Monitoring-aware Optimal Deployment for Applications based on Microservices

Edoardo Fadda, Pierluigi Plebani, and Monica Vitali

Abstract—Modern cloud applications are required to be distributed, scalable, and reliable. The microservice architectural style enables developers to achieve this goal with reduced effort. Nonetheless, microservices deployment is not trivial due to the heterogeneity of the microservices in terms of both functional and non-functional requirements. This is also true when considering the monitoring requirements that are specific to each microservice and must be satisfied in order to enable the verification of the application objectives satisfaction. However, not all providers offer the same set of metrics with the same quality.

The goal of this paper is to provide an approach for supporting the deployment of microservices in multi-cloud environments focusing on the Quality of Monitoring. Adopting a multi-objective mixed integer linear optimisation problem, our approach supports the application owner in finding the optimal deployment for satisfying all the constraints and maximising the quality of monitored data, while minimising the costs. To this end, a knowledge base is introduced to mediate between the perspectives of the cloud provider and the application owner, while a Bayesian Network is adopted to enhance the provider's monitoring capabilities by estimating metrics requested by the application owners that the cloud provider is not able to monitor.

Index Terms—Monitoring system, microservice, multi-cloud, cloud computing, deployment optimisation

1 Introduction

SOFTWARE solutions based on the microservice architectural style [1] are increasingly gaining the attention of software engineers, as their underlying pattern promises to enable the development of highly reliable and scalable applications with reduced effort compared with traditional approaches. The benefits of adopting a microservice-based architecture are clear, especially in the development phase. Unlike monolithic applications, this new approach requires an application to be organised into a set of distributed, communicating microservices, the development of which has boosted the adoption of agile methodologies [2]. Based on the application design, microservices collaborate at runtime to satisfy the global objectives of the application (e.g. performance, security, and scalability) and an effective monitoring system is required to check whether such objectives are actually achieved.

While on the one hand the adoption of the microservice architectural style has several advantages in terms of agility, on the other it makes appropriately supporting the monitoring of microservice-based applications more challenging [3]. In fact, due to the independence of the teams involved in the development of the microservices and the different characteristics of such microservices, it results in different monitoring requirements. In some cases, microservices are CPU intensive, while others are more storage intensive. As a result, different metrics with different details may be required. These different requirements affect deployment, as the microservices must be placed in a site where the cloud provider's capabilities reflect the application owner's requirements. Moreover, additional complexity comes from the scalable and flexible nature of microservice-based applications, thus the place in which they are deployed needs to offer a scalable monitoring solution as well.

The goal of this paper is to support the deployment of microservice-based applications by considering the Quality of Monitoring (QoM), i.e., the ability of a set of cloud platforms to satisfy the monitoring requirements of the microservice-based application, as a key factor at deployment time, in order to make the management and collection of monitoring data simpler and to optimise the monitoring capabilities. QoM is relevant since monitoring data about the application provides the instruments for analyzing the efficiency and effectiveness of the application and for taking informed decision about improvements and modifications. To this end, the main contributions of this paper are:

- A definition of QoM for microservice-based applications that takes into account the ability of a cloud provider to cover the set of required metrics while also providing a predetermined degree of completeness.
- A framework allowing developers of microservice-based applications to establish the requirements for the deployment of the application and that supports them in the establishing the relevant metrics by means of a *knowledge base*.
- An API that allows cloud providers to express their capabilities in terms of QoM.
- The implementation of a multi-cloud deployment optimiser to suggest the best match between application requirements and cloud capabilities using a multi-objective mixed-integer linear optimisation problem (MOMILP) algorithm while taking into account also issues relating to the potential scalability of the application. The optimiser exploits dependencies between

- *Edoardo Fadda, DAUIN department, Corso Duca degli Abruzzi 24 - 10129 Turin, Italy. E-mail: edoardo.fadda@polito.it*
- *Pierluigi Plebani and Monica Vitali, DEIB department, Politecnico di Milano, Piazza L. da Vinci 32 - 20133 Milan, Italy. E-mail: [pierluigi.plebani,monica.vitali]@polimi.it*

Manuscript received XXXX XX, XXXX; revised XXXX, XXX, XXX.

monitoring metrics expressed in the knowledge base through a Bayesian Network (BN).

This paper is an extension of a previous work [4] in which a simpler version of the matchmaking algorithm was applied to the deployment of Virtual Machines (VM) as opposed to microservices hosted in containers. With respect to the previous scenario, this new approach makes it possible to express the monitoring capabilities also at a dimensional level and thus in a more abstract way. Moreover, the scalability that may be required by the microservices is considered in the optimisation model to find a deployment that is valid also during the application scaling in and out.

The rest of the paper is organized as follows. Sect. 2, using a running example, sets out the motivation for developing a monitoring-aware deployment for microservice-based applications. Sect. 3 provides a definition of QoM while highlighting the research challenges that should be considered at deployment time. Sect. 4 discusses the details of our approach, whereas Sect. 5 contains a formal description of the optimisation problem that drives the monitoring-aware deployment. Sect. 6 validates the approach in terms of response time also investigating scalability issues. Related work is discussed in Sect. 7, while Sect. 8 contains some concluding remarks and suggests possible future directions for research in this area.

2 Motivation

The need for an approach to deploying a microservice-based applications that also takes into account the issue of monitoring is illustrated by the application shown in Fig. 1. The goal of this application is to provide support in predicting the trajectories of eels in the ocean based on previous observations stored in an oceanographic database. The application consists of three microservices: (i)**MS1** performs a pre-processing of raw data about oceans conditions and historical data about the behaviour of eels. The data are cleaned and restructured (considering eels' cohorts and years of observations) to make processing more efficient; (ii) **MS2** calculates the trajectories for a single year of a cohort of eels. Depending on the number of years to be considered and the size of the cohort, several instances of this microservice may be required and executed in parallel. (iii) **MS3** aggregates the results of the instances of MS2 and performs some post-processing to help with visualization of the data. We can assume that three different development teams are involved in developing these microservices. In addition to the functional requirements – such as the number of CPUs or the amount of memory for the VM – these teams correspond to different requirements in terms of monitoring capabilities depending on the objectives of the microservices:

- MS1: I/O throughput, Availability and Free Storage.
- MS2: I/O throughput, CPU usage and Availability.
- MS3: CPU usage, Memory usage and Availability.

As the efficiency of MS1 and MS2 depends on the ability to read and write data from data sources, monitoring the I/O speed is fundamental. At the same time, MS2 and MS3 perform CPU-intensive tasks and their efficiency depends on the CPU usage level as well as on the ability to not saturate the memory. The operator must also take into

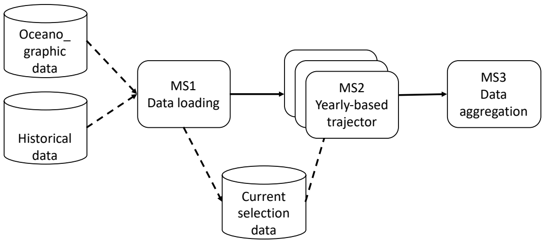


Fig. 1. Eels' trajectory computation case study.

account these monitoring requirements in addition to the usual infrastructure and economic constraints. Since different cloud providers may have very different monitoring capabilities and contracts, selecting an optimal deployment approach is not an easy task. The differences regard (i) the metrics that each cloud provider is able to monitor; (ii) the degree of customisation that allows the user to define custom metrics; (iii) the rate at which the metrics are collected; and (iv) the cost of the monitoring service provided. An examination of the services of the main cloud providers shows these differences clearly. Google Cloud[1] charges its customers according to the amount of monitoring data collected per billing account or the number of monitoring API calls. Moreover, the metrics directly provided by the Google Cloud Platform (GCP) are free, while other metrics (e.g., user metrics or AWS metrics) are provided for a supplementary charge. The price of the monitoring service[2] is mainly defined by: (i) the amount of resources, (ii) the number of metrics, (iii) whether the metrics are GCP metrics or not, and (iv) the rate at which the metric data is written (affecting the monitoring data volume). Similarly, Amazon Cloud Watch[3] provides a set of predefined metrics and the option to define custom metrics. Two main service levels[4] are offered to the user: (i) a free service that provides only basic metrics at a 5-minutes rate and up to 10 metrics at a 1-minute rate with a limited amount of monitoring APIs calls, (ii) a paid service that enables custom metrics to be defined and provides metrics at a 1-minutes rate. The cost depends on (i) the number of metrics, (ii) the monitoring rate, (iii) the number of instances, and (iv) the geographical region of deployment. The IBM Cloud Monitoring service[5] collects all metrics at a 1-minute rate and also allows custom metrics to be defined to complement the basic ones. Two plans ("lite" and "premium") are offered at a cost depending on the amount of monitored metrics. Other monitoring services also provides different pricing plans for a different number of instances or metrics (e.g., CloudMonix[6] and New Relic[7]). Finally, as the number of microservices of an application can be significant, deployment can involve different sites, making it necessary to deal with heterogeneous monitoring systems. Some monitoring solutions (e.g., New Relic or Datadog[8]) have been proposed to integrate the monitor-

[1] https://cloud.google.com/monitoring/
[2] https://cloud.google.com/stackdriver/pricing
[3] https://aws.amazon.com/cloudwatch/features/
[4] https://aws.amazon.com/cloudwatch/pricing/
[5] https://console.bluemix.net/catalog/services/monitoring
[6] http://cloudmonix.com
[7] https://newrelic.com/application-monitoring/pricing
[8] https://www.datadoghq.com/

ing data gathered from the different monitoring platforms on which the microservices are deployed with a view to support the analysis of these data to find bottlenecks, and managing alerts in the event of problems.

This complex situation means that the deployment of an application based on microservices – in addition to the functional constraints – is required to manage several aspects. Therefore, to simplify this step, the proposed approach offers a solution that identifies the variables that are relevant to the problem and computes the optimal deployment. In doing so, monitoring aspects are considered to address one of the relevant issues raised by the scheduling of microservices as discussed in [3].

3 Quality of monitoring

In this paper we introduce the concept of QoM:

Definition 1. *QoM is the fitness of the cloud provider capabilities in relation to the specific microservice-based application requirements in terms of QoS measurements that considers both the ability to monitor a metric and to provide a suitable measurement. QoM can be influenced by several factors, such as the level of confidence in the data (e.g., precision, accuracy, completeness) and the overhead needed to manage the monitoring system (in terms of resources for managing, storing, and retrieving the data).*

As we consider QoM a key to the optimal deployment of microservices, the overhead of the monitoring system can be considered equivalent for all the cloud providers and can be ignored since it has no impact in the deployment decision. In fact, given the application requirements, a similar volume of monitoring data would be generated by each of the cloud providers and how these data are managed by the storage system is not of interest to the application owner.

Conversely, we are interested in the level of confidence provided by the data, which can vary significantly between different cloud providers. Accuracy and precision are quality metrics used to evaluate the quality of the monitoring system sensors. Although these metrics can be of interest, they are rarely published by cloud providers and difficult to estimate. Estimation requires us to know the expected behaviour of the monitoring data to be compared with the result of the monitoring system. It is not feasible to obtain this information prior to deployment. As a result, these two metrics are overlooked unless they are explicitly declared in the capabilities of cloud providers. In this work we focus on the completeness of the monitoring system. This property is measured by (i) the extent to which monitoring information available for a cloud provider covers the monitoring requirements of the application, and (ii) the extent to which the sampling rate offered by the cloud provider satisfies the requirements of the application owner (see Sect. 5.1).

While much of the literature in this field focuses on the ability of a cloud provider to measure a set of required QoS metrics [5], none considers the quality of the monitored data. In particular, as the focus here is on how to optimise deployment, we assume that the following are available: (i) a declaration of QoM capabilities on the part of a set of cloud providers as well as the associated cost, and (ii) a list of the set of microservices comprising an application along with QoM requirements and the total budget. At deployment time, it is necessary to identify the combination of cloud providers and microservices that best satisfies all the QoM requirements while remaining within budget. There are several potential challenges to achieving this:

- *Mismatched definitions of QoS metrics* (C1): the way in which the metrics offered by the cloud providers and required by the microservices are defined may be different. As a result, supply and demand cannot be matched as terms of comparison are unavailable. Moreover, application owners may have insufficient knowledge of the finer details of the aspects that they wish to monitor, and may need support in defining their monitorability requirements. To mitigate this issue a knowledge base is introduced.
- *Mismatched quality of QoS metrics* (C2): the monitoring capabilities offered by cloud providers are usually expressed in terms of predefined packages. A cloud provider may be unable to monitor a specific metric or its monitoring may be of lower quality (e.g. sampling rate lower than requested). As cloud providers allow certain monitoring capabilities to be customised, the proposed approach takes into account also the work required to implement such customisation.
- *Unavailability of QoS metric monitoring* (C3): customisation can be implemented even when a required metric is not available at all. This especially in the case of application level metrics that are rarely supported natively by cloud providers but for which tools to include them in the monitoring systems are provided. In some cases, however, even when these tools are available, certain metrics may not be monitored. The matchmaking approach proposed here also considers the option of using a BN to estimate the values of the unavailable metrics.
- *Application scalability* (C4): scalability is a basic feature of microservice-based applications. This means that, at runtime, several instances of one or more of the microservices comprising the considered application can be created. All of these instances must therefore be monitored. As scalability is managed at runtime, optimisation of QoM at deployment time must consider also this variable.

How the approach proposed here deals with these challenges is investigated in the following sections.

4 An approach to monitoring-aware deployment

The approach to supporting QoM-aware deployment of microservice-based applications proposed here is illustrated in Fig. 2. Two classes of stakeholders are considered: the *application owner*, who expresses QoM requirements for the microservices, and the *cloud provider*.

The approach is built on the information contained in a *Knowledge Base* (Sect. 4.1) consisting of two parts: a *Hierarchical Model of Dimensions* (HMD), representing the metrics at different levels of granularity, and a *Bayesian Network* (BN) of metrics providing information about the dependencies between different metrics. Application owners specify quality and monitoring constraints for each of

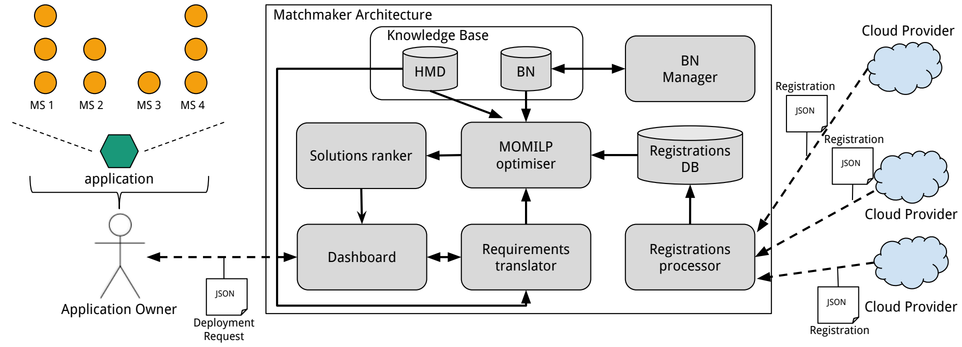


Fig. 2. The general architecture of the proposed approach.

the microservices composing the application by interacting with the framework through a *Dashboard* that can be used to specify requirements. Such requirements are then translated into the constraints that define the optimisation problem by the *Requirements Translator* (Sect. 4.2). On the other side, each cloud provider declares its services as a set of quality parameters and the list of metrics provided by its monitoring system together with the quality associated with the monitoring of each specific metric (Sect. 4.3). The services offered by individual cloud providers are translated into an appropriate format by the *Registration Processor* and stored in a dedicated database called *Registrations DB*. The core of the architecture is the *MOMILP optimiser* that is used to find the best deployment plan for matching the application owners requirements and the capabilities of the cloud provider. The results of the optimisation algorithm are presented to the application owner through the dashboard.

4.1 The knowledge base

The primary role of the knowledge base is to solve the issue of *mismatched definitions of QoS metrics (C1)* introduced in Sect. 3 by providing a common framework for application owners to specify their monitoring requirements and cloud providers to specify their monitoring capabilities while taking into account the different perspectives that these two stakeholders may have. For this reason, the HMD is built around three main concepts: dimensions, metrics, and metric measurements, hierarchically organised (see Fig.3(a)).

Definition 2. *A **dimension** is a high-level characteristic and is not measured directly. It is defined by a name and a set of metrics used for its evaluation:*

$$d_i \in \mathcal{D} =< name, \{m_j\} >$$

As shown in Fig. 3(a), for the running example, we assume that nine dimensions are considered. A microservice can be executed in a container or in a VM, thus the *status* of the container or of the virtual machine running the microservice is of interest, as well as the status of the physical machine where the VM or container is instantiated. The same holds for the *performance* and *sustainability* dimensions.

Definition 3. *A **metric** is a quantitative measure of the degree to which an application or a microservice possesses a given attribute. Each metric is linked to a dimension and expresses a way to quantify such a dimension. A metric is a low-level requirement and is defined by a name and a function used for its computation based on a number of measurements in the environment:*

$$m_j \in \mathcal{M} =< name, f(mm_k) >$$

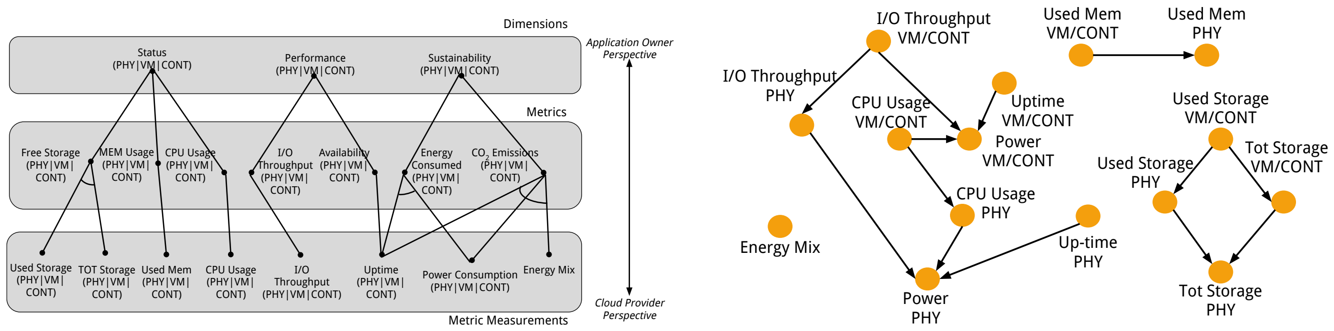
For instance, the *VM status* dimension is influenced by a set of metrics, i.e. *VM Free storage*, *Mem Usage*, and *CPU Usage* metrics. Generally speaking, the metrics connected to a dimension represent non-exclusive alternatives for evaluating the dimension (OR relation). For this reason, someone could represent the VM status only in terms of CPU Usage, some other might be interested in all of the metrics.

Definition 4. *A **metric measurement** is a measurement provided by the monitoring system used to quantify the value of a metric. It is defined by a name and the sampling rate at which it is collected:*

$$mm_k =< name, samplingRate >$$

Each metric is linked to one or more metric measurements. For instance, the *CONT Free Storage* metric is computed according to a function that uses the following metric measurements as inputs: (i) *CONT TOT Storage* related to the total storage space allocated; (ii) *CONT Used Mem* related to the amount of used memory in the container. As both of these metric measurements are required to compute the *CONT Free Storage* metric, an AND relation (represented by an arc linking all the required metric measurements) is used to represent this dependency.

In the literature, a number of approaches have been proposed to model the characteristics of quality attributes that can be adopted for the implementation of the HMD. For instance, ISO 25010:2011 [6] defines quality of service hierarchically as a set of characteristics, sub-characteristics, quality properties, and measurements. The model can thus capture the different stakeholder perspectives that are distinguished in the standard between primary users (application owners in our case) and the secondary users (cloud providers). The NOVI project [7] proposes an information model organised into a set of ontologies, including: (i) a monitoring ontology that allows quality parameters to be defined in terms of measurement levels and units, and (ii) a resource ontology to associate measurements with the objects to be monitored. It is worth noting that, although the creation of a shared knowledge base has the advantage of providing a common framework to specify requirements and capabilities [8], creating one often requires considerable effort. At the same time, the way in which metrics are currently published by



(a) The HMD. Dimensions, metrics, and metric measurements can be specified for the physical (PHY), virtual (VM) or container (CONT) layers.

(b) The metrics Bayesian Network.

Fig. 3. Elements of the knowledge base [4]

cloud providers (e.g. AWS Cloudwatch) often makes the task of understanding their meaning and characteristics a lengthy one. Our approach, therefore, supports the idea - also followed by the proposers of ontologies for monitoring systems in various domains such in buildings, processes, and data centres - that the advantages of a shared, hierarchical model, requiring a significant effort in the beginning, may save time during the microservice design and deployment phases.

The second component of the knowledge base is the Bayesian Network (BN). While the HMD maps the relations that exist at different levels between dimensions, metrics, and metric measurements, the BN focuses on the lower layer, modelling relations between metric measurements. The BN can predict trends in some metrics without collecting actual data from the monitoring system. Trends express the observed or predicted behaviour of a metric measurement, i.e. whether the value of a metric measurement is increasing or decreasing. The BN model is based on the assumption that metrics might not be independent, and that any dependencies can be either explicit (declared in the formula through which a metric is computed, see Def. 3), or implicit when their relation is hidden. Relations between metrics can be extracted by analysing data collected by the monitoring system, and represented through a BN expressing causal relations among them as discussed in [9]. The proposed BN is a general model computed from monitoring data collected by several cloud providers. In the network, nodes are metric measurements of interest, while edges map causal relations between metric measurements. Trends in a metric measurement are dependent only on the knowledge of the values and/or behaviour of its parent nodes in the BN. The estimated trend is associated with a confidence value contained in the Conditional Probability Table (CPT) associated with the node that quantifies the influence of the parents on the child node. An example of a BN based on the set of metric measurements contained in Fig. 3(a) is shown in Fig. 3(b).

The BN represents shared knowledge built from a generalisation of the observed relations between metric measurements from different cloud providers. It can be used by the optimiser as an alternative to collecting actual metrics in the event of *unavailability of QoS metric monitoring (C3)* and when the cost of customisation does not justify the benefit obtained. In fact, the application owner may not be interested in the detailed metrics but only in an aggregated value that can be provided at no cost starting from other information already available. Our approach allows for this kind of specification (Sect. 4.2).

4.2 Specifying deployment and QoM requirements

Application owners wishing to deploy the application specify the requirements through which the best match with the cloud providers can be identified. They provide general information about the application and specific information about deployment requirements. QoM requirements can be specified for the whole application and for each microservice. For each microservice, application owners indicate the *id* and the number of instances required, together with the expected variation. This deals with the issue of *application scalability (C4)* by providing the optimiser with useful information. With regard to QoM constraints, the application owner specifies monitoring system quality requirements, thus dealing with the issue of *mismatched quality of QoS metrics (C2)*. Requirements can be specified for the whole application or at the level of each microservice comprising the application, allowing different perspectives to be specified according to the features of the different microservices. Moreover, requirements can be specified at the *metric level* (owners select the set of metrics that they wish to monitor for each dimension), or at the *dimension level* (application owners do not go into the details of the metric selection, and specify only the dimension to metric translation policy exploiting the HMD). The translation policies are: (i) *ALL*: selects all the metrics related to the dimension; (ii) *XOR*: a metric related to the dimension is selected automatically; (iii) *N*: at least N metrics for the dimension specified by the application owner are selected automatically. The policies deal with the issue of *mismatched definitions of QoS metrics (C1)*. Finally, application owners specify the level of precision required for each of the selected dimensions or metrics: they may require (i) *detailed* monitoring, indicating the desired sampling rate at which the information should be collected, or (ii) *trends* in a specific dimension or metric (also specifying the update rate for the trend). In this case, the optimiser will use the BN when solving the deployment optimisation problem. Application owners also specify *deployment constraints*. For instance, they may ask the same provider to deploy a set of microservices

```
{"application_owner_request":
 { "date": "01-03-2017", "user_id": "usr_34675",
   "requests": [
     {"microservice": "ms1",...}
     {"microservice": "ms2",
      "instance_mean": 10,
      "instance_var": 5.0,
      "qom_constraints":
       {"constraints":
          [{"dimension": "d0",
            "name": "status",
            "trans_policy": "none",
            "acc": "none", "prec": "none",
            "metrics":
              [{"metric": "m8",
                "name": "CONT CPU usage",
                "sampling": 60000, "prec": "detailed"
                    },]},
           {"dimension": "d1",
            "name": "Performance",
            "trans_policy": "ALL",
            "sampling": 60000, "prec": "detailed"},
           {"dimension": "d2",
            "name": "Sustainability",
            "trans_policy": "XOR",
            "sampling": 1800000, "prec": "trend"}]}}
      {"microservice": "ms3",...}],
   "dep_constraints": [
     {"ms_list": ["ms1", "ms2"], "type": "same"},
     {"ms_list": ["ms1", "ms3"], "type": "diff"},
     {"ms_list": ["ms1"], "type": "pref",
      "providers":["cp3"]}],
   "budget":{
     "currency": "$",
     "maxbudget": 1000, "estimated": 700}}}
```

Listing 1. Example of JSON code for a deployment request

(e.g. microservices sharing a large amount of data), or in contrast they may require the microservices to be deployed in different locations (e.g. microservices that may interfere with each other). They may also require a microservice to be deployed by a specific cloud provider. Finally, application owners specify *budget information*, i.e. the budget they are willing to spend on deployment as well as the maximum budget that can be allocated for the solution.

An extract from the deployment request, expressed using JSON, can be seen in Listing 1. Based on the example in Fig. 1, the monitoring requirements for the three microservices comprising the application for analysing eel movements described above are expressed. With regard to QoM for *MS2*, three dimensions are considered, specifying each metric of interest for *Status* (only CPU usage), the ALL translation policy for *Performance* (both availability and I/O throughput are of interest for the application), and the XOR translation policy for *Sustainability*, specifying that it is an extra item of information for which no specific metric is required. A hierarchical rule is applied so that requirements at the higher level of abstraction are inherited by lower-level elements if not overwritten. The application owner also specifies deployment constraints and budget information. In the example, microservices *MS1* and *MS2* are deployed together, while *MS1* and *MS3* are deployed in two different locations. Moreover, a specific location is specified for *MS1*, e.g. forcing *MS1* to be deployed nearer the data sources for performance reasons. Finally, budget constraints are specified. The deployment request, generated by the interaction of the application owner via the *Dashboard* module, is then translated by the *Requirements translator* module into goals and constraints in a format that can be used by the *MOMILP optimiser*.

4.3 Specifying monitoring capabilities

The requirements specified by application owners must be matched with the capabilities of the various cloud providers to find the best solution. Providers typically publish information about declared QoS. Our approach enhances this information by also declaring the QoM characteristics of the cloud provider as well as the set of metric measurements and their sampling rates. With regard to QoM, several cloud providers offer different sets of metric measurements at different conditions in terms of costs and accuracy. The same cloud provider may offer different monitoring services at different costs, as discussed in Sect. 2.

In our approach, cloud providers may specify cost information defining several *budget profiles*: the basic profile expresses the cost for each microservice instance for collecting basic monitoring information at the declared sampling rate. Premium profiles may offer more accurate monitoring at a higher cost. Monitoring capabilities are declared through the set of metric measurements collected by the monitoring system. Our approach focuses on an extensible monitoring approach where this set of metric measurements is complemented by custom metrics. In fact, several cloud providers allow customers to specify custom metrics that can be collected by implementing new hardware or software probes, the cost of which in some cases can be high. An illustrative extract from one cloud provider's registration expressed using JSON can be seen in Listing 2. The cloud provider's monitoring capabilities are translated by the *Registrations processor* into a format that can be used by the *MOMILP optimiser*. With reference to the running example in Sect. 2, the cloud provider specifies two of the metrics of interest (CPU usage and Availability through the Downtime metric measurement) at the required sampling rate with a premium profile, while I/O throughput and sustainability metrics such as Energy are provided as custom metrics (through the Power Consumption custom metric measurement) at an additional cost.

5 Optimisation

In this section, we describe the MOMILP optimiser module of the architecture proposed in Fig. 2. The goal of the optimiser is to find the best deployment strategy in order to match the requirements of application owners with the capabilities of the registered cloud providers. In the optimisation phase, all of this information is transformed into the formulation of a set of goals and a set of constraints of a MOMILP, exploiting the additional information contained in the HMD and in the BN. The definition of the goals of the multi-objective problem is described in Sect. 5.1, while Sect. 5.2 formulates the optimisation problem.

5.1 Determining goals: monitoring quality and cost

The deployment optimisation problem considered in this paper has two main goals: to provide a deployment solution

```
{"cloud_provider_offer":
 {"date": "01-01-2017", "user_id": "cp_321",
   "budget_profile": [
    {"id_profile": "basic",
     "cost_per_inst": 20, "sampling": 300000},
    {"id_profile": "prem1",
     "cost_per_inst": 30, "sampling": 60000},
    {"id_profile": "prem2",
     "cost_per_inst": 50, "sampling": 30000}],
   "basic_metric_measurements": {
       "cost": 0,
          "mms": [
          {"mm": "mm3", "name": "PHY Used Mem"},
          {"mm": "mm6", "name": "PHY CPU Usage"},
          {"mm": "mm20", "name": "PHY Uptime"},
          {"mm": "mm5", "name": "CONT Used Mem"},
          {"mm": "mm8", "name": "CONT CPU Usage"},
          {"mm": "mm14", "name": "CONT Uptime"}]},
   "custom_metric_measurements": {
       "cost": 5,
       "mms": [
          {"mm": "mm0", "name": "PHY Used Storage"},
          {"mm": "mm0", "name": "PHY Tot Storage"},
          {"mm": "mm2", "name": "CONT Used Storage"},
          {"mm": "mm2", "name": "CONT Tot Storage"},
          {"mm": "mm11", "name": "CONT Throughput"},
          {"mm": "mm17", "name": "CONT Power"}
          ]}}}
```

Listing 2. Example of JSON code for the registration of a cloud provider

that collects as much information as possible (based on the application requirements) about the microservices by monitoring them while meeting budget constraints. We thus face a multi-objective optimisation problem that requires us to maximise the monitoring quality for the application owner for each dimension or metric, and to minimise the cost of the solution. For the sake of simplicity, we have translated this multi-dimensional problem into a bi-dimensional problem in which the maximisation of the monitoring quality for each metric is aggregated into a single value. In short, the aim is to maximise monitoring quality while minimising costs. This simplification makes the solution easier to understand for the application owner who can decide the priority to give at each of the two perspectives. It is worth noting that this decision does not affect the complexity of the problem while improving the interpretation of the results.

5.1.1 Computing Quality of Monitoring

Before formulating the optimisation problem, it is necessary to formalise how the QoM is computed. As introduced in Sect. 3, we focus on the completeness as a property of the monitoring system, which is considered as a combination of (i) the coverage in the probes available for a cloud provider compared to the requested metric measurements required for the application and (ii) the percentage of samples collected by the monitoring system compared to the required sampling rate. The application owner is asked to define monitoring requirements supported by the HMD at the dimension or (optionally) at the metric level. Therefore, it is important to define how the completeness is computed at each level of the HMD.

Definition 5. *The* ***completeness of a dimension*** *is the minimum completeness of the metrics selected to assess the dimension according to the translation policy selected by the application owner.*

As described in Sect. 4.2, this can involve either all the metrics measuring that dimension, a subset of them, or a specific set selected by the application owner.

Definition 6. *The* ***completeness of a metric*** *is the minimum completeness of the metric measurements used to quantify the value of the metric in a specific site.*

According to the knowledge base, several metric measurements are needed to compute a metric. For each cloud provider, the metric measurement with the lower quality is the one that influences the completeness of the whole metric.

Definition 7. *The* ***completeness of a metric measurement*** m *for a specific cloud provider* p*, indicated as* $a_{mp}^{(A)}$*, is proportional to the ratio between the sampling rate required in the deployment request and the sampling rate of the probe provided by the cloud provider for measuring that metric measurement.*

Completeness may also depend on the variability of the metric measurement. In fact, if the metric measurement is almost steady, monitoring with a lower sampling rate does not affect completeness. Contrairwise, meeting the required sampling rate is important in the case of a highly variable signal. Thus, completeness is computed considering the entropy of the monitored property, estimated using the Kullback-Leibler divergence (KLD) [10] [11]: a specialisation of the Rényi entropy [12]. KLD expresses the divergence between two signals, in terms of the amount of information gained using a signal, P, compared to another signal, Q. In our scenario P is the probability distribution of samples at the required sampling rate and Q is the probability distribution of samples at the sampling rate provided by the cloud provider. The Kullback-Leibler divergence (KLD) between P and Q is expressed as:

$$D(P||Q) = \sum_i p_i \log \frac{p_i}{q_i} \tag{1}$$

To express the completeness as proportional to the ratio between the required sampling rate and the sampling rate offered by the monitoring system, and to the loss of information, we can model completeness as follows:

$$a_{mp}^{(A)} = 1 - s_T \cdot (1 \exp^{-D(P||Q)}) \tag{2}$$

where $s_T = (1 - min(1, \frac{T_P}{T_Q})$ with T_P the sampling rate required by the application and T_Q the sampling rate offered by the cloud provider. According to Eq. 2, completeness of a metric measurement is dependent on s_T in a way that is proportional to the loss of information in downsampling the metric measurement. If the metric measurement is not provided (i.e., $s_T = 0$ and $D(P||Q) = 0$) completeness for the considered metric measurement is 0. If the metric measurement is provided at the required rate ($s_T = 1$ and $D(P||Q) = 0$), completeness is 1. For values in between, the relevance of s_T in computing completeness is proportional to the loss of information: completeness is 1 if the two distributions are equivalent ($D(P||Q) = 0$) and 0 if the distributions are completely different ($D(P||Q) \rightarrow \infty$). KLD computation can be easily evaluated at design time by appropriately sub-sampling a monitoring set of samples generated by monitoring the application in a test environment.

If the cloud provider does not supply any information about a metric measurement, then $a_{mp}^{(A)} = 0$. In this case, the cloud provider may implement a new probe to monitor the metric measurement at an additional cost.

Definition 8. *The completeness of the implementation of a probe for monitoring a metric measurement m for a specific cloud provider p, indicated as $a^{(I)}_{mp}$, is defined as proportional to the ratio between the sampling rate required in the deployment request and the sampling rate the probe implemented by the cloud provider for measuring that metric measurement. The gain in completeness of implementing a probe for a cloud provider is defined as:*

$$\Delta a^{(I)}_{mp} = a^{(I)}_{mp} - a^{(A)}_{mp}$$

that is positive only if a physical sensor is not yet implemented or if its completeness is lower than the completeness required in the deployment request.

As before, $a^{(I)}_{mp}$ can be expressed as described in Eq. 2.

If a probe is not available for a metric measurement, the proposed solution can instead estimate the trend for that metric measurement from the information coming from other probes by exploiting the knowledge contained in the BN metrics described in Sect. 4.1. This is also an alternative if the application owner has indicated the value *trends* as the level of precision for that metric or dimension in the deployment request. In this case, even if a probe is available, the cost of using estimated metric measurements may be lower than the direct monitoring, if it is based on metric measurements already collected by the monitoring system.

Definition 9. *The completeness of estimating the trend of a metric measurement m for a specific cloud provider p, indicated as $a^{(E)}_{mp}$, is defined as proportional to the ratio between the sampling rate required in the deployment request and the minimum sampling rate of the probes involved in the estimation (all the parents of the node to be estimated in the BN). The gain in completeness of estimating a probe for a cloud provider is defined as:*

$$\Delta a^{(E)}_{mp} = a^{(E)}_{mp} - a^{(A)}_{mp}$$

that is positive only if a physical sensor is not available or if its completeness is lower than or equal to the completeness required in the deployment request.

Detailed algorithms for the completeness computation with a simpler formulation are described in [4].

5.1.2 Deployment cost

The other perspective to consider is the total cost of a deployment solution that is obtained by summing the cost of deployment of all the microservices and the cost of monitoring, estimating and implementing probes. The information about costs is contained in the cloud provider registration and is taken into consideration by the optimiser.

5.2 Formulation of the optimisation problem

In this section, we provide a formulation of the optimisation problem, using the following notation:

- $\mathcal{S}$ is the set of all microservices with cardinality S.
- $\mathcal{P}$ is the set of all cloud providers with cardinality P.
- $\mathcal{D}$ is the set of all dimensions with cardinality D.
- $\mathcal{M}$ is the set of all metrics with cardinality M.
- $\mathcal{M}_d,\ d = 1 : D$ is the set of all metrics that contributes to dimension d, the cardinality is $M_d\ \forall d = 1 : D$. These sets are not a partition since a metric can contribute to the completeness of several dimensions.
- $\mathcal{MM}$ is the set of all metric measurements. The cardinality of this set is $\check{M}$.
- $\mathcal{MM}_m,\ m = 1 : M$ is the set of metric measurements that contributes to metric M. The cardinality of this set is $\check{M}_m$. These sets are not a partition since a metric measurement can contribute to the completeness of several metrics.
- $\mathcal{SP}$ is the set of couples of microservices (s_0, s_1) such that microservice s_0 and s_1 must be deployed by using the same cloud provider. The cardinality is SP[9].
- $\mathcal{DP}$ is the set of sets of microservices $dp = \{s_0, s_1, \dots\}$ such that all of them must be deployed by using different cloud providers. The cardinality is DP.
- $\mathcal{SR} \subseteq \mathcal{S} \times 2^{\mathcal{P}}$ is the set of couples $(s, \{p_i\})$ such that microservice s must be deployed by using one cloud provider in the set $\{p_i\}$.
- $\mathcal{MR} \subseteq \mathcal{V} \times \mathcal{M}$ is the set of couples (s, m) such that we want to measure the metric M for microservice s.
- $\mathcal{DV}_0$ is the set of all triples $(s \in \mathcal{S}, d \in \mathcal{D}, n_d \in \mathbb{N})$ such that for microservice s the application owner wants to estimate at least n_d metrics for dimension d. Its cardinality is DV_0.
- $\mathcal{DV}_1$ is the set of couples $(s \in \mathcal{S}, d \in \mathcal{D})$ such that for microservice s the application owner wants information about no more than 1 metric. Its cardinality is DV_1.

For the parameters, we use the following notation:

- $\lambda_i\ \forall\ i = 1 : D$ is the relative importance of a dimension with respect to the others. We assume that $\sum_{i=1}^{D} \lambda_i = 1$ and that the application owner specifies these values. This last assumption is not strictly necessary as application owners implicitly describe these values from the minimum completeness requested for each metric (the parameters α_m in the following).
- $F_p,\ \forall\ p = 1 : P$ is the cost of deployment of a microservice by using cloud provider p.
- $n_s(\omega),\ \forall\ s = 1 : S$ is the number of instances of microservice s. It is a random variable.
- $CI_{mp}\ \forall\ m = 1 : \check{M},\ p = 1 : P$ is the cost that cloud provider p charges for implementing a probe for metric measurement m.
- $CE_{mp}\ \forall\ m = 1 : \check{M},\ p = 1 : P$ is the cost that cloud provider p charges for estimating a probe for metric measurement m.
- $a^{(A)}_{smp} \in [0, 1]\ \forall\ m = 1 : \check{M},\ p = 1 : P$ is the completeness of metric measurement m supplied by cloud provider p for microservice s.
- $\Delta a^{(E)}_{smp} \in [0, 1]\ \forall\ m = 1 : \check{M},\ p = 1 : P$ is the improvement in completeness obtained if cloud provider p evaluates metric measurement m for microservice s.
- $\Delta a^{(I)}_{smp} \in [0, 1]\ \forall\ m = 1 : \check{M},\ p = 1 : P$ is the improvement in completeness obtained if cloud provider p implements a probe for metric measurement m for microservice s.

[9] Although the constraint is based on couples, also cases in which more than two microservices must be deployed on the same cloud can be expressed by using several couples (e.g., including (s_0, s_1) and (s_1, s_2) in SP requires that all the three microservices must be deployed on the same cloud).

- β is the budget that the application owners estimate, which we assume to be different from the maximum amount that they wish to pay.
- $\alpha_M \; \forall \; m = 1 : M$ is the minimum degree of completeness that we require for metric M.
- ρ is the percentage of the budget such that the application owners are not willing to pay more than $\beta + \rho\beta$.

We use the following variables:

- x_{sp}, binary variable, true if microservice s is deployed by using cloud provider p.
- y^s_{mp}, binary variable, true if microservice s is deployed by using cloud provider p and metric measurement m is made because not available (option M).
- z^s_{mp}, binary variable, true if microservice s is deployed by using cloud provider p and metric measurement m is estimated because not available (option E).
- w_{Ms}, binary variable, true if the completeness of metric M is greater than zero for microservice s.
- l, which measures by how much the solution implemented exceed the estimated budget.

Our problem [10] is then:

$$\text{max. } \lambda_1 \sum_{s=1}^{S} \min_{M \in \mathcal{M}_1} \min_{m \in \mathcal{MM}_M} \sum_{p=1}^{P} (a^{(A)}_{smp} x_{sp} + \Delta a^{(E)}_{smp} z^s_{mp} + \Delta a^{(I)}_{smp} y^s_{mp}) +$$

$$\ldots$$

$$+ \lambda_D \sum_{s=1}^{S} \min_{M \in \mathcal{M}_D} \min_{m \in \mathcal{MM}_M} \sum_{p=1}^{P} (a^{(A)}_{smp} x_{sp} + \Delta a^{(E)}_{smp} z^s_{mp} + \Delta a^{(I)}_{smp} y^s_{mp})$$

$$\text{minimize } l \tag{3}$$

subject to:

$$\sum_{p=1}^{P} x_{sp} = 1 \quad \forall \; s \; \in \mathcal{S} \tag{4}$$

$$z^s_{mp} + y^s_{mp} \leq x_{sp} \quad \forall \; p \in \mathcal{P}, \; s \in \mathcal{S} \tag{5}$$

$$a_{smp} \leq 1 - z^s_{mp} \quad \forall \, p \in \mathcal{P}, \; m \in \mathcal{MM}, \; s \in \mathcal{S} \tag{6}$$

$$x_{sp} = x_{s'p} \quad \forall \; p \in \mathcal{P}, (s, s') \in \mathcal{SP} \tag{7}$$

$$\sum_{s \in dp} x_{sp} \leq 1 \, \forall \, p \in \mathcal{P}, dp \in \mathcal{DP} \tag{8}$$

$$\sum_{p \in (s, \{p\}) \in \mathcal{SR}} x_{sp} = 1 \, \forall \, (s, \cdot) \in \mathcal{SR} \tag{9}$$

$$w_{Ms} = 1 \; \forall \, (M, s) \; \in \mathcal{MR} \tag{10}$$

$$\sum_{M \in M_d} w_{Ms} \geq n_d \; \forall \, (s, d, n) \; \in \mathcal{DV}_0 \tag{11}$$

$$\sum_{M \in M_d} w_{Ms} \geq 1 \; \forall \, (s, d) \; \in \mathcal{DV}_1 \tag{12}$$

$$\sum_{s=1}^{S} \sum_{p=1}^{P} \Big[F_p n_s(\omega) x_{sp} + \sum_{m=1}^{\check{M}} CE_{mp} z^s_{mp} + \sum_{m=1}^{\check{M}} CI_{mp} y^s_{mp} \Big] \leq \beta + l \tag{13}$$

$$\alpha_M w_{Ms} \leq \min_{m \in \mathcal{MM}_M} \sum_{p=1}^{P} a^{(A)}_{smp} x_{sp} + \Delta a^{(E)}_{smp} z^s_{mp} + + \Delta a^{(I)}_{smp} y^s_{mp} \;\; \forall M \in \mathcal{M}, s \in \mathcal{S}. \tag{14}$$

$$x_{sp} \in \{0, 1\} \quad \forall \; s, p; \;\; w_{Ms} \in \{0, 1\} \quad \forall \; M, s; \;\; l \in \Big[0, \rho\beta\Big];$$

$$z^s_{mp} \in \{0, 1\} \quad \forall \; m, p, s; \;\; y^s_{mp} \in \{0, 1\} \quad \forall \; m, p, s$$

The objective function (3) maximises the completeness of the dimensions and minimises the difference between the actual cost and the budget β estimated by the application owner. Please, notice that since different users may have different sensibilities to budget and completeness, we consider a multi objective optimisation problem. Constraint (4) ensures that every microservice must be deployed. Constraint (5) entails that, only in the site where the microservice is deployed, a metric measurement might be implemented or estimated, and that it cannot be implemented and estimated at the same time. Furthermore, constraint (6) entails that a metric measurement can be estimated if and only if the cloud provider does not have any measure related to the metric measurement in the site. Constraints (7) and (8) mean that the application owner can require some microservices to be deployed by using the same cloud provider or by using different cloud providers. Constraint (9), in contrast, allows the application owner to specify that a microservice can be deployed only in a particular subset of cloud providers. For example, constraint (9) can be used to prevent a microservice from being deployed using cloud providers with a low QoS. Constraints (10), (11), and (12) allow the application owner specify which metrics must be measured for a given microservice, if for a certain microservice the application owner wants to measure at least n metrics for a given dimension or if he wants to measure not more than one metric for a given dimension. Finally, constraint (13) models the cost of the services and constraint (14) imposes a minimum degree of completeness for a metric if the application owner wants to measure it. Note that the variable l is constrained to lie within the interval $[0, \rho\beta]$, thus a solution with a cost of more than $\beta(1 + \rho)$ is not acceptable for the application owner. We have chosen to not set the budget constraint as a "hard" constraint in order to allow for flexibility in terms of cost.

It should be noted that, although the focus of this paper is QoM, additional constraints on QoS (e.g., requirements on response time or availability) can be easily added to the formulation of the optimisation problem by: (i) considering $\mathcal{P}$ to be the set of all cloud providers that fulfill the constraints of QoS; (ii) enriching the model with constraints related to QoS.

Problem (3)–(14) is non-linear but can be linearized by adding two sets of variables v_{Ms} and u_{ds}. The former represents the minimum completeness of metric M for microservice s and the latter represents the minimum completeness for dimension d for microservice s. The objective function expressed by Eq. 3 thus becomes:

$$\text{max. } \textstyle\sum_{d=1}^{D} \lambda_d \sum_{s=1}^{S} u_{ds} \qquad \text{min. } l$$

The constraints of the optimisation problem (Eq. 4 to Eq. 13) remain the same, with the exception of Eq. 14:

$$\alpha_M w_{Ms} \leq v_{Ms} \;\; \forall M \in \mathcal{M}, s \in \mathcal{S}. \tag{15}$$

$$v_{Ms} \leq \sum_{p=1}^{P} a^{(A)}_{smp} x_{sp} + \Delta a^{(E)}_{smp} z^s_{mp} + + \Delta a^{(I)}_{smp} y^s_{mp} \;\; \forall M \in \mathcal{M}, s \in \mathcal{S}. \tag{16}$$

$$u_{ds} \leq v_{Ms} \;\; \forall d \in \mathcal{D}, M \in \mathcal{M}_d, s \in \mathcal{S}. \tag{17}$$

$$v_{Ms} \in [0, 1] \quad \forall \;\; M, s; u_{ds} \in [0, 1] \quad \forall \;\; d, s$$

This problem is a multi-objective stochastic mixed-integer linear program that can be optimally solved by means of stochastic programming techniques. However, we have chosen to consider the expected value problem (i.e.,

[10] The proposed formulation assumes that the utility of the optimiser can be well approximated by a linear function as it is reasonable to assume that the second order iterations between the completeness of different metric measurements is negligible.

the deterministic problem obtained by considering the mean value of each random variable in the problem). Hence, the only change is in constraint (13), which becomes:

$$\sum_{s=1}^{S}\sum_{p=1}^{P}\Big[F_p\bar{n_s}x_{sp}+\sum_{m=1}^{\check{M}}CE_{mp}z^{s}_{mp}+\sum_{m=1}^{\check{M}}CI_{mp}y^{s}_{mp}\Big]\le\beta+l \tag{18}$$

where the $\bar{n_s}$ are the expected values of the random variables n_s. This approximation was chosen as developing a stochastic analysis of the problem is beyond the scope of the present paper. By means of this approximation we obtain a multi-objective mixed-integer problem.

6 Validation

Since Problem (3)-(14) is multi-objective, we compute the Pareto front solutions (i.e., the set of solutions such that there are not better ones given the available set of resources) by using the ϵ constraint method. This is the standard approach to find all the Pareto front solutions by solving single objective problems obtained by considering one objective functions and by bounding the other one. For a more detailed explanation of this method and its properties, see [13]. Specifically, we choose to maximise the completeness using extra cost as a constraint, i.e. $l \le \epsilon$. In order to find all the points, we solve the problem with the constraint $l \le \rho\beta$ and we obtain the extra cost of the optimal solution l^*. We then solve the problem again by adding the constraint $l \le l^* - 0.01$. We choose 0.01 since any other smaller change in the value of l has no sense (we do not have one thousand of euros). We continue in this way until $l^* = 0$ or the problem is unfeasible (i.e., we do not have enough money to satisfy all of the constraints). As we add the constraint $l \le \epsilon$, we can delete the variable l by replacing constraint (18) and $l \le \epsilon$ with

$$\begin{aligned}\sum_{s=1}^{S}\sum_{p=1}^{P}F_p\bar{n_s}x_{sp}+\sum_{m=1}^{\check{M}}\sum_{p=1}^{P}CE_{mp}Z_{mp}+&\\+\sum_{m=1}^{\check{M}}\sum_{p=1}^{P}CI_{mp}Y_{mp}\le\beta+\epsilon.&\end{aligned} \tag{19}$$

In this way, at each step we solve an integer problem whose complexity is $\mathcal{O}(2^{\max[MS,SP,\check{M}SP]})$. To do so, we implement an exact algorithm using a C++ software integrated with the commercial software Gurobi[11] on an Intel R CoreTMi7-5500U CPU @2.40 Ghz with 8 GB RAM and Microsoft Windows10 Home installed. On top of this, we developed a web application prototype using Node.js[12].

To test our framework we executed several validations. First of all, we clarify the practical implications of the proposed approach applying it to the running example (Sect. 6.1). Then, we apply the framework to a more realistic scenario where we compare three executions: starting with the same problem size and changing the requirements we show how the constraints expressed by the application owner and the scalability of the microservices influence the solution (Sect. 6.2). Finally, in Sect. 6.3 we check the computational complexity and the scalability of the framework.

	Status	Performance	Sustainability
MS1	Free Storage (10m, D)	ALL (10m, D)	-
MS2	CPU Usage (1m D)	ALL (1m, D)	XOR (30m, T)
MS3	CPU Usage (1m D), Mem Usage (5m, D)	Avail. (5m, D)	-

TABLE 1
Summary of QoM requirements

Since, to the best of our knowledge, no other approach focusing on the systems monitorability matching exists in the literature, we cannot compare our solution directly with other approaches. For this reason, our experiments aim to demonstrate that the time required to solve the problem is reasonable for supporting the application developer in deciding where to deploy the microservices as well as the effects in changing the constraints. Finally, it should be noted that, since the exact solver Gurobi is able to solve even the most complex feasible instances in a reasonable amount of time, there is no point in applying heuristics to the problem. We are not looking for faster solution methods that provide an approximated solution. We have chosen Gurobi as a solver for its performance and flexibility.

6.1 Deployment optimisation for the running example

To show how the framework can be used in a real-life scenario, we refer to the microservice-based application represented in Fig. 1. For convenience, a summary of the requirements expressed in the deployment request (Listing 1) is shown in Tab. 1. As described in Sect. 4, the capabilities of the registered Cloud Providers are stored in the Registrations DB. During the registration phase the cloud provider matches the metric measurements that it offers with the metric measurements expressed in the HMD. By way of example, we matched the metrics provided by Google Stackdriver[13] with the HMD as shown in Tab. 2. A comparison between the capabilities of two cloud providers can be found in Tab. 3[14]. Given the request, the total estimated cost for deploying the application on Google is around $1.50 due to the need to collect a custom metric for the 5 to 15 instances of MS2. In contrast, on Amazon a premium plan is required, with an estimated cost of $35. However, with Google full completeness cannot be obtained since the power metric is not available and will be estimated exploiting the knowledge contained in the BN (see Fig.3(b)). A solution may be to implement a mixed deployment approach, deploying MS2 on Amazon, with a higher cost but greater completeness, and MS1 and MS3 on Google. This is feasible if the costs do not exceed the budget. If, instead, the budget is set to $25 (for example), deployment must be exclusively on Google, with limitations on QoM.

6.2 Examples of deployment optimisation

The problem discussed in Sect. 2 is too limited to be realistic for any validation process that wishes to analyse the scalability of the approach. We therefore considered a problem

[11] http://www.gurobi.com

[12] Node.js and C++ source code, is available at https://bitbucket.org/plebanip/optimon

[13] https://cloud.google.com/monitoring/api/metrics

[14] These costs were estimated using the tools provided by the two cloud providers: Amazon (https://calculator.s3.amazonaws.com/index.html) and Google (https://cloud.google.com/products/calculator/)

Metric Measurement	Google Stackdriver
Total Storage	`disk/bytes_total`
Used Storage	`disk/bytes_used`
CPU Usage	`CPU/utilization`
Used Mem	`memory/bytes_used`
I/O Throughput	`disk/read_ops_count`
Uptime	`uptime`
Power	NA

TABLE 2
Match between Google Stackdriver metrics and HMD metric measurements

	Google Cloud	Amazon
Coverage	all except power	all
Sampling Rate	60 sec free	5 min free, 60 sec 2.10$ x inst
Custom Metrics	0.010$ x metric (60s)	0.10 x metric (60s)

TABLE 3
Cloud provider capabilities comparison

with 100 microservices, 10 cloud providers, 10 dimensions, 30 metrics (about 3 metrics for each dimension), and 45 metric measurements (about 1.5 metric measurements for each metric), with an additional budget set to $160. It is pointed out that this additional budget is an extra amount of money that the application owner is willing to spend in order to obtain greater completeness.

As a first step, we compared the outcome of the framework for three different scenarios, differing in terms of the underling knowledge base and the parameters of the deployment request. In particular, in `case 1` the knowledge base has 10 metric measurements per metric and 10 metrics. In this situations, the productivity of the investment is higher: with less budget the user increases the quality of monitoring of more metric (see Eq. (14)), thus increasing the dimension of the solution space. In `case 2`, the knowledge base is characterized by 5 metric measurements per metric and 20 metrics and the application owner expresses stricter requirements for QoM than in `case 1`. Finally, in `case 3` the knowledge base consists of 2 metric measurements per metric and 50 metrics. Fig. 4 illustrates the three Pareto front computed from those instances. Each point in the Pareto front represents a trade-off between the QoM and the budget. The average time to compute a single point of the Pareto front is about 2.88 sec, while the average time to compute the whole Pareto is 172.53 sec (less than 3 minutes). The fronts are neither concave nor convex, as a result of the integrality requirements of some variables. It will be noted that the QoM is directly proportional to the budget (i.e., the more money available, the greater the achievable completeness). It is pointed out that all the curves can reach the 100% completeness with a sufficiently large additional budget only if a feasible solution exists. In `case 2`, the maximum completeness is reached with a budget lower than the maximum, while in `case 1`, the maximum completeness obtained with the given budget is around 0.9. This might suggest to the developer to try increasing the budget to obtain higher completeness. An interesting case can be observed in `case 3`: here the completeness

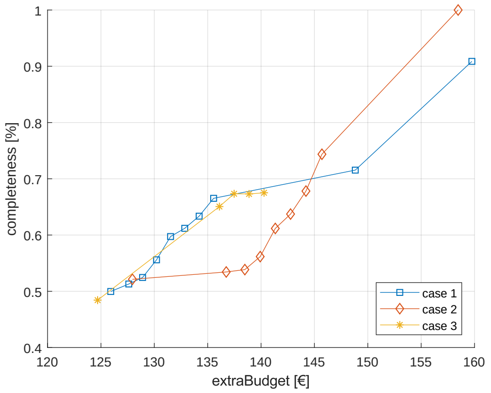


Fig. 4. Three Pareto fronts obtained by using the ϵ constraint method.

stops increasing with an additional budget of $140 due to the constraints expressed in the request (related both to QoM and to deployment), which make several solutions unfeasible. In this case, the developer can choose to relax some constraints in order to seek a better solution.

As already discussed in the paper, the problem that we are solving is subject to uncertainty due to the variation in the number of instances for each microservice according to the provided distribution. We studied how the cost of the solutions varies according to the variation in the number of instances. Starting with the problem size discussed in the previous experiment, we simulated $5,000$ scenarios extracting the number of microservices from their distribution and solving the expected value problem (taking the average instance value as a benchmark). If the application owners are more risk adverse, they can use some quantiles instead of the expected value. The two distributions obtained using the two approaches are shown in Fig. 5. The lines represent the density of the normal distributions that best fit the data. As can be observed, the mean value of the cost distribution obtained from the solution of the average value problem is 1% lower than the one from the solution obtained by using the 0.9 quantile. This difference reduces the length of the right tail of the distribution of the costs. Furthermore, in order to verify that this increment in the cost decreases the risk of paying more, we define a risk measurement close to the Expected Shortfall (interested readers are referred to [14]), consisting of the expected value of the right tail of the distribution above a certain quantile. If we call $\mathcal{X}_\alpha$ the set of all the observations greater than the quantile q_α, we define our risk indicator $\hat{ES}_\alpha$ as follows:

$$\hat{ES}_\alpha = \frac{\sum_{x\in\mathcal{X}_\alpha}}{|\mathcal{X}_\alpha|}. \tag{20}$$

By computing $\hat{ES}_{0.9}$ we see that it decreases in the distribution of the costs obtained by using the 0.9 quantile by the 6%. We shall discuss the stochastic nature of the problem in more details in a future paper.

6.3 Computational complexity and scalability

To validate also the scalability of the proposed solution, we start with a problem with the same dimension as the one described in Sect. 6.2, and we explore the effect on the execution time required to find the Pareto front while increasing the size of three variables: (i) number of metric

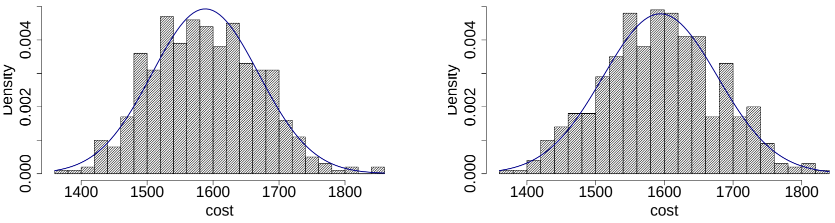


Fig. 5. On the left: cost distribution of the expected value problem, on the right: cost distribution of the 0.9 quantile value problem

measurements (from 45 to 800); (ii) number of microservices (from 100 to 3,500); (iii) number of cloud providers (from 10 to 200). When we increase the number of metric measurements, we necessarily increase the number of metrics required to maintain a reasonable knowledge base. In particular, we assume that multiplying by n the number of metric measurements, we also multiply by $n/2$ the number of metrics. In solving the problem, the computation of 10 points of the Pareto front are assumed to be sufficient to allow the application owners to identify the solution that fits their needs, as these points satisfy requirements in terms of QoM and cost. Fig. 6 shows the results of the experiments where the values are obtained as the mean of 20 observations. As can be observed, the time for the computation of the Pareto front is almost linear with the increase in microservices (maximum time 220 seconds) while it grows exponentially with the number of metric measurements (maximum time 55 seconds). As concerns the number of cloud providers, we observed a linear growth and solving the problem with 200 cloud providers required 300 seconds. However, with a more realistic number of 50 cloud providers, the problem is solved in 50 seconds. In order to discover the scalability limits for our approach, we also considered an extreme setting, with 800 metric measurements, 100 microservices, and 10 cloud providers, taking 3,400 seconds (around 58 min). If we take 3 min as a maximum time limit, a feasible problem size might include 100 metric measurements, 1,500 microservices and 50 cloud providers, which is a more than reasonable configuration.

It should be pointed out that this solution method is used at design time, hence a response time measured in minutes and not seconds is acceptable. The resolution time allows also the application owner to iterate the process several times, changing requirements by either adding or relaxing constraints, in order to find the most suitable solution.

7 Related work

The work presented in this paper touches upon several aspects: monitoring with heterogeneous cloud providers, customisability of monitoring metrics, and monitoring-aware optimisation of application deployment.

The challenges and patterns for monitoring microservices are described in [15]. In an application split into several microservices, the authors pointed out that it is difficult to detect the cause of a failure at run time. In [16], the authors provide a model for guiding decisions with a view to selecting the best monitoring system for a specific microservice architecture. Dealing with the development of applications requiring the interaction with different providers increases the

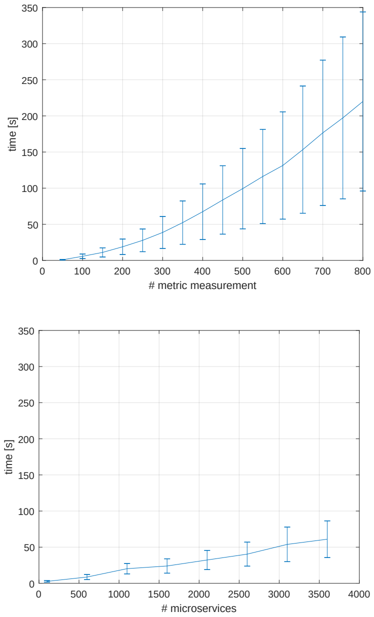


Fig. 6. Time required to find a single Pareto Point in relation to variations in number of metric measurements and microservices.

complexity of managing the application itself. As discussed in [17] there is a rich variability in the configuration options offered by cloud providers, and managing this variability is not an easy task. As in our work, the authors support the assignment of microservices to cloud providers with a matchmaking mechanism between the cloud provider capabilities and the developer requirements but, unlike our approach, they do not focus on the monitoring aspects. Although [18] points out that this is one of the most important issues in microservices literature, most existing studies analyse how to measure a single or several microservices to assess to what extent the SLAs are satisfied [19] or how to manage the resulting application [20]. The problem of dealing with heterogeneous cloud providers is not considered at all. When such heterogeneity is taken into account, the approach is usually provider-centric and focuses on solutions to mask the heterogeneity of the adopted monitoring platforms [21], [22] by means of a common interface. Our aim is to consider multi-cloud solutions with a client-centric perspective: the end-user is the application that coordinates access to and utilization of different cloud providers to meet the application requirements, as also discussed in [23]. Here, a cloud broker, as seen by the NIST [24], facilitates the relationships between providers and the application owners (consumers) by providing an intermediation service. In this paper, the matchmaker (a sort of cloud broker) supports the cloud consumer in the deployment of the application by providing a deployment strategy that optimises the monitoring capabilities required by the application. Dealing with different providers entails mediating between different models used to describe dimensions and monitoring metrics. Here, semantic web techniques can enable the knowledge base to be developed and managed. Indeed, semantic technologies are gaining increasing attention also in cloud computing sphere [25] and, specifically with regard to monitoring issues. In [26], linked data are used to handle the heterogeneity of the collected data, whereas [27] provides a semantic meta-model for classifying dimensions and metrics.

Metric customisability is a relevant issue as witnessed by the existence of several cloud platforms, differing in terms of: (i) the set of metrics provided by the monitoring system; (ii) sampling times for each metric; (iii) the costs of hosting the application (or microservice); (iv) the flexibility in terms of the option of adding new metrics or reducing sampling rate at an additional cost. On-demand, customised metrics can be requested on several platforms and monitoring solutions like Nagios, PCMONS, and Sensus[15] [28]. Nonetheless, application owners may be unable to find the provided metrics. Some work has modelled and discovered the relations between different metrics that can contribute to a knowledge base for predicting the value of a missing metric. In this way, the application owner may be able to collect information about the desired metric even if the actual value is not directly provided by the monitoring system. In [29], a framework for building a dependency tree of mutual influences between metrics is proposed. The tree is created using machine learning, while the influential factors among indicators are statically and manually defined. Google [30] employs neural networks for modeling and predicting the outcome of some modifications over the monitored variables. In [9] the authors automatically extract relations between metrics provided by the monitoring system of a data centre and represent their relations through a BN automatically created from historical values. The BN adapts to the updates that can occur in the data centre and is continuously refined. The approach can be used to express trends about the behaviours of the metrics and predict their future values from other metrics that have a causal relation with it. In this work, we have modified this approach to enable users to predict missing metrics.

With regard to optimising application deployment, our work takes its cue from the vast literature about the deployment of VMs in cloud environments. In [31] a multi-objective algorithm is employed for VM placement in a cloud system. The algorithm minimises overall resource wastage and power consumption by providing a Pareto set of solutions. In [32], a greedy allocation algorithm is used to optimise the cloud provider's profit, considering energy efficiency, virtualisation overheads, and SLA violation penalties as decision variables. In these approaches, a single cloud provider is considered, thus monitoring services are not compared. The issues of deployment in multi-cloud environments is discussed in [33], using a multi-cloud deployment in a federated cloud with a view to maximising profit for cloud providers. To achieve the automatic deployment in this context, an approach to describing customers' desired application deployments is provided as a topology model. Similarly, the providers can adapt this description to their capabilities. With regard to optimised deployment driven by the monitoring capability, to the best of our knowledge, our previous paper on this topic [4] is the first one to address this issue. Nevertheless, a strand of optimisation literature deals with the optimal deployment of cloud-based applications in servers, whose main goal is to maximise the reliability of the application. Interested reader, are referred to [34] and the references therein.

8 Final remarks and future work

In this paper we have proposed an architecture for managing the deployment of microservice-based applications involving several cloud providers. The approach provides a set of modules to manage both application owner requirements and cloud provider monitoring services semi-automatically, adopting a MOMILP to identify the best solutions for matching those requirements and monitoring services. The architecture stresses the importance of the QoM offered by the different cloud providers. This aspect is relevant since monitoring data enable the application owners to analyse the efficiency and effectiveness of the application and to make informed decisions regarding improvements and modifications. The optimiser also considers the variability in terms of number of instances to be deployed for each microservice comprising the application and makes it possible to specify requirements for microservices at different levels of detail. Our results shown that the approach proposed here is capable of providing a solution within a reasonable time interval for realistic application sizes and is also scalable to larger numbers of microservices, of cloud providers, and of metrics provided by their monitoring system. A prototype of the framework, with the capability to translate monitoring services supplied and demanded offers in an optimisation problem and find the Pareto front according to the constraints, has also been implemented.

In future work, we plan to develop an improved cost evaluation model for the implementation of metrics when they are neither available nor estimated. Such a model may involve third-party services, such as New Relic, which provides external support in measuring those metrics that impact costs and completeness. From an implementation standpoint, the possibility of directly connecting our prototype with cloud providers in order to be informed about their services and potentially to integrate the approach with TOSCA [35] will be investigated. Finally, the prototype will be extended to consider monitoring services offered along with Quality of Infrastructure and Quality of Service requirements as additional constraints.

Acknowledgments

DITAS project is funded by the European Union's Horizon 2020 research and innovation programme under grant agreement RIA 731945.

[15] https://www.nagios.org; https://code.google.com/p/pcmons; https://sensuapp.org

References

[1] J. Lewis and Fowler, “Microservices: a definition of this new architectural term,” 2014. [Online]. Available: http://martinfowler.com/articles/microservices.html

[2] M. McLarty, “Microservice architecture is agile software architecture.” [Online]. Available: http://www.infoworld.com/article/3075880/application-development/microservice-architecture-is-agile-software-architecture.html

[3] M. Fazio, A. Celesti, R. Ranjan, C. Liu, L. Chen, and M. Villari, “Open issues in scheduling microservices in the cloud,” *IEEE Cloud Computing*, vol. 3, no. 5, pp. 81–88, Sep. 2016.

[4] E. Fadda, P. Plebani, and M. Vitali, “Optimizing monitorability of multi-cloud applications,” in *International Conference on Advanced Information Systems Engineering*. Springer, 2016, pp. 411–426.

[5] K. Kritikos, K. Magoutis, and D. Plexousakis, “Towards knowledge-based assisted iaas selection,” in *Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on*. IEEE, 2016, pp. 431–439.

[6] ISO, “ISO/IEC 25010:2011: Systems And Software Engineering Systems And Soft-Ware Quality Requirements And Evaluation (Square) System And Software Quality Models,” ISO/IEC, Tech. Rep., 2010.

[7] J. van der Ham, J. Stger, S. Laki, Y. Kryftis, V. Maglaris, and C. de Laat, “The novi information models,” *Future Generation Computer Systems*, vol. 42, pp. 64 – 73, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13002811

[8] M. Natu, R. K. Ghosh, R. K. Shyamsundar, and R. Ranjan, “Holistic performance monitoring of hybrid clouds: Complexities and future directions,” *IEEE Cloud Computing*, vol. 3, no. 1, pp. 72–81, Jan 2016.

[9] M. Vitali, B. Pernici, and U.-M. O’Reilly, “Learning a goal-oriented model for energy efficient adaptive applications in data centers,” *Inf. Sciences*, vol. 319, pp. 152–170, 2015.

[10] J. M. Joyce, “Kullback-leibler divergence,” in *International Encyclopedia of Statistical Science*. Springer, 2011, pp. 720–722.

[11] A. Pescape, “Entropy-based reduction of traffic data,” *IEEE Communications Letters*, vol. 11, no. 2, 2007.

[12] T. Van Erven and P. Harremos, “Rényi divergence and kullback-leibler divergence,” *IEEE Transactions on Information Theory*, vol. 60, no. 7, pp. 3797–3820, 2014.

[13] K. Miettinen, *Nonlinear multiobjective optimization*. Springer Science & Business Media, 2012, vol. 12.

[14] P. Embrechts, T. Mikosch, and C. Klüppelberg, *Modelling Extremal Events: For Insurance and Finance*. London, UK: Springer-Verlag, 1997.

[15] S. Newman, *Building microservices: designing fine-grained systems*. ” O’Reilly Media, Inc.”, 2015.

[16] S. Haselböck and R. Weinreich, “Decision guidance models for microservice monitoring,” in *Software Architecture Workshops (ICSAW), 2017 IEEE Int’l Conference on*. IEEE, 2017, pp. 54–61.

[17] G. Sousa, W. Rudametkin, and L. Duchien, “Automated setup of multi-cloud environments for microservices applications,” in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, June 2016, pp. 327–334.

[18] N. Alshuqayran, N. Ali, and R. Evans, “A systematic mapping study in microservice architecture,” in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, Nov 2016, pp. 44–51.

[19] D. Ardagna, G. Casale, M. Ciavotta, J. F. Pérez, and W. Wang, “Quality-of-service in cloud computing: modeling techniques and their applications,” *Journal of Internet Services and Applications*, vol. 5, no. 1, p. 11, 2014.

[20] G. Toffetti, S. Brunner, M. Blöchlinger, F. Dudouet, and A. Edmonds, “An architecture for self-managing microservices,” in *Proc. of the 1st Int’l Workshop on Automated Incident Management in Cloud*, ser. AIMC ’15. New York, NY, USA: ACM, 2015, pp. 19–24.

[21] J. M. Alcaraz Calero, B. Knig, and J. Kirschnick, *Using Cross-Layer Techniques for Communication Systems*. Hershey: IGI Global, 2012, ch. Cross-Layer Monitoring in Cloud Computing.

[22] C. Zeginis *et al.*, “Towards cross-layer monitoring of multi-cloud service-based applications,” in *ESOCC 2013, Malaga, Spain. Proceedings (LNCS 8135)*. Springer, 2013.

[23] A. N. Toosi, R. N. Calheiros, and R. Buyya, “Interconnected cloud computing environments: Challenges, taxonomy, and survey,” *ACM Comp. Surv.*, vol. 47, no. 1, pp. 1–47, 2014.

[24] F. Liu *et al.*, *NIST Cloud Computing Reference Architecture: Recommendations of the National Institute of Standards and Technology (Special Publication 500-292)*, USA, 2012.

[25] A. Sheth and A. Ranabahu, “Semantic modeling for cloud computing, part 1,” *Internet Comp., IEEE*, vol. 14, no. 3, pp. 81–83, 2010.

[26] A. Portosa, M. Rafique, S. Kotoulas, L. Foschini, and A. Corradi, “Heterogeneous cloud systems monitoring using semantic and linked data technologies,” in *IFIP/IEEE Int’l Symp. on Integrated Network Mgmt*, May 2015, pp. 497–503.

[27] W. Funika, P. Godowski, P. Pegiel, and D. Król, “Semantic-Oriented Performance Monitoring of Distributed Applications,” *Computing and Informatics*, vol. 31, no. 2, pp. 427–446, 2012.

[28] G. Aceto, A. Botta, W. de Donato, and A. Pescap, “Cloud monitoring: A survey,” *Computer Networks*, vol. 57, no. 9, pp. 2093 – 2115, 2013.

[29] R. Kazhamiakin *et al.*, “Adaptation of Service-Based Applications Based on Process Quality Factor Analysis,” in *ICSOC/ServiceWave 2009 Workshops*, 2010, pp. 395–404.

[30] J. Gao, “Machine Learning Applications for Data Center Optimization,” Google, Tech. Rep., 2014.

[31] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu, “A multi-objective ant colony system algorithm for virtual machine placement in cloud computing,” *Journal of Computer and System Sciences*, vol. 79, no. 8, pp. 1230–1242, 2013.

[32] Í. Goiri, J. L. Berral, J. O. Fitó, F. Julià, R. Nou, J. Guitart, R. Gavaldà, and J. Torres, “Energy-efficient and multifaceted resource management for profit-driven virtualized data centers,” *Future Generation Comp. Syst.*, vol. 28, no. 5, pp. 718–731, 2012.

[33] A. Panarello, U. Breitenbücher, F. Leymann, A. Puliafito, and M. Zimmermann, “Automating the deployment of multi-cloud applications in federated cloud environments,” in *Proceedings of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools*, 2017, pp. 194–201.

[34] F. L. Pires and B. Barán, “Virtual machine placement literature review,” *CoRR*, vol. abs/1506.01509, 2015. [Online]. Available: http://arxiv.org/abs/1506.01509

[35] D. Palma and S. Moser (eds.), “Topology and orchestration specification for cloud applications v. 1.0,” 2013. [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.pdf

Edoardo Fadda received received his degree in Mathematical Engineering and Ph.D. in Information Technology and System Engineering from the Politecnico di Torino, Torino, Italy, in 2014 and 2018, respectively. He is a post-doc with the Dipartimento di Automatica e Informatica, Politecnico di Torino. His main interests are optimisation under uncertainty, statistics and their applications.

Pierluigi Plebani is Assistant Professor at Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, where he also received the Ph.D. in Information Engineering. His research interests concern Service Oriented Architectures based on Fog and Cloud environments, Business Process Management with IoT, and Green IT.

Monica Vitali received her Ph.D. in Information Technology from the Politecnico di Milano, Milan, Italy, in 2014. She is currently post-doc at the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano. She has been visiting Ph.D. candidate at MIT, Cambridge (Massachusetts - US) and visiting researcher at UMU (Sweden). She is interested in adaptation and monitorability in cloud computing, in adaptive and self-adaptive systems and services, and in machine learning techniques for adaptation.