Non-Functional Requirements Optimisation for Multi-Tier Cloud Applications: An Early Warning System Case Study

Polona Štefanič
Faculty of Civil and Geodetic Engineering
Faculty of Computer and Information Science
University of Ljubljana
Email: polona.stefanic@fgg.uni-lj.si

Dragi Kimovski
University of Innsbruck
Institute of Computer Science
Email: dragi@dps.uibk.ac.at

George Suciu Jr
BEIA Consult International
Bucharest, Romania
Email: george@beia.ro

Vlado Stankovski
Faculty of Civil and Geodetic Engineering
University of Ljubljana
Email: vlado.stankovski@fgg.uni-lj.si

***Abstract*—Modern software engineering tools, technologies and approaches can radically speed up the development and engineering of multi-tier cloud applications and may be applied to the cloud/edge/fog computing paradigm. The engineering of such cloud applications must take into account Non-Functional Requirements (NFRs), such as end-user, software and cloud infrastructure requirements for low-power computing, performance, availability, elasticity, operational cost and similar. Such requirements should be identified and considered early in the software engineering process and decisions must be taken on their trade-offs, e.g. greater service availability while balancing operational costs. This paper introduces a new multi-criteria decision making approach via the use of the Pareto method to aid the software engineers with NFR trade-off adjustment possibilities. Adjustment possibilities may include geographic distribution of application tiers, horizontal and vertical scaling of virtual resources (such as Virtual Machines and containers) and similar. This work builds on top of technologies and tools developed under the SWITCH and ENTICE projects. The impact of this approach is the ability to fully tailor the adjustments of non-functional properties of the multi-tier cloud application according to the decisions of its engineer. An early warning system cloud application is used to present the approach.**

***Keywords*—Software engineering, Multi-tier cloud applications, Edge/Fog computing, Non-functional requirements.**

I. Introduction

Various cloud computing applications have diverse needs for optimization of their Non-Functional Requirements (NFRs). This diversity is particularly evident with the emergence of smart phones and the Internet of Things (IoT) in general. In some cases, it is necessary to be able to run the applications on the smart phones, while in other cases it is more reasonable to run the services in micro-data centres close to the smart phones or in the actual core data centre, where high frequency processors may be used. All these applications fit in the context of modern cloud/edge/fog computing approaches and are facilitated by new middleware technologies including distributed storage repositories for both data and software, such as Storej.io, orchestration tools, such as Kubernetes, virtualization technologies, such as Docker, and software engineering environments, such as Juju or Fabric8.

Thus, the selection of software technologies in the overall software engineering process plays an increasingly important role and contributes to the design of dependable applications and systems. Dependability is a complex non-functional requirement, which includes, inter alia aspects of availability, resilience, reliability, security etc. Although the (traditional) software engineering domain has been studied in the last decades extensively, it is obvious that the emergence of cloud computing, Big Data, federated and distributed storage repositories, micro-service architectures, orchestration and virtualization tools, cloud design patterns, and other new cloud-based technologies and approaches, may provide new possibilities to address NFRs of applications and to make them radically more adaptive.

Some approaches to address the complete life-cycle of cloud applications, including their NFRs are restricted to the use of dedicated infrastructures or cloud providers (e.g. Amazon EC2, Google Cloud Platform), and various cloud models (e.g. private, public or community cloud) which make it possible to ”personalize” the applications. This, however, is not sufficient, particularly when it comes to the needs to provide highly self-* (=*adaptive, healing etc.) cloud applications based on the use of various (micro-)services. A range of these approaches leads to vendor lock-in, thus, it is necessary to consider life-cycle of cloud applications and their NFRs, which is cloud approach, technology and vendor agnostic. This has lead us to the emergent need to provide an approach for the engineering of NFRs, which relies on a great diversity of infrastructures, virtualization technologies, networking and other solutions.

Cloud applications increasingly rely on multi-tier architectures, which are particularly suitable to address cloud, edge and fog computing concepts. Multi-tier refers to a type of

application that is organized in several tiers that are usually the independent modules and can be reused in other applications. For example, the most common is three-tier architecture, where presentation, logic and data layer can build the entire and fully operational application.

Managing the NFRs of such cloud applications requires detailed understanding of their trade-offs. For example, achieving greater service availability, requires an increase of the operational cost. With multiple NFRs this problem becomes a multi-criteria decision making problem. The goal of the present paper is therefore to design a new approach for requirements engineering, which includes decision making process based on a study of trade-offs among conflicting NFRs for each application tier and create application deployment patterns for the deployment and migration options of the entire cloud application to the federated cloud environment.

The rest of the paper is structured as follows. Section II presents related work. In section III cloud, edge and fog computing paradigms are briefly described. The design phase of a proposed software engineering approach to a solution is introduced in section IV. Section V thoroughly describes the example of an early warning system that suits for fog/edge and cloud computing paradigm and presents results and Pareto front of the first phase in the software engineering process of cloud applications. Finally, section VI concludes the paper and reveals further plans for our future implementation work and research.

II. Related work

When considering the need to optimise NFRs for multi-tier cloud applications it is necessary to compare the present work, with existing scientific works, prototypes, tools, technologies and frameworks. Several important criteria were made in order to prepare such comparison: first, it is necessary to assess modern software engineering Interactive Development Environments, and the extent to which they allow the management of NFRs; second, the level of which it is possible to freely combine functionalities into multi-tier applications; third, it is necessary to assess if such software engineering tools utilise advanced standards, such as the OASIS standard for Topology and Orchestration Specification for Cloud Applications (TOSCA) [1]. In the end of this section a short comparison among the present approach and the studied existing works is presented.

Currently, two leading software engineering tools that are used to create cloud applications and services are Juju [3], [14] and Fabric8 [6]. Juju is an open source universal component-based graphical modeling tool for service oriented architectures and application deployments. It contains a collection of charms (e.g. software assets or components: a knowledge, how to properly deploy and configure selected services in the cloud and relationships among them) that can be graphically managed or written using various programming languages, such as Python and Ruby [3]. Juju allows that application can be vertically scaled by adding or removing charms. It offers also sets (called bundles) of predefined relationships and configurations among charms [14]. Fabric8 is an open source platform that is based on Docker as virtualisation, and Kubernetes as orchestration technology. Fabric8 provides a developer console for creating, building and deploying micro-services and run and manage them with continuous improvement [6]. However, none of the above described software engineering editors consider neither the management of the trade-offs between conflicting NFRs nor they allow user's involvement in the decision making process. Juju packs software assets onto Virtual Machines and LXD containers, while present approach uses Docker containers.

In order to guide developers to the optimal software design and development, a variety of cloud application design patterns have been proposed by traditional software engineering companies [9] as well as by enterprise integrators [11]. Fehling et al. [7] present a catalogue of cloud computing patterns that describe good solutions and practices to address NFRs, including different types of cloud models and provide guidelines how to deploy and migrate cloud applications on the basis of the cloud patterns. The paper [13] presents a catalogue of service-based cloud migration patterns that are focused on the ability to run the applications across multiple clouds.

Furthermore, some bigger cloud providers, such as Amazon [21] and Microsoft [12] offer their own platform-dependent cloud patterns with tutorials for software installation and migration. However, generally speaking, the provided design patterns in more or less all existing works cited above, are only guidelines for developing and deploying cloud applications. The end-users of these applications may all have different QoE requirements at runtime, which may significantly influence the resource consumption and the decisions on elasticity (up or down scaling of the use of resources). In order to address NFR concerns, software engineers may want to organise applications into several tiers, which in turn may all be elastic and scale on specific cloud providers in the cloud/edge environment. Such applications may significantly differ from an application to an application, and this kind of scenarios are not supported by commercial cloud providers. Besides, application patterns provided by cloud providers are usually platform-dependent, and represent significant vendor lock-in that is part of the NFRs of many end-users. In addition to this, cloud application design patterns are developed having in mind only the deployment and migration of the application to the cloud, and elasticity to a limited extent (e.g. horizontal scaling). Other phases of the life-cycle of cloud applications are not addressed, e.g. software design and development or wider range of NFRs.

The following two papers [18], [17] present patterns for migration only of data layers in the cloud. The contribution of [17] is an initial catalog of Cloud Data Patterns, which deal with challenges for the data tier to be realized in the clouds. The authors are mostly focusing on two aspects: how to enable store scalability and ensuring data confidentiality. The later paper [18] introduces cloud data patterns that take into account functional, non-functional and privacy-related aspects for the migration of the data layer to the cloud. However, the authors provide the initial catalogues of cloud data patterns only for the data tier, the other two tiers, e.g. the presentation and the logic tier, are not considered. Moreover, no (data) tier-related NFRs management is available, nor any user's involvement in the decision making process.

Current state-of-the-art approaches for orchestration of micro-services use and address new developed topologies for

orchestration, automation and application provisioning and management [8], [10], [20]. However, they do not handle NFRs, but explain that TOSCA should be extended to allow handling and defining NFRs using TOSCA policies and steer topology completion and similar [8]. [20] demonstrates an approach to include NFRs in TOSCA using policies and provides a mechanism for automatic processing of the policies in a TOSCA compliant runtime environment. However, these approaches neither handle NFRs nor provide the software engineer with ability to study the trade-offs between conflicting NFRs but explain that TOSCA should be extended to allow handling and defining NFRs using TOSCA policies and steer topology completion and similar.

III. Background

The emergence of the Internet of Things (IoT) and its intended applications opens a whole new range of Big Data and real-time computing problems. Software engineering intends to follow this progress by radically shortening the life-cycle of cloud applications through component-based software development and engineering. The underlying computing platforms and technologies are designed to further support cloud, edge and fog computing concepts. Multi-tier application patterns seem to be predominant approach for delivering applications that can run both at core data centres and simultaneously at the edge/fog devices. Currently, in this context, there does not exist a systematic approach for the management of NFRs, and this study intends to address this gap. Following is an account of the key design principles underpinning an approach to incorporate NFRs management in the overall life-cycle of multi-tier cloud applications.

Edge, fog, osmotic [19] and other new computing concepts have been emerging recently in order to address the requirements of a plethora of new applications for smart phones, cars, homes, and so on. For such applications decentralized computing infrastructures can be advantageous. The general idea of cloud/edge/fog computing is to extend the traditional cloud computing model to the network edge. There exist virtualized platforms (e.g. micro-data centres) that are located between the centralized core data centres and the end users' devices and offer strong support for IoT-based applications.

Cloud, edge and fog computing platforms and approaches generally allow to move the processing, storage and communication capabilities from actual core data centres towards the edge of the network and run services in micro-data centres that are closer to the end users. That is, the main idea of cloud/edge/fog computing paradigm is to properly push/move the computing resources, application services and data away from centralized nodes to the edge of the network and vice-versa in order to achieve higher efficiency [4]. When application's services run at the edge of the network, e.g. in micro-data centres closer to the end users: (1) lower amount of data is transported that normally would be processed and analysed in the core data centres, which means for example lower data transportation costs, lower latency and delays, faster data (e.g. virtual machine images, containers) delivery and communication with the end users, specific geographic availability of the services i.e. location awareness of services and so on [16].

Nowadays, new container-related technologies make it more straightforward to realize the cloud/edge/fog computing concepts. Various companies enable edge devices, such as routers to perform operations with containers, such as delivery, storage, deployment and so on. Due to the nature of the technologies used, the movement, starting, stopping and other operations with services and networks are much faster and efficient. This makes it possible to address NFRs, such as latency, bandwidth, availability, reliability, security, operational cost, and so on, in a much more efficient and effective manner. The goal of the present study is to inform the software engineering process about the potential offered by the underlying cloud technologies so that essential NFRs are taken into account early in the cloud application engineering process i.e. in editors such as Juju.

IV. Software engineering: multi-tier cloud applications and their NFRs

Multi-tier application architecture is a type of architecture, where application layers run on various server instances or on several virtualized server instances somewhere between the core data centres and the end user. A most commonly used multi-tier architecture is a three-tier architecture, which usually consists of presentation, logic and database tiers.

The presentation tier (also called layer) is the highest level of the application and it usually presents the user interface. Its main function is to render the tasks to the end user that he/she can understand them. Users access this layer directly, e.g. via Web browser on the client-side. The logic tier controls the application's functionality and processes data between other tiers. This tier communicates with the other two tiers (presentation and data tier). The data tier presents the storage, e.g. database or file, where the information can be stored and retrieved from. The back-end logic layer serves both the client application (user interface) and the data storage. The presentation layer (user interface) does not have a direct access to the data storage.

Using a three-tier architecture allows flexible scaling and modifying one of the tiers without affecting the other two tiers. In contrast, a two-tier architecture is an application that executes a back-end logic and data on the same server, and the presentation tier is on the client side. The deployment of cloud/edge/fog applications (e.g. IoT) in a two-tiered architecture already fails short of meeting the requirements related to low latency, mobility, location awareness and so on. Two-tier architecture is not really applicable to cloud/edge/fog computing concepts.

Multi-tier architectures are designed to address the needs of these new computing concepts, and therefore inherit the benefits of the underlying technologies. For example, they can support the replication of data and services, self-* (*=healing, adaptation, configuration, maintenance etc.) properties and so on, which immediately affects the perceived Quality of Experience (QoE) and Quality of Service (QoS).

In conclusion, a multi-tier cloud application architecture can be suitable to address a range of new NFRs across cloud, edge and fog infrastructures. In this study, we focus on a three-tier application architecture as being sufficient to address the objectives of our study.

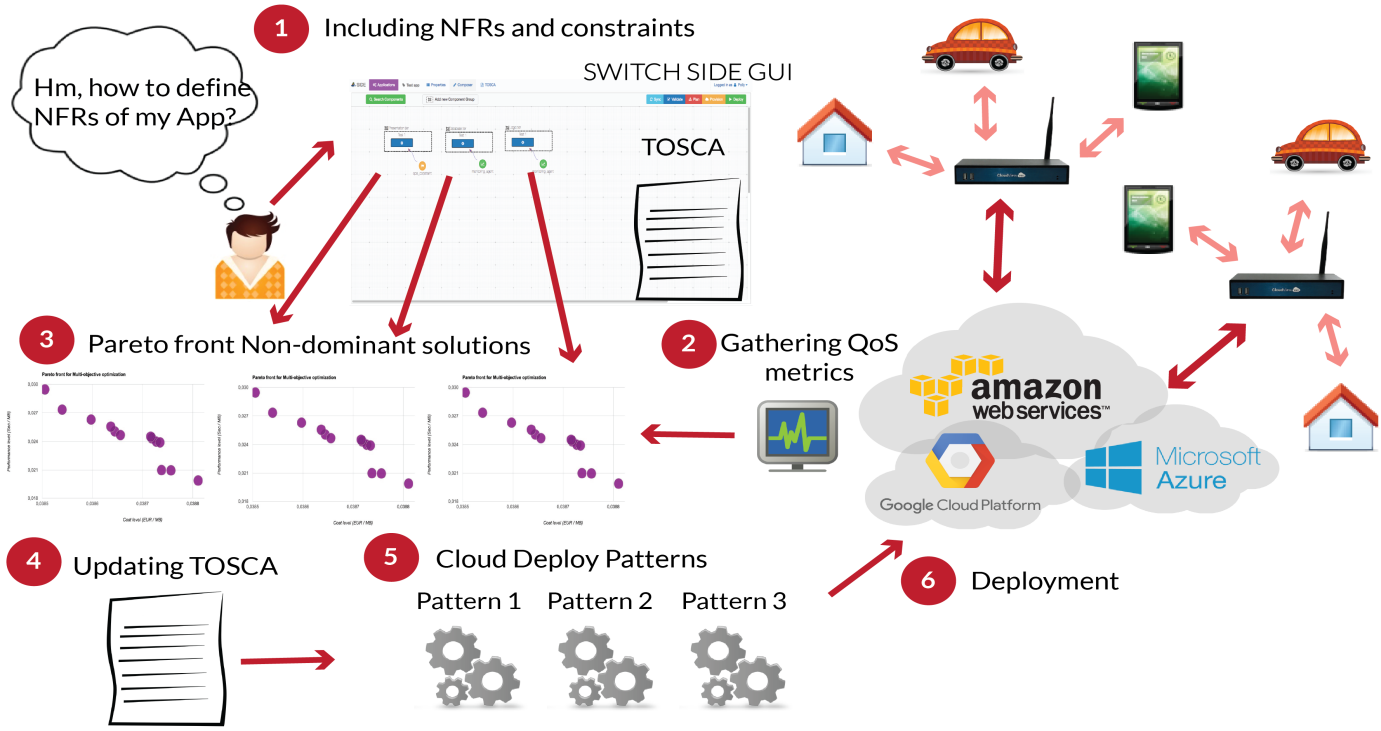

Fig. 1: Software development of multi-tier cloud applications on top of SWITCH SIDE. Phase 1 presents including NFRs and constraints into the development process; in phase 2: QoS metrics are gathered; phase 3 offers to the engineer Pareto front for each application component; in a phase 4 TOSCA file is updated according to the engineer's decisions; in phase 5 cloud design patterns will be generated and according to the engineer's decision making multi-tier application will be deployed to the cloud/edge/fog environment in phase 6.

A. *NFRs management in the software engineering process*

A major novelty of this work, is to provide a process that allows the software engineer to study the NFR trade-off possibilities for each application tier during the development of a multi-tier cloud application. The approach presented in this paper builds on the top of a new software engineering editor of project SWITCH[1], which is a graphical user interface developed in the course of the running project SWITCH. This new graphical editor allows the software engineer to consider various NFRs (e.g. thresholds on Quality of Service (QoS) metrics, allowed physical locations where to run the services, and restrictions to such locations, and other). The SWITCH SIDE (software interactive development environment) relies on the TOSCA standard, which makes it possible to integrate with existing orchestrators capable of using TOSCA with ability to take into account NFRs early in the software engineering process, by creating trade-off decisions about NFRs by further automatically proposing cloud design patterns suitable for the cloud/edge/fog computing paradigm. Finally, these functionalities are packed in Docker containers and can be deployed and orchestrated across multiple clouds (e.g. data centres, micro-data centres, IoT devices). Figure 1 shows the steps in the software development of multi-tier cloud application on top of the SWITCH SIDE. The process is managed by a software engineer who is the decision maker.

The first three steps of the process represent the application design phase and the last three steps (out of a total of six steps) represent the deployment phase for the application. In the following, we briefly describe all steps of our newly designed process. Particular focus is given on the first three steps that highlight the novelty of the process.

1) Designing and including NFRs and constraints: A software engineer can create services and application components in the SWITCH SIDE GUI by gathering application components from public or private repositories (e.g. Docker Hub) and by creating them on the SIDE canvas. In many situations several useful components may be retrieved, and the engineer has to decide which one to use. The SWITCH SIDE the overall application editing process is TOSCA compliant. A crucial step in the process is that the software engineer can consider which NFRs should be defined for each application component (application tier) or service separately, e.g. what is the minimum CPU frequency required for the component to run, what is the minimum amount of Memory that has to be allocated to that particular component so that the specific application tier will run properly. Obviously, not all NFRs of the application components of the multi-tier cloud application share the same grade of the importance to work optimally (e.g. high security for higher cost is probably more important for the data tier and not that much important for the presentation tier). The user can include these constraints in an updated version of the TOSCA application specification.

For illustration, the use case, which is presented later on

[1] www.switch-project.eu

in this paper (in section V) has the following NFR constraints: jitter should be less than 1 millisecond, querying the database should be less than 2 milliseconds and so on.

2) SLAs of cloud providers and monitoring data: Since we are dealing with cloud computing, Service Level Agreements (SLAs) of the cloud providers should be considered, such as availability, cost and performance. SLAs of individual cloud providers are publicly available documents. Some cloud providers expose their SLAs via APIs, and can be used for various calculations and cost estimation. In addition to the SLAs, some applications may require real-time monitoring of the provided Quality of Service (QoS). Monitoring the entire cloud, edge and fog computing spectrum is a work in progress (see V-B).

When considering monitoring, various QoS metrics can already be measured by using systems, such as JCatascopia and the SWITCH agent-based Monitoring System. Metrics that can already be collected include application level metrics (availability, cost, performance), network level metrics (e.g. throughput, latency, response time etc.) and container-based metrics (CPU, memory etc.). When performing our tests, we have considered performance of the databases, throughput, latency and response time of connecting clients to the databases and running server tests in the clouds (see subsection V-B).

3) Application tier-based Pareto decision making: In this step, the monitoring data, SLA information of cloud providers and NFRs (e.g. application tier constrains) are defined in TOSCA, and sent to the multi-criteria decision module in order to offer the software engineer the optimal trade-offs of the NFRs for every application tier of the multi-tier application. At this point, the decision maker (software engineer) chooses the optimal solution on the Pareto front, which represents a NFRs trade-off solution in accordance with the given objective functions (see V-C). The concept of Pareto optimality has been utilized to simplify the decision making process and to provide guidance to the software engineer how to efficiently deploy their application.

4) Updating TOSCA file: In the next step of the process, the decision maker (i.e. software engineer) selects a completely independent non-dominant solutions for each application component. The changes of his/her choices are updated in the TOSCA application specification and are made ready to be processed in the next steps.

5) Creating deployment patterns: The last step before deployment is achieved by creating deployment patterns that will provide the software engineer a set of few optimal deployment options. For example, the engineer may select on which nodes – cloud/edge – the application tiers should be deployed in order to meet the desired NFRs. In this step the previous decision maker's choices will be considered when creating deployment patterns for optimal deployment to the cloud infrastructure.

6) Deploy to the cloud infrastructure: In the last step of this process the software engineer chooses the application pattern that suits the most his/her multi-tier application and deploys the application on the selected cloud/edge computing infrastructure.

V. USE CASE

A multi-tier cloud application was designed and developed in order to illustrate the NFR optimisation approach. The application represents an early warning system, and was designed in collaboration with the company BEIA Consult[2]. The main idea of an early warning systems is in case of natural disasters, such a flood, earthquakes and similar disaster events to save lives – warn people for example to leave homes, hospitals to be prepared to receive more patients, a fire department or army to be prepared for rescuing people and to enable the authorities to prepare and provide any help – and to minimize costs when preventing efficiency. An early warning system often collects data from sensors in real time, processes the gathered information using various predictive simulation tools on the warning system and provides data to the warning services or interactive facilities for the public to obtain more information.

A. An Elastic Disaster Early Warning System Case Study

BEIA Consult elastic disaster warning system application represents "cloudified" solution for natural disasters. The application should be capable of collecting and processing sensor in nearly real-time and thus allowing very rapid response to urgent events. In addition to that application should provide sufficient reliability, availability and scalability to the increasing numbers of sensors. The main QoE metric of the application that should be met is the system's reaction time i.e. the time from sensor data acquisition to the notification sent to the operator. The design of this cloud application is presented in Figure 2.

Sensors transmit the field data that is collected by Remote Telemetry Unit (RTU) and forwarded to IP Gateway that transmits the data to the database server. Here the reports/statistics are created. The notification server periodically checks the database server and statistics in real time and sends notifications to operators if the threshold is violated. Notifications are sent to the operator that processes the event. The operator checks statistics received and makes a decision whether or not to alert the emergency system.

This is a multi-tier cloud application, where two relational data bases (database and notification server), representing two tiers can be horizontally scaled since the amount of data can start growing quickly. Therefore, we have done some measurements using MySQL and SQLite databases and deploy them to Amazon EC2 in order to perform benchmark tests. The results are collected in V-B and presented on Pareto front in V-C.

B. Gathering QoS metrics for the modelling process

Tests were run using Amazon Web Services (AWS) Elastic Compute Cloud (EC2), since Amazon is a major cloud provider. Amazon EC2 is hosted on various regions in a separate geographic areas world-wide. Currently, Amazon offers 14 regions on various continents. For the purpose of the evaluation we have set instances and run tests on four different regions: North Virginia, USA (us-east-1), London, Europe (eu-west-2), Sydney, Australia (ap-southeast-2) and São Paulo, South America (sa-east-1). In each region we have created three

[2]http://www.beiaro.eu/

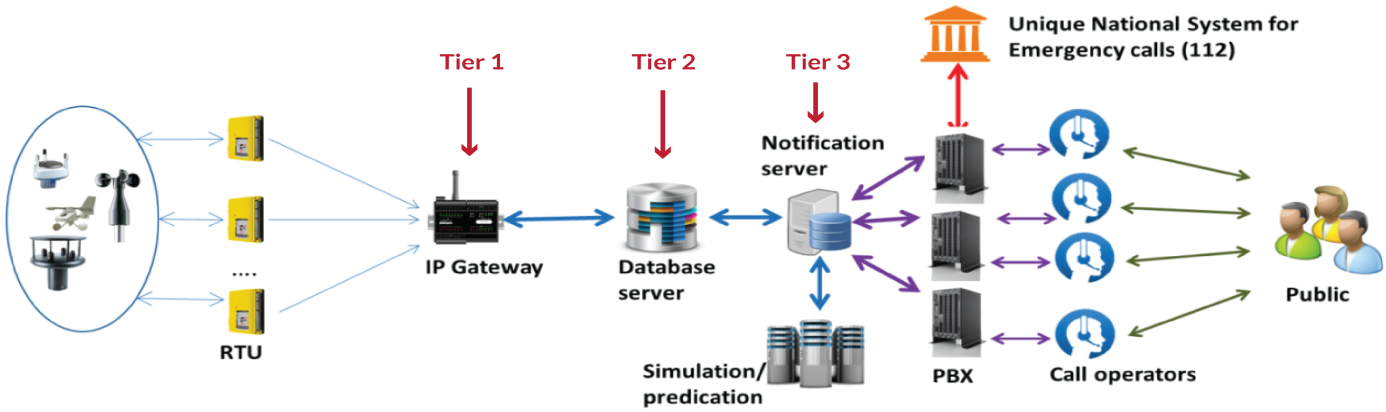

Fig. 2: BEIA elastic disaster early warning system.

different virtual machines: t2.micro[3], m4.large[4] and i3.large[5]. In Table I the basic properties[6] of the above mentioned VMs are described.

TABLE I: AWS E2 Instances and their basic properties.

	vCPU [cores]	ECU [CPU unit]	RAM [GB]
t2.micro	1	variable	1
m4.large	2	6.5	8
i3.large	2	7	15,25

On each VM instance we have deployed two relational databases: MySQL and SQLite. For each database we have run several server and client tests. As a server tests we have run the following functionalities: (1) deleting database, (2) creating database, (3) creating tables, (4) populating tables; We have populated tables with various number of records, whereby we have built a tree. For each node we have stored the information about all its parents up to the root. As mentioned above, for each database we have run more tests every time with different number of records in the database, e.g. 13, 85, 781 and 9331. We defined the number of records according to the following formula $No.\ of\ Records = \frac{N^{L-1}}{N-1}$, where N represents the number of childs each node (parent) has and L represents the level (depth) of a tree. We decided to populate the tables with data in an indexed tree, because it is greedy operation over databases.

We have also created clients, who connect to all the VMs on several locations and perform actions, such as download the biggest database (with 9331 records). We measured how long the data transfer takes (transfer rate metric: throughput). The results have been summarized with those gathered when measuring the creation and insertion of the database with different amount of records. This data represents our performance that is measured in milliseconds and the cost is presented in USD/h. The measured results can be found in Tables II and III.

[3]t2 instances are low-cost and for general purpose and provide a baseline level of CPU with the ability to burst above the baseline.

[4]m4 instances are for general purpose, with a balance of compute, memory, and network resources.

[5]i3 instances are optimized on storage.

[6]https://aws.amazon.com/ec2/pricing/on-demand/

TABLE II: The data for performance and cost for SQLite database.

VM	Instance	Performance [ms]	Cost $[USD/h]$
us-east-1	t2.micro	327.9364671	0,012
	i3.large	286.565312	0,156
	m4.large	313.203688	0,108
eu-west-2	t2.micro	299.335739	0,014
	i3.large	303.966593	0,181
	m4.large	317.950037	0,125
ap-southeast-2	t2.micro	332.465423	0,016
	i3.large	318.667166	0,187
	m4.large	307.410629	0,134
sa-east-1	t2.micro	361.49492	0,02
	i3.large	333.903969	0,286
	m4.large	328.825433	0,171

TABLE III: The data for performance and cost for MySQL database.

VM	Instance	Performance [ms]	Cost $[USD/h]$
us-east-1	t2.micro	290.032196	0,012
	i3.large	259.175117	0,156
	m4.large	296.150911	0,108
eu-west-2	t2.micro	276.665274	0,014
	i3.large	267.977918	0,181
	m4.large	262.342901	0,125
ap-southeast-2	t2.micro	301.305107	0,016
	i3.large	291.266964	0,187
	m4.large	283.23083	0,134
sa-east-1	t2.micro	295.705113	0,02
	i3.large	310.696244	0,286
	m4.large	301.424925	0,171

C. Multi-criteria decision making

The process of cloud applications engineering requires consideration of multiple different NFRs. Those requirements are mutually conflicting, thus altering one parameter can have profound effects on the others. For example, increasing the availability of a given application will require increased system redundancy, which will eventually lead to higher operational costs for the application owner. Contrarily, reducing the operational cost can easily limit the possibility for renting more

computational resources in the cloud, causing high VM deployment, instantiation and execution overheads. Selecting the most optimal trade-offs between multiple application run-time parameters can be time-consuming and error-prone process, especially when considering complex cloud environments.

In order to assist the software engineers to efficiently deploy and operate multi-tier applications in cloud environment we have introduced essential models from the field of multiple-criteria decision making. To be more concrete, we have resorted to the concepts of Domination and Pareto optimality to develop a tool which will simplify the application deployment process in the cloud by consider multiple conflicting non-functional parameters.

The concept of domination has been used in the field of multi-criteria decision making to compare multiple solutions on the basis of two or more conflicting objective functions [15]. For a solution $a \in A$ *is said to dominate over* $a' \in A$ if a' is greater than a in relation to all objective functions, while a' has worse value for at least one of them. Furthermore, a solution $a' \in A$ is *non-dominated* if there are no other solutions $a \in A$ that dominate over a'. A set of solutions $P' \in P$ is called *Pareto optimal* set if there are no other solutions in P that dominate any solution in P'. The set of all Pareto optimal solutions is called *Pareto optimal set*, or simply *Pareto front*. The Pareto front can be considered as a efficient tool for aiding the decision making process, especially in complex environments. The shape and spread of the Pareto front can provide insights that enable efficient exploration of the space of non-dominated solutions, thus allowing certain properties and regions of particular interest to be easily explored.

For our particular problem of Pareto front construction we have utilized the JMetal Multi-objective optimization framework [5]. The Pareto front is constructed on the basis of the QoS metrics and benchmark tests described in Section V-B. The algorithm uses the QoS metrics data to properly sort the available VM instances in relation to the instance's performance and cost as conflicting objectives. The tools has been created in a generic way, thus allowing the dominance sort to be extended to more than two objective functions. The process of Pareto construction is initiated by marking all available instances that meet the minimal objectives' criteria required by the user. Afterwards, the pre-selected instances are compared pair-wise and the actual non-domination sorting is performed. This results in a construction of the Pareto front, which encloses the optimal solutions in relation to both conflicting objectives. Therefore, the tool reduces the number of total available VM instances to only those ones that provide the most optimal balance between cost and performance.

To illustrate the concepts presented in this section we have evaluated the multi-criteria decision making tool with the data provided in Tables II and III. Figures 3 and 4 show the Pareto optimal sets constructed for the SQLite and MySQL VM instances respectively. It can be observed that the proposed tool can easy the burden on the developers and simplify the decision-making process by removing VM instance types which are not optimal in the given objective space. In our experiments we have reduced the number of possible VM instance types from 12 to only few. Nevertheless, this tool would be most suitable in the cases in which the number of possible instance types and locations is in the range of hundreds. Furthermore, this tool allows for graphical representation of the objective space and the Pareto front, thus providing easy to use interface. On the vertical axis of the Pareto front we have set cost (USD/h) of each VM instance and on the horizontal axis the performance (milliseconds), which is the sum of running server tests (e.g. time needed for inserting, creating, populating databases and so on) and client tests (throughput).

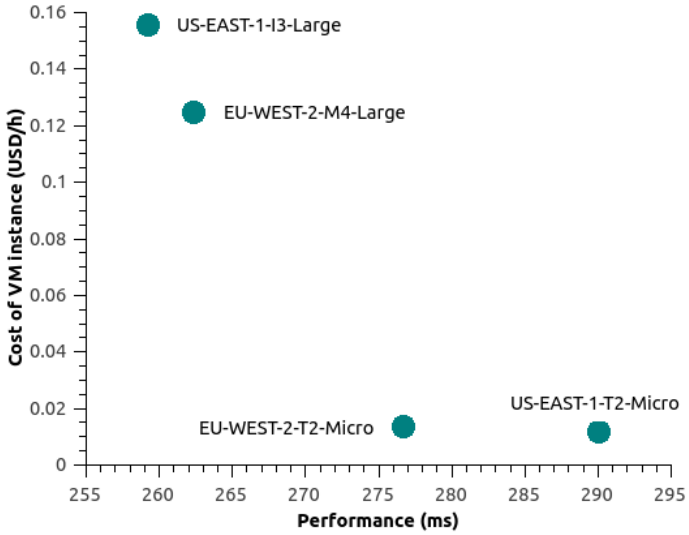

Fig. 3: Pareto set for the SQLite VM instances.

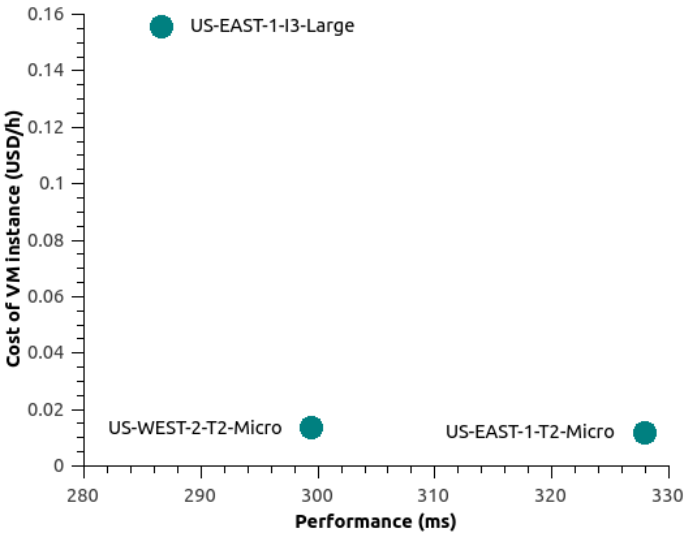

Fig. 4: Pareto set for the MySQL VM instances.

VI. Conclusion

New software engineering methods and practices, such as component-based software engineering may radically improve the life-cycle of cloud applications. Our work focuses on a very important aspect of the engineering process: the management and optimisation of Non-Functional Requirements in multi-tier cloud applications.

This work presents a new approach that raises the level of abstraction, so that NFRs can be managed and optimised very early in the application engineering process. As a proof of concept, the process is included in the SWITCH SIDE GUI, but, it can be used in other software engineering IDE, such as Juju. The key novelty is in the use of available knowledge about the software artifacts (components, such as containers, Virtual Machines, etc.), the SLAs of the cloud providers, and the QoS monitoring metrics, so that the software engineer is provided with a Pareto front to study her/his optimal trade-off solution. This process can apply to all tiers in particular, and to the application as a whole. Once the NFRs are optimised and a decision is taken, the NFRs are recorded in a semantic description of the cloud application, which is a proposed extension to the OASIS TOSCA standard.

Compared to other existing works discussed in section II the novelty of our work is an approach, that offers software engineers to study, optimise, and manage, NFR trade-offs possibilities in a component-based software Interactive Development Environment.

While the first three steps (1-3) of the process have been fully completed and are described in this study, the next three steps steps (4-6) have complexities, e.g. due to the possibility to horizontally scale the application components at each application tier, which need to be addressed by a separate study (see IV-A). The focus of the new study will be the overall optimal deployment, including scaling, migration and other run-time adaptation possibilities for optimal operation of the multi-tier cloud application. This will be achieved by creating pre-prepared patterns for the software engineer to study and choose based on the trade-offs. For example, the SIDE GUI could offer the user optimal combination of application tiers, some of which should be deployed in cloud core data centres and some tiers to be moved towards the edge of the network.

Generally speaking, our future works will include analysis on how to include NFR management principles to achieve self-adaptive behaviour of applications, infrastructure planning, middleware for the optimisation and delivery of software components (e.g. ENTICE delivery mechanisms for Virtual Machines and their fragments).

$$Noofrecords = \frac{N}{N-1}$$

ACKNOWLEDGMENT

The research work of this paper bases on two projects, which have received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 643963 (SWITCH project) and under grant agreement No 644179 (ENTICE project).

REFERENCES

[1] Oasis topology and orchestration specification for cloud applications version 1.0, November 2013.

[2] A. A. Adewojo, J. M. Bass, and I. K. Allison. Enhanced cloud patterns: A case study of multi-tenancy patterns. pages 53–58, Nov 2015.

[3] 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.

[4] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[5] Nebro A. J. Durillo, J. J. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771, 2011.

[6] Fabric8. Fabric8 documentation, December 2016.

[7] Christoph Fehling, Frank Leymann, Ralph Retter, Walter Schupeck, and Peter Arbitter. *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer Publishing Company, Incorporated, 2014.

[8] Nicolas Ferry, Alessandro Rossini, Franck Chauvel, Brice Morin, and Arnor Solberg. Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems. In *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing*, pages 887–894, Washington, DC, USA, 2013. IEEE Computer Society.

[9] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 edition, 1994.

[10] Pascal Hirmer, Uwe Breitenbücher, Tobias Binz, and Frank Leymann. Automatic Topology Completion of TOSCA-based Cloud Applications. In *Proceedings des CloudCycle14 Workshops auf der 44. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, volume 232, pages 247–258, Bonn, September 2014. Gesellschaft für Informatik e.V. (GI).

[11] G. Hohpe and B.A. WOOLF. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. The Addison-Wesley Signature Series. Prentice Hall, 2004.

[12] A. Homer, J. Sharp, L. Brader, M. Narumoto, and T. Swanson. *Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications*. Patterns & practices. Microsoft Developer Guidance, 2014.

[13] Pooyan Jamshidi, Claus Pahl, Samuel Chinenyeze, and Xiaodong Liu. Cloud migration patterns: A multi-cloud service architecture perspective. In *Service-Oriented Computing - ICSOC 2014 Workshops - WESOA; SeMaPS, RMSOC, KASA, ISC, FOR-MOVES, CCSA and Satellite Events, Paris, France, November 3-6, 2014, Revised Selected Papers*, pages 6–19, 2014.

[14] Ubuntu Juju. Juju docs, December 2016.

[15] D. Kimovski, N. Saurabh, V. Stankovski, and R. Prodan. Multi-objective middleware for distributed vmi repositories in federated cloud environment. *Scalable Computing: Practice and Experience*, 17(4):299–312, 2016.

[16] I. Stojmenovic and S. Wen. The fog computing paradigm: Scenarios and security issues. In *2014 Federated Conference on Computer Science and Information Systems*, pages 1–8, Sept 2014.

[17] S. Strauch, V. Andrikopoulos, U. Breitenbuecher, O. Kopp, and F. Leyrnann. Non-functional data layer patterns for cloud applications. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pages 601–605, Dec 2012.

[18] Steve Strauch, Vasilios Andrikopoulos, Uwe Breitenbücher, Santiago Gómez Sáez, Oliver Kopp, and Frank Leymann. Using Patterns to Move the Application Data Layer to the Cloud. In *Proceedings of the 5th International Conference on Pervasive Patterns and Applications (PATTERNS'13)*, pages 1–8. Xpert Publishing Services, May 2013.

[19] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan. Osmotic computing: A new paradigm for edge/cloud integration. *IEEE Cloud Computing*, 3(6):76–83, Nov 2016.

[20] Tim Waizenegger, Matthias Wieland, Tobias Binz, Uwe Breitenbücher, Florian Haupt, Oliver Kopp, Frank Leymann, Bernhard Mitschang, Alexander Nowak, and Sebastian Wagner. *Policy4TOSCA: A Policy-Aware Cloud Service Provisioning Approach to Enable Secure Cloud Computing*, pages 360–376. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[21] Marcus Young. *Implementing Cloud Design Patterns for AWS*. Packt Publishing, 2015.