who makes the resource available (e.g., *IT-H*, *US-R*, *ARG-R*), $credentials$ specifies how to access to the resource in order to create the DMN, and $resources$ is related to the information about the amount of resources in terms, for instance, of CPUs, memory, storage, which can be used in that node.

Requirements. The requirements expressed by a data consumer are assessed through a set of metrics that are used to configure the monitoring module of the related DAA. We define $\mathcal{M}$ as the set containing the metrics, e.g., "availability" and "response time".

Definition 1. *We define the function $mi : m, t \rightarrow \mathbb{R}$ as a function provided by the monitoring module of a DAA returning the value of a metric $m \in \mathcal{M}$ at time t.*

On this basis, we can define a requirement rq as:

$$rq = < m, v, p > \tag{2}$$

where $m \in \mathcal{M}$ is a metric; $v \in \mathbb{R}$ is a reference value for the metric; p is a predicate used to compare a measured value of a metric $m \in \mathcal{M}$ at time t with the desired one. p is defined as $p(mi(m,t), v) \in [true, false]$ and returns true if the requirements are satisfied. We can also define the set of requirements defined for a data consumer as $\mathcal{RQ}$.

Data Set. Given the data set made available by the data provider, as discussed, many replicas can be managed by the DaaS environment through the DMNs. Thus, each DMN is in charge of managing a specific copy (complete or partial) of a data set. For the sake of clarity, we now consider a DMN in a more abstract way, as a combination of an accessible data set and the fog resource in which the data set is stored.

Let $\mathcal{DMN}$ be the set of DMNs in the DaaS, a $dmn \in \mathcal{DMN}$ is defined as

$$dmn = \langle URL, r \rangle \tag{3}$$

where URL is the access point through which the DMN can be accessed (i.e., the data access interface) and r is the resource in the Fog infrastructure where the DMN is currently deployed. As the *Data Movement* adaptation action has the ability to change the resource in which the DMN is located, the value of r can change during the time. The information about the available data sets and their derived DMNs is saved in a shared *Data Source Knowledge Base*, better described in Section 5.3.

Multi-Agent System. The core of the approach is represented by a Multi-Agent System (MAS) composed of the several DAAs which are in charge of satisfying the data consumers' requirements through the selection and enactment of the supported adaptation actions.

As done for the DMN, to better define the proposed approach, a more abstract definition of the DAA is proposed in this formalization. More specifically, given $\mathcal{DAA}$ the set of DAAs existing in the DaaS environment, a $daa \in \mathcal{DAA}$ is defined as:

$$daa = \langle URL, r, \{\langle \mathcal{RQ}, dmn \rangle\} \rangle \tag{4}$$

where URL uniquely identifies the DAA and r represents the fog resource in which the daa is deployed[5]. Finally, as a data consumer could access from the same application more than one data sets, a set of pairs $\langle \mathcal{RQ}, dmn \rangle$ specifies the connected DMNs through which is possible to access to the data sets and, for each of them, the requirements. For the sake of simplicity, we hereafter consider only a scenario in which a DAA access to a single data set, i.e., it is connected to a single DMN. Nevertheless, the proposed solution does not loose generality as this simplification only reduce the number of requirements, and not the structure of the decision system at the core of the DAA.

At the same time, more than one DAA can share the same DMN with different requirements and from different locations. Therefore, each modification to a DMN potentially affects all DAAs linked to it. To better formalize this concept we introduce the definition of *neighborhood* and *neighbors*:

Definition 2. *Given a $dmn \in \mathcal{DMN}$, the* neighborhood *of dmn is defined as the set of DAAs accessing the same DMN. We call* neighbors *the set of DAAs belonging to the neighborhood of dmn.*

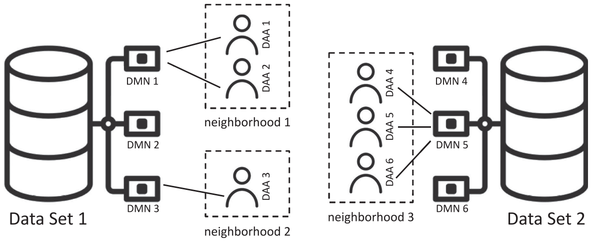


Fig. 4. Example of neighborhoods.

Fig. 4 shows an example of a scenario in which two data sets are made available through the DaaS, each of them with three DMNs. In the example, three neighborhoods exist: a first neighborhood is composed of DAA 1 and DAA 2, linked to DMN 1, a second one includes only DAA 3, accessing to DMN 3. The last neighborhood includes DAA 4, DAA 5, and DAA 6, accessing to DMN 5. A DAA may belong to multiple neighborhoods if it uses multiple data sets.

Adaptation Actions. An adaptation action $a \in \mathcal{A}$ is defined as:

$$a =< type, params, tp > \tag{5}$$

where $type \in \mathcal{A}ction\mathcal{T}ype = \{DM, DD, CR, NULL\}$ denotes the type of the action (DM = Data Movement, DD = Data Duplication, CR = Change Reference Copy); $params$ depends on the action: i.e., if $type = DM|DD$ then $params = \{dmn_s, r_t\}$; if $type = CR$ then $params = \{dmn_s, dmn_t\}$; if $type = NULL$ then $params = \emptyset$. These parameters are:

- $dmn_s \in \mathcal{DMN}$ is the DMN which has to be moved, duplicated, or no longer referenced by a DAA.
- $dmn_t \in \mathcal{DMN}$ is the already existing DMN which has to be referenced by a DAA.
- $r_t \in \mathcal{R}$ is the fog resources where a new DMN is deployed.

The propagation time tp represents the time required to apply the action and observe its effects. We can reasonably assume that different adaptation actions have different propagation times. For instance, the execution time of a data movement or data duplication action is affected by the network latency and the size of the data set considered. Estimating tp is not trivial. However, at execution, its value can be obtained from activity logs keeping track of the action enactment (e.g., when the new copy of a data set is up and running in the execution environment). In the action definition, the value indicated for tp is the average of the propagation time observed from the action logs. In addition to the considered aspects, the model could be extended also considering the cost of the action that depends on the execution time of the action and its storage cost (i.e., the cost for allocating storage resources for the new location of the data set). For the sake of simplicity, the cost is not currently included in the proposed model and it will be considered in future work.

Being daa the DAA enacting the adaptation action, Table 1 summarizes the effects of the enactment of these adaptation actions to the state of the overall system where the function `createDMN` corresponds to the deployment of a new DMN in the resource r_t.

[5] Assuming that the data consumer is not nomadic, the resource in which the DAA is initially deployed is fixed during the time.

TABLE 1
Effects on the System When Enacting the Supported Adaptation Actions

Action	Changes
$DM(dmn_s, r_t)$	$dmn_t = createDMN(r_t)$ $\mathcal{DMN}' \leftarrow \mathcal{DMN} \cup dmn_t$ $\forall daa \in \mathcal{DAA} \mid daa.dmn = dmn_s,$ $daa.dmn = dmn_t$ $dmn_s \leftarrow \emptyset \wedge \mathcal{DMN}' \leftarrow \mathcal{DMN} - dmn_s$
$DD(dmn_s, r_t)$	$dmn_t = createDMN(r_t)$ $\mathcal{DMN}' \leftarrow \mathcal{DMN} \cup dmn_t$ $\forall daa \in \mathcal{DAA} \mid daa.dmn = dmn_s$ $\wedge dist(daa, dmn_t) < dist(daa, dmn_s),$ $daa.dmn = dmn_t$
$CR(dmn_s, dmn_t)$	$\forall daa \in \mathcal{DAA} \mid daa.dmn = dmn_s,$ $daa.dmn = dmn_t$
NULL	-

While a Data Movement action maintains a single copy of a data set in the new location, a Data Duplication action maintains both the original copy and the new generated copy at the same time. Duplication has some advantages and disadvantages when compared to movement. The main advantage is that a duplication action does not have a direct effect on the original data set: thus, the neighbours of that data set are not affected since they can keep accessing the data set from its original location. However, maintaining several copies of the same data set in different locations increases the costs of storage (more storage resources are needed) and the cost of management (the several copies have to be kept consistent). Additionally, all the possible adaptation actions can be subject to transformations when data is moved or duplicated from a storage node to another. Transformations consist in the manipulation of the content of a data set. Examples of transformations are anonymization, encryption, and aggregation. For the sake of simplicity and without affecting the generality of the approach, transformations will not be considered in the rest of this paper.

It is also important to note that:

- In a realistic environment not all the actions are always applicable, according to the context. For example, some data movement actions might be limited or forbidden among sites due to privacy constraints imposed by the data provider. Without loss of generality, we assume that all adaptation actions are available. The definition of constraints on adaptation actions that may be specified by the data provider at deployment time have been discussed in [31] and are out of scope of this paper.
- Some of the actions could be unavailable at run-time depending on the resources status, for example due to lack of disk space to host data. The availability of the actions is evaluated at run-time.
- A system that includes replicas also requires a mechanism for selecting and locating them [19]. A search of the existing copies is thus needed to prevent unnecessary movement and copy actions, saving time and costs. We can assume that the information about the position of the existing copies in the environment is available and shared among the DAAs.
- The duplication action DD could take into consideration which parts of the the data set are relevant for the user, in order to perform *partial* duplication. Similarly to a common replication placement strategy used in Data Grid Systems and Content Delivery Networks (CDNs), the selection of the portion of the data set to copy might depend on the actual usage of the data set in terms of access frequency or data utility for the user. In the case of partial copy, the new copy will be useful only for the user who created it and for its neighbors using the same subset of the data. In this work, without loosing generality, we are not considering partial duplication.

*Event.*An *event* $e \in \mathcal{E}$ represents the enactment of an action and it is defined as:

$$e = < a, t > \tag{6}$$

where $a \in \mathcal{A}$ is the performed action, and t is the time of enactment of the adaptation action.

5 Distributed Decision System for DaaS

This section describes the decision process in its two phases: the local decision and the global decision. In fact, each DAA is in charge of satisfying the requirements for a single data consumer, the Decision System module of the DAA is responsible of this task. It needs to map the effect of each action on the metrics used to assess the requirements to be able to select a proper action when requirements violations occur. This aspect will be discussed in Section 5.2. However, the selected action could modify the state of the overall system, therefore, it could affect also the neighborhood of the DMN interested by the adaptation action. Thus, DAAs integrate in their models those collateral effects, by considering feedback produced by other DAAs. This issue will be discussed in Section 5.3. Both phases need to include an adaptation mechanism in order to dynamically adapt to the modifications of the environment. In fact, both the effects of an action for a single agent and for its neighbors might change during time.

5.1 Notation

Before going into the details of the decision system, Table 2 summarizes the adopted notation, connected to the formalization discussed in the previous section. Moreover, Table 3 describes the set of parameters used in our distributed decision system approach.

5.2 Local Adaptation

This section proposes an adaptive mechanism to make a DAA able to keep the user's requirements satisfied. As described in previous sections, an agent is composed of a monitoring module and a decision system module: the former is in charge of monitoring the current status of the user's requirements through a set of metrics, while the latter is in charge of ensuring their satisfaction by enacting an adaptation action.

As an adaptation action modifies the state of the environment in which each agent operates, its enactment affects the requirements satisfaction and could have a different measurable effect over different metrics. To select the proper adaptation action, each DAA keeps an internal knowledge of the expected impacts of each actions on the set of metrics $\mathcal{M}$ on which the data consumer has expressed its

TABLE 2
Notation Table

Notation	Description
$dmn \in \mathcal{DMN}$	a data management node and the respective set of all data management nodes
$daa \in \mathcal{DAA}$	a data access agent and the respective set of all data access agents
$r \in \mathcal{R}$	a storage resource in the fog environment and the set of resources
$m \in \mathcal{M}$	a metric and the respective set of all metrics
$k = \lvert\mathcal{M}\rvert$	the cardinality of the set of all metrics
$mi : m, t \rightarrow \mathbb{R}$	the value of a metric m at time t
$v \in \mathbb{R}$	the quantitative value associated with a metric
p	the predicate expressed in a requirement associated with a metric value
$rq \in \mathcal{RQ}$	a requirement of the consumer and the respective set of all requirements of the data consumer
$\mathcal{S} \subset \mathcal{RQ}$	the set of satisfied requirements for a data consumer
$\mathcal{NS} \subset \mathcal{RQ}$	the set of unsatisfied requirements for a data consumer
$a \in \mathcal{A}$	an adaptation action and the respective set of all the adaptation actions
$l = \lvert\mathcal{A}\rvert$	the cardinality of the set of all the adaptation actions
$tp \in \mathbb{R}^+$	the propagation time of an action
$e \in \mathcal{E}$	an event and the respective set of events
$i_{a,m} \in [-1, 1]$	the impact of an action a on a metric m
IM	the matrix of the impacts of all the actions in $\mathcal{A}$ on all the metrics in $\mathcal{M}$
$l\_score(a) \in [0, 1]$	a local score associated with the enactment of an action $a \in \mathcal{A}$ based on the impact matrix IM
f	a feedback received after the observation of an event e performing an action $a \in \mathcal{A}$
$f_{val} \in [0, 1]$	a quantitative value associated with a feedback
$g\_f_e \in [0, 1]$	the global feedback associated with an event e performing an action $a \in \mathcal{A}$
$g\_score(a) \in [0, 1]$	a global score associated with the enactment of an action $a \in \mathcal{A}$ based on the global feedback g_f_e

requirements $\mathcal{RQ}$ (see Definition 1). This knowledge is represented in an impact matrix defined as follows.

Impact Matrix. An impact matrix IM describes the overall impacts of all the actions types on all the metrics for a given DAA, i.e.,:

$$IM = \begin{pmatrix} i_{a_1,m_1}, \ldots, i_{a_1,m_k} \\ \vdots \\ i_{a_l,m_1}, \ldots, i_{a_l,m_k} \end{pmatrix} \tag{7}$$

where $l = |\mathcal{A}|$ and $k = |\mathcal{M}|$ are, namely, the cardinality of the adaptation actions set and the metric set.

Impact. A generic element $i_{a,m} \in \mathbb{R} : -1 \leq i \leq 1$ of the IM predicts the impact in case of the enactment of an adapation action a on metric m_n monitored by the DAA. Positive values of $i_{a,m}$ indicate positive effects of action a on metric m, while negative values indicate negative effects. The nearer $i_{a,m}$ is to 1, the higher is the positive effect, the nearer $i_{a,m}$ is to -1, the higher is the negative effect.

A critical aspect is related to the quantification of each $i_{a,m}$. This information could be provided at design time by experts. However, it is very difficult, if not impossible, to know in advance all the impacts of an action, thus the values of the IM. Moreover, in a dynamic environment where nodes can join and leave the network frequently, and changes in the environment caused by variations of latency and bandwidth due to network congestion can occur, the information about the impacts can become obsolete quickly. The impact of an action over the metrics can change over time and this value has to be kept updated, so that the

TABLE 3
Parameters Table

Parameter	Description
$TW \in \mathbb{R}^+$	a time window of validity of an event
$\alpha \in [0, 1]$	weight of the most recent impact observed in updating the impact value
$\beta \in [0, 1]$	weight of the most recent feedback received in updating the feedback value
$\omega \in [0, 1]$	weight of negative impact on satisfied requirements in the action selection process
$\gamma \in [0, 1]$	weight of the penalty due to negative feedback in the action selection process
$\rho \in [0, 1]$	stepness of the qualitative function $\mathcal{Q}(x)$

system adapts its behavior to the current situation. Additionally, noise in the environment might affect the quality of the impact observed by enacting an action and the effect of noise has to be limited.

Thus, we opt to assess impacts at run-time, from the observation of the status of metrics before and after the enactment of an adaptation action, whenever an event is registered. We estimate the impact using a reinforcement learning approach, in which the impact associated with an action is a cumulative estimate of past and recent values [18]. According to this approach, whenever an action is enacted, it receives a reward that has to be integrated with the previously recorded rewards. We thus need to define an update rule for the value of impact, taking into consideration dynamism and noise:

$$i'_{a,m} : \begin{cases} (1-\alpha)\ i_{a,m} + \alpha\ \mathcal{Q}(\triangle(e,m)) & \text{if } m \text{ increasing} \\ (1-\alpha)\ i_{a,m} - \alpha\ \mathcal{Q}(\triangle(e,m)) & \text{if } m \text{ decreasing} \end{cases} \tag{8}$$

where:

- $\alpha \in [0,1]$ is the *adaptation rate*, that is the factor that decides how much emphasis give to the historical values and to the last reward;
- $i_{a,m}$ is the previously computed value of the impact of action a on metric m (we can assume that $i_{a,m} = 0$ at time zero);
- $\mathcal{Q}(\triangle(e,m))$ is the observed impact (reward) computed for the event e on the metric m with $e.a = a$. Depending on the nature of the metric, i.e., decreasing or increasing, the reward positively or negatively affects the impact. For instance, in case of throughput the goal is to increase the value, while in case of response time, the goal is the opposite.

The value of α should be set according to the characteristics of the execution environment. High values of α increases the ability of the model to adapt to changes in the environment, giving more importance to the last reward. However, in this case the model will be more sensitive to the noise (e.g., wrong reward due to side effects of other events in the environment). The noise effect is instead reduced when past executions are considered.

Reward. The reward $\mathcal{Q}(\triangle(e,m)) \in \mathbb{R} : -1 \leq i \leq 1$ for the event e on the metric m, quantifies the impact of an action based on the variation of the metric value before and after the enactment of such an action, i.e.,:

$$\triangle(e,m) = mi(m, e.t + a.tp) - mi(m, e.t) \tag{9}$$

where:

- $e = \langle a, t\rangle \in \mathcal{E}$ is the registered event;
- $a = \langle type, params, tp, c\rangle \in \mathcal{A}$ is the action enacted during the event;
- $m \in \mathcal{M}$ is a metric;
- $mi(m, e.t + a.tp)$, defined in Definition 1, is a function that returns the value of the metric m measured at time $e.t + a.tp$ with $e.t$ the time of enactment of the action and $a.tp$ its propagation time;
- $mi(m, e.t)$ returns the value of the metric m measured at time $e.t$ with $e.t$ the time of enactment of the action.

To obtain a *normalized* value for the variation in the range $[-1,+1]$, the $\triangle(e,m)$ is subject to a $\mathcal{Q}(x)$ qualitative function:

$$\mathcal{Q}(x) : \begin{cases} -1 & x < -\rho \\ \arctan(x\frac{\pi}{2}\frac{1}{\rho}) & -\rho \leq x \leq \rho \\ +1 & x > \rho \end{cases} \tag{10}$$

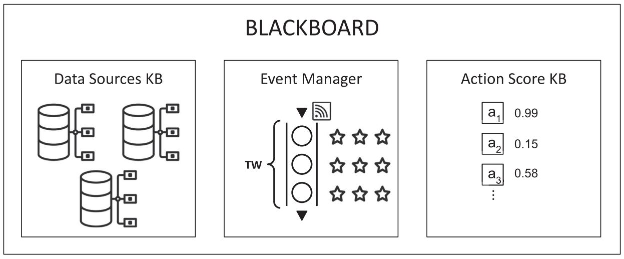


Fig. 5. Structure of the blackboard.

Function $\mathcal{Q}$ takes values in the desired $[-1,+1]$ interval, reaching its maximum value at the point corresponding to the parameter ρ, and its minimum for $x = -\rho$. The parameter ρ (with $\rho > 0$) represents a significant increase (or decrease) of the considered metric. Given the differences among the metrics, in terms of unit of measurement and wished target thresholds, this value is chosen for each metric, taking into consideration minimum, maximum, and variance of the observed values in the history. The choice of the parameter ρ allows to state how fast the minimum (-1) and maximum $(+1)$ value will be reached, shaping the steepness of the qualitative function $\mathcal{Q}$ for each metric. The parameter ρ can be derived considering the maximum and minimum thresholds of the analysed the metric. If there are no upper or lower bounds of the values of the metric, we suggest to consider a broad range in order to include all possible increases or decreases of the metric values.

5.3 Global Adaptation

Section 5.2 has focused on the local adaptation issue, determining which is the most effective adaptation action to enact in case of a violation. As we have discussed, a DMN is shared between several data consumers with different requirements. Thus, any modification to the DMN affects not only a single DAA, but also all its neighbors. It is necessary to enrich the method considering also the outcomes that an action may cause on the other involved actors, and define how multiple agents can interact in order to coordinate their actions.

To enable cooperation between agents in a decentralized way, we adopted the concept of *blackboard* proposed in [10]. A blackboard is a repository, accessible by all the parties, in which common knowledge can be shared among them. In our framework, the blackboard contains knowledge bases and event logs shared between the DAAs. The agents can consult it, and integrate the obtained information with their own knowledge, in order to improve their decision making. At the same time, they can enrich the blackboard with new information helping other agents in performing their tasks. Even though the blackboard is a centralized component, it acts only as a repository of limited shared information, thus not affecting the overall performance of the agents and not representing a bottleneck in the architecture.

In the proposed framework, the blackboard is composed of: (i) a *Data Source Knowledge Base*, (ii) an *Event Manager*, and (iii) an *Action Score Knowledge Base* (Fig. 5).

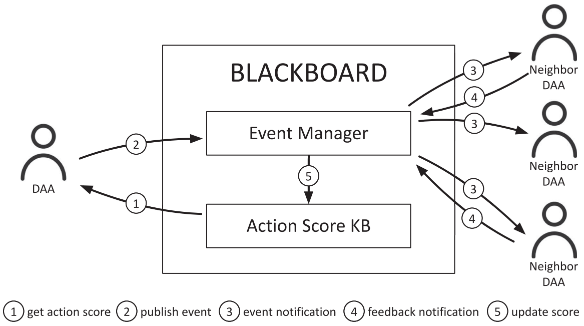


Fig. 6. Blackboard and Data Access Agents for event handling.

The *Data Source Knowledge Base* is the representation of the organization of the data provided by the DaaS service. From this Knowledge Base (KB) it is possible to retrieve the list of data sets and their characteristics, as well as the existing DMNs for each data set and the URL through which their data is accessible. The DAA exploits this KB to get the location of a DMN. Additionally, the KB is exploited during the enactment of adaptation actions: in case of data movement and data duplication, the KB is consulted to detect if a copy of the DMN already exists in the selected location and the information contained in the KB are updated after the action is performed; in case of the change reference copy action, the KB is queried to get the list of copies and their URLs.

The *Event Manager* provides the actual coordination mechanism between different agents. The coordination is based on a Publish-Subscribe pattern [34]. When a DAA performs an adaptation action, it publishes the event (Eq. (6)) in an event queue managed by the Event Manager. The event manager automatically notifies all the neighbors of the DMN involved by the action. In this way, all the agents are aware of actions taken by the other agents, that can potentially impact them. If a violation is observed after the notification, a neighbor replies to the notification with a negative feedback. The list of feedback notifications is collected by the event manager. An event is active for a limited period of time. When this time expires, the performed action is not considered responsible any longer for violations occurred to other neighbors. DAAs interact with the Event Manager by publishing events and sending feedback notifications in reply to occurred events.

The *Action Score Knowledge Base* collects a global score for the actions that have been enacted in the shared environment. The global score is computed starting from the feedback notifications received by the neighbors after the enactment of an action. When the active period of an event expires, the action score is updated using the received feedback and stored in the Action Score KB. DAAs interact with the Action Score KB by querying it for the global score assigned to actions they are planning to enact. They can use this score to predict possible negative outcomes of their decisions. The interaction between the Blackboard and DAAs for event handling is shown in Fig. 6.

In this paper we opted for an *a-posteriori* approach (the feedback is provided after the execution of the action) rather than an *a-priori* one (the feedback is provided before the agent execute the action). This is motivated by the uncertainty of the considered environment. In an *a-priori* approach, the agent would consult its neighbors before the enactment of an action in order to select an adaptation action able to improve its own requirements without negatively affecting its neighbors. Even if theoretically optimal, the neighbors don't have enough information to predict accurately the effects that an action might cause on them. With an *a-posteriori* approach, instead, the given opinion will be based on the direct observation of the real effects caused by the action. The adoption of this kind of approach has some advantages:

- it reduces the communication costs, since not all the agents have to communicate when a violation is detected, but only the interested ones (that in general constitute a small part of the total) and only in specific situations;
- the DAA does not need to wait for a response from other actors before making a decision. The opposite approach, i.e., wait for external feedback before taking action, would in fact increase the time required to restore the user's requirements.

The effect of an action is limited in time, i.e., other neighbors can observe a negative outcome only during a time window after its enactment. Thus, events are saved in the event queue for a limited time period, defined by a Time Window (TW), during which neighbors can send negative feedback if violations in their requirements are observed. During the validity of the time window, an event is considered *active*. A boolean function $active(e, t)$ evaluates if an event e is still active at a specific time t.

As shown in Fig. 5, each event related to an action can be associated with a set of feedback notifications. A *feedback notification* f is a message generated during the active period of an event $e \in \mathcal{E}$ by the neighbors of the DMN involved in the adaptation action. A feedback notification is defined as:

$$f = < e, f\_val, t > \tag{11}$$

where $e \in \mathcal{E}$ is the event, t is the time of creation of the feedback notification, and $f\_val \in \mathbb{R} : 0 \leq f\_val \leq 1$ is defined as:

$$f\_val = 1 - \frac{t_{violation} - e.t}{TW} \tag{12}$$

denoting with $t_{violation}$ the time stamp in which the requirements violation is detected. According to Eq. (12), f_val is proportional to the time elapsed between the enactment of the action and the violation, thus giving more importance to violations detected nearer to the action enactment, and lower importance to violations detected nearer to the time window expiration. In this paper, we are making two assumptions about the feedback:

- all violations have the same relevance, we are not associating the feedback with the severity of the violation observed. This is due to the fact that evaluating the severity (how much the metric deviates from the desired value) changes metric by metric and might introduce bias in case of data consumers with very restrictive QoS requirements on some metrics;
- all metrics have the same time window, we are not considering that different metrics are characterised by a different time span to observe an impact. This can be solved by using different time windows for different metrics. For the sake of simplicity we are not considering this aspect in this formulation.

When the Time Window TW expires, a global feedback g_f for the event is computed by aggregating all the feedback notifications received as in Eq. (13):

$$g\_f_e = \frac{1}{n-1} \sum_{i \in K} f\_val_i \tag{13}$$

where n is the total number of neighbors of the DAA and K is the number of notifications received. Thus, the global feedback is given by the average value of the feedback received by the neighbors, also including not participating neighbors, i.e., neighbors that did not detect any violations, whose estimated feedback is 0.

The new information about the action outcome acquired by the agent through the global feedback, is a useful piece of data for all the neighbors. In fact, neighbors share both the DMN but also (most of) the storage resources in which it can be moved or copied. Thus, the same action might be selected in the future by another agent with similar outcomes. Thus, the feedback value is stored in the *Action Score KB*.

The global score of an action a, $g\_score(a) \in [0,1]$ is an indicator of the negative feedback assigned to an action a throughout the history. Every time the action is enacted, the $g\_score(a)$ is updated according to the global feedback.

$$g\_score(a) \leftarrow (1-\beta)\; g\_score(a) + \beta\; g\_f_e \tag{14}$$

where the parameter $\beta \in [0,1]$ is a weight used to control the importance given to history or to new observations. In this way, a continuous and noise-aware improvement approach is provided. As with the local parameter α, the value assigned to β should depend on the dynamicity and noise of the execution environment.

5.4 Action Selection

In Sections 5.2 and 5.3 we have introduced the proposed method for enabling an adaptive approach for mapping the local and global effects of actions enactment through reinforcement learning.

Starting from the knowledge acquired through the interaction with the environment, the Decision System of the DAA can take independent decisions on which action is better to enact in a given context. In this section we illustrate the methodology for action selection.

Action selection is performed whenever an agent detects a violation in the requirements. The selected action (Table 1) should be the one that maximizes the positive impact on the violated metrics, while minimizing the negative effect on the satisfied goals, to avoid new violations after the enactment. During the selection process a score is associated with each action according to the knowledge acquired from the impact analysis as defined in Eq. (8).

Indicating with $\mathcal{S} \subset \mathcal{RQ}$ the set of requirements that are in a satisfied state, while with $\mathcal{NS}$ the set of requirements that are in a not satisfied state, the *Local Score* for each action can be computed as:

$$l\_score(a) = \omega\; \frac{\sum_{rq \in \mathcal{NS}} i_{a,rq.m}}{|\mathcal{NS}|} + (1-\omega)\; \frac{\sum_{rq \in \mathcal{S}} min(i_{a,rq.m}, 0)}{|\mathcal{S}|} \tag{15}$$

where $\omega \in [0,1]$ is a weight used to balance the two components of the equation. The first component takes into consideration the ability of the action a to improve unsatisfied requirements ($rq \in \mathcal{NS}$). Through this component, we can select the action maximizing the likelihood of improving the current situation. The second component considers negative effects of the actions (i.e., their likelihood to bring a satisfied requirement in a violated state). It takes into account only the negative value of impacts ($min(i_{a,rq.m}, 0)$)) on satisfied requirements ($rq \in \mathcal{S}$). We consider negative impacts since they might generate a violation.

In Learning Automata theory [26], at each point in time an action is associated with a probability to be selected $p_a(t)$ which is proportional to its local score. The higher is the score, the higher is the probability to select the action. However, the local score only accounts for the local impact. Thus, we need to integrate into the selection probability also the global feedback released by the neighbors as discussed in Eq. (13). This can be done introducing a *penalty factor* $\gamma \in [0,1]$.

$$\begin{aligned} fit_a(t) &= \; max(0, l\_score(a) - \gamma\; g\_score(a)) \\ p_a(t) &= \frac{fit_a(t)}{\sum_{aa \in \mathcal{A}} fit_{aa}(t)} \end{aligned} \tag{16}$$

where the first equation compute the fitness for each action $a \in \mathcal{A}$. With $\gamma = 0$ only the local score is considered, while with $\gamma \geq 0$ the probability of selecting the action is mitigated by the negative global score provided by the neighbors in previous iterations. If the penalty is higher than the local score, the fitness value is set to 0. In the second equation, the fitness values are transformed in a probability using fitness proportionate selection [11].

Algorithm 1. Action Selection Algorithm

```
Input: metrics states $\mathcal{M}(t)$, Impact Matrix $IM(t)$
if violated metrics set $NS \neq \emptyset$ then
    filter from $\mathcal{A}$ invalid and unavailable actions;
    generate random $rand \in [0,1]$
    if $rand \geq 1 - p$ then
        Exploration: select $a^*$ as a random action from $\mathcal{A}$;
    else Exploitation:
        for $a_i \in \mathcal{A}$ do
            compute local score:
            $l\_score(a_i) = \omega \times \frac{\sum_{rq \in \mathcal{NS}} i_{a_i,rq.m}}{|\mathcal{NS}|} + (1-\omega) \times \frac{\sum_{rq \in \mathcal{S}} min(i_{a_i,rq.m}, 0)}{|\mathcal{S}|}$
            compute probability:
            $fit_{a_i}(t) = max(0, l\_score(a_i) - \gamma\; g\_score(a_i))$
            $p_{a_i}(t) = \frac{fit_{a_i}(t)}{\sum_{aa \in \mathcal{A}} fit_{aa}(t)}$
        end for
        selected action: $a^* = argmax_i(p_{a_i}(t))$
    end if
    apply $a^*$;
    if $a^* \in \{CR, DD\}$ then
        unsubscribe from events of old DMN;
        subscribe to events of new DMN;
    end if
end if
```

As we operate in a dynamic environment where the impact and feedback of actions change over time, always selecting the action with the highest score could limit an agent's potential. While the agent must prefer actions that it has tried in the past and that it has found effective (*exploitation*), on the other hand it must also attempt actions that it has not selected before or that have not been selected for a long time since their result may have changed (*exploration*). In reinforcement learning, finding a good compromise between these two behaviors is a common problem, known as the *exploration-exploitation dilemma* [36]. A commonly applied method to balance exploration and exploitation is the ϵ-greedy approach [38]: the agent selects at each step a random action with a small probability p, instead of greedily selecting the optimal one. The selection is guided by the extraction of a uniform random number $rand$. In case of random action (when $rand \geq 1 - p$), the probability of choice is the same for all actions, otherwise the optimal action is selected.

The action selection procedure is summarized in Algorithm 1. When a violation is detected (line 1), the list of valid actions in the current context is extracted (line 2). Then, if exploration is selected a random action is selected from the list (lines 4-5) otherwise the action with the highest score is selected (lines 6-14). The selected action is executed, eventually modifying the subscription to the event manager if change reference or duplication is executed (lines 15-19). In the proposed algorithm, only the ability of the action to improve the requirements satisfaction is considered in the selection, through the evaluation of Eq. (16). A multi-objective approach could be also applied to take into account also the cost of the action (see Eq. (5)) and the trade-off between the cost and the fitness.

6 Evaluation

This evaluation section aims at analysing the ability of the method proposed in this paper to adapt to the frequent changes in a Fog environment and its scalability. These aspects are very relevant in the considered scenario, where the number of data consumers and of nodes available to host data can change very dynamically. Moreover, the number of consumers simultaneously accessing to the same data set can be significant.

The first objective considers if and how quickly the proposed distributed decision system brings the Fog computing environment to a stable environment, in which no requirement violations are detected. The second objective considers how the method deals with an increasing complexity in the fog computing environment.

We will first introduce the elements of the simulation environment that have been used in the experiments. Then we will define the set of tests executed and we will show the results.

6.1 Simulated Fog Infrastructure

In this work, we decided to use a simulated Fog environment instead of a real one to better control the effect of the several independent variables playing a role on the performance estimation. To simulate the execution of a Fog environment we defined the following elements:

- *full nodes* are resources as defined in Eq. (1) hosting a single DAA. In the infrastructure we will have a full node (and consequently a DAA) for each data consumer accessing the data set. A full node is also a potential location for storing the data, thus it can host one or several DMNs;
- *data nodes* are resources as defined in Eq. (1) that can be provided by the DaaS or the data consumers to host the data. Thus, a data node might host one or several DMNs, but not a DAA. In the infrastructure we will have from 0 to several data nodes;
- a network implemented with Toxiproxy[6] to simulate the variability of the network performance (i.e., the latency between the nodes of the infrastructure). Through Toxiproxy we can simulate the proximity or distance between nodes, thus simulating the topological features of a fog environment.

For the purposes of this evaluation, a pool of 20 full nodes (up to 20 concurrent data consumers) and of 10 data nodes (up to 30 locations for storing the data) have been considered. Adaptation actions are enacted by DAAs because of violations of the requirements.

The Fog environment was simulated using the Kernel Virtual Machine (KVM)[7] virtual environment. We used 2 KVM virtual machines, both with 20 cores, 32 GB or memory and 50 GB or hard disk. More information and the images of the KVM virtual machines can be found at [22]. On top of the KVM virtual machines we used Docker[8] virtual environment to simulate the fog network, the full and data nodes. The computation tasks executed by consumers, that used data provided by the DMNs, was simulated using Apache Spark[9]. We used Minio[10] to store data sets managed by DMNs, while we used MySQL[11] to store information shared by the blackboard.

Effort has been put on the reproducibility of the experiment. The repository at [22] contains all the instructions, data and code to create an environment as described in this section. Moreover, to facilitate the reproducibility, we made available the two KVM virtual machines we used to simulate the environment and run the experiments. The execution of the tests is scripted with Bash.

6.2 Metrics and Monitoring

The Fog environment has to monitor a set of metrics that can be used to asses the QoS provided by the DaaS. A monitoring module is part of the DAA and its evaluation is out of scope of this paper. However, its output is essential to detect the violations of the data consumer requirements. We, therefore, partially implemented a monitoring module, only for the metrics that are subject to high variability, while we simulated other metrics by associating a static value to each node.

The metrics used for the evaluation are:

- latency: it measures the delays in the communication between nodes, in our scenario the delay between the node hosting the DMN and the node hosting the DAA. This value can be easily retrieved by every monitoring service activated in an execution environment. For testing purposes, in the experiments latency is simulated using Toxiproxy: for each network connection between two fog nodes, random values of latency are generated using a normal

[6] https://github.com/Shopify/toxiproxy
[7] https://www.linux-kvm.org
[8] https://www.docker.com
[9] https://spark.apache.org
[10] https://min.io
[11] http://mysql.com

TABLE 4
Example of Initial Configuration for 3 Full Nodes(fn) and 3 Data Nodes(dn)

Latency (ms)	fn1	fn2	fn3	dn1	dn2	dn3
fn1		0	1543	1615	246	600
fn2			654	964	723	0
fn3				1336	542	1627
dn1					589	339
dn2						0
dn3						
Availability	98	95	85	78	99	93

distribution, whose average and standard deviation are based on values that we were able to collect by running some tests on a real distributed execution environment (see Table 4);

- response time: it measures the time period between receiving the request for accessing data from the data consumer of the DaaS and the collection of the required data from the DMN. It is measured by the monitoring module;
- execution time: it measures the time used to perform a computing tasks using the data provided by the DMN. It is measured by the monitoring module;
- throughput: it measures the number of requests to access the data served per time unit. It is measured by the monitoring module;
- availability: it measures the uptime of a node. This is an intrinsic property of a node, easily retrievable by every monitoring service. For testing purposes, availability is statically defined for each node when the simulation is created, as reported in Table 4;
- data consistency: it measures the matching between the different copies of a data set. It is a traditional Data Quality metric that can be assessed as in [3]. For testing purposes, in our experiment we artificially inject a change in data consistency, which is maximum at the beginning of the simulation, when only one DMN is available, but decreases proportionally with the number of DMNs of the same data set generated during the simulation.

The *configuration of a Fog environment* is defined as the values of latency between each pair of nodes, the availability property of each node, and the number and position of datasets. The values of other metrics are collected during the simulations. In particular, we implemented a Spark job to simulate an execution of an computation task and we measured its response time and execution time. Table 4 shows an example of values for the initial configuration of a simulation with 3 full nodes and 3 data nodes. Results may be influenced by (not) favourable configurations. To avoid bias, several simulations are executed using different random configurations, where the values of latency are generated from ranges that are reasonable in a Fog environment from our experience and from reports as [2], [35]. Initial configurations used for the tests can be found in [22].

6.3 Requirements Definition and Violation's Detection

One of the inputs necessary to test the solution is the set of requirements of the data consumers, as defined in Eq. (2). In a simulated environment we don't have real consumers, thus we needed to generate synthetic but realistic requirements to associate to each DAA. Getting inspiration from the typical applications using DaaS solutions, we defined two types of consumer's application and, consequently, two types of performance requirements: analytics applications and interactive applications. For an analytics application, throughput, availability, and data consistency are usually more relevant, while for interactive applications latency and response time are prioritised. We specified the requirements using the approach defined in [31], with a goal-driven hierarchical model. In order to set the thresholds for the relevant metrics, the generated sets of requirements are tuned to the simulated network. In fact, setting too strict requirements would have led to perpetual violations, while setting loose requirements would not generate any violation.

For the simulation we generated 20 sets of requirements, one for each DAA. The complete set of requirements can be found in [22]. Violations are detected by the monitoring module of the DAA, comparing the requirements with the simulated metrics.

6.4 Tests Definition

Once all the elements necessary for the simulation of the proposed approach have been defined, we present how the tests are performed. Each test is characterised by a predefined number of full and data nodes, a configuration for the metrics, and an initial data set hosted by a DMN placed in one of the full nodes. For the sake of simplicity and without loosing generality, in the experiments a single data set is considered. In particular, the dataset is a JSON file of 322 MB containing generated information on blood analysis. The file is stored using Minio and is is accessed with Spark jobs that process the file. Starting from an initial DMN linked to the data set made available by the data provider, several copies might be created during the execution to satisfy the requirements of all data consumers. This restriction does not impact the results since the decisions taken on the placement of one data set are not going to affect others.

During a test, each agent performs a local training offline to learn the effect of the actions on its requirements. Once an initial local model is created, all agents cooperate in a shared environment feeding the global and local adaptation models through the blackboard and the impact. An experiment is considered concluded when the system reaches a stable environment, i.e., no violations in the requirements of any data consumer participating in the simulation are detected for 10 minutes. An experiment is considered successful if this condition of stability is reached. The outcome of the experiment is measured in terms of the number of actions that have been necessary to reach a stable environment.

To evaluate the performance of the proposed approach, we tested it under different conditions, exploring different values for the independent variables listed in Table 5. More specifically we explored:

- the number of full nodes considering configuration with 3, 5, 10, 15, and 20 DAAs. The exploration of this variable aims to observe how the number of concurrent users affects the performance;

TABLE 5
Independent Variables and Mitigation Actions

Independent variable		Action applied
# of nodes	# of full nodes	Varied their numbers
	# of data nodes	
Fog nodes configurations		Used multiple configurations
Initial position DMN		Run tests an all possible initial positions

- the number of data nodes exploring configuration with 0, 3, 5, and 10 data nodes. The exploration of this variable aims to check the effect on performance of an additional set of locations available to store the data and the trade-off between more available solution and higher complexity;
- the location of the initial DMN. The choice of the initial position might affect the performance; we tested how this assignment affects the outcome of the experiment.

To explore different aspects of the proposed solution, two types of tests have been defined:

- *stress tests* where starting from scratch, a random configuration is generated and a set of DAAs is added to the Fog environment simultaneously;
- *incremental tests* where a more realistic scenario is simulated in which, starting from a situation of equilibrium between a set of DAAs, new DAAs join affecting the stability.

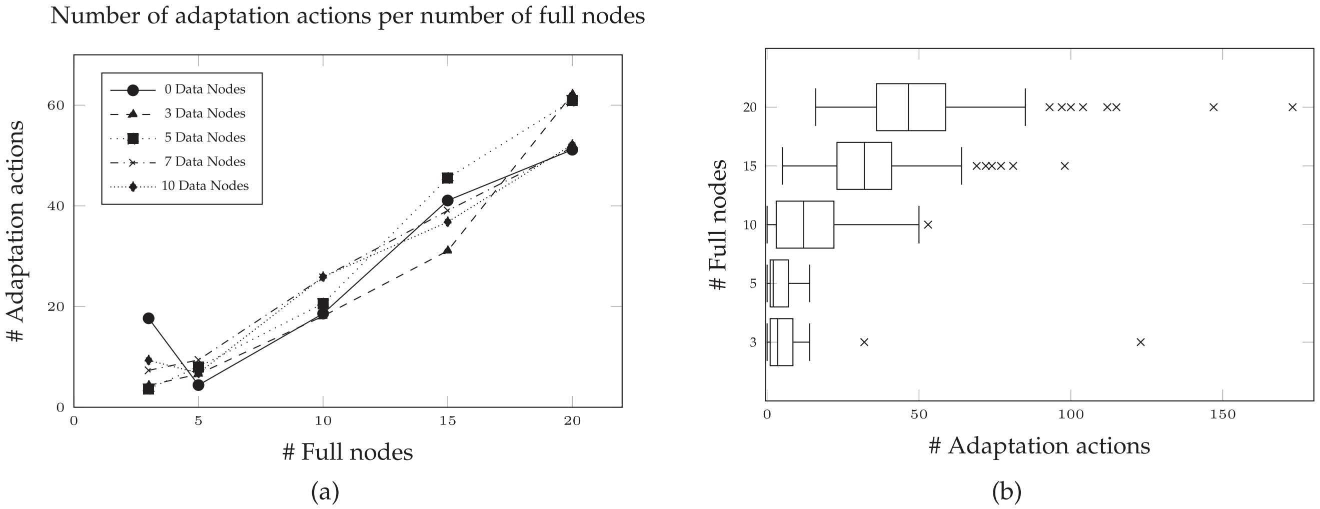
Each test can also be influenced by the initial position of the data provider, thus the first DMN. Therefore, tests for all possible initial positions of the first DMN on the generated network are executed. To make tests with an increasing number of data nodes with the same number of full nodes easily comparable, only full nodes are considered as capable to act as initial position.

We organized the experiments in batches. Each batch consists in 53 tests: 5 sets of full nodes (3, 5, 10, 15, 20 full nodes) where the initial position of the DMN is placed in each possible full node. For each of these configurations, test batches are performed with 0, 3, 5, 7 and 10 data nodes (53 x 5 = 265 tests). As mentioned before, by executing tests using all possible initial positions, the possible effects of a (un)favorable position is mitigated, while varying the number of full nodes and data nodes permits to test if they impact on the scalability of the method.

This experiment is repeated using different randomly generated fog computing configurations to remove the variability introduced by the configuration of the Fog environment. We generated and used 4 different fog environment configurations, for a total of 265x4 = 1060 stress tests and 40 incremental tests.

6.4.1 Stress Tests Results

Fig. 7a shows the aggregated results of the stress tests for one random configuration, changing both the number of full nodes and the number of data nodes. In these tests we wanted to show if and how the number of full and data nodes impacts the convergence time of the algorithm. Each line represents a different number of data nodes. Each point is the average number of adaptation actions required to reach a situation with no violations, starting from same initial configuration but with different initial position of the DMN. Reading the graph, we can draw the following conclusions: (i) observing each single line (fixed number of data nodes), the results show that the number of adaptation actions required to reach a situation where no requirements are violated, generally grows with the number of full nodes, i.e., with the number of consumer's applications, but the growth is proportional ensuring scalability; (ii) observing



(a) (b)

Fig. 7. Variation in the number of adaptation actions enacted, (a) per number of full nodes and data nodes (b) for different initial initial placements of the DMN with increasing number of full nodes.

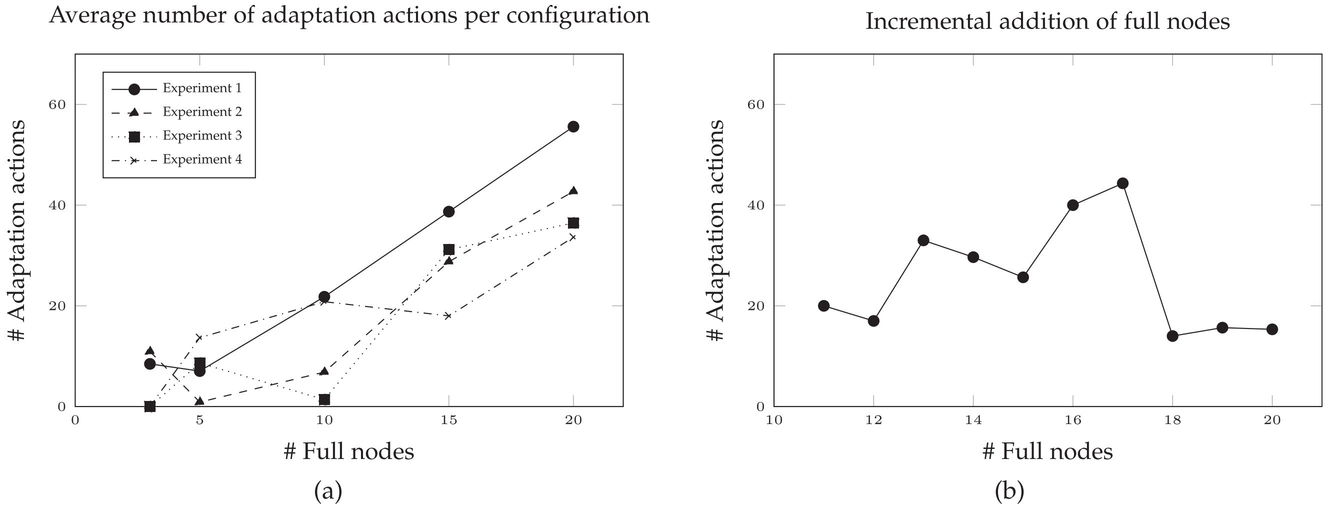


Fig. 8. Number of adaptation actions (a) in four different configurations (b) with incremental adding of DAAs.

the results with a fixed number of full nodes, we can see that the trend is not significantly influenced by the number of data nodes (i.e., the number of actions required is not directly affected by the number of data nodes employed). In other words, the results show that the complexity added to the solution space by adding data nodes does not impact on the number of adaptation actions executed.

Fig. 7b shows the results of the stress tests, using a box plot. In these tests we wanted to check if and how the initial position of the DMN affects the convergence time of the algorithm. In this case, experiments have been executed again by changing the number of full nodes. As in the previous tests, fixed the number of full nodes, all the possible initial placements of the DMN have been explored (e.g., 3 with 3 full nodes, 5 with five, and so on). As shown in the graph, the positioning of the data source may trigger a different set of violations (e.g., a "lucky" initial position may not trigger any violation.

A final set of stress tests explores how the configuration parameters affect the results. We have tested the solution comparing 4 different random configurations, i.e, with random values of latency and availability for each node (see Section 6.2). Fig. 8a shows the results of this set of tests: the behaviour is only marginally influenced by different configurations and keeps its scalability properties in all the configurations tested.

6.4.2 Incremental Tests Results

The number of adaptation actions required may seem high, especially considering that each adaptation action may involve expensive operations like movement and duplication of a DMN. Such a high number is due to the very "stressed" type of tests performed, where multiple DAAss are added simultaneously and have to learn their mutual influence from scratch. In a realistic scenario, data consumers increases and decreases. As the critical element is the increasing trend, the evaluation is focused on that. More specifically, due to a limitation of available resources, at this stage we consider an incremental arrival of customers in small numbers, that, anyway, reflects a reasonable scenario. A different arrival trend, that considers a higher number of new customers in each iteration will be considered in future work. In particular, when a data consumer joins the service, a global model for the existing consumers is already created and it needs to adapt to the requirements of the new consumers. To test the ability of the proposed solution to adapt in such a situation, other simulations are executed by adding a DAA after a stable point (i.e., where no violations are detected for 10 minutes) is reached. In particular, a first run of simulation considers 10 DAAs. When a stable environment is reached, one additional DAA is added and the number of adaptation actions required to reach again an environment with no violation is noted.

Fig. 8b shows the results of the experiment. The tests are executed by adding 10 different full nodes, one at the time. The figure shows the average result of the experiments performed. We repeated the experiment 3 times, with the same initial configuration and initial position of the DMN, to consider non-deterministic behaviours such as the exploration that each DAA may trigger. The average number of adaptation actions required to reach a stable environment is 20, lower with respect to the stressed environment used in the first set of tests. There are two peaks when adding nodes with configurations 13 and 17, due to their positions and their internal requirements. It is worth noticing that the number of adaptation actions required at each step is not influenced by the growing number of DAAs, because at every addition the system adapts only to the requirements of the new data consumer. This is different from the "cold start" situation shown in Fig. 8a.

6.4.3 Local versus Global Decisions

Fig. 9 shows the number of adaptation actions enacted by a decision system considering only local feedback (a.k.a local decision) or including global feedback (a.k.a global decision). The results reported in this Figure were generated with more than 200 stress tests. The difference in the number of adaptation actions enacted is noticeable when the number of full nodes grows over 10 nodes. In particular, with 20 full nodes the number of actions enacted with a global decision is almost half than local decision. Moreover, the standard deviation of global decision, with 20 full nodes,

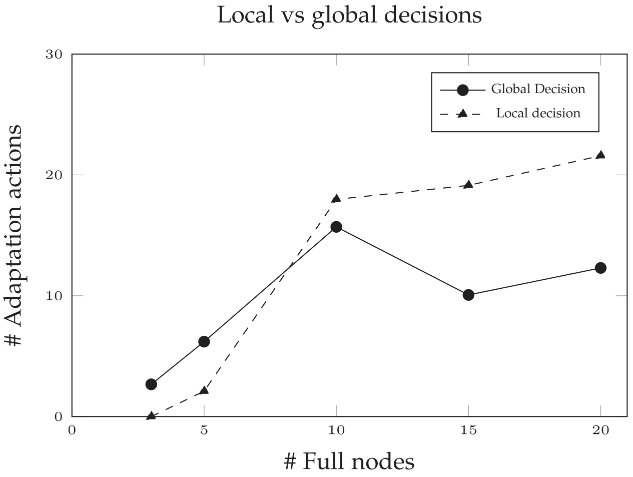


Fig. 9. Number of adaptation actions for global and local decisions.

is 58.3% lower than local decision. These results indicate that the global feedback mechanism proposed in this paper, helps avoiding adaptation actions that, possibly, maximise local requirement satisfaction but negatively impact the fog environment as a whole. When the number of full nodes is low, i.e., lower than 5, the global feedback has not enough impact on the decision system. In other words, the value of $g\_score(a)$, defined in Eq. (13), is too low to influence the decision of the actions defined in Eq. (16). To increase the impact of $g\_score(a)$, γ penalty factor should be in increased, yet this would give too much weight to global feedback letting the external fog environment to influence too much the local decisions.

6.5 Experiment's Conclusions

The discussed results show that the method proposed in this paper is able to manage the conflicting requirements of different data consumers of a DaaS and to reach stability in a limited number of steps. The learned model enables adaptation when additional data consumers join the service. The method is able to work in a very stressed environment, where requirements of multiple DAAs are violated. This is clearly shown in Fig. 7a. The same figure shows that the method is scalable since the number of adaptation actions grows linearly with the number of DAA, without being influenced by the number of data nodes. This is a relevant result for a method oriented to Fog computing, where the number of connected devices that can be used to host data is potentially huge [9]. Fig. 8b shows more realistic values of number of adaptation action that are required to reach a convergence when fog node with a DAA joins the fog environment by accessing to a shared data set.

7 Related Work

In order to compare our approach with the existing literature, we have classified the most relevant contributions in three sets: (i) adaptive approaches based on machine learning; (ii) resource allocation problems; (iii) data management.

Adaptive Approaches. In [16], it is presented a Reinforcement Learning (RL) based QoS controller, with the ability to guarantee differentiated response time requirements for different service classes. A waiting queue is assigned to each service class and managed by a dedicated controller. All the requests are however received and monitored in a centralized way. A solution to monitor, detect, and predict QoS violations is proposed in [39], in the context of web-applications auto scaling. A RL-based decision module triggers management actions. The adoption of this method requires a huge amount of data about the past executions of the service (training data) that may not be available for all fog nodes. Furthermore, the Q-learning offline policy adopted in this work requires that all candidate actions are tested in each state until the convergence is reached during the training period. In [4] authors apply RL in order to build a self-adaptive system capable of monitoring and improve its behavior through corrective actions, for multi-component applications. The approach doesn't scale with the number of applications and metrics, since it is necessary to evaluate all the combinations of the metrics values. Similarly, an algorithm for the selection of web services according to users' QoS requirements is presented in [24]. A MARL approach based on a coordination mechanism is used to learn the service composition task cooperatively. Agents share their experience and select together the best adaptation action. However, the framework proposed in the paper assumes that all agents share the same objective, which is not the case in the considered scenario.

Resource Allocation. The placement of data sources in a Fog environment can be considered as a dynamic allocation problem. The main challenge is due to the lack of a central authority in control of the nodes and their resources. Deep RL for scheduling task assignment in Fog Computing for Mobile Crowdsensing is used in [17]. The main objective is to minimize computing and bandwidth costs and maximize the overall fog nodes allocation. However, the only QoS metric considered is the percentage of completed tasks. A decentralized, cooperative agent-based load balancing mechanism for service placement in IoT networks is proposed in [25]. Agents locally generate possible assignments of tasks to available resources, that are then globally optimized to maximize edge utilization and minimize execution costs. In [8] the authors implemented a QoS enforcement mechanism for service deployment in edge computing, considering the failure risk of edge devices.

The research work analyzed so far focuses on service provisioning. Limited attention has been paid to the link between the service oriented paradigm and *data management*. A decision making method for database container placement in Fog environment based on Markov Decision Process (MDP) is proposed in [13]. This method is able to satisfy QoS metrics by monitoring the environment and taking corrective actions when necessary. However, QoS metrics are application independent and are generally set for the data source and not for its users. Another typical approach to improve end-to-end QoS and ensure scalability consists in adopting *replication strategies*. This strategy is adopted in Content Deliver Network (CDN) [15], where data are cached or moved on servers closest to the data consumers who request them most. Their main limitation is that resources used for caching data are predefined, owned and managed by the provider. This approach cannot be applied in an environment as dynamic as Fog computing, where fog nodes change and there is only a partial control of the environment. Moreover, CDNs only address performance and availability optimization of all the data consumers, not taking into account the objectives of individual consumers.

Data Management. Data intensive applications require proper data management techniques, especially when data are shared among different applications. In [21], the authors proposed a data placement method for data-intensive scientific workflows based on Bayesian networks used to allocate data sets in different distributed data centers. In [20], a RL-based framework is used to learn the optimal data placement policies in a distributed storage system in order to reduce the latency, performed by a single agent. The agent observes the outcome of its placement decisions on the environment considering time-varying user request patterns, network conditions, and end-to-end performance metrics. RL is also used in [14] to perform data migration in hierarchical storage systems. The RL agent is proactive, and it schedules data migration based on the device's characteristics, including transfers per second, read/write per second average and queue size measurements.

The approaches described so far don't take into consideration the requirements of the different users. An attempt in this direction is described in [40], where a database-as-a-service framework offers an adaptive and dynamic provisioning of a database tier based on application-defined policies for satisfying the data consumer QoS requirements. The framework continuously monitors the application-defined SLA and automatically triggers the execution of necessary corrective actions (i.e., scaling out/in the database tier) when required. Users' QoS requirements are also considered in [1]. Data placement problem is addressed using fuzzy rules, associated with a set of action. QoS requirements are here limited to storage resources performance (e.g., read and write speed, number of replicas, security requirements).

Although all of these methods aim to automate data management, none of them considers a scenario in which data are shared among different applications in a dynamic environment as Fog computing, giving the right relevance to the QoS requirements that each application might have against the data. However, all these research work have shown that the adoption of machine learning techniques can also be effectively applied in data management.

This work exploits our previous results discussed in [31] and [7], where the concept of Adaptation Action in a Fog Environment for DaaS has been introduced and defined as a modification of the distribution of data sets or applications, undertaken in order to react to a violation of the user's requirements. For the scope of this paper, we adopted this definition focusing on data set modifications, extending the set of possible actions. In that case, the control on which action to execute was demanded to a central decision system, managing the information about all the data sets and all the data consumers' requirements. Here we have proposed a scalable solution based on a distributed decision approach.

8 Conclusion

This paper has presented a solution to provide a DaaS approach able to exploit the advantages of both cloud and edge environments by relying on the Fog Computing paradigm. The approach enables the satisfaction of the QoS requirements when several users need to share the same data source. The adoption of a multi-agent system to orchestrate the data movement among the different nodes, driven by a reinforcement algorithm, has been demonstrated by the conducted experiments to be a viable solution in terms of reliability, scalability, and sensitivity.

In the paper, we proposed a distributed decision-making system based on the combination of local and global adaptation, in which the interests of both individual agents and global interactions are taken into account. With the proposed approach, each agent can make independent decisions on how to manage shared data sets, integrating its internal knowledge with the feedback received from its neighbors. This allows agents to react quickly to violations of the consumer's requirements without requiring a central controller to coordinate all data placement activities. The results of the trials have shown that the proposed approach is able to converge towards a stable situation in which all consumer needs are met in a limited number of adaptation actions. Furthermore, we have shown that the amount of actions required grows linearly with the number of neighbors involved and is not significantly affected by the number of locations where data can be moved or copied. Through incremental testing, we also demonstrated that the system is capable of integrating new consumers and adapting to new sets of requirements.

As future work, heuristics to further reduce the number of adaptation actions required to reach a stable configuration will be studied. Moreover, the possibility to introduce additional adaptation actions which, for instance, focus on the possibility to move also the computation, will be considered as well. Yet, a more precise role of cost when deciding the adaptation action will be studied. Finally, an extension which studies the possibility to exploit the proposed approach to reduce the energy footprint of a DaaS leveraging on the data movement reduction will be investigated.

References

[1] A. R. Abdurrab and T. Xie, "FIRE: A file reunion based data replication strategy for data grids," in *Proc. IEEE/ACM Int. Conf. Cluster Cloud Grid Comput.*, 2010, pp. 215–223.

[2] S. Agarwal, "Public cloud inter-region network latency as heat-maps," 2018. Accessed: Nov. 2022. [Online]. Available: https://medium.com/@sachinkagarwal/public-cloud-inter-region-network-latency-as-heat-maps-134e22a5ff19

[3] C. Batini et al., *Data and Information Quality*, Cham, Switzerland: Springer, 2016.

[4] N. Belhaj, D. Belaïd, and H. Mukhtar, "Framework for building self-adaptive component applications based on reinforcement learning," in *Proc. IEEE Int. Conf. Serv. Comput.*, 2018, pp. 17–24.

[5] D. Bermbach et al., "A research perspective on fog computing," in *Proc. Int. Conf. Service-Oriented Comput.*, 2018, pp. 198–210.

[6] M. Cantarutti, P. Plebani, and M. Salnitri, "Fast replica of polyglot persistence in microservice architectures for fog computing," in *Proc. Int. Conf. Service-Oriented Comput.*, 2020, pp. 45–55.

[7] C. Cappiello et al., "Improving health monitoring with adaptive data movement in fog computing," *Front. Robot. AI*, vol. 7, 2020, Art. no. 96.

[8] F. Carpio, A. Jukan, R. Sosa, and A. J. Ferrer, "Engineering a QoS provider mechanism for edge computing with deep reinforcement learning," in *Proc. IEEE Glob. Commun. Conf.*, 2019, pp. 1–6.

[9] F. Dahlqvist, M. Patel, A. Rajko, and J. Shulman, "Growing opportunities in the Internet of Things," McKinsey and Co., 2019. Accesed: Nov. 2022. [Online]. Available: https://www.mckinsey.com/industries/private-equity-and-principal-investors/our-insights/growing-opportunities-in-the-internet-of-things

[10] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," *IEEE Access*, vol. 6, pp. 28573–28593, 2018.

[11] P. J. Hancock, “An empirical comparison of selection methods in evolutionary algorithms,” in *Proc. AISB Workshop Evol. Comput.*, 1994, pp. 80–94.

[12] *IEEE Standard for Adoption OpenFog Reference Architecture for Fog Comput.*, IEEE Standard 1934–2018, Aug. 2018.

[13] P. Kochovski, R. Sakellariou, M. Bajec, P. Drobintsev, and V. Stankovski, “An architecture and stochastic method for database container placement in the edge-fog-cloud continuum,” in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2019, pp. 396–405.

[14] T. Lakshmi and R. Sedamkar, “Proactive and adaptive data migration in hierarchical storage systems using reinforcement learning agent,” *Int. J. Comput. Appl.*, vol. 94, no. 9, pp. 46–52, 2014.

[15] F. T. Leighton and D. M. Lewin, “Content delivery network using edge-of-network servers for providing content delivery to a set of participating content providers,” U.S. Patent 6 553 413, Apr. 22, 2003.

[16] D. Li and D. Levy, “A reinforcement learning based self-optimizing QoS controller framework for distributed services,” in *Proc. IEEE Chin. Control Decis. Conf.*, 2010, pp. 2917–2922.

[17] H. Li, K. Ota, and M. Dong, “Deep reinforcement scheduling for mobile crowdsensing in fog computing,” *ACM Trans. Internet Technol.*, vol. 19, no. 2, pp. 1–18, 2019.

[18] K. Li, A. Fialho, S. Kwong, and Q. Zhang, “Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition,” *IEEE Trans. Evol. Comput.*, vol. 18, no. 1, pp. 114–130, Feb. 2014.

[19] K. C. Li, *Handbook of Research on Scalable Computing Technologies*. Hershey, PA, USA: IGI Global, 2009.

[20] K. Liu et al., “A learning-based data placement framework for low latency in data center networks,” *IEEE Trans. Cloud Comput.*, vol. 10, no. 1, pp. 146–157, First Quarter 2022.

[21] F. Ma, Y. Yang, and T. Li, “A data placement method based on Bayesian network for data-intensive scientific workflows,” in *Proc. IEEE Int. Conf. Comput. Sci. Serv. Syst.*, 2012, pp. 1811–1814.

[22] G. Mangiaracina, P. Plebani, M. Salnitri, and M. Vitali, “Experiment results and test resources,” DEIB, Politecnico di Milano, 2021. [Online]. Available: https://github.com/MattiaSalnitri/DaaSin FogComputing

[23] P. Mell and T. Grance, “The NIST definition of cloud computing,” NIST Special Publication 800–145, 2011.

[24] A. Moustafa and M. Zhang, “Learning efficient compositions for QoS-aware service provisioning,” in *Proc. IEEE Int. Conf. Web Serv.*, 2014, pp. 185–192.

[25] Z. Nezami, K. Zamanifar, K. Djemame, and E. Pournaras, “Decentralized edge-to-cloud load-balancing: Service placement for the Internet of Things,” 2020, *arXiv: 2005.00270*.

[26] A. Nowé, K. Verbeeck, and M. Peeters, “Learning automata as a basis for multi agent reinforcement learning,” in *Proc. Int. Workshop Learn. Adaption Multi-Agent Syst.*, 2005, pp. 71–85.

[27] C. Papagianni, A. Leivadeas, and S. Papavassiliou, “A cloud-oriented content delivery network paradigm: Modeling and assessment,” *IEEE Trans. Dependable Secure Comput.*, vol. 10, no. 5, pp. 287–300, Sep./Oct. 2013.

[28] J. Pasquale, “Problems of decentralized control: Using randomized coordination to deal with uncertainty and avoid conflicts,” in *Coordination Theory and Collaboration Technology*. London, U.K.: Psychology Press, 2001, pp. 369–389.

[29] P. Plebani et al., “Improving data-intensive applications in Fog Computing with DITAS,” 2019. [Online]. Available: https://www.ditas-project.eu/wp-content/uploads/2019/12/Improving-data-intensive-applications-in-Fog-Computing-with-DITAS.pdf

[30] P. Plebani et al., “DITAS: Unleashing the potential of fog computing to improve data-intensive applications,” in *Proc. Eur. Conf. Service-Oriented Cloud Comput.*, 2018, pp. 154–158.

[31] P. Plebani, M. Salnitri, and M. Vitali, “Fog computing and data as a service: A goal-based modeling approach to enable effective data movements,” in *Proc. Int. Conf. Adv. Inf. Syst. Eng.*, 2018, pp. 203–219.

[32] D. A. Popescu, “Latency-driven performance in data centres,” Univ. Cambridge, Computer Laboratory, Cambridge, U.K., Tech. Rep. UCAM-CL-TR-937, Jun. 2019.

[33] D. L. Sackett, W. M. Rosenberg, J. M. Gray, R. B. Haynes, and W. S. Richardson, “Evidence based medicine: What it is and what it isn’t,” *BMJ*, vol. 312, pp. 71–72, 1996.

[34] O. K. Sahingoz and A. C. Sonmez, “Agent-based fault tolerant distributed event system,” *Comput. Informat.*, vol. 26, no. 5, pp. 489–506, 2012.

[35] D. Strom and J. F. van der Zwet, “Truth and lies about latency in the cloud,” InterxionTM, 2012. [Online]. Available: https://www.interxion.com/whitepapers/truth-and-lies-of-latency-in-the-cloud

[36] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[37] Cloud Performance Benchmark - 2019–2020, ThousandEyes, 2020. [Online]. Available: https://www.thousandeyes.com/resources/cloud-performance-benchmark-report-november-2019

[38] M. Tokic, “Adaptive ε-greedy exploration in reinforcement learning based on value differences,” in *Proc. Annu. Conf. Artif. Intell.*, 2010, pp. 203–210.

[39] S. Vakilinia, C. Truchan, J. Kempf, and H. Elbiaze, “Automated enforcement of SLA for cloud services,” in *Proc. IEEE 11th Int. Conf. Cloud Comput.*, 2018, pp. 49–56.

[40] L. Zhao, S. Sakr, and A. Liu, “A framework for consumer-centric SLA management of cloud-hosted databases,” *IEEE Trans. Services Comput.*, vol. 8, no. 4, pp. 534–549, Jul./Aug. 2015.

Giulia Mangiaracina received the MSc degree in computer science and engineering from the Politecnico di Milano, Milan, Italy, in 2020. She is interested in data management, business process analytics, and data science.

Pierluigi Plebani received the PhD degree in information engineering from Politecnico di Milano. He is an associate professor with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano. His research interests concern service oriented architectures based on fog and cloud environments, IoT-based multi-party business process management, and green IT.

Mattia Salnitri received the PhD degree in computer science and telecommunication from the University of Trento. He is a research assistant with Politecnico di Milano and a visiting researcher with Bournemouth University. Before that, he was a post-doc researcher with University of Trento. His research focuses on fog computing and security in socio-technical systems.

Monica Vitali received the PhD degree in information engineering from Politecnico di Milano. She is an assistant professor with Politecnico di Milano. Since 2020, she is also visiting researcher with Umeå University. Her research interests concern adaptive information systems, application management in cloud and fog computing, and green IS.

Open Access funding provided by ‘Politecnico di Milano’ within the CRUI CARE Agreement