Developing and operating time critical applications in clouds: the state of the art and the SWITCH approach

Zhao, Z.; Martin, P.; Wang, J.; Taal, A.; Jones, A.; Taylor, I.; Stankovski, V.; Garcia Vega, I.; Suciu, G.; Ulisses, A.; de Laat, C.

Published in:
Procedia Computer Science

DOI:
10.1016/j.procs.2015.09.220

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 68 (2015) 17 – 28

Procedia Computer Science

HOLACONF - Cloud Forward: From Distributed to Complete Computing

Developing and operating time critical applications in clouds: the state of the art and the SWITCH approach

Zhiming Zhao[a*], Paul Martin[a], Junchao Wang[a], Ari Taal[a], Andrew Jones[b], Ian Taylor[b], Vlado Stankovski[c], Ignacio Garcia Vega[d], George Suciu[e], Alexandre Ulisses[f], Cees de Laat[a]

[a]*University of Amsterdam, Science Park 904, Amsterdam, 1098XH, the Netherlands *contact: z.zhao@uva.nl*
[b]*Cardiff University, Queen's Buildings, 5 The Parade, Cardiff CF24 3AA, Unite Kindom*
[c]*University of Ljubljani, Slovenia*
[d]*Wellness Telecom SL, Spain*
[e]*BEIA Consult International SRL, Romania*
[f]*MOG Technologies SA, Portugal*

Abstract

Cloud environments can provide virtualized, elastic, controllable and high quality on-demand services for supporting complex distributed applications. However, the engineering methods and software tools used for developing, deploying and executing classical time critical applications do not, as yet, account for the programmability and controllability provided by clouds, and so time critical applications cannot yet benefit from the full potential of cloud technology. This paper reviews the state of the art of technologies involved in developing time critical cloud applications, and presents the approach of a recently funded EU H2020 project: the Software Workbench for Interactive, Time Critical and Highly self-adaptive cloud applications (SWITCH). SWITCH aims to improve the existing development and execution model of time critical applications by introducing a novel conceptual model—the application-infrastructure co-programming and control model—in which application QoS and QoE, together with the programmability and controllability of cloud environments, is included in the complete application lifecycle.

Peer-review under responsibility of Institute of Communication and Computer Systems.

Keywords: Time critical applications, Cloud, quality of user experience, infrastructure programming, self-adaptable system

1. Introduction

Many time critical applications (applications that must respond immediately to time-sensitive events over a prolonged period) often have very high business value (e.g. on-demand business collaboration platforms) or social impact (e.g. disaster early warning systems). These applications demand a high standard of Quality of Service (QoS)

Peer-review under responsibility of Institute of Communication and Computer Systems.
doi:10.1016/j.procs.2015.09.220

(e.g. tsunami emergency response time) or quality of experience (QoE) (e.g. smooth delivery of ultra-high definition audio and video for live events), but are very difficult to develop and operate because of their distributed nature and the high requirements they impose on the runtime environment—in particular the sophisticated optimization mechanisms needed to develop and integrate system components that must interact seamlessly with one another.

Cloud environments are capable of providing virtualized, elastic, controllable and high quality on-demand services for supporting these kinds of complex distributed application. Indeed, many cloud providers already provide many of the technologies needed to develop and deploy these applications. However, what time-critical applications still need from the cloud is the ability to control the selection and configuration of infrastructural components in response to changing requirements and environmental pressures. Unfortunately current Cloud environments lack the tools and application programming interfaces that would allow the developers to exert such control on the underlying infrastructure in an intelligent, semi-autonomous manner.

This paper reviews the state of the art of the technologies involved in developing time critical cloud applications, and present the approach in a recently funded EU H2020 project: the Software Workbench for Interactive, Time-Critical and Highly self-adaptive cloud applications (SWITCH). First we analyse the requirements for time critical cloud applications via three use cases, and then review the state of the art of supporting technologies for such applications. Afterwards we discuss the design of a software workbench to be developed in SWITCH.

2. Requirements and State of the art

The development and operation of time critical applications in clouds are both very difficult, because they have such high requirements for system performance and service quality. In this section, we first use some examples to discuss the general requirements for developing, deploying and operating time critical applications in clouds, and then, after reviewing the technological state of the art, identify the key technical gaps that must be bridged.

2.1. Time critical applications and requirements

The development of time critical applications faces multiple challenges, which can be seen in several use cases, a few of which are described below.

Use case 1—A collaborative real-time business communication platform. Real-time communication plays an increasingly important role in many business applications, whether for videoconferencing, establishing a cooperative working environment for geographically dispersed colleagues, or performing remote diagnosis. However, renting very high bandwidth or private connection links is not affordable for many business users. The Web Real-Time-Communications (WebRTC)[*] project enables real-time communications directly in a web browser, but is limited when handling many-to-many situations—as the number of peers increases, so do the resources needed. When using a mixing architecture, this resource cost is concentrated on the server-side where all the video and audio data mixing is done. An ideal real-time business communication platform should be 1) very scalable with the number of peers, 2) very high quality in regard to its communication service, and 3) competitive in cost.

Use case 2—The elastic disaster early warning system. Early warning for natural disasters is an important challenge for many countries. An early warning system often collects data from real-time sensors, processes the information using tools such as predictive simulation, and provides warning services or interactive facilities to allow the public to obtain more information. The implementation of this kind of system faces several challenges, as the system must: 1) collect and process the sensor data in nearly real time, 2) detect and respond to urgent events very rapidly, 3) predict the potential increase of load on the warning system when users increase, 4) operate reliably and robustly throughout its lifetime, and 5) be scalable with greater deployment of sensors.

Use case 3—A cloud studio for directing and broadcasting live events. In a live event, like a football match or a music concert, the broadcaster or production company has to deploy a large number of personnel and many items of equipment to fully cover it. Multiple cameras are placed around the event venue to cover all the different angles that the director considers relevant. The cameras are then connected to an OB (outdoor broadcast) van through

[*] Web Real-Time Communication, http://www.webrtc.org/

dedicated links. The OB van contains equipment used by an operator to control and mix the signal before sending it to the head office. Each camera is usually associated with an SDI (Serial Digital Interface) link that records the content at 1,485 Gbits/s (HD-SDI 1080i and 720p) and generally more than 3 cameras are present at a typical event. Furthermore, other cameras are used for interviews with the players and the bands, each one with similar characteristics to the ones recording the actual event. By virtualising the basic components that an OB van may have (typically the video switches) on the Cloud, directors and producers could then interact with different video sources remotely via the cloud environment instead of physically staying in the OB van. In order to realise this scenario however, not only does the network for delivering different video streams need to be of high quality, but also the connectivity between streams needs to be fully reconfigurable, based on different broadcast program scenarios. Due to the malleable nature of clouds, the network also has to be continually monitored and changes in state immediately responded to. In the meantime, the video material should also be archived and streamed to users who want to watch it afterwards.

Several requirements can be enumerated from the above scenarios:

1) **System-level performance requirements:** there are constraints on quality of service that must be satisfied to deliver acceptable performance, regarding for instance latency in real-time collaborative activities in use case 1, emergency response time in use case 2, and delivery quality of high definition television in use case 3.
2) **Verifiability:** the programmer must find a way to prove that the application will actually achieve the desired QoS. For example, in use case 1, the load of the potential business activities and the number of real-time data switching services to be deployed in the cloud should be verified before activities are delivered to customers.
3) **Integration complexity:** the programmer must also deal with the integration and communication aspects of his or her application, which is a complex task, given the specialised knowledge needed of both the software and infrastructure parameters that can be controlled—for instance both uses cases 2 and 3 involve external hardware that must be able to feed data directly and reliably to application components deployed in the cloud.
4) **Use of virtualized resources:** in order to provide properties such as portability, replicability and availability, as well as reduced operational cost and flexibility of resource utilization, the application must be virtualized and made independent of physical infrastructure. This is usually perceived as a complex task that can be achieved only by cloud programming experts and necessitates the selection of appropriate cloud platforms and programming models—for example in use case 1 different parts of the collaboration platform are deployed in their own domains according to their own scaling requirements in anticipation of the services demanded by clients.
5) **Configuration of the infrastructure:** implementing the QoS constraints necessitates knowledge of the underlying infrastructure and furthermore of the configuration of components on the network. Hence there is the need for a common, reusable interface for programming the infrastructure (e.g. software-defined networks and infrastructures)—for example in use case 3, there is a need to monitor the QoS of the streaming service and respond to changes immediately, requiring an understanding of how to respond to quality issues programmatically.
6) **Data intensive communication:** there is a need to collect and store large volumes of data, and decisions must be taken on where and how to process data in order to cope with problems of data volume, variety, velocity and veracity. There is therefore a need to define virtualized storage and communication configurations based on functional descriptions and to characterise data throughput and latency between application components. In situations where the supply of data can dramatically change, it is also necessary to provide support for elasticity and modify the configuration accordingly, for example in use case 2, where an event can create a surge of data from existing sensor deployments.
7) **Adaptability for quality-on-demand:** applications increasingly need to be configured to cope with quality-on-demand in the run-time environment. For example, when live events are being broadcast (as in use case 3), the follower might want the best possible video quality when watching a particular content, while otherwise being satisfied with lower quality when watching something else.
8) **Adaptability to changing infrastructure:** in many use cases (including the three described above), maintaining QoS without interruption is of particularly high importance. In such cases the programmer must

implement code to address changes of infrastructure, for example by designing and implementing the ability to change packet routing if QoS attributes begin to deteriorate.

9) **SLA negotiation:** closely related to the necessity to adapt to changing infrastructure, it is also necessary to dynamically enforce explicit SLAs (Service Level Agreements) with cloud providers at runtime; this is particularly of concern where prolonged violations of SLAs can have a measurable business impact, such as for use cases 1 and 3.

These requirements cover the entire lifecycle of the time critical cloud applications: including development, verification, programming, deployment, and runtime control.

Programmable infrastructures, such as clouds and software defined networking, provide an elastic and flexible way of configuring and reconfiguring the infrastructure as needed (much more flexible than the traditional approach of configuring individual switches, firewalls, etc., directly). However, it is becoming increasingly apparent that the development of such time critical cloud applications presents complex requirements to programmers. We review existing technologies in three contexts: application development, infrastructure customization and deployment, and runtime QoS control.

2.2. State of the art

Support for time critical applications in cloud environments is still at a very early stage, especially for applications that should be self-adapting in order to maintain the required system performance. We shall review the state of the art from the three most relevant technical aspects: 1) distributed application programming, 2) advanced infrastructure (and in particular programmable infrastructure), and 3) self-adaptive performance control.

Programming distributed applications often depends on the adoption of a specific computing architecture or platform; typical examples include Message Passing Interface (MPI)-based parallel computing in distributed memory cluster architectures, service platform-based workflow applications, and cloud-based Map Reduce processing. Time critical applications may likewise involve some MPI or other parallel computing based components for high performance data processing; however, the distributed nature of the system components often makes the time critical application difficult to deploy in a parallel computing program based on technologies like MPI. Quality constraints are used in workflow applications for describing the abstract workflows, and for creating the runtime enactment, such as by Zhao et al.[1] However, in those applications, the creation of the application logic is mostly separated from the customisation of the runtime environment; in particular, a formal model is rarely utilised in verifying time constraints. Co-programming of application and the runtime environment requires a formal underlying model for both applications and infrastructure, continuous interaction between the programming environment and the execution platform, and the ability to verify that the specified application is consistent and executable with the requested QoS.

1) **Application and infrastructure co-specification** benefits from the use of a formal taxonomy for application and infrastructure profiles, which would assist in developing the necessary mappings from QoS constraints to application/infrastructure characteristics to data acquired during runtime monitoring. A quality-aware model-driven engineering approach (such as described in Soley[2]) where non-functional requirements for applications can be rigorously specified (for example using OCL[3] or a dedicated UML profile like MARTE[4]) could be used by developers to describe applications in a way that could then be used to effectively deploy those applications on the cloud.

2) **Internal coordination** is necessary to provide live feedback between developer and execution platform. With regard to the programming interface via a Web interface, it is necessary to use lightweight, non-invasive technologies; for example high-level web frameworks that provide support for RESTful APIs[5], and allow the use of message queuing systems (e.g. based on AQMP[6]) to manage the execution and monitoring of distributed components in conjunction with lightweight workflow management tools.

3) **Formal reasoning and verification** is needed to maintain consistency between application and infrastructure views. There is a lack of well-defined methodology for translating between system-objective QoS attributes and human-subjective QoE attributes. The use of application profiles (e.g. modelled using MARTE) that can be translated into formal models like timed Petri nets[7], makes it possible at an early stage to formally verify the satisfiability of certain non-functional requirements. For the profiling of infrastructures, the use of network-

oriented description languages such as the Infrastructure and Network Description Language[8] appear promising if they can be adapted for the cloud context. The use of Semantic Web technologies would make it easier to ensure interoperability with other information models such as used for SLA negotiation, monitoring or QoS specification.

Advanced infrastructures enable quality guaranteed runtime environments for time critical applications, in which we can see two important foci directly related to time critical applications. The first one is from the transferral of High Performance Computing (HPC) environments to virtualized infrastructure such as the Cloud, for instance HPC cloud services in the European Grid Initiative (EGI)[9]. In those environments, supercomputers, and HPC and GPU clusters, are deployed in a cloud environment to support tasks with very high performance requirements, but underlying most of them is an MPI-based parallel computing model. The second focus is driven by the emergence of advanced network technologies—not only advanced hardware (e.g. quality-guaranteed optic networks) but also advanced protocols for controlling network behaviour and quality of service, such as Software Defined Networking (SDN)[10]. These advanced infrastructures provide developers opportunities to program and customise qualified runtime environments for time critical applications. However, to do this effectively also requires 1) an effective planning model for defining the virtual runtime environment, 2) advanced network services for optimised communication, 3) agile encapsulation of application components, and 4) technologies for issuing Service Level Agreements (SLAs) and real-time negotiation. In detail:

1) **An effective planning model for defining virtual runtime environment** involves selecting and matching resources from a resource pool with specific requirements. Semantic modelling and searching technologies are commonly used. The semantic model for describing virtual infrastructure, in particular network topologies, is important in this context. Ghijsen et al.[8] describe a semantic web based description language for virtual resources and network known as the Infrastructure Network Description Language (INDL), based on the Network Modelling Language (NML). For search, semantic matching and optimisation technologies such as genetic algorithms and Ant Colony Optimisation (ACO) have been explored extensively[11].
2) **Advanced network services** provide applications extra opportunities to optimise data communication. Software Defined Networking (SDN) protocols such as OpenFlow[12] and Network Service Interface (NSI)[13] have attracted substantial attention from both industry and academia. Compared to purely network level protocol optimisation, such as multiple path TCP, these SDN technologies allow applications 1) to customize network connectivity between services by defining suitable flow forwarding tables, or by reserving dedicated links, 2) to virtualize the network resources for different partition schemas by tuning the network slice for given set of computing and storage nodes, and 3) to control the network quality of service by either advanced reservation of links or dynamically controlling the packet flows. However, including these new features in data delivery services is still at a very early stage.
3) **Agile encapsulation of application components** is required to deploy non-monolithic applications effectively on clouds. A number of virtualisation solutions exist, but containers have recently gained an increased profile as a more lightweight and easily extensible mechanism that can be used in lieu of full virtualisation if a single standard operating system kernel can be employed[14]. Container technologies such as Kubernates or Rancher deployed on a dedicated container operating system such as CoreOS or RancherOS can be employed if supported by adequate scheduling mechanisms to ensure high availability of components[15], though more sophisticated monitoring facilities than are currently provided with these technologies may be needed.
4) **Service Level Agreement (SLA) issuing and real-time negotiation technologies** depend heavily on the complexity of the mapping between application requirements and the available resources, and the matching among quality requirements at different service layers. Most mapping approaches are based on graph mapping using key quality parameters such as execution time; however limited association between the application and infrastructure during application development makes the searching procedure over large resource graphs very time consuming. In this context, the main approach currently taken to improve the search procedure is to include different types of heuristics and optimization technologies, for instance parallelizing the searching procedure for matching resources and applications[16], pre-processing the resource information by clustering the resource information based on the SLA request, and multi objective optimisation for searching alternative solutions[17].

Self-adaptable software architecture and performance control has attracted substantial attention amongst software

engineering researchers during the past decade. Self-configuration permits services to reconfigure themselves in response to changes in their environment. The elasticity of clouds makes them a natural context for self-adaptive applications, with the ability to scale both vertically (conscripting more powerful resources) and horizontally (parallelising computation across more nodes). Despite this, current cloud providers have yet to solve all the problems associated with automating scaling strategies and providing runtime configuration features for virtualised applications. The self-adaption problem is often modelled as a looped procedure of monitor-analyse-plan-take action[18].

1) **Monitoring the system performance** has been extensively studied in the context of distributed systems[19]. Application quality can be monitored directly by the application itself or via the monitoring of the runtime environment. In distributed applications, such as workflow management systems, QoS monitoring is often associated with logging of the application's execution. The workflow provenance system employs a semantic model to annotate logged events and allows users to query the events based on specific criteria and to reconstruct the execution sequence of the application[20]. Monitoring of QoS aspects of the infrastructure, such as network quality, has been recognised as an important service, especially in cloud environments. MonPaaS[21] is an example. These basic monitoring services provide useful support for obtaining quality of service information, but they mainly focus on specific quality attributes of the infrastructure or the application. Semantically harmonising different kinds of monitoring information, and in particular harmonizing this information with the application logic, still remains challenge for self-adaptive systems.
2) **The analysis and planning mechanism** can be implemented using a centralised or distributed paradigm. SanGA[22] and ReviveNet[23] are two examples of the centralised paradigm—the advantage of such an approach is low communication overhead and easy implementation. In the distributed paradigm, the activities in the looped procedure are distributed. Advantages over the centralized approach include the fact that there is no single point of failure, and greater flexibility in monitoring and controlling distributed components, although the communication overhead may have a negative impact. Nallur and Bahsoon[24] mention time related issues as a basis for self-adaptability (as a cost for selecting services in an auction procedure), but do not directly refer to the time critical attributes we highlighted in the proposal.
3) **The self-adaptive mechanism** has been implemented mainly using a so-called *architecture style*[25], in which the internal structure of the application architecture is explicitly modelled and the application can manipulate the structure at runtime based on the results of certain decision-making procedures. Esfahani et al.[26] proposed a different approach based on the application features and use a machine learning mechanism to implement adaptability. These early works either focus only on the application control, or only on the quality of the service from the provider point of view; they do not fully explore the adaptability of systems that contain programmable infrastructure.

All these aspects (distributed application programming, advanced infrastructure, and self-adaptive performance control) need to be addressed to realise the deployment of time critical applications in cloud environments and have been subject to considerable prior research; but they all still far from solved problems, especially given recent developments in programmable infrastructures and cloud platforms.

2.3. Technical gaps

From the above review, we can identify a number of technical gaps that need to be addressed:

1) Current time-critical application programming models lack consideration of the controllability of the infrastructure; they thus do not exploit the potential benefits offered by the programmable infrastructure provided by cloud environments. In particular, there is a lack of effective description mechanisms for specifying application logic, system quality constraints, and infrastructure controllability. Moreover, tools providing effective support for application-infrastructure co-programming have not yet been produced.
2) There have been many studies on the application of various optimisation mechanisms in selecting resources. However, there is currently no semantically well-modelled mapping between the application quality of user experiences, and infrastructure-level QoS attributes. This renders the modelling and negotiation of a Service Level Agreement between application and the resource providers difficult at best. Although optimisation of network protocols in data communication has been extensively studied in the network and communication

domain, the inclusion of network controllability (such as SDN technologies) in applications' data delivery services is still at a very early stage[1].

3) Although self-adaptive software has already been studied in software engineering, and autonomous systems, there lack effective mechanisms to semantically couple the various kinds of monitoring with application logic, and to make use of this information in order to adapt system-level performance by controlling both application and infrastructure. The existing self-adaptation intelligence focuses either on controlling application architecture *or* on the service quality of the runtime infrastructure; there is a lack of co-controlling mechanism addressing *both* aspects.

3. The SWITCH approach

We propose a software workbench, namely the Software Workbench for Interactive, Time Critical and Highly self-adaptive cloud applications (SWITCH). The overall objective of the SWITCH project is to address the entire life-cycle of time-critical, self-adaptive cloud applications by developing new middleware and front-end tools to enable users to specify their time-critical requirements for an application using an interactive user interface, then deploy their applications and adapt the infrastructure to changing requirements either automatically (using the specified requirements) or by human intervention if desired.

3.1. The basic idea

The SWITCH project addresses the technical gaps described in section 2.3 by providing an interactive and flexible software workbench that, by using discovery tools at the networking level and QoS requirements from the application level, can provide the tools necessary to control the lifecycle for rapid development, deployment, management and dynamic reconfiguration of complex distributed time critical cloud applications. In particular, SWITCH provides integrated support for defining, optimising and controlling time critical constraints while programming, testing, deploying and executing the applications. Using a fully responsive web based interface and backend components for coordinating the data flows across the networking infrastructure, the SWITCH workbench can define dynamic application-level mappings for the time critical control rules and strategies to be employed on an application-by-application basis.

At the core of SWITCH environment is a new development and execution model—an application-infrastructure co-programming and control model—that will be developed for time-critical cloud applications. This model brings together application composition, execution environment customisation, and runtime control, which are normally handled by separate processes, into one optimisation loop based on time critical requirements. In this model: 1) the application logic will be programmed with full regard to QoS/QoE constraints together with the programmability and controllability of the cloud environment such that both application and the virtual runtime environment for executing the application can be optimised at the design phase; 2) the virtual runtime environment can be customised for time critical application requirements, and can be provisioned in the cloud with a time critical application oriented Service Level Agreement (SLA); and 3) the application can autonomously adapt its own behaviour and that of the virtual runtime environment when performance drops during runtime. The SWITCH environment employs formal performance reasoning mechanisms to guide each step in the development and the tools are delivered to the users via three subsystems, which are shown in Fig 1.

3.2. SWITCH integrated development environment

The SWITCH Interactive Development Environment (SIDE) subsystem provides interfaces for all user- and programmer-facing tools, by exposing a collection of graphical interfaces and APIs that tie SWITCH's services to a Web-based environment. SIDE will be engineered using fully responsive HTML5 on the front end, providing interactivity and end-user device portability, and the back-end Web services (hooks) and APIs will be constructed using Python and tools such as Django, Flask, or similar.

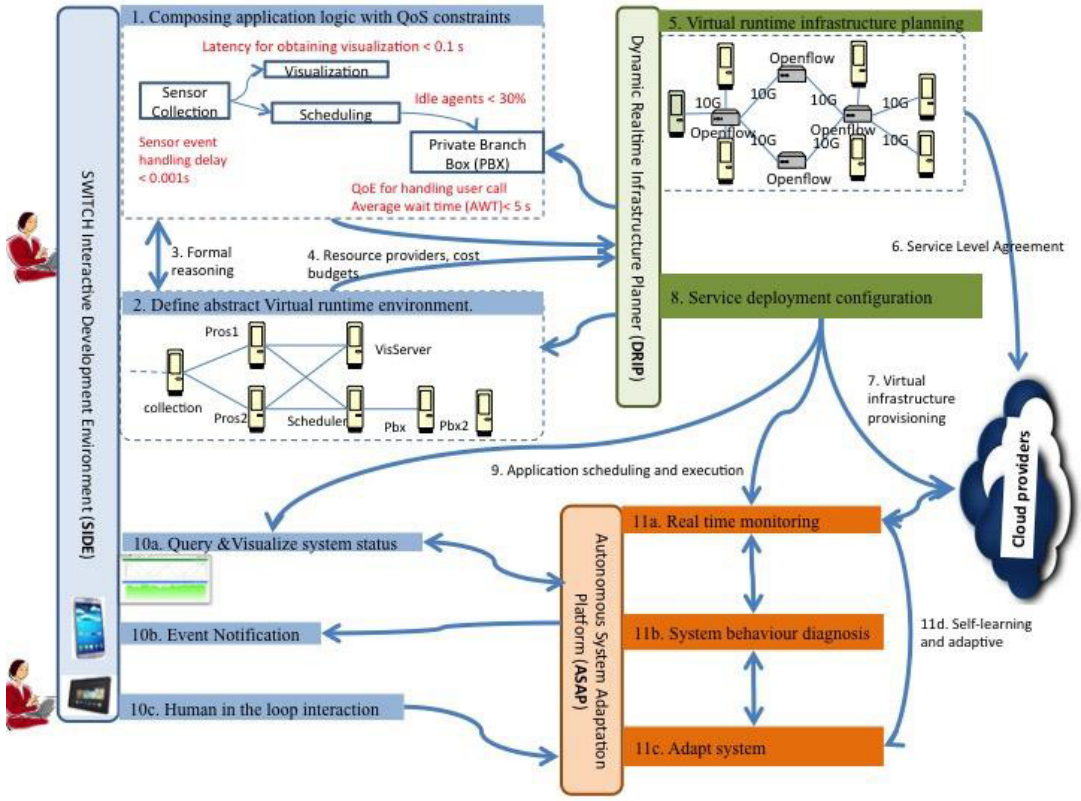

Figure 1. How SWITCH works

3.3. Dynamic real-time infrastructure planning

The Dynamic Real-time Infrastructure Planner (DRIP) subsystem prepares the execution of the applications developed in the SIDE subsystem by 1) semantic modelling and linking of different QoS/QoE attributes, 2) defining an optimal virtual runtime environment, 3) creating a Service Level Agreement with the resource provider, and 4) deploying the platform required by the application.

Autonomous System Adaptation Platform

The Autonomous System Adaptation Platform (ASAP) 1) monitors the status of the application and the runtime environment, 2) examines the actual performance of the required quality attributes, 3) autonomously manipulates the application and runtime environment to maintain optimal system level performance against the time critical constraints, and 4) learns from its own decision history to improve its intelligence in making future decisions for autonomous reconfiguration of the application.

Basic scenario

Figure 1 shows a basic scenario of SWITCH. The application developer begins with composing the application logic and defining the QoS constraints (e.g., maximum latency for state visualisation or maximum sensor event handling delay) that apply to it (step 1). The developer can also provide an abstract network overlay to define the runtime environment (step 2). These activities can be optimized and aided using a knowledge base of successful patterns of applications and infrastructure, which employs a formal reasoning component (step 3). The results will be passed from SIDE to DRIP; the developer can also specify requirements such as specific resource providers to be used, and

the total cost budget for application execution (step 4). DRIP plans the concrete virtual runtime environment of computing, storage and network elements by reasoning about the provided application-level QoS constraints (step 5). DRIP then generates SLAs with the resource provider(s) (step 6), and the resource provider(s) provision the virtual environment accordingly (step 7). After that, DRIP customises the virtual environment and deploys necessary services for the application (step 8), and unbundles and executes the application (step 9). At runtime, SIDE allows the user to: query and visualise the run-time status of the application and run-time environment (step 10a); receive notification of system status and inform the user (step 10b); directly manipulate the system execution (step 10c); and expose real-time monitoring information from the database, which is generated by ASAP. The ASAP subsystem can monitor the runtime status of the application and its environment (step 11a), pass this information (via the backend database) to SIDE, diagnose system performance and make decisions on control actions needed to restore performance where necessary (step 11b), take action to maintain system performance (step 11c), and learn from history in order to improve the subsequent effectiveness of decision making (step 11d).

4. A use case

The functionality of SWITCH can be better illustrated by referring back to one of the use cases of section 2.1. The SWITCH environment can contribute to the use case of the cloud studio for directing and broadcasting live events with all three of its subsystems:

1) The SIDE subsystem can provide the cloud studio developer an intuitive, interactive interface to 1) describe the application logic among: video camera sources, streaming services, archiving, video switch service and possible video processing services, 2) define quality requirements at system-level or each individual process level, 3) describe an abstract virtual runtime environment for switching videos, and 4) describe the cost and quality requirements for the runtime environment on Cloud, and select potential cloud providers. The formal reasoner included in SIDE can then validate the quality constraints a developer has put in the description.

2) The DRIP subsystem will create a concrete virtual runtime environment based on the input received from SIDE, and create a Service Level Agreement (SLA) to negotiate with the resource provider, and deploy the virtual video switch services to the virtual runtime infrastructure after it is provisioned.

3) The ASAP subsystem will detect the key quality attributes such as video quality, and packets lost during streaming, and dynamically tune the network quality by adding new flows (network paths) or rescheduling the network traffic.

With the support of SWITCH, the development and deployment effort associated with a cloud based production service for live events can be dramatically reduced. Instead of transporting large numbers of equipment and crew members to the event venues, the broadcaster will only need to rent sufficient bandwidth network for connecting the video content to the virtual video switch in the cloud using the SWITCH environment. This use case concentrates on the broadcast scenario, but actually this technology can bring much broader business potential, for instance for enterprises to broadcast their own live events, or even for individuals to start up their own personalized TV stations.

5. Summary

In this paper, we introduced the basic idea and approach of a newly funded EU H2020 project called SWITCH. The project started in February 2015 and will last three years. The software of SWITCH will be open source. The consortium draws half of its members from industry, and the other half from academic institutions; the development of the SWITCH software will be tested and validated using industrial-strength use cases. Moreover, the industrial partners will actively explore the future market value of developments.

5.1. Relevant projects and innovations

Related in scope with SWITCH, several other projects have tackled similar problems in cloud service development, provisioning and QoS control, including MOSAIC[27,28], MODAClouds[29], NetIDE[30], DICE[31], SeaClouds[32] and U-QASAR[33].

From the software engineering point of view, the FP7 project MODAClouds (MOdel-Driven Approach for design and execution of applications on multiple Clouds) focuses on methodology and tools for designing multi-cloud systems with guaranteed QoS. MODAClouds follows an MDE approach, where the applications are defined at 3 different levels (Cloud-enabled Computation Independent Model, Cloud Provider Independent Model and Cloud Provider-Specific Model) using a meta-model called MODACloudML, allowing developers to specify both functional and non-functional requirements. For validation, the CPIM and CPSM models can be transformed into Palladio Component Models (PCM) so that the software engineer can run quality prediction tools over those models to verify in an early stage if the non-functional requirements can be satisfied. The project shares a similar mission with SWITCH, finding a new approach for managing the lifecycle of cloud applications; however, SWITCH has a specific focus on the programmability of infrastructures, using techniques such as SDN.

The FP7 project NetIDE ("An integrated development environment for portable network applications") developed an Eclipse-like IDE and associated tools for developing network applications based on SDN. NetIDE follows a Model Driven Architecture (MDA)-like approach, where the network applications are defined using a platform-independent model written in a domain-specific language called IRF (NetIDE Interchange Representation Format). The project aims at developing network control plane programs, defining mechanisms to abstract SDN programming independently from the underlying SDN flavour. NetIDE contributes valuable input for SWITCH's development, however the SWITCH project has a different focus, namely on combining infrastructure programming (e.g., at SDN-level) with the verification of time critical quality constraints at the application level.

The FP7 project MOSAIC ("Open Source API and Platform for multiple clouds") focused on provisioning and deploying applications on multiple Clouds. The open source platform offered APIs for developing cloud applications with abstractions of IaaS (Infrastructure-as-a-Service) services that enable the migration of these applications from one cloud to another, implementing a semantic engine for discovering cloud API components, resources and services driven by functional and application domain concepts, cloud patterns and inference rules. In SWITCH, semantic components will be used to describe not only application service components, but also infrastructure properties, such as network topology and QoS. In contrast to MOSAIC, SWITCH highlights the time critical application oriented programming and control model.

Several other projects have also addressed application QoS control in clouds. The EU H2020 DICE ("Developing Data-Intensive Cloud Applications with Iterative Quality Enhancements") project aims to continuously enhance data-intensive cloud applications with the goal of optimizing their service level by making quality-aware MDE accessible to developers of Big Data applications. DICE tools rely on UML meta-models annotated with information about data, data processing and data transfers. The DICE QA tool chain covers simulation, verification and architectural optimization. These tools are coupled with feedback analysis methods to help the developer iteratively improve the application design, based on monitoring data obtained from test or production environments. The EU FP7 U-QASAR (Universal Quality Assurance & Control Services for Internet Applications with Volatile Requirements and Contexts) project created a flexible Quality Assurance, Control and Measurement Methodology to measure the quality of Internet-related software development projects and their resulting products. The development of SWITCH will carefully review the achievements from these projects when modelling and controlling quality attributes of services and user experiences for time critical applications. SWITCH focuses not only on these QoS control and optimization intelligence at specific stages, but more importantly on putting them in the context of a user-centred software workbench during the entire lifecycle of application development, provisioning and operation.

Aside from cloud-specific projects, there are other projects, specifically in the research infrastructure space, that could benefit from SWITCH. The European Strategy Forum on Research Infrastructures[34] has produced a roadmap for European research infrastructures in various scientific domains; in particular, the ENVRI^{PLUS} project[35] seeks to provide common services to research infrastructure projects in the Environment cluster. Many research tasks can benefit from the exploitation of cloud technologies, and many of these tasks could be deemed time critical—especially tasks involving the continuous and timely processing of data from large-scale sensor deployments (similar to our use case 2, the disaster early warning system described in section 2.1). The SWITCH workbench could provide a tool for developers of research applications in much that same way as it provides a tool for commercial applications, especially for 'long-tail' researchers who are not members of larger research collectives and do not have privileged access to high-performance computing resources.

5.2. Agenda

The SWITCH environment builds on proven technologies and is therefore capable of being developed to a fully operational standard. The workbench will be developed in six phases over three years: technology review and requirement analysis, system design, technology development, tool integration, validation by use cases, and release. Development will initially be conducted within private clouds, but project results will be pushed into public clouds and the federated clouds in order to identify gaps between the current orthodoxy of public cloud provision and the requirements for supporting real-time applications on clouds.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements No 643963 (SWITCH project) and No 654182 (ENVRI^{PLUS} project).

References

1. Zhao, Z., Dumitru, C., Grosso, P., & de Laat, C. (2012). Network resource control for data intensive applications in heterogeneous infrastructures, *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum*, pp. 2069-2076.
2. Soley, R. (2000). Model driven architecture. *OMG white paper*, *308*(308), 5.
3. Richters, M., & Gogolla, M. (1998). On formalizing the UML object constraint language OCL. In *Conceptual modeling–ER'98* (pp. 449-464). Springer Berlin Heidelberg.
4. Koudri, A., Aulagnier, D., Vojtisek, D., Soulard, P., Moy, C., Champeau, J., Vidal, J. & Le Lann, J. C. (2008, March). Using MARTE in a co-design methodology. In *MARTE UML profile workshop co-located with DATE'08*.
5. Fielding, R. T., & Taylor, R. N. (2002). Principled design of the modern Web architecture. *ACM Transactions on Internet Technology (TOIT)*, *2*(2), 115-150.
6. ISO/IEC 19464. (2014). Information technology—Advanced Message Queuing Protocol (AMQP) v1.0 specification.
7. Louati, A., Barkaoui, K., & Jerad, C. (2014). Time properties verification of UML/MARTE real-time systems. In *Information Reuse and Integration (IRI), 2014 IEEE 15th International Conference on* (pp. 386-393). IEEE.
8. Ghijsen, M., Van Der Ham, J., Grosso, P., Dumitru, C., Zhu, H., Zhao, Z., & de Laat, C. (2013). A semantic-web approach for modeling computing infrastructures. *Computers & Electrical Engineering*, *39*(8), 2553-2565.
9. EGI. (2014). European Grid Initiative, http://www.egi.eu/.
10. Meyer, D. (2013). The software-defined-networking research group. *Internet Computing, IEEE*, *17*(6), 84-87.
11. Dong, W., Liu, Y., Chen, C., Bu, J., Huang, C., & Zhao, Z. (2013). R2: Incremental reprogramming using relocatable code in networked embedded systems. *Computers, IEEE Transactions on*, *62*(9), 1837-1849.
12. Jin, H., Pan, D., Liu, J., Pissinou, N., (2013) *OpenFlow-Based Flow-Level Bandwidth Provisioning for CICQ Switches*, IEEE Transactions on Computers, vol. 62, no. 9, pp. 1799-1812, Sept.
13. Zhao, Z., Van Der Ham, J., Taal, A., Koning, R., Dumitru, C. O., Wibisono, A., Grosso, P. & De Laat, C. (2012). Planning data intensive workflows on inter-domain resources using the Network Service Interface (NSI). In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:* (pp. 150-156). IEEE.
14. Felter, W., Ferreira, A., Rajamony, R., & Rubio, J. (2014). An updated performance comparison of virtual machines and linux containers. *Technology*, *28*, 32.
15. Li, W., Kanso, A., & Gherbi, A. (2015). Leveraging Linux Containers to Achieve High Availability for Cloud Services. In *Proceedings of the IEEE International Conference on Cloud Engineering (IC2E 2015)*.
16. Muller, C., Oriol, M., Franch, X., Marco, J., Resinas, M., Ruiz-Cortés, A., & Rodriguez, M. (2014). Comprehensive explanation of SLA violations at runtime.*Services Computing, IEEE Transactions on*, *7*(2), 168-183.
17. Antonescu, A. F., Oprescu, A. M., Demchenko, Y., De Laat, C., & Braun, T. (2013, December). Dynamic optimization of SLA-based services scaling rules. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on* (Vol. 1, pp. 282-289). IEEE.
18. He, Q., Han, J., Yang, Y., Jin, H., Schneider, J. G., & Versteeg, S. (2014). Formulating Cost-Effective Monitoring Strategies for Service-Based Systems. *Software Engineering, IEEE Transactions on*, *40*(5), 461-482.
19. Meng, S., Kashyap, S. R., Venkatramani, C., & Liu, L. (2012). Resource-aware application state monitoring. *Parallel and Distributed Systems, IEEE Transactions on*, *23*(12), 2315-2329.
20. Chebotko, A., Lu, S., Chang, S., Fotouhi, F., & Yang, P. (2010). Secure abstraction views for scientific workflow provenance

querying. *Services Computing, IEEE Transactions on*, *3*(4), 322-337.

21. Alcaraz Calero, J. M., & Gutierrez Aguado, J. (2015). MonPaaS: An Adaptive Monitoring Platform as a Service for Cloud Computing Infrastructures and Services. *Services Computing, IEEE Transactions on*, *8*(1), 65-78.
22. Klein, A., Ishikawa, F., & Honiden, S. (2014). SanGA: A self-adaptive network-aware approach to service composition. *Services Computing, IEEE Transactions on*, *7*(3), 452-464.
23. Yan, G., Han, Y., & Li, X. (2011). ReviveNet: A self-adaptive architecture for improving lifetime reliability via localized timing adaptation. *Computers, IEEE Transactions on*, *60*(9), 1219-1232.
24. Nallur, V., & Bahsoon, R. (2013). A decentralized self-adaptation mechanism for service-based applications in the cloud. *Software Engineering, IEEE Transactions on*, *39*(5), 591-612.
25. Cheng, B. H., de Lemos, R., Giese, H., Inverardi, P., & Magee, J. (2009). Software Engineering for Self-Adaptive Systems: A Research Roadmap. In *Software Engineering for Self-Adaptive Systems, Lecture Notes In Computer Science* (Vol. 5525, pp.1-26). Springer-Verlag.
26. Esfahani, N., Elkhodary, A., & Malek, S. (2013). A learning-based framework for engineering feature-oriented self-adaptive software systems. *Software Engineering, IEEE Transactions on*, *39*(11), 1467-1493.
27. MOSAIC. (2010). EU FP7 project Open Source API and Platform for multiple Clouds, http://www.mosaic-project.eu/.
28. Petcu, D., Di Martino, B., Venticinque, S., Rak, M., Máhr, T., Lopez, G. E., ... & Stankovski, V. (2013). Experiences in building a mOSAIC of clouds. *Journal of Cloud Computing*, *2*(1), 1-22.
29. MODAClouds. (2012). EU FP7 project MOdel-Driven Approach for design and execution of applications on multiple clouds, http://www.modaclouds.eu/.
30. NetIDE. (2014). EU FP7 An integrated development environment for portable network applications, http://www.netide.eu/.
31. DICE. (2015). EU H2020 Developing Data-Intensive cloud Applications with Iterative Quality Enhancement, http://www.dice-h2020.eu/.
32. SeaCloud. (2013). EU FP7, Seamless Adaptive multi-cloud management of service based applications, http://www.seaclouds-project.eu/.
33. U-QASAR. (2012). EU FP7, Universal Quality Assurance & Control Services for Internet Applications with Volatile Requirements and Contexts, http://www.uqasar.eu.
34. ESFRI. (2015). European Strategy Forum on Research Infrastructures, http://ec.europa.eu/research/esfri/.
35. ENVRI^{PLUS}. (2015). Common operations of environmental research infrastructures, http://envri.eu/.