QoS-Aware Orchestration of Network Intensive Software Utilities within Software Defined Data Centres

An Architecture and Implementation of a Global Cluster Manager

Uroš Paščinski[1] Jernej Trnkoczy[2] Vlado Stankovski[2]
Matej Cigale[3] Sandi Gec[1]

Abstract

Although the cloud computing domain is progressing rapidly, the deployment of various network intensive software utilities in the cloud is still a challenging task. The Quality of Service (QoS) for various gaming, simulations, video-conferencing, video-streaming or even file uploading tasks may be significantly affected by the quality and geolocation of the selected underlying computing resources, which are available only when the specific functionality is required. This study presents a new architecture for geographic orchestration of network intensive software components which is designed for high QoS. Key elements of this architecture are a Global Cluster Manager (GCM) operating within Software Defined Data Centres (SDDCs), a runtime QoS Monitoring System, and a QoS Modeller and Decision Maker for automated orchestration of software utilities. The implemented system automatically selects the best geographically available computing resource within the SDDC according to the developed QoS model of the software component. This architecture is event-driven as the services are deployed and destroyed in real-time for every usage event. The utility of the implemented orchestration technology is verified qualitatively, and in relation to the potential gains of selected QoS metrics by using two network intensive software utilities implemented as containers: an HTTP(S) file upload service and a Jitsi Meet videoconferencing service. The study shows potential for QoS improvements in comparison to existing orchestration systems.

Keywords autonomic orchestration, Quality of Service, Software Defined Data Centre, event-driven cloud applications

[1]University of Ljubljana, Faculty of Computer and Information Science
[2]University of Ljubljana, Faculty of Civil and Geodetic Engineering
[3]Cardiff University, School of Computer Science

1 Introduction

There is a variety of popular Web-based and Internet of Things applications [6] including online gaming, videoconferencing, robot vision, remote navigation and similar, which make use of software utilities that are compute-, memory- and network-intensive.

The development and management process for this kind of applications has considerably improved in the past few years with the emergence of various new editors and tools (such as Juju [4] or Fabric8 [14]) and methodologies for component based software engineering [23]. According to these new practices, commonly known as DevOps [13], applications are developed as collections of loosely coupled software components, e.g. databases, simulation algorithms, speech-to-text translation services, computer-vision services and alike. Software developers can then search for them in public Open Source software repositories, connect them together by using specialized editors (such as Juju), and deploy them in cloud infrastructures. Such applications can be fairly complex. Their components during the runtime would normally communicate via standardised interfaces, while exposing their services to end users via the Web.

Another important recent development is the emergence of Software Defined Data Centres (SDDC) [2, 8], a term nowadays interpreted in many different ways. In the present study, a SDDC is defined as set of Virtual Machines and physical computing resources on which common middleware services are deployed, thus forming a virtual cluster. Such cloud infrastructure can be composed by computing elements of greatly varying physical characteristics and can span geographically in order to serve customers World-wide. A cloud application can be deployed and orchestrated on top of such SDDCs.

The above mentioned advanced software engineering and cloud computing trends promise to improve the make-span of the software life-cycle, reduce operational costs and improve Quality of Service (QoS) aspects to achieve greater applications' acceptance and improved services for the end users.

However, these new and promising trends have been followed by new problems, one of which is the difficult management of the QoS within the SDDCs. Acquired computing resources from various cloud providers usually differ significantly regarding their geolocation, the connectivity between the users (clients) and the running software services, the quality of the allocated computing resources such as the processor frequency, the amount of allocated memory to applications, the operational cost including storage cost [17], and many other aspects, which add to the complexity. It is therefore paramount to design and implement specific autonomic orchestration technology that can be used to address the QoS concerns of the end users.

Existing orchestration systems [27, 19, 48] have been designed mainly for single data centres to help improve the make-span of workflow-based applications. QoS aspects related to networking have been traditionally addressed with Software Defined Networking (SDN) approaches [12], such as the one of Wang et al. [47]. However, as previously elaborated, there exists a range of network-intensive software utilities for which these existing approaches cannot be applied. Examples include interactive Web applications, exposing data from cameras, microphones or other sensors, which require that the clients use the services via the open Internet. In these cases, it is reasonable to assume that high QoS can be achieved by careful selection of the computing infrastructure (including its geolocation) each time the particular software utility has to be used by the end users.

This work focuses on the design of a new architecture for the autonomic orchestration of network-intensive software utilities. It is designed to achieve high QoS of applications operating on top of a globally distributed SDDC. One of the major requirements for this architecture is to support diverse network-intensive software components, which are required by end users accessing the services from various geographic regions.

Quality itself is multifaceted [15] and it is a difficult problem to assess it for each and every network-intensive software component, it is difficult to know in what way the underlying computing infrastructure and the environmental conditions affect the end users' experience. Recently, various new cloud monitoring systems [41, 43] have been designed that measure QoS metrics at various levels, that is, infrastructure-level, network-level, Virtual Machine (VM)-level, container-level, application-level and even user-level. This plethora of QoS metrics could be effectively used to assess what affects the quality for every single software utility, and then use the collected information in order to implement an intelligent orchestration capability contributing to high QoS. An important undertaking in this work is therefore to make advantage of runtime QoS measurements to be able to generate and make use of QoS models in the designed autonomic orchestration capability for SDDCs.

1.1 Goals of the study

Based on the above rationale the goals of this study are therefore to:

- develop an architecture for global orchestration of network-intensive software utilities with high QoS;
- design and implement an orchestration technology and associated services, implementing this architecture;

- design and implement a consistent QoS modelling approach by taking into an account end users' locations, current clusters' load, network conditions and pre-runtime QoS measurements of the software utilities,
- perform trace-based evaluation of the novel orchestration technology by using two exemplary software utilities, a File Upload HTTPS service, and a videoconferencing service based on the Jitsi Meet Open Source software,
- present and evaluate an exemplary approach to QoS modelling and decision making aiming at high QoS, which may be achieved within a globally distributed SDDC.

1.2 Organisation

This work falls in the area of autonomic orchestration. It leads to a new architecture and implementation of a Global Cluster Manager. Section 2 discusses related works. Section 3 presents an novel architecture and implementation of a Global Cluster Manager (GCM) – technology for geographic orchestration of network-intensive software utilities, with all its components. Description of the workflow between these components is also presented. The following Section 4 elaborates the implemented QoS modelling and decision making process, which is used by the GCM. QoS measurements for two cloud applications (File Upload and Jitsi Meet Videoconferencing) that benefit from the GCM, are presented in Section 5. Section 6 analyses the utility of the developed platform in the context of the Software-as-a-Service (SaaS) business model. Section 7 summarizes the work with focus on the significance and relevance of the obtained results.

2 Related work

The area of autonomic computing [20, 28], aims to address the increasing heterogeneity and dynamics of networks, systems and applications. Its central goal is to contribute to the design of computer and software systems and applications that can manage themselves in accordance with high-level guidance from humans. For example, the study of Liu et al. [28] presents a component-based programming model for autonomic applications.

From the viewpoint of orchestration, autonomic orchestration strategies are considered as the highest level of sophistication in orchestration. The work of Weerasiri et al. [48] presents a taxonomy and survey of cloud resource orchestration techniques, which includes autonomic orchestration strategies.

They have found that mainly human-assisted techniques (including user-defined and rule-based strategies), are currently being used as approaches to autonomic orchestration. Our present work relies on specific QoS models that are generated for the software utilities and then used in the orchestration process to more precisely address the end users' QoS requirements. Although the survey of Weerasiri et al. is very recent, it does not consider some important technologies that can be used for geographic orchestration of cloud services, such as Kubernetes [38], which is used in our present study.

The studies of Toosi et al. [44], Zhan et al. [52], Cheng et al. [11], Kacsuk et al. [24] have focused on self-management of cloud resources applied to maintain the expected QoS in the presence of various faults, variable environmental conditions, and changes in user requirements. Their works contribute to the area of autonomic orchestration. These authors list a number of benefits that could be obtained with self-management of cloud resources, such as the potential to enhance high availability of cloud resources through dynamic (re-)configuration, and scaling up or down running applications by analyzing the recent resource consumption statistics.

The survey of Singh [34] has analysed the possibilities for runtime management of QoS for applications. It presents an autonomic (self-* properties) resource management taxonomy that covers aspects related to runtime QoS management of cloud-hosted applications. It considers that several elements are necessary in order to achieve high QoS including QoS metrics monitoring, fault tolerance, workload consolidation, and scheduling objective function. All considered techniques relate to autonomic management of resources within a single data centre, while autonomic resource management within multi-cloud, federated-cloud and other inter-cloud scenarios, are not considered. The present work concentrates on SDDCs which emerge as a more flexible mechanism for global delivery of software services.

Some very recent works have focused on aspects of QoS assurance with light virtualization [18], container orchestration in fog computing infrastructures [32], many-objective Virtual Machine placement [29], and autonomic mobile cloud computing [33], however, none of these studies have analysed the potential of global orchestration of network-intensive software utilities. Hence, the present study contributes to the overall agenda of achieving event-driven, highly flexible and extensible, fault-tolerant, fast and lightweight global autonomic orchestration capability for cloud services with QoS-awareness through the use of container technologies.

An important goal undertaken by the present work is to address the QoS-requirements of various network-intensive software utilities. This requires the development of specific QoS models for every software utility, which are then used in the operation of the autonomic orchestrator. Generally, the modelling of software utilities for high QoS is an NP hard (non-deterministic

polynomial-time hardness) problem. Detailed QoS modelling approaches have been developed for specific types of applications, such as workflow-based applications for which the makespan can be optimized [47], or QoS modelling for financial services [10].

Software Defined Networking (SDN) approaches and technologies have also been designed to address the QoS requirements of network intensive applications [3], specific technologies which are used include vSwitch and OpenFlow. However, such approaches cannot be applied to connectivity problems in the open Internet, for example, they cannot be applied to User Datagram Protocol (UDP) traffic for video-communications, gaming and similar applications, between the clients and the running software services. The open Internet is best efforts, and various circumstances such as the geolocation of the end users, the geographic placement of the software service, the current quality of the connections, bandwidth and similar can significantly affect the QoS. Existing approaches for QoS modelling are therefore not suitable for the orchestration of network-intensive software components, and this study makes a systematic approach to address this problem.

3 Architecture and Implementation

3.1 High-Level Architecture

Many software companies today focus on Web application development, which is generally represented by the Software-as-a-Service (SaaS) business model. Within such applications they need to implement various network intensive functionalities and to reach end users on the global market. While the cloud computing technology helps improve the software engineering life cycle, for the software companies it is equally important to deliver high QoS applications to the end users. Autonomic orchestration mechanisms could therefore help to further simplify the delivery process for cloud applications. Major expected benefits are the handling of larger number of non-functional requirements, including geolocation, improved resources utilization and reduction of operational costs, practically unlimited scalability, and most importantly, greater end users acceptance.

This work is motivated by a major trend in the cloud computing domain, that is, the increased use of containers (instead of VMs) as a method for packaging and delivery of software utilities. Major underlying technologies in this area are Docker [36] for the management of containers, and Kubernetes [38] as a general purpose orchestration technology. The File Upload and Videoconferencing applications are therefore packed into Docker containers for the purpose of this study. In contrast to Virtual Machines, con-

tainers can be spinned on and off much faster, practically within seconds, thus forming basis for fine grained services orchestration based on events. Whenever a specific service is required, it can be dynamically deployed and started for a specific end user (or indeed another service).

In order to motivate this development, several potential applications have been studied, including live-event broadcasting, videoconferencing, early-warning systems and similar. Two software utilities have been selected in order to illustrate the approach. For the purpose of the study they have been implemented in containers:

- File Upload is a Web application designed for a single user to be able to upload a single file to a cloud storage, and
- videoconferencing is much more complex Web application designed to serve unified communications for multiple users at the same time.

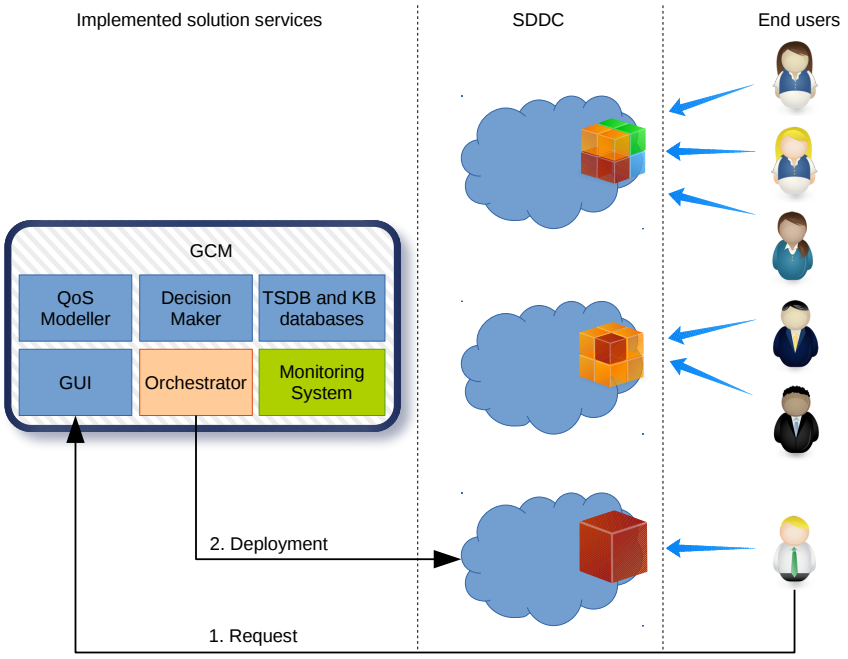


Figure 1: A high-level architecture of an autonomic orchestrator for network-intensive software utilities.

Based on the above rationale and the extensive literature review, a new, high-level architecture for an autonomic orchestrator was designed and is presented in Figure 1. The overall solution is called a Global Cluster Manager (GCM). It can be used to orchestrate services within a geographically distributed SDDC.

The architecture consists of a GUI that can be used by software engineers

and end users to steer the operation of the Orchestrator of software utilities, a Monitoring System for the runtime measurements of multi-level QoS metrics and two databases – a Time-Series Database (TSDB), which is used to store QoS metrics collected from the runtime environment and a Knowledge Base (KB), which is used to store already developed QoS models of software utilities, adaptation rules and other information. The database services can be accessed and used by the Orchestrator via convenient Application Programming Interfaces (APIs).

Two very important elements of the GCM are the QoS Modeller component and the Decision Maker, which are elaborated in detail in a separate section. In the following sub-sections, key components of this solution are discussed by following a logical order.

3.1.1 Monitoring System

Prerequisite for the functioning of the GCM as an autonomic orchestrator is the existence of a monitoring system. The implemented Monitoring System is available as Open Source, and can be used as a separate component of the developed system [41]. Baseline technologies which were used to implement the system are JCatascopia [45] and Netdata [39]. Its detailed architecture is presented in Figure 2. The Monitoring System consists of three main components: (i) Monitoring Server, (ii) Monitoring Agent(s) and (iii) Monitoring Probe(s). It uses the TSDB to store the measurements.

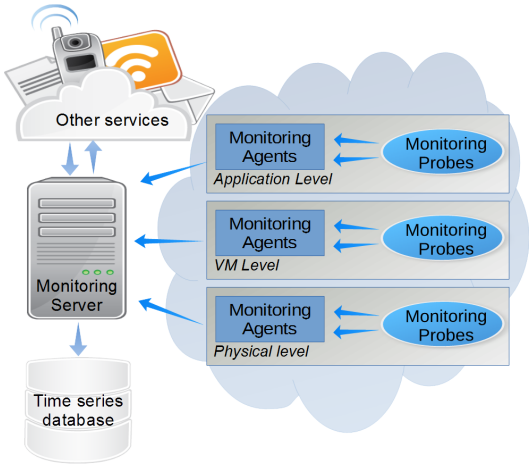


Figure 2: Monitoring System architecture.

The Monitoring System has an important role in the system since it provides multi-level QoS metrics. These include application, container and VM metrics, including network properties (between running services and clients)

and physical (infrastructure) metrics. Collecting monitoring metrics from the cloud infrastructure and the application is therefore done continuously in order to notify the system about the actual resources' availability and quality. The Monitoring Server has been designed for elasticity across the SDDC and it adapts seamlessly together with the scaling of the applications.

Usually the monitoring process starts when an application (software utility packed in a container) is deployed in a VM or on a physical resource, and when the user starts using the service via the Web. An Alarm Trigger is used to react in cases when monitoring metric data are exceeded and violated by predefined threshold-based rules. Those thresholds are initially defined in a monitoring environment, but can be also be extended to more complex structure by using an OWL2 compliant ontology [26]. Alarms due to threshold violations can be used to trigger specific autonomic adaptation services by using implemented rules in the KB.

While the KB is used to store higher-level complex structured data, such as the QoS model and adaptation rules for the application, the TSDB has a simpler design as it is used to store only the monitored metrics. Thus, the TSDB can be optimised for fast CRUD (create, read, update, and delete) operations and does not require complex data queries.

The TSDB has been implemented by using the Apache Cassandra technology, which is a distributed TSDB available as Open Source. The schema used to store QoS metrics encapsulates all the needed monitoring information (e.g. identity numbers of monitoring probes, agents, metrics, metric types, frequencies of measurements and similar).

3.2 Global Cluster Manager for autonomic orchestration

A limited number of technologies today allow the geographic scaling of applications across multiple host systems. This requires capability to abstract away the management of individual hosts and containers available across multiple clouds. For this purpose it is possible to use some container orchestration tools that combine a set of independent bare-metal or Virtual Machines into a cluster, and provide a unifying interface facilitating the deployment, maintenance, resource management and scaling of containerized applications across geographically distributed hosts. Existing orchestration tools, however, are usually used to orchestrate relatively self-contained cluster systems. Due to the design of scheduling and network routing mechanisms of the orchestrating tools each cluster is usually setup within a relatively performant, reliable and low cost network, which in practice bounds the cluster boundaries to a single data centre or single availability zone of a cloud provider. Global multi-instance applications on the other hand should

be running across different cloud providers and multiple data centres and availability zones. In order to allow cloud applications to run across multiple geographically distributed data centres, we developed a Global Cluster Manager (GCM). It can be used through well-defined APIs. A Docker example version of the developed GCM is made available via an Open Source Apache License V2 [35].

The conceptual design of a GCM is presented in Figure 3. As it can be seen, the GCM leverage the capabilities provided by the Kubernetes container orchestration tool. Kubernetes already provides management API server controlling a single self-contained cluster. However, our GCM is able to issue orchestrating commands globally, over a set of globally distributed clusters. In order to achieve this, the GCM component handles various authentication mechanisms (e.g. password-based, certificates-based) required by different management API servers belonging to individual Kubernetes clusters. In order to be able to achieve high QoS in its operation, it communicates with the KB, which stores QoS models for software utilities as well as other information needed for the orchestration of applications across clusters, for example, information about the cluster credentials, master nodes endpoints, status of the environment and so on.

The functionality provided by the GCM includes running/stopping application component instances in the selected geographic location (selection of cluster and host machine), management of the component resources (RAM and CPU), and monitoring of the status of application components. The GCM exposes an Orchestrator API through which the federated clusters can be manipulated. Thus, the developed API provides technical means to realize QoS-aware orchestration of network-intensive software utilities, which is based on QoS models.

3.3 Components workflow

Description of the data flow among the components of GCM is summarized by using a sequence diagram for the videoconferencing (Jitsi Meet) application as a particular network-intensive software utility. Figure 4 shows what happens in the underlying system when an user wishes to use the videoconferencing service via a Web browser (WebRTC protocol). The developed autonomic orchestration system and its functionalities are used to obtain best possible quality of experience for the users of the Jitsi Meet videoconferencing application.

When using the system functionalities the following individual steps take place:

1. An end user who wants to start a videoconference opens a GUI form,

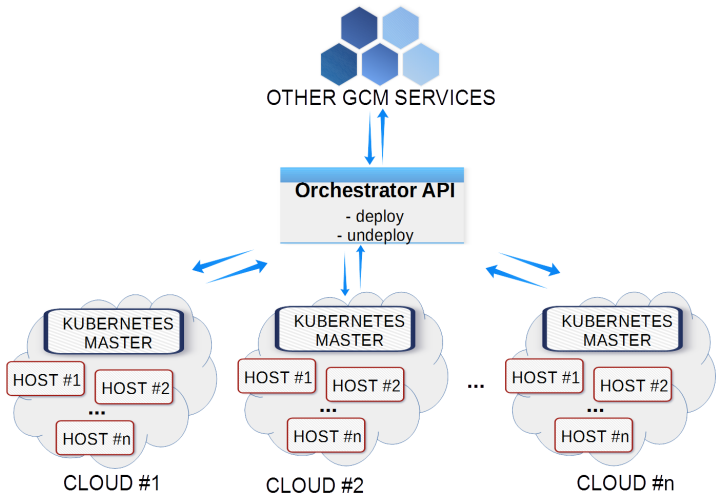


Figure 3: Conceptual design of GCM API.

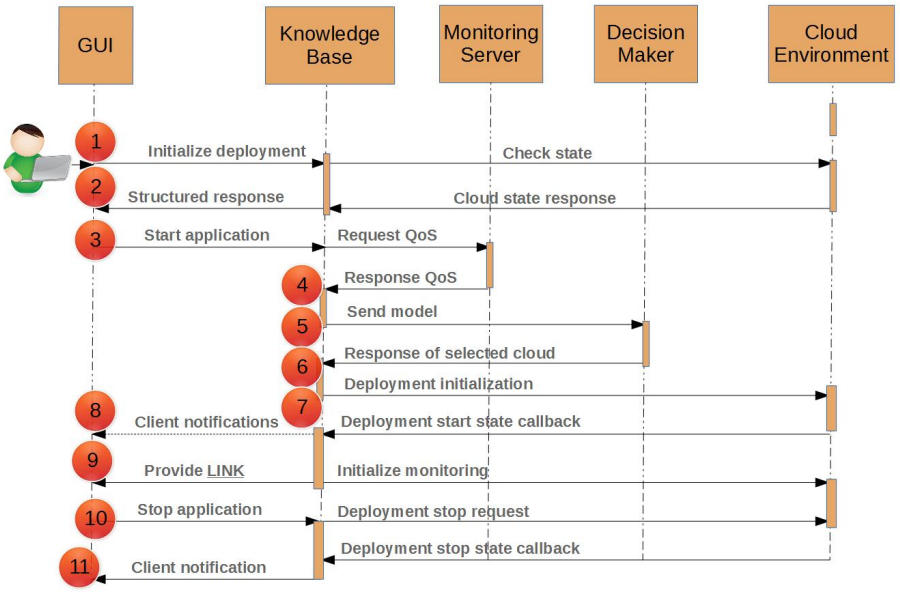


Figure 4: Sequence diagram showing roles of the GCM components for the videoconferencing application.

that is a Web application in its Web browser. This event triggers a query in the KB, which returns the availability of resources in the SDDC, taking into account the type of physical resources required for the particular application.

2. The response of the current SDDC status (e.g. availability of currently running VMs and/or containers and their QoS metrics) is provided to the Decision Maker through the KB component.
3. System state parameters and participating clients' Internet Protocol (IP) numbers are collected.
4. The Monitoring System provides monitoring metric data of all required metrics (for the specific Jitsi Meet application) to the KB component.
5. Monitoring response data is aggregated with RDF-based data to match decision model scheme definition and is passed to the Decision Maker component.
6. The Decision Maker component returns the most suitable host on which the service should be deployed. The decision making process itself is explained in Section 4.4.
7. The KB component initializes the application on the selected cloud and passes all needed application definition and cloud credentials in YAML format.
8. While deploying the application on the cloud the user is presented with the deployment status through real-time callbacks. In case of errors the user is notified and the execution is attempted again on the same cloud or on another cloud, depending on the error type.
9. On successful deployment of the videoconferencing application the Monitoring Server starts monitoring the running application, which is bundled with a Monitoring Agent to collect specific application-level metrics. Clusters are already monitored on the container, VM and/or infrastructure level with permanently present agents. User's GUI is presented with a link for the private videoconference call. For this particular application, the user might wish to share the link with other participants.
10. After an end user or a group of users are finished with using the application, the undeployment process of the container starts (when a user requests to stop the application).
11. Similar to step 9. The user is also notified about the state of the action in real-time and gets the final notification on successful undeployment. In this step also application-level monitoring agents are undeployed and the corresponding Cloud resources are released.

The discussed process can also be followed on the GUI presented in Section 6.

4 QoS modelling and decision making

QoS modelling and decision making for network-intensive software components are two important aspects of the implemented autonomic, event-driven global orchestration technology. In this section, the methodology and the implementation of a specific approach for QoS modelling is presented. The modelling approach is designed in a way that it can benefit from the capabilities of the GCM. Of course, as is the case with all optimisation techniques, further improvements of the presented QoS modelling approach could be possible.

4.1 Background

In order to implement a QoS modelling and decision making approach it is necessary to consider available adaptation mechanisms supported by the underlying platform. Commonly used are horizontal scaling approaches, such as those implemented by the ROAR project [40]. The ROAR solution scales the number of running Virtual Machines, while considering QoS and operational costs of the application. The horizontal scaling approach, however, is not sufficient to address the QoS requirements for network-intensive functionalities. Another adaptation possibility is to select or dynamically change the Virtual Machine instance types (e.g. T2, M4, C4, X1, P2 etc.), the amount of allocated RAM or number of CPU cores. And a third option, investigated in this study, is to select geographically the best possible infrastructure for the required service utility.

Thus, the QoS modelling approach has to take into account from one side, the autonomic adaptation capabilities supported by the orchestrator, and at the same time the plethora of available QoS metrics that can be collected and analysed before deploying the application, but also at runtime. The later relies heavily on the capabilities of the implemented QoS monitoring system. For example, Salman et al. presented an example for Internet of Things domain [42].

The prediction goals for the QoS modelling approach should be also clearly specified. One possibility is to create application models that predict the performance of the software utility on different systems. For example, vPerfguard [51] uses a two-step modelling approach. In the first step, an aggressive detection and removal of correlated metrics takes place, which reduces the computational complexity for the implemented algorithm. To further reduce the complexity, in the second stage, only a subset of uncorrelated metrics is selected and used for the prediction. Such reduction of the search space could negatively affect the prediction accuracy. Their used approach generates several possible models, such as a linear regression model,

k-nearest neighbour, regression tree and boosting approach, and selects the one with lowest error for the application. Another approach used by Xiong et al. [51] has concentrated on finding those QoS metrics that can be used to predict the performance of the application. This is an interesting approach, however, it is based on aggressive metric pruning, and the fact that the models take a lot of time to calculate makes them hard to use in most scenarios.

Furthermore, the models created by vPerfguard are highly susceptible to noise, which has been addressed by fuzzy logic to some extent [22]. The approach relies on taking concepts from experts in the field to fuzzify the variables. This approach works for a smaller set of common variables. If the number of variables grow, or are hard to qualify, for example, by using application-level QoS metrics, the fuzzification of the metrics becomes impossible.

Most of these modelling approaches do not take into account network-level metrics, such as throughput, latency, packet loss and jitter, which are in the focus of this study. These are often most important for network-intensive software utilities. The values of such network-related QoS metrics often determines the end users' quality of experience, and thus are highly relevant and have to be taken in the modelling process. The CA-DAG model developed by Kliazovich et al. [25] accounts for network-level metrics, however, it is merely a conceptual approach and has not been applied in real-world scenarios.

An important additional consideration to be made is the quality of the network-related metrics observed for InfiniBand connections within specific data centres, which is usually very high, and the quality of open Internet connections between the end users and the running software services. The later is much more critical, and has not been sufficiently addressed by existing studies. In this work, a new approach based on a qualitative modelling technique is presented that is suitable for a variety of network-intensive software utilities and for its use to predict the QoS within geographically distributed multi-cloud environments. Qualitative modelling is a well-known technique [31], used mostly in control and analysis of complex systems. It is resilient to noise, requires small learning sets and is more intuitive for human understanding than most other Machine Learning systems. In recent years it has been used to detect faults in complex systems, such as Centrifugal compressors [30]. In some applications, qualitative modelling is augmented with rule-based approaches [9].

While quantitative modelling techniques, such as statistical and machine learning methods are used to induce exact models (e.g. expressed with equations, decision trees, rules and other formalisms) of the investigated phenomenon, qualitative modelling approaches aim at inducing models only as

incentives to be used in a decision making process. Qualitative modelling is usually viewed as abstraction of quantitative modelling and is extensively elaborated elsewhere [7] and [16]. Examples of applications include prediction of ozone concentration [53] and modelling of phytoplankton [46]. In this work, qualitative modelling is used to provide support for the decision making process of the developed GCM.

4.2 Qualitative QoS modelling approach

Real time applications, such as videoconferencing and online gaming, are time-critical since they rely heavily on UDP traffic. Different aspects of the quality of such traffic can affect significantly the experience of the end users.

It is extremely hard to predict what affects the quality of experience of individual end users, even when the application is concise enough that all the users expect the same thing. Some users may expect quality sound, other ultra high definition for the video, for third, it is important that communication flows without noticeable delay. The usual way to find out what satisfies individual users is by preparing a questionnaire and collecting feedback from them. For example, popular online services ask the users to rate the various aspects of their experience (with 1–5 stars), after using the online service. While this is not an ideal approach (as it does not provide complete information to the developer to adapt the application), it still provides some useful information about the users' experience that can be used to implement an autonomic orchestration capability.

In our approach, feedback from potential end users can be collected during the beta testing phase of the application or when the application is in production, along with multi-level (infrastructure, networking, Virtual Machine, container and application-level) QoS metrics [41]. High-level (application-level) QoS metrics are, for example, the Peak Signal-to-Noise-Ration (PSNR), which is important for videoconferencing applications [50], video delay or file upload speed in case the end user wishes to upload a file to a cloud service, and similar. The users' feedbacks on various quality aspects may also be considered as high-level (user-level) QoS metrics, for example, rating the sound quality in a videoconference with 1–5 stars after the session. High-level QoS metrics may be selected and used in the QoS modelling process, in order to predict what affects the QoS of the application.

Therefore, the question to be answered by the QoS model is which underlying (infrastructure, networking, Virtual Machine, container, etc.) QoS metrics affect the application at runtime; how, and to what extent. This should be predicted at fine-grained level for every single software utility. Such QoS model can be used in a predictive manner to steer and make use

of the capabilities of the global autonomic orchestrator. In the current approach this manifests as prediction of what QoS metrics are influencing the performance of a specific software utility.

The implemented technology is based on qualitative modelling [1]. As such, this approach does not provide concrete prediction for the experience of the end users, but a way to compare the performance of the same software utility, when running in two different geolocations or on two different infrastructures. It can thus be used to predict what are the important QoS metrics for the particular software utility, which pairs of metrics are correlated, and so on.

4.3 Implementation of the QoS modelling approach

The implemented qualitative QoS modelling process goes through several steps. First, the developer has to run the software utility for which she wishes to generate a QoS model. This process involves the collection of multi-level monitoring data, which is conveniently stored in a time-series database. Once an initial set of runs is completed, the collected data can be used to generate a QoS model.

The implemented algorithm for building QoS model from measurements is presented in Listing 1 as code in Python. For illustration, the algorithm is shown in steps as a manipulation of tables in the following. The collected data (in our example Table 1) are used to create a differential table, which contains element-wise differences between all of the row pairs. The differential Table 2 shows examples of row pair differences between rows five and one, rows six and one, and rows six and five. Note the presence of the QoS metric in the first column of Table 1 and Table 2. Next, for each row, all of the values for non-QoS metrics differences are compared to the respective QoS metric difference. If the direction of change matches, the correlation is positive; otherwise, it can be either negative or neutral. Correlation is negative if the direction of change between QoS metric difference and non-QoS metric difference is opposite. An example of this step is presented in Table 3. The last step (shown in Table 4) simply adds all the values along the columns together. Positive values (in the table marked as green) denote positive correlation with the QoS metric, while negative values (red in the table) denote negative correlation. The resulting QoS model is then normalisation of the summed values. The model does not contain QoS metric, but encodes it in the rest of the metrics. This allows for performing measurements and making decision based on the non-QoS metrics only without having the application deployed.

Thus, the implemented algorithm simply calculates the probability of the

Listing 1: Python implementation of the QoS modelling algorithm.

```
from itertools import combinations
import numpy

# M is an input numpy 2d array of metrics.
# The QoS metric is stored in the first column.
# M has n rows (i.e. examples) and m columns (i.e. metrics)
    .
n, m = M.shape

# Compute the total number of all unique pairs of the rows
    of M.
nn = n * (n-1) / 2

# Initialise regression vector to zeros for all non-QoS
    metrics.
r = np.zeros(m-1, dtype=int)

# For every unique pair of rows i, j in metric table M
for i, j in combinations(range(n), 2):
    # compute vector of differences between rows i and j of
        M
    d = M[i,:] - M[j,:]

    # keep only the direction of change in d, i.e. the sign
    s = numpy.sign(d).astype(int)

    # add +1 to the regression vector r for all non-QoS
        metrics
    # that match in the direction of change with the
        direction
    # of change of the QoS metric, and -1 for those non-QoS
    # metrics that are opposite in the direction compared
        to
    # the QoS metric.
    r += s[0] * s[1:]

# Finally, normalize the regression vector such that all
    values
# are bound to the [-1, 1] interval.
qos_model = r * (1 / nn)
```

Table 1: An example of the monitoring data represented as an input for the qualitative modelling.

QoS	RTT	No. Hops	Memory Free [MB]	No. Cores	CPU
2	33	20	1000	1	12
2	32	20	600	1	50
5	12	5	200	1	30
4	27	15	100	2	20
3	12	5	100	1	100
4	40	25	100	2	25
1	44	20	500	1	60

Table 2: An example showing a selection of calculated differentials from Table 1.

QoS	RTT	No. Hops	Memory Free [MB]	No. Cores	CPU
1	-21	-15	-800	0	18
2	7	5	-900	1	13
1	28	20	0	1	-75

Table 3: An example correlation table computed from Table 2 showing coloured correlations between non-QoS metric differences and the QoS metric differences – red is negative, blue is neutral and green is positive correlation.

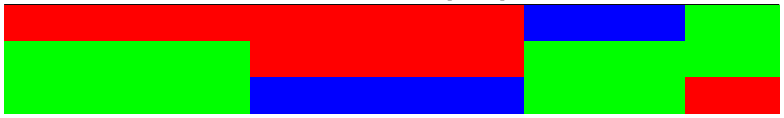
RTT	No. Hops	Memory Free [MB]	No. Cores	CPU
				

Table 4: An example QoS model before normalisation computed from Table 3. Rows of the previous table reduce to a single row, encoding a QoS model.

RTT	No. Hops	Memory Free [MB]	No. Cores	CPU
				
1	1	-2	2	1

positive correlation of each QoS metric compared to one selected, important (governing), high-level "Quality of Experience" / QoS metric. When correctly modelled, the function modelling the QoE metric is monotonic. If the qualitative modelling process results in a non-monotonic function, this is usually the result of noise. The result of the technique is interpreted as

probability value that determines the parameters (in our case monitoring metrics, such as throughput, latency or software component constraints), which have the greatest influence on the experience of the end users. In our case, the existing qualitative modelling algorithm has been simplified due to the nature of the investigated modelling problem.

The final result of the process is therefore a vector of weights that correspond to the strength and correlation direction (positive or negative) of the QoS metrics with regard to the governing QoS (QoE) metric. Generated models by using the presented algorithm have been shown in the Results section for both developed applications.

The generated model for the particular software component can then be conveniently stored in the KB by using the KB APIs, so that the decision making component of the orchestrator can use it to make decision where and with what computing resources to start the service.

Examples of realistically generated QoS models can be found in Section 5 based on the two cloud applications used in the experimentation.

4.3.1 Runtime multi-level QoS measurements

In the presented architecture, the operation of the Global Cluster Manager relies heavily on the existence of QoS models as described in the previous sections, and on runtime measurements of all multi-level (infrastructure, networking, etc.) metrics affecting the quality of the users' experience. The implemented Monitoring System [41] is designed in a way that supports horizontal and vertical scaling mechanisms as well as geographic placement of software utilities. Once the software component is launched, it is monitored and data are collected in a time-series databases. The Monitoring System relies on a great number of Monitoring Probes for various metrics to be collected (see results section) when the software component runs in various cloud providers and Virtual Machine/container configurations. Particular focus has been given on the development of network-related Monitoring Probes. If the Internet Protocol (IP) numbers of the clients are known, these Monitoring Probes can measure network-related metrics (e.g. jitter) from various Virtual Machines running in the SDDC and the clients. This allows to select the best possible placement for the software utility (which requires to minimize jitter for high quality of experience) and for the particular client.

The implemented Monitoring System also allows the implementation of custom Monitoring Probes in case the type of existing probes is insufficient for the monitoring of a particular software utility. This runtime information

is therefore used in the decision making process used by the orchestrator, which is elaborated in the following subsection.

4.4 Decision making

When an application (network-intensive software utility) is to be deployed, an automated decision has to be made on where to place it. The decision making is done in two stages. In the first stage the KB filters out all the deployment possibilities that do not reach the minimal requirements for the application. For example, it removes all the configurations that do not have enough memory for the application or the CPU instruction set of the computer system is not compatible with the application code (e.g. in edge clouds an ARM CPU architecture might be more common than x86, but the container is only compatible with one of them).

After the first stage a subset of computing resources with different configurations remains that could successfully be used to run the application, but not necessarily with the sufficient QoS. Therefore, at the second stage the decision making module is invoked. It retrieves the model from the KB and starts comparing the configurations according to the relevant metrics. It queries the Monitoring System running on all the appropriate locations (VMs or physical computing resources) to provide the QoS metrics that were used for the model creation. Once all this information is gathered the algorithm computes average values of the measurements, resulting in a single vector (i.e. a row) of data per available machine. What follows is a computation of a score metric. For each of the machines the score is computed by subtracting element-wise all of the rows resembling the performance of the other machines, preserving only the direction of change, and adding up all the elements which have the same sign as the corresponding QoS metric of the model. The resulting vector is then weighted with the weight from the QoS model, to capture the relative importance of the metrics in a single score. The higher score means better machine.

5 Results

The architecture and design of the GCM has been tested and validated by using two software utilities: File Upload and Jitsi Meet videoconferencing. The File Upload is a rather simple usage scenario; however, it is still network-intensive, and therefore represents the core problem addressed by this work. A user may wish to upload a file to a Web service as quickly as possible from anywhere on Earth, which is an adequate representation for the required QoS. The second, Jitsi Meet software utility is much more complex. A

videoconferencing Web application was developed based on the Jitsi Meet Open Source software, and it can be used from a Web browser via the WebRTC protocol. It allows several users to engage in video-conversation in real time. Such users, potentially film directors or medical doctors, may have stringent high-level quality of experience requirements, for example, Ultra High Definition (UHD) video resolution without noticeable delay.

The complete workflow of QoS model building, decision making and service placement by using the developed GCM was tested for both applications. This section summarizes the obtained results.

5.1 Cloud applications

5.1.1 File Upload

The File Upload application (i.e. network-intensive service utility) allows users to upload their files to the cloud. Services like this already exist for a long time and one may argue that such functionality is best provided via a solution with permanently running servers, like in the traditional Content Delivery Networks (CDNs). However, such solutions are costly, because the computing resources are consumed even, if there are no user requests to upload files, meaning that the storage services are operating in an idle mode. In our solution the service is created, served and destroyed for every file upload request. This approach can reduce costs and allows to dynamically select/modify all the aspects of the service operation. With wise selection of the location and the infrastructure, where the service is instantiated it is possible to provide best possible QoS, while at the same time reduce the operational cost or satisfy any other operational strategy defined by the application administrator.

The File Upload application is relatively simple and in its basic form does not require any kind of configuration; however, in some cases it might be beneficial to collect the information about the file size prior to deciding upon the location of the service. For very small files the distance and network characteristics between the client and server might not really matter in the decision making steps, because provision time of container and the application might outlast the upload time when uploading to the more distant server. (In any case, it should be beneficial for medium- to large-sized files.) The simplest way to implement such context gathering step is to provide a form to an end user and ask for file size (or an order of the size). Based on the file size the server would either handle the request itself (for small files) or invoke the orchestrator to prepare the service at suitable location and provide a user with a link.

Let us assume that the quality of experience for the File Upload service is directly proportional to the time needed for file upload. This QoS metric depends on many factors, such as client to service network QoS and the server-side resources (disk I/O, RAM and CPU). The File Upload service is designed to work over the best-effort public Internet with clients from anywhere and unpredictable network traffic patterns. The challenge is therefore to determine the best possible VM/cloud provider from the SDDC, and its resource characteristics, where the service should be deployed to serve a particular file upload event with high QoS. We expect that this location will primarily depend on the network conditions between the client and the service; however, the computing resources allocated to the service also need to be taken into account.

Depending on the performance characteristics of the VMs and physical hosts, both container and service provision times might vary between different machines. In our simplistic case the difference was not significant and therefore as a rule of thumb, the metric Round-Trip-Time (RTT) was considered reasonable to describe the quality of the network paths between the client and the potential service endpoints. Another estimate is to perform a Transmission Control Protocol (TCP) throughput test, which might require more bandwidth and more tests compared to the RTT. Furthermore, many TCP throughput tests, such as iperf3 require to install a piece of software on both the client and server side of the link. Therefore, in order to develop a monitoring probe for the File Upload application, one can use the ping tool to estimate an average RTT from the server to the client. Alternatively, pinging can be implemented on a client side from a browser script.

5.1.2 Videoconferencing

The videoconferencing example application is represented by a WebRTC multiparty videoconferencing application that is based on real-time multimedia streaming RTP/UDP protocols. For our purposes we selected an open source Selective Forwarding Unit (SFU)-based videoconferencing application called Jitsi Meet. The application uses a centralized architecture, with media processing component that only decrypts, encrypts and forwards RTP packets, optionally changing their headers, but without processing the payload, such as computationally intensive video and audio transcoding. In general, this makes the component more CPU-efficient, but less bandwidth-efficient. The SFU component is called Jitsi Videobridge. Other centralized units are included. Jicofo acts as a conference focus initiating sessions between the endpoints. Web server hosts Jitsi Meet JavaScript WebRTC application. Prosody XMPP server allows for the exchange of the signalling messages between components. The client part of the application runs as a

javascript in modern Web browsers that support WebRTC protocol, such as Google Chrome and Mozilla Firefox.

Similarly to the File Upload application, we are not using dedicated servers to host the Jitsi Meet services. Instead, the services are created and destroyed dynamically for each individual video conference. Again, the application is intended to work over the public Internet, providing no means of network QoS tuning or assurance. Besides high bandwidth requirements and low packet loss this kind of applications require also low network latency, due to the interactive nature of the communication. Although the service is SFU-based it still requires substantial amount of CPU power, which could also represent the application bottleneck when many users are involved in a videoconferencing event. It is therefore expected that by wise selection of the location of the service and the computational resources where the service will be instantiated it should be possible to leverage the required quality of experience. Additionally, it could be possible to reduce the operational cost, because the service will never run in idle mode. Also, elasticity to a varying number of users will be achieved automatically, as long as a single Jitsi Meet instance can handle the resources required for all of the participants involved in a single videoconferencing event.

5.2 Testing environments

Testing was done by using on-premise and public and private cloud infrastructures. The SDDCs are not the same for the two example applications due to various reasons, so they are separately explained in the following.

5.2.1 File Upload

The File Upload application is composed of a single client and a single server computing node. QoS models were built based on measurements performed between a client VM running on our on-premise servers, and servers running in the SDDC. The VM configuration for machines deployed in the SDDC was the following: 4 vCPUs VM with hardware accelerated CPU virtualization powered by Intel Xeon E5649 CPUs clocked at 2.53GHz, 2 GB of RAM, 20 GB of disk backed by RAID 6 network-shared disks, Ubuntu 15.10 operating system, and was running on top of VMware hypervisor. A client VM was located in Ljubljana, Slovenia.

The SDDC is composed of cloud servers running in public and private cloud infrastructures. For public cloud we used Google Container Engine with Kubernetes clusters deployed in five different regions: Asia East (Changua County, Taiwan), Asia Northeast (Tokyo, Japan), Asia Southeast

(Sydney, Australia), Europe West (St. Ghislain, Belgium), and US West (The Dalles, Oregon, USA). Every cluster comprised two n1-standard-1 VM worker nodes, each with 1 vCPU, 3.75 GB of RAM and 100 GB of locally mounted hard disk. Every vCPU was run on a dedicated hardware thread backed by powerful Intel Xeon processors. Kubernetes nodes communicated via hardware-switched network. The network interface per node was at least 1 Gbps. The software running on each node was Container-Optimized operating system from Google, and Docker container version 1.11.2.

Moreover, private infrastructure were also included in the SDDC. These included Arnes cloud located in Ljubljana, Slovenia, and an infrastructure of FlexiOps located in Edinburgh, UK. Kubernetes cluster in Arnes cloud comprised two worker nodes, each with a single vCPU accelerated by Intel Haswell processor, 4 GB of RAM, and 80 GB large network-shared disks. The installed software was CoreOS operating system and Docker version 1.10.3.

FlexiOps' Kubernetes cluster comprised three worker nodes with AMD Opteron backed 4 vCPUs, 4 GB of RAM and 100 GB of network shared disk (Ceph) each. The installed software was CoreOS operating system and Docker version 1.11.2. Different to the commercial GCP solution, both Arnes and FlexiOps clouds vCPUs were scheduled pre-emptively and Kubernetes nodes were communicating via flannel network overlay.

5.2.2 Videoconferencing

The videoconferencing application requires two client computers hosting in-browser WebRTC application and one server node hosting Jitsi Meet application components. Laptops with mounted Web camera and microphone were required for the experiment. The software installed on laptops was Windows operating system and recent Google Chrome browser. Both client computers were connected into the local network over a physical network cable to remove the potential issues that could arise from using WiFi connection. Because the Jitsi Meet application relies on SFU-based videobridging, the video and audio transcoding process happens – if so required – on the client side. Therefore, it was necessary to assure that the client computers are powerful enough for the task.

Server part of the videoconferencing application was run on a HP ProLiant on-premise server with 16 computing threads, 8 GB RAM, 2 x 2 TB disk and 1 Gbit network interface. The installed software was Ubuntu 16.04.3 LTS and Docker version 17.06.0-ce. Network simulations were carried out by using tc (i.e. traffic control) netem command line utility from the iproute2 Debian package. The Linux kernel version was 4.4.0-66-generic. The Jitsi

Meet application was containerized and it is available freely under an Open Source license on Docker hub [37].

5.3 Experimentation

The goal of experimental evaluation was to show that the proposed architecture is operational and at the same time – with respect to the QoS model – allows for improved QoS based on the network metrics. Due to prerequisites to our solution that software engineer performs the measurements required for building the QoS model, the appropriateness of the model depends a lot on the careful consideration of the performed measurements. Based on the experience we had obtained previously with videoconferencing applications and preliminary observations carried out, we have decided to perform an experiment in which network conditions were emulated and controlled in-house rather than captured through observations of Jitsi Meet running in different regions over the World. This way it was possible to achieve greater variety in network conditions, perform experiments much faster and study effects of network conditions for arbitrary settings. Contrary to this, for the File Upload application we performed measurements in a scenario that mimics realistic use case. Further details on the experimental setup for both applications are given in the following.

5.3.1 File upload

For the File Upload application in order to obtain the metrics required to build the QoS model a series of file uploads of various sizes were performed. The client was fixed at one location while the server part was floating around all of the available machines in all of the clusters. The end user was simulated on the client machine with the script executing the file upload action. While in realistic scenario besides the upload time the end user would perceive also the pre-deployment time of the File Upload application and then also spending some time with the File Upload GUI to select the file and click the upload button, the experiment focused only on the upload time.

Table 5 shows metrics that were of interest for the File Upload application. The QoS metric chosen was the upload speed – the ratio between the size of the uploaded file and the time needed to upload it. This metric is reasonable choice for the QoS, because it reflects the time needed to perform the upload. Therefore, it could directly correspond with the QoE perceived by the end user. Instead of choosing upload time needed for the QoS we decided for the upload speed to hide the file size. However, while clearly the upload time generally increases for larger files, even the upload speed metric still depends on the file size, as this ratio increases for larger files too. The other three

Table 5: Metrics collected to create the QoS model for the File Upload application.

Metric	Description
Upload speed (B/s)	The goodput: the average upload speed for the client–server connection. This is the QoS metric.
Delay (ms)	The packet round-trip-time as measured for the client–server connection.
Jitter (ms)	Jitter as measured for the client–server connection. This metric was computed from the samples of delay metrics: as a sampled standard deviation of the delay.
Packet loss (%)	Packet loss as measured for the client–server connection.

metrics of interest were packet delay, jitter and packet loss. Throughout the course of each upload session, a NetData [39] monitoring server was deployed on the same machine as the File Upload server application in order to measure these three metrics against the client. This setting required the client to be accessible to ping echo requests over the ICMP protocol. In realistic scenario such assumption is not generally viable, but due to the simplicity and known methods to mitigate this problem we assumed that ICMP traffic towards the client does not end in a black hole. The ping measurement interval was 200 ms and every ping request was sent with a small payload.

Because the client application was executed from a VM that usually serves as a server machine, it largely eliminated other possible issues that otherwise can occur in realistic scenarios, such as overloaded client machine, competition for network bandwidth with other applications and even other users in the same local network, and WiFi connectivity problems. This is reasonable assumption as we focused on what can be achieved on the server side.

5.3.2 Videoconferencing

For the videoconferencing application we performed a subjective QoE measurement experiments in a scenario with two end users connected to a videoconferencing server and exposed to varying network QoS settings. Similar to the File Upload application, the network QoS was represented by three network parameters, i.e. jitter, delay and packet loss. Because in real networks these parameters cannot be controlled, we performed a simulation with Linux Traffic Control tool. We setup a videoconference room and after

that we modified the settings for the three network parameters described in Table 6 to simulate different network QoS conditions. For each setting the human observers were asked to rate the quality of the videoconferencing session. Of course, the quality of videoconference potentially depends on many parameters, such as the processing power on both clients, as well as on the media server. However, it was necessary to simplify the model for the study, and these parameters were not taken into account. The generated model therefore assumes that all other infrastructure-level parameter values are in the range, where the quality of experience is not affected significantly. To make sure that during the experiment the infrastructure-level parameters do not affect the quality of experience (i.e. a governing QoS metric), we assured that both client machines had enough computing resources, i.e. powerful personal computers were used, and during the experimentation all other applications except the Web browser were closed. On the server side we installed the NetData [39] monitoring tool, and we made sure that during the videoconference the CPU, memory consumption, I/O, software interrupts and other important metrics were far below the thresholds, where they could potentially start to affect the quality of experience. For example, throughout the course of experiments the CPU usage on the server, even in peaks, never surpassed the 10% mark.

Table 6: Metrics measured to create the QoS model for the videoconferencing application.

Metric	Description
Star rating (MOS)	QoE metric. Number of stars collected per each videoconferencing event for each of the users. The range is 1–5 stars where 1 star is the worst and 5 stars is the best quality as perceived by the end user, respectively.
Delay (ms)	The average inbound and outbound packet (IP layer) delay on the network interface. The total end-to-end delay is therefore a sum of current inbound and outbound packet delay.
Jitter (ms)	The average inbound and outbound packet (IP layer) jitter on the network interface. The total end-to-end jitter is therefore a combination of these two values. This metric is computed as a sampled standard deviation of the delay.
Packet loss (%)	The average inbound and outbound packet (IP layer) loss on the network interface. The total end-to-end packet loss is therefore a combination of current values of these two parameters.

In the following, few additional details on the server side setup are provided. On the videoconferencing server the Jitsi Meet application with all of its components was run in a single Docker container. A separate bridge network interface was created beforehand to only affect that interface with performed network QoS simulations, while keeping the server otherwise normally operational. The Jitsi Meet application was attached to this bridge network interface. The network simulations were performed as described on the Linux Foundation Wiki page [49]. In particular, we configured the interface to simulate delay, jitter and packet loss symmetrically in both directions. Because both clients were located in the same local network and communicated to the same server, also the network conditions were nearly symmetrical for both of them. The configuration for jitter was selected to be normally distributed around the specified mean value and the packet loss was configured to appear more likely in bursts rather than in a uniform distribution. However, the bandwidth was not controlled. The monitoring agent with ping probes was deployed in a separate Docker container on the same server and bridge interface as the videoconferencing application. This caused similar network QoS conditions against both the videoconferencing application server and the monitoring agent in order to measure degraded delay, jitter and packet loss against both clients. The selected ping interval was one second with small payload.

It is well known that in-depth subjective quality assessment of telemeetings is a very complicated task. To verify the perceived quality in this kind of applications, formally agreed test methods were published: the most relevant of them is the ITU-T Recommendation P.1301 [21]. Many factors can affect telemeeting quality and the situation gets more complicated if there are several participants using different types of equipment. This leads to a complicated and costly test methods. Because the purpose of our work was not to build the application model with the highest possible accuracy, we did not follow any formally defined test method, and the tests were simplified considerably.

In our experiment only two participants were involved. The two users were non-experts, and were asked to have an arbitrary videoconferencing conversation, displayed on their desktop screens and using their own personal computers equipped with Web cameras and microphones. After each traffic configuration change the conversation of the two participants lasted for 3 minutes and after that they were asked to independently evaluate an overall session Mean Opinion Score (MOS) rating from 1 to 5. The traffic configuration was then changed and next score collected from the users. To minimize the effect of uncontrollable additional QoS degradation caused by the real network between both users and centralized media server, both users and the server were located in the same local network. The obtained results are presented in Section 5.4.2.

5.4 QoS measurements

By following the procedures described in Section 5.3, we carried out measurements required to build QoS models for both example applications. In general, the results match with our expectations, and show that the proposed QoS metrics are to some degree influenced by the proposed network-level QoS metrics. This opens possibilities for our solution to make informed decisions about application deployment that lead to achieving high QoS.

5.4.1 File upload

Figure 5 shows the results of the measurements for the File Upload application. Not surprising, the upload speed declines as the network delay increases. From the plot it is hard to see any effects of jitter, since the measured jitter was low for all of the measurements. Packet loss was only occasionally observed in one of the distant regions and is not shown on the plot. For the sake of the QoS model generation, in this experiment it would have been probably better to perform network QoS simulations, similar as it was done for the videoconferencing example, in order to end up with a training data better representing diverse network conditions (e.g. heavy Monday morning traffic).

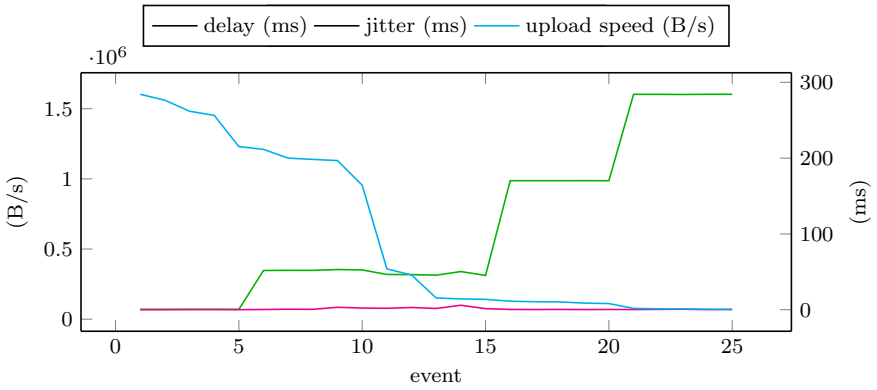


Figure 5: The results of the QoS measurements for the File Upload application sorted by the upload speed, descending. Events denote each of the performed upload session. Only uploads for the file size of 1 MB are presented here.

5.4.2 Videoconferencing

The results of the QoE evaluation of the videoconferencing application in different simulated network conditions are presented in Figure 6. The event

represents a particular traffic configuration (i.e. delay, jitter and packet loss configuration) and respective MOS score for one of the end users. In total, 49 different traffic configurations were performed and two users were grading the QoE: this resulted in 98 "events" shown on the graph. The value of 49 traffic configurations was chosen by a rule of thumb. On one hand we did not want to stress the human participants too much, since tiredness of the evaluators can have impact on the MOS grading they give. On the other hand, the number of experiments (i.e. learning set) should be large enough for the model maker to produce a meaningful model. Also, the set of traffic configurations should be diverse, with the values in the range that can be typically expected in real global Internet network infrastructures. For this purpose we first measured the typical values that can be expected in real networks, i.e. we measured the delay, packet loss and jitter on six different clusters of our test bed. After obtaining these orientation values, we performed a test phase, in which two participants started a videoconference, while an expert was modifying the delay, jitter and packet loss settings and tried to find the thresholds when the QoE started to deteriorate. Based on these observations the expert selected a suitable set of 49 different network settings. Some outlier values (i.e. values that cannot be typically expected in real networks) were also consciously included in this set in order to prove the effect on the quality of experience, and evaluate how the model deals with them.

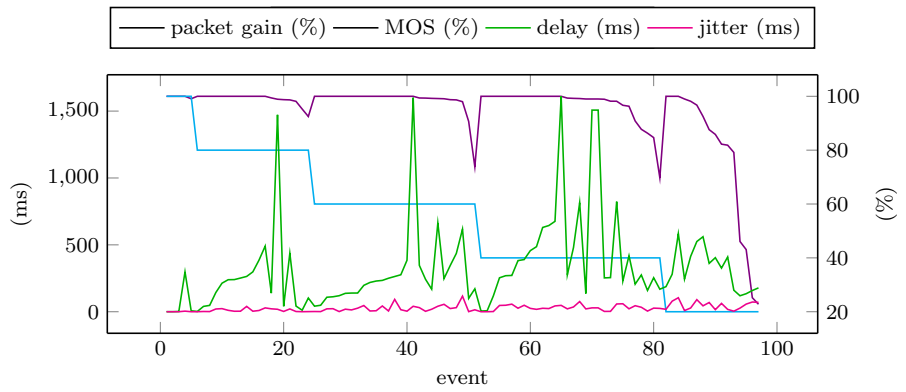
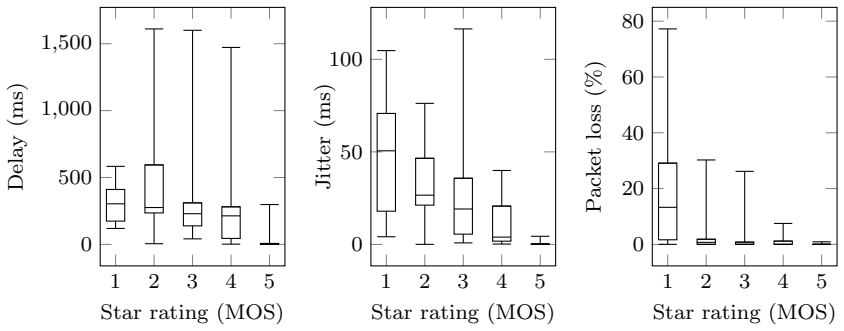


Figure 6: The results of the quality assessment for the videoconferencing application sorted by the MOS and then by the packet loss, descending. The MOS expressed as a percentage corresponds to star rating, where 100% MOS score here translates to five stars (i.e. the best quality of experience), 80% MOS corresponds to four stars, and so on down to 20% MOS matching one star. For the clarity of presentation the packet loss is shown as 100 - packet gain.

The obtained results are in line with human reasoning. It can be seen that the MOS score falls with the increase of packet loss and jitter. One

observation is that the delay does not seem to be strongly correlated with the given MOS score. However, it was shown in the previous studies that the influence of delay is not always mirrored in test results, but might affect the conversational quality of videoconference quite substantially [5]. Special purpose quality evaluation methods and separate questionnaires should be used for delay tests, which was not the case in our study.



(a) Delay (b) Jitter (c) Packet loss

Figure 7: Delay, jitter and packet loss per star rating for the videoconferencing application. Box plots show the first, the second and the third quartile, respectively; whisker plots show the minimum and the maximum values.

Because in our experiment the network QoS values were selected by a rule of thumb and not in any systematic way, Figure 7 shows the basic statistics of the three observed metrics in relation to the quality of experience. While outliers tend to be present in all star rating classes below five stars, again the trend is clearly seen: the higher each of the observed metrics is, the worse the star rating gets.

5.5 QoS models

With the QoS measurements of Section 5.4 and the QoS modelling technique presented in Section 4.3 the QoS models for each of the two example applications were created. The interactivity of the File Upload and videoconferencing application differ drastically, with File Upload having low interactivity and videoconferencing having high one. Despite that, the degraded network conditions tend to have similar effects with respect to the QoS on both applications, although the exact effect of individual parameters might vary. The resulting QoS model of both applications show negative correlation of all of the three observed network metrics with the QoS. This confirms the usefulness of the model, although the degree of the acceptance would require more thorough evaluation.

5.5.1 File Upload

Generated QoS model for the File Upload application is presented in Figure 8. The model suggests that by large the packet delay negatively influences the upload speed, followed by the packet loss and jitter. The result is inline with the common use of geolocation, since the network delay generally increases with the increasing physical distance between the endpoints of interest. Nevertheless, even though the results might be correct for the delay, the exact effect of the other two parameters, especially the packet loss is less clear since in our experiment the latter only occurred for one of the servers that was the furthest away from the client. As already mentioned, network simulations might help.

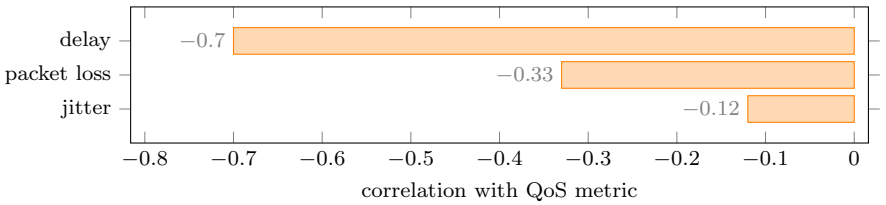


Figure 8: A QoS model for the File Upload application, measured for 1 MB file upload. All three observed metrics are negatively correlated with the QoS.

An important observation had occurred during the evaluation of the model. Due to the fact that the selected QoS metric – the upload speed – tends to increase with the increased file size, the model obtained from measurements with varying size of the uploaded file tend to show less confidence in either of the observed parameters. In other words, while the increase in parameters still indicate negative impact on the QoS, their correlation approaches zero. The reason for that is clear: because the modelling technique compares every pair of measurements with each other, the situation might occur in which one example has much smaller delay than the other, yet it also has smaller upload speed than the other. This usually happens if the two measurements represent two servers with increasing distance with respect to the client, but in the course of the measurement for the first much smaller file was uploaded than for the second. Therefore, during the model creation when subtracting the second measurement from the first one both the delay and the upload speed difference is negative, indicating the positive correlation between the delay and the QoS metric.

The solution to the above problem is either selection of a QoS metric which does not depend on any hidden parameter or clear separation of the examples with respect to any of the hidden parameters. At the same time the weakness of the proposed QoS modelling technique has been exposed.

Consider, for example, that the observed metrics are CPU and memory usage. Then two notably different machine configurations in terms of these two parameters can show different behavioural patterns. Most importantly, without prior normalisation the measurements between such different machines should not be compared. Our testing infrastructure comprised more or less machines with similar configurations; besides they were compared in network performance only, which in most cases depends less on the sheer performance of the individual machines. Therefore, we allowed ourself to mix the measurements from various machine configurations.

5.5.2 Videoconferencing

Quality of experience model for the videoconferencing application is depicted in Figure 9. Compared to the File Upload application the importance of jitter and delay metrics are swapped. The difference between the metrics importance is, however, less evident. As a consequence, the decision making should take into account all of the metrics, not only a single one. Their relative difference is also small enough such that even a slight variance in the experimental setup could produce different results in the order of importance. For that to assert more thorough evaluation is needed. Nevertheless, if the model fairly represents the effect of these parameters on the quality of experience, the effectiveness of geolocation for this kind of problem becomes less evident. One should be reminded that we drastically simplified the videoconferencing application by only considering two parties, while both communicated on nearly symmetrical network configuration compared to each other. In more realistic scenario many users could participate with much higher variety than we were able to achieve. The variety can appear in many ways, including geographical diversity of the participants for which the effectiveness of the geolocation is also less evident.

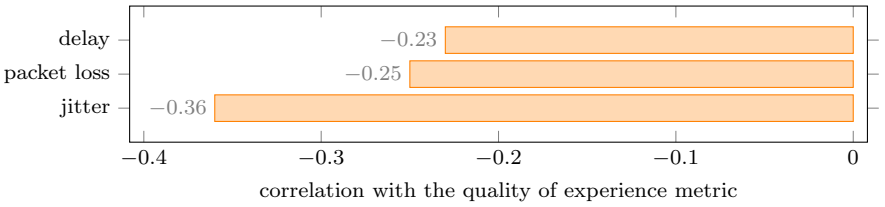


Figure 9: A QoS model for the videoconferencing application. All three observed metrics are negatively correlated with the selected quality of experience metric.

5.6 Automated decision making

Decision making is a process not directly connected to modelling, but uses its outputs. Decisions on where to start applications are adapted to the end users. To make a decision the created model identifies which measurements should be performed to support the decision about the placement of the application. Once the measurements are performed, which should take only a few seconds in time, they directly guide the decision maker in the process of making the decision. By following the second stage of the procedure described in Section 4.4, we performed measurements for delay, jitter and packet loss against one fixed VM located in Ljubljana, Slovenia, from all of the available machines in our testing infrastructure, except for FlexiOps, which was not available at the time of measurements. The ping echo requests were sent simultaneously 50 times with interval 200 ms from each of the server against the client. The averaged measurements results are presented in Table 7. The results show that packet loss was not sensed, the delay was nearly constant from the VMs residing in the same geographic region, and the jitter was slightly less predictable, although the trend of increased jitter with the distance can be noticed. Based on these measurements we used the respective QoS model for each of the example applications.

Table 7: Delay (ms), jitter (ms) and packet loss (%) for all of the machines available in the testing environment, except for FlexiOps.

VM	Delay	Jitter	Packet loss
arnes2	0.53	0.05	0.00
arnes3	0.52	0.04	0.00
gke-eu-west1	29.45	0.27	0.00
gke-eu-west2	29.53	0.23	0.00
gke-us-west1	169.99	0.57	0.00
gke-us-west2	169.99	0.47	0.00
gke-asia-east1	283.92	0.60	0.00
gke-asia-east2	284.31	0.06	0.00
gke-asia-northeast1	259.60	0.50	0.00
gke-asia-northeast2	259.28	0.68	0.00
gke-asia-southeast1	323.73	0.18	0.00
gke-asia-southeast2	325.85	0.62	0.00

5.6.1 File Upload

The QoS model of Section 5.5.1 and the measurements from Table 7 give the results presented in Table 8. In general, the results match the expectation: the servers with lower delay are better, which turned out to be the case also

for all of the upload tests we performed. Even though the decision maker considers all of the available metrics, in this case the ranking of the servers aligns with the decision made solely on the delay metric, except for the two servers in Eastern Asia region, where the difference in jitter was big enough to place the server with slightly higher delay and significantly lower jitter before the other.

Table 8: Decision maker's ranks and scores for the File Upload application. Higher score means better machine based on the pre-deployment time measurements and the respective QoS model.

VM	Rank	Score
arnes3	1	0.71
arnes2	2	0.65
gke-eu-west1	3	0.55
gke-eu-west2	4	0.51
gke-us-west1	5	0.42
gke-us-west2	6	0.38
gke-asia-northeast2	7	0.28
gke-asia-northeast1	8	0.26
gke-asia-east2	9	0.20
gke-asia-east1	10	0.18
gke-asia-southeast1	11	0.13
gke-asia-southeast2	12	0.01

Note that the score is a relative metric depending on the set of available servers. Therefore, expanding or contracting the set of servers would result in smaller or larger relative differences in score between servers, respectively. The maximum possible score is 1.0, but can be achieved only if there exists a server which is better than other servers in all of the metrics. Clearly, in our case ping was unable to capture any packet loss; therefore, the best performing server (i.e. arnes3) is not superior to any other in the packet loss. The relative weight of the packet loss is 0.29 – an amount that complements 0.71 to 1.

5.6.2 Videoconferencing

Using the quality of experience model from Section 5.5.2 for the videoconferencing application and pre-deployment time measurements from Table 7, the decision maker ranks the machines as presented in Table 9. Clearly, the packet loss played no role in the decision making process; therefore, the competing metrics were delay and jitter. As expected, machines located in Ljubljana (i.e. Arnes) won over more distant Google Cloud Platform

machines with the closest location in Belgium. In general, the results are somewhat similar to the File Upload case. Because the jitter is – according to the model – the most important metric, servers with smaller jitter sometimes precede the servers with larger jitter but smaller delay.

From the experiments that we performed with videoconferencing on all of the machines available in the testing environment, we were rarely able to perceive any significant degradation of the quality of experience, except for the increased delay, which can only be noticed in more interactive conversations. Another perceived degradation was occasionally stalled video for more distant servers, but is not always easy to sense, even when present. Thus, with the experiments done so far, it is not straightforward to assess whether the decision maker prefers better machines over worse ones. More thorough evaluation is needed.

Table 9: Decision maker's ranks and scores for the videoconferencing application. Higher score means better machine based on the pre-deployment time measurements and the respective QoS model.

VM	Rank	Score
arnes3	1	0.70
arnes2	2	0.64
gke-eu-west2	3	0.47
gke-eu-west1	4	0.46
gke-asia-east2	5	0.40
gke-us-west2	6	0.34
gke-asia-southeast1	7	0.34
gke-us-west1	8	0.29
gke-asia-northeast1	9	0.26
gke-asia-east1	10	0.15
gke-asia-northeast2	11	0.12
gke-asia-southeast2	12	0.04

An issue could arise from the inability to capture the packet loss, while it does occur from the perspective of the videoconferencing application. This is a challenge related to monitoring and is not the goal of this work.

Regarding the measurements presented in Table 7 an important detail arises when comparing both applications. Videoconferencing application is multi-party while File Upload is single-party. The measurements were performed in a single-party fashion since only one client was pinged. We can justify such experimental design with the following claims. Firstly, only one peer is required to setup the videoconference: the peer creates a room and sends the resulting link to other participants. There is no guarantee that all of the peers engaged in a videoconferencing will be present at the setup time –

some may even join the call later, sometime during the session. As a result, we can only rely on the presence of the conference organiser, that is, a single person, prior to the session. Secondly, if at the setup time more peers are known, the presented measurements should be treated as averaged metrics over all the peers. We only considered a simplistic case with two peers, in which both are almost symmetric in the performance behaviour; therefore, observing metrics for one client is almost the same as taking the averaged values from observing both.

6 Using the GCM

In this section, the utility of the GCM is analysed in relation to the stakeholders. A GUI is presented, which was developed specifically to explain the operation of the autonomic orchestrator, once the cloud application(s) (containing network-intensive software components) are deployed in a SDDC, and can be used massively by end users World-wide.

6.1 Stakeholders and usage scenarios for the GCM

Within a wider context the developed GCM is designed to be beneficial for two types of end users: software developers (DevOps) and end users.

By using usual software engineering tools, such as Juju, Fabric8 or SWITCH, a software developer could select important QoS metrics for an application. The same tools can be used in the testing and tuning phases of the application. If some QoS metrics have not been implemented in the underlying monitoring system, the developer can implement and include additional probes for monitoring QoS metrics. The testing can involve the use of diverse computing resources. Collecting QoS metrics in this phase is important, as it can help develop a QoS model of the developed software utility, and store it in a knowledge base. This means, that the QoS model of the software utility can further be used by the autonomic orchestrator, when the application is deployed and used by users World-wide, and thus achieve high QoS for the end users (Figure 10).

Much wider user base for the GCM are the application's end users. They can use the capabilities of the autonomic orchestrator to decide where and at what cost they wish to use their services. Recent study conducted in the context of the ENTICE project[17] showed that Multi-Objective Pareto-based modelling can be used to graphically inform the end users, and let them chose the non-functional properties of their service. For example, an end user from China could decide to use services only deployed in China, another end user from Australia could decide to use only cheapest possible

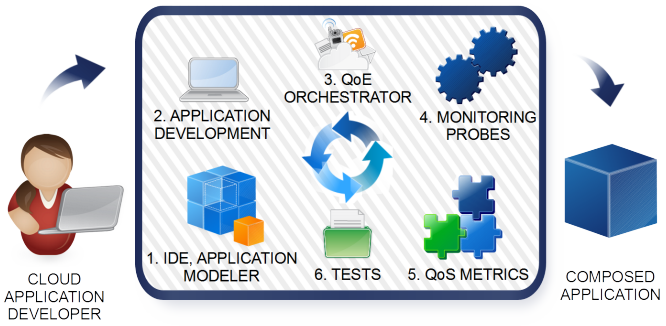


Figure 10: Life cycle perspective of a cloud application developer for event-driven and short-lived applications.

videoconferencing services, meaning that each time when the service utility is required, it will be deployed on cheapest cloud resources (Figure 11).

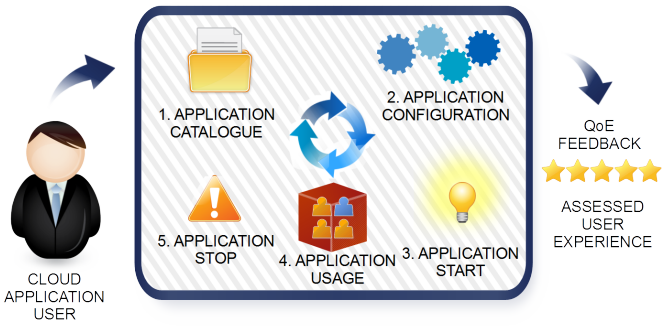


Figure 11: Life cycle perspective of an application end user for event-driven and short-lived applications.

6.2 Presentation layer of the system's workflow

A simple Web application was developed to present the idea of an end user who uses various network-intensive software utilities. Of course, in a more realistic scenario, such network-intensive software utilities can be included in stand-alone Web applications.

From the Web-based interface presented in Figure 12, the end users can select software utilities to run, perform simple applications setup, run and terminate applications and rate their experience. One can see that each end user must be able to use the available applications in a transparent way by simply running them and stopping them on demand, being not concerned with irrelevant information (e.g. state of the available cloud resources, the monitoring mechanisms and similar). Moreover, the orchestrator can deploy the container-based software utilities on different platforms (e.g. computer, mobile devices) to serve various types of clients including the use of the WebRTC protocol implemented in Web browsers, and similar. By carefully selecting computing resources, geographically, the system provides the highest possible QoS for the end users. In case when an end user runs several software utilities (applications) at the same time, the Web GUI registry reflects their actual state, including important metrics such as application status, service running time, cost and other, as presented in Figure 12.

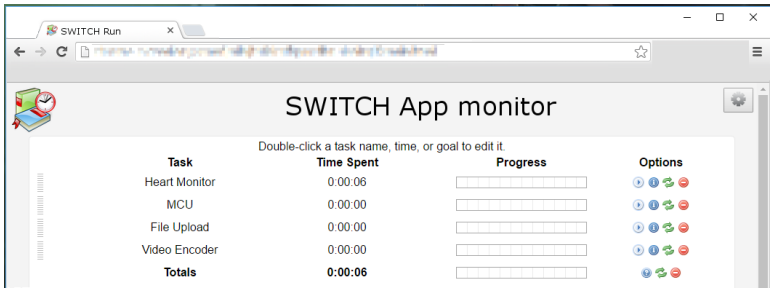


Figure 12: Application manager form – allowing end users to run and stop software utilities.

Another Web application presented in Figure 13 was developed to demonstrate the process of launching the cloud application and the functioning of the autonomic orchestrator i.e. the developed GCM. All process related information is presented, including the list of available geographically distributed cloud providers, where the container-based service can be started, the acquired IP numbers of the end users' clients that require to use the service (based on which the Decision Making component operates), the application's run and stop buttons, runtime QoS monitoring information, which can be seen in real-time for all selected QoS metrics, other real-time reports as well as form to collect various aspects of the end user's quality of experience by using 1-5 star feedback collection mechanism.

The developed system can effectively operate in case the following assumptions are met:

- a monitoring system is established that can help collect network-related metrics between services running in a SDDC and clients that wish to

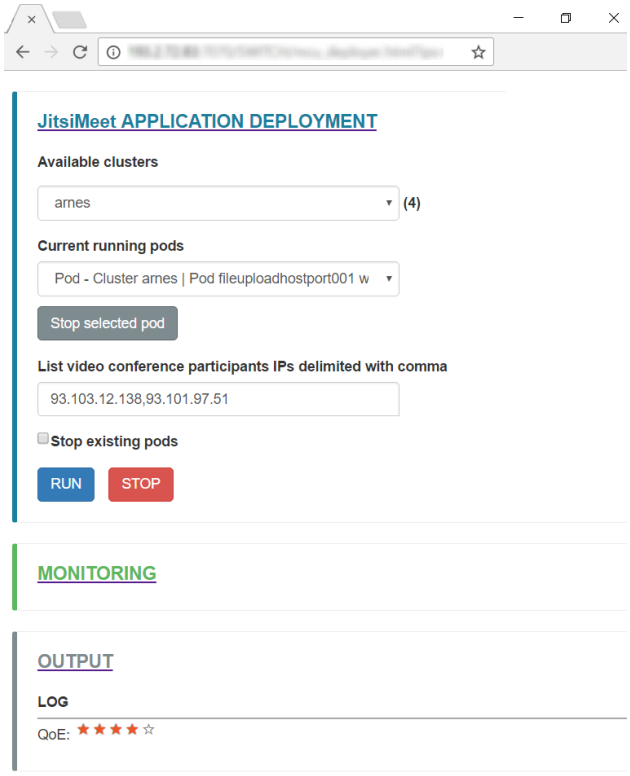


Figure 13: Presentation layer of the system workflow.

use a software utility. Such clients could potentially access the services from anywhere. Once the QoS requirements are taken into account and the service is established, the system cannot cater for late joiners or FW/NAT blocked peers. The only way this problem can be addressed, to the best of our knowledge, is by re-establishing the service,

- an application developer is able to select appropriate QoS metrics that can affect the performance of the developed cloud application. If additional metrics are required, the developer has to implement her own Monitoring Probes for the specific metrics,
- the proposed solution does not provide dynamic adaptation to mitigate network congestions or overloaded servers. Should such unfortunate situation occur, the application has to be restarted,
- due to technological (network overlays) difficulties, the current solution supports multi-component cloud applications only when they are deployed in a single data centre for the particular user. This is only a current technical difficulty and it may be expected that will be improved in near future.

7 Conclusions

According to many studies, the implementation of various compute, memory and network intensive functionalities within cloud applications is still a challenging task. The achievement of adequately high QoS for the application can be affected by various external and internal aspects such as the geolocation of the end users and the type and quantity of computing resources, which are acquired for the application to run. Our work focused on addressing these complex aspects through the development of a new autonomic orchestration architecture and its implementation with a Global Cluster Manager.

The solution relies on multi-level (processor, memory, networking, container and so on) QoS metrics collected in the runtime and a QoS modelling and decision making solution. The QoS and modelling solutions could potentially be further enhanced depending on the modelling technique used. They indicate the type of needed computing resources, their quantity as well as geographic location given important QoS metrics which are measured during the runtime.

Various cloud monitoring systems use hundreds of metrics for their operation. The collection of such metrics may or may not be relevant for a particular software component. Generally, it is relatively difficult to define a single high-level QoS metric (a governing QoE metric), which is correlated with the experience perceived by the end users. The final selection of such a QoE governing metric is left to the discretion of the software engineer. The applied qualitative modelling technique is designed to be generic, so that it can be used for a wide spectrum of software utilities and contribute to achieving higher QoS of diverse applications. The KB component allows for different QoS models to be plugged in the system, so that other strategies (such as other QoS models, low-cost, low-power and similar) can be used.

This new orchestration architecture is primarily developed for event-driven functionalities, such as those triggered by an user who requests a videoconference. Such events are short-lived, for example, they could last from 5 minutes to 1 hour. Nevertheless, the quality of experience for some users (e.g. medical doctors offering telemedical services) is paramount, and the developed architecture may offer obvious benefits to such users.

For the implementation of this new architecture container technologies were used. They allow for much fast orchestration of services, such as starting and stopping services, and fine-grained allocation of computing resources to various containers comprising the cloud application. The Global Cluster Manager and its APIs were developed by using Kubernetes as an advanced Open Source orchestration technology enabling seamless container configu-

ration and independence from the infrastructure providers.

The developed system is primarily intended for Software as a Service (SaaS) providers that would like to implement network-intensive cloud applications. The results presented in this work support the conclusions that an adequate QoS modelling approach coupled with geographic orchestration technology can significantly improve the quality of network-intensive software utilities, perceived by the end users. Moreover, this event-driven autonomic orchestration approach helps spare computing resources since the software components are started and executed in the cloud only when needed. After their use, they are immediately stopped, and the allocated resources are released to be used for another purpose. This allows for much finer grained use of resources.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement no. 643963 (SWITCH project: Software Workbench for Interactive, Time Critical and Highly self-adaptive cloud applications) and under grant agreement no. 644179 (ENTICE project: dEcentralised repositories for traNsparent and efficienT vIrtual maChine opErations). The authors are thankful to the Academic and Research Network of Slovenia (ARNES) for using their public cloud infrastructure.

References

[1] Jarmo J. Ahonen. On qualitative modelling. *AI & SOCIETY*, 8(1):17–28, Mar 1994.

[2] A. I. Avetisyan, R. Campbell, I. Gupta, M. T. Heath, S. Y. Ko, G. R. Ganger, M. A. Kozuch, D. O'Hallaron, M. Kunze, T. T. Kwan, K. Lai, M. Lyons, D. S. Milojicic, H. Y. Lee, Y. C. Soh, N. K. Ming, J. Y. Luke, and H. Namgoong. Open cirrus: A global cloud computing testbed. *Computer*, 43(4):35–43, April 2010.

[3] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba. Policycop: An autonomic qos policy enforcement framework for software defined networks. In *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, pages 1–7, Nov 2013.

[4] Kent Baxley, Jose De la Rosa, and Mark Wenning. Deploying workloads with juju and maas in ubuntu 14.04 lts, May 2014. A Dell Technical White paper.

[5] Gunilla Berndtsson, Mats Folkesson, and Valentin Kulyk. Subjective quality assessment of video conferences and telemeetings. In *Proceedings of the 19th International Packet Video Workshop*, pages 25–30. IEEE, 2012.

[6] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. Fog computing: A platform for internet of things and analytics. In Nik Bessis and Ciprian Dobre, editors, *Big Data and Internet of Things: A Roadmap for Smart Environments*, pages 169–186. Springer International Publishing, Cham, 2014.

[7] Ivan Bratko and Dorian Suc. Learning qualitative models. *AI magazine*, 24(4):107, 2003.

[8] Rajkumar Buyya, Rodrigo N. Calheiros, Jungmin Son, Amir Vahid Dastjerdi, and Young Yoon. Software-defined cloud computing: Architectural elements and open challenges. *CoRR*, abs/1408.6891, 2014.

[9] J. P. Carvalho and J. A. B. Tome. Qualitative modelling of an economic system using rule-based fuzzy cognitive maps. In *2004 IEEE International Conference on Fuzzy Systems (IEEE Cat. No.04CH37542)*, volume 2, pages 659–664 vol.2, July 2004.

[10] Victor Chang and Muthu Ramachandran. Financial modeling and prediction as a service. *Journal of Grid Computing*, 15(2):177–195, Jun 2017.

[11] Betty H. C. Cheng, Kerstin I. Eder, Martin Gogolla, Lars Grunske, Marin Litoiu, Hausi A. Müller, Patrizio Pelliccione, Anna Perini, Nauman A. Qureshi, Bernhard Rumpe, Daniel Schneider, Frank Trollmann, and Norha M. Villegas. *Using Models at Runtime to Address Assurance for Self-Adaptive Systems*, pages 101–136. Springer International Publishing, Cham, 2014.

[12] N.M. Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization. *Comput. Netw.*, 54(5):862–876, April 2010.

[13] DevOps. Devops official web page, December 2016.

[14] Fabric8. Fabric8 Documentation, December 2016.

[15] M. Fiedler, T. Hossfeld, and P. Tran-Gia. A generic quantitative relationship between quality of experience and quality of service. *IEEE Network*, 24(2):36–41, March 2010.

[16] Kenneth D. Forbus. Qualitative modeling. In Frank van Harmelen, Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, chapter 9, pages 361–393. Elsevier Science, 2007.

[17] Sandi Gec, Dragi Kimovski, Uros Pascinski, Radu Prodan, and Vlado Stankovski. Semantic approach for multi-objective optimisation of the entice distributed virtual machine and container images repository. *Concurrency and Computation: Practice and Experience*, pages e4264–n/a, 2017. e4264 cpe.4264.

[18] P. Heidari, Y. Lemieux, and A. Shami. Qos assurance with light virtualization - a survey. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 558–563, Dec 2016.

[19] S. Hoque, M. S. d. Brito, A. Willner, O. Keil, and T. Magedanz. Towards container orchestration in fog computing infrastructures. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 294–299, July 2017.

[20] Markus C. Huebscher and Julie A. McCann. A survey of autonomic computing—degrees, models, and applications. *ACM Comput. Surv.*, 40(3):7:1–7:28, August 2008.

[21] ITU-T. P.1301 : Subjective quality evaluation of audio and audiovisual multiparty telemeetings. Recommendation P.1301, International Telecommunication Union, Geneva, November 2012.

[22] Pooyan Jamshidi, Claus Pahl, and Nabor C. Mendonca. Managing Uncertainty in Autonomic Cloud Elasticity Controllers. *IEEE Cloud Computing*, 3(3):50–60, 2016.

[23] He Jifeng, Xiaoshan Li, and Zhiming Liu. Component-based software engineering. In Dang Van Hung and Martin Wirsing, editors, *Theoretical Aspects of Computing – ICTAC 2005: Second International Colloquium, Hanoi, Vietnam, October 17-21, 2005. Proceedings*, pages 70–95. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[24] Peter Kacsuk, Gabor Kecskemeti, Attila Kertesz, Zsolt Nemeth, József Kovács, and Zoltán Farkas. Infrastructure aware scientific workflows and infrastructure aware workflow managers in science gateways. *Journal of Grid Computing*, 14(4):641–654, Dec 2016.

[25] Dzmitry Kliazovich, Johnatan E. Pecero, Andrei Tchernykh, Pascal Bouvry, Samee U. Khan, and Albert Y. Zomaya. CA-DAG: Modeling Communication-Aware Applications for Scheduling in Cloud Computing. *Journal of Grid Computing*, 14(1):23–39, 2016.

[26] Elena Kornyshova and Rébecca Deneckère. Using an Ontology for Modeling Decision-Making Knowledge. pages 1553—-1562, 2012.

[27] Changbin Liu, Jacobus E. Van Der Merwe, Yun Mao, and Mary F. Fernández. *Cloud resource orchestration: A data-centric approach*, pages 241–248. 2011.

[28] Hua Liu, M. Parashar, and S. Hariri. A component-based programming model for autonomic applications. In *International Conference on Autonomic Computing, 2004. Proceedings.*, pages 10–17, May 2004.

[29] Fabio López-Pires and Benjamín Barán. Many-objective virtual machine placement. *Journal of Grid Computing*, 15(2):161–176, Jun 2017.

[30] Yunsong Lu, Fuli Wang, Mingxing Jia, and Yuanchen Qi. Centrifugal compressor fault diagnosis based on qualitative simulation and thermal parameters. *Mechanical Systems and Signal Processing*, 81:259–273, 2016.

[31] J. Lunze. Qualitative modelling of linear dynamical systems with quantized state measurements. *Automatica*, 30(3):417 – 431, 1994.

[32] C. Pahl and B. Lee. Containers and clusters for edge cloud architectures – a technology review. In *2015 3rd International Conference on Future Internet of Things and Cloud*, pages 379–386, Aug 2015.

[33] D. M. Shila, W. Shen, Y. Cheng, X. Tian, and a. X. S. Shen. Amcloud: Toward a secure autonomic mobile ad hoc cloud computing system. *IEEE Wireless Communications*, 24(2):74–81, April 2017.

[34] Sukhpal Singh and Inderveer Chana. A survey on resource scheduling in cloud computing: Issues and challenges. *Journal of Grid Computing*, 14(2):217–264, Jun 2016.

[35] Software. Autonomous Self-Adaptation Platform, 2017.

[36] Software. Docker Official Web page, 2017.

[37] Software. Jitsi Meet Docker container, 2017.

[38] Software. Kubernetes, 2017.

[39] Software. Netdata, 2017.

[40] Yu Sun, Jules White, Sean Eade, and Douglas C. Schmidt. ROAR: A QoS-oriented modeling framework for automated cloud resource allocation and optimization. *Journal of Systems and Software*, 116:146–161, 2016.

[41] Salman Taherizadeh, Taylor Ian, Andrew Jones, Zhiming Zhao, and Vlado Stankovski. A network edge monitoring approach for real-time

data streaming applications. In *In Proceedings of the 13th International Conference on Economics of Grids, Clouds, Systems and Services (GECON 2016), ACM, Athens, Greece*, 2016a.

[42] Salman Taherizadeh and Vlado Stankovski. Quality of service assurance for internet of things time-critical cloud applications. In *Proceedings of the 6th International Congress on Advanced Applied Informatics (AAI 2017)*, July 2017.

[43] Salman Taherizadeh, Ian Taylor, Andrew Jones, Zhiming Zhao, and Vlado Stankovski. *A Network Edge Monitoring Approach for Real-Time Data Streaming Applications*, pages 293–303. Springer International Publishing, Cham, 2017.

[44] Adel Nadjaran Toosi, Rodrigo N. Calheiros, and Rajkumar Buyya. Interconnected cloud computing environments: Challenges, taxonomy, and survey. *ACM Comput. Surv.*, 47(1):7:1–7:47, May 2014.

[45] Demetris Trihinas, Chrystalla Sofokleous, Nicholas Loulloudes, Athanasios Foudoulis, George Pallis, and Marios D. Dikaiakos. *Managing and Monitoring Elastic Cloud Applications*, pages 523–527. Springer International Publishing, Cham, 2014.

[46] Daniel Vladusic, Boris Kompare, and Ivan Bratko. Modelling lake glumso with q2 learning. *Ecological Modelling*, 191:33–46, 2006.

[47] Junchao Wang, Arie Taal, Paul Martin, Yang Hu, Huan Zhou, Jianmin Pang, Cees de Laat, and Zhiming Zhao. Planning virtual infrastructures for time critical applications with multiple deadline constraints. *Future Generation Computer Systems*, 75:365 – 375, 2017.

[48] Denis Weerasiri, Moshe Chai Barukh, Boualem Benatallah, Quan Z. Sheng, and Rajiv Ranjan. A taxonomy and survey of cloud resource orchestration techniques. *ACM Comput. Surv.*, 50(2):26:1–26:41, May 2017.

[49] Wikipage. Linux Foundation Wiki Web page, 2017.

[50] S. Winkler and P. Mohandas. The evolution of video quality measurement: From psnr to hybrid metrics. *IEEE Transactions on Broadcasting*, 54(3):660–668, Sept 2008.

[51] Pengcheng Xiong, Calton Pu, Xiaoyun Zhu, and Rean Griffith. vperfguard: an automated model-driven framework for application performance diagnosis in consolidated cloud environments. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, pages 271–282. ACM, 2013.

[52] Zhi-Hui Zhan, Xiao-Fang Liu, Yue-Jiao Gong, Jun Zhang, Henry Shu-Hung Chung, and Yun Li. Cloud computing resource scheduling and a survey of its evolutionary approaches. *ACM Comput. Surv.*, 47(4):63:1–63:33, July 2015.

[53] Jure Žabkar, Rahela Žabkar, Daniel Vladušič, Danijel Čemas, Dorian Šuc, and Ivan Bratko. Q^2 Prediction of ozone concentrations. *Ecological Modelling*, 191(1):68 – 82, 2006. Selected Papers from the Fourth International Workshop on Environmental Applications of Machine Learning, September 27 - October 1, 2004, Bled, Slovenia.