ADAPTIVE MECHANISMS FOR MOBILE SPATIO-TEMPORAL APPLICATIONS

VASILEIOS THEODOROU

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ARTS

GRADUATE PROGRAMME IN INFORMATION SYSTEMS & TECHNOLOGY
YORK UNIVERSITY
TORONTO, ONTARIO

SEPTEMBER 2013

ABSTRACT

Mobile spatio-temporal applications play a key role in many mission critical fields, including Business Intelligence, Traffic Management and Disaster Management. They are characterized by high data volume, velocity and large and variable number of mobile users. The design and implementation of these applications should not only consider this variablility, but also support other quality requirements such as performance and cost. In this thesis we propose an architecture for mobile spatio-temporal applications, which enables multiple angles of adaptivity. We also introduce a two-level adaptation mechanism that ensures system performance while facilitating scalability and context-aware adaptivity. We validate the architecture and adaptation mechanisms by implementing a road quality assessment mobile application as a use case and by performing a series of experiments on cloud environment. We show that our proposed architecture can adapt at runtime and maintain service level objectives while offering cost-efficiency and robustness.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my research supervisor, Professor Marin Litoiu, whose advice and support have been pivotal for the completion of this work as well as for my academic endeavours at York University. He has been not only my supervisor but also a role-model for me and I feel truly honoured and inspired, having worked under the guidance of such a distinguished personality in both academia and industry.

I also want to express my thankfulness and respect to my thesis committee – Professor Radu Campeanu who has been extremely kind to provide me with mentoring and advice as well as to share exciting traveling stories; Professor Sotirios Liaskos who has significantly contributed in my deeper understanding for Software Architectures and Requirements Engineering and Professor Vassilios Tzerpos who has introduced me to the world of Software Reengineering. I honestly appreciate their time and guidance.

I would also like to thank my colleagues and friends at the Adaptive Systems Research Lab of York University who were always there for me and whose experience and help have been invaluable. In particular I would like to express my gratefulness to Dr. Mark Shtern, with whom I have worked most closely. I sincerely feel that his contribution has been great not only in the conducting of this work but also in my maturity as a researcher.

I would also like to express my gratitude to the person who contributed the most to my journey in Canada, Professor Kostas Kontogiannis. His supervision and support throughout my undergraduate studies was outstanding and his advice that ‘sky’s the limit’ has been truly inspirational.

At this point I could not omit to express my gratefulness to Professor Timos Sellis whose teaching and generous help have been priceless.

Finally, I am very grateful to my family who taught me all the important principles that define my personality and have always supported and inspired me. Special thanks also belong to my new family and friends in Toronto and especially Tina Bekeros who has provided me with crucial support.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1
Introduction

1.1 Motivation

The widespread adoption and use of smart devices is driving for new models of ubiquitous computing. Smart devices usually offer a large set of functionalities including portable media players, GPS navigation and wireless internet connectivity. In order to do so they are equipped not only with powerful memory, processing and network resources but also with sensors such as GPS, accelerometer, camera, touchscreen; and the list seems to continuously grow with light-, temperature-, air quality-, humidity-sensors and so on. What is common about all these sensors is that they can continuously sample and produce rich data about the user's context and environment, which can potentially be transmitted to an information management system.

The capabilities described above can support the development applications with functionalities such as crowdsourcing for a common goal, personalization and context awareness. Despite this hardware innovation, it appears that the traditional paradigm for mobile application deployment is not sufficient to facilitate such applications. The model of context-agnostic design which contain a centralized server provisioned for a certain level of traffic is not only incompatible with those applications but also costly and rigid.

The difficulties mentioned above are also true for a special category of mobile applications which are marked by their high mobility and change in time – spatio-

temporal applications. Spatio-temporal applications are generally described as applications where spatial changes occur over time. Although the conceptual model and semantics for this field are mature, there still needs to be a lot of work for their efficient deployment on supporting infrastructures. In addition, modern devices can easily enable their client-side realization, which indicates that now is the right time to examine the optimal architectural designs and methods to effectively obtain the desired benefits from their use.

The main challenge faced during the implementation of spatio-temporal applications is the tremendous variance in data traffic – both in time and space. As an example, we could consider a navigation mobile application. Users of the application who happen to be at the same point of road congestion, would likely all consult their navigation application at the same time, leading to increased data traffic at that specific point. Similar examples can be found in multiple other fields such as Business Intelligence, Disaster Management and so on, where unexpected events can triger specific activity and consequently localized peaks in data traffic.

Another major challenge concerns the QoS characteristics of such applications and their ability to provide differentiated services based on time, space and user activity in a transparent manner. Services such as location awareness are not very useful when they are frequently interrupted, especially in cases where sensitivity plays an important role. In addition, such vigorous systems should be robust and resilient to state changes, which is almost impossible to achieve with inflexible centralized components. They should

innately adjust to changes in geographical distribution and unpredicted events or patterns with the least possible human intervention, while maintaining important characteristics including low latency and security.

In addition to performance, it is very important to consider the optimal utilization of available resources. There are two main reasons that dictate careful resource management for spatio-temporal mobile applications. The first reason is that in spite of the continuous improvement of mobile devices' features, they still suffer from many weaknesses such as limited computational power and battery lifespan. Concurrent applications constantly running on the background can drain battery very fast, particularly when they demand frequent wireless transmission of data. The second reason is the enormous number of users and consequently the vast amount of data that is produced, transmitted and processed. In this respect, traditional implementations on the server side would require expensive hardware appliances provisioned for maximum capacity, which could be avoided considering such applications' load variability over space and time. Moreover, one significant parameter of cost is localization, since there are currently many examples where cost varies among different regions of server farms.

One promising model which could provide viable solutions to many of the above-mentioned concerns is Cloud Computing. The most indubious meaning of Cloud Computing would be "on demand access to elastic computing resources" and this paradigm has gained a lot of popularity over the past few years. The promise of this model is automation in the sense of seamless deployment and running of applications.

However, it is currently not a mature model and many of its aspects and capabilities are yet to be explored. For instance, it is not an obvious task to select the optimal architectural deployment on the cloud according to distinct characteristics of an application. The availability of diverse resources and alternative technologies comes at the cost of increasing the complexity of the optimization problem. Furthermore, the degrees of freedom require many architectural and management decisions to be made such as which should be the modularization of the system and where automation should take place.

The work presented in this thesis explores the use of Cloud Computing for the deployment of spatio-temporal applications and addresses many of the challenges mentioned. We consider adaptivity as one crucial aspect of such systems and propose an adaptivity-enabling architecture which can facilitate scaling, context-aware adaptivity and dynamic configuration of real-time data transmisson parameters. To support these functionalities we apply techniques from the area of Autonomic Computing and we evaluate our methodologies using a mobile application for the assessment of road quality.

1.2 Research Objectives

The research objectives of this thesis can be summarized as follows:

1) Define an architecture for mobile spatio-temporal applications, which can facilitate multiple levels of adaptation.
2) Define and evaluate adaptation mechanisms for mobile spatio-temporal applications for dynamic adjustment to changes in traffic load and users'

spatial distribution, while maintaining QoS characteristics such as low response time.

3) Evaluate alternative configurations and database technologies for real-time applications deployed on a cloud environment.

The hypotheses tested in this work are:

1) Our proposed architecture can support the functionalities of a spatio-temporal application and be implemented in alternative configurations.
2) In the context of three-tier, real-time applications, a NoSql Database System has significantly better performance than Sql Database System.
3) Location-awareness and elastic scaling can be supported for mobile spatio-temporal applications, while maintaining response time lower than transmission period and infrastructural cost to a minimum.

We propose a three-tier architecture for mobile spatio-temporal applications with the data tier and the logic tier deployed independently on the cloud. In addition, we define adaptive methods in order to continuously monitor the running instances of the application and proceed to corresponding adjustments to number of instances, geographic coverage and configuration parameters, as required according to defined policies. We also apply adaptation on the client side with mobile clients sharing partial state information, adjusting their sampling and transmission parameters and having some degree of freedom as to which server to direct their requests to, thus providing context-aware adaptivity.

As a case study to validate and test our proposed architecture we consider a smartphone application for monitoring and assessing the condition of roads in a city. This application collects data about each vehicle position, speed and 6-axis acceleration and sends the users information about traffic congestion and road quality to a server. This application is ideal for demonstrating the potential of harnessing currently available technologies in an efficient fashion, to carry out tasks which would appear almost impossible until recently, such as applying a full-scale traffic monitoring using common smartphones.

1.3 Research Contributions

This research work introduces an architecture and adaptive techniques to implement mobile spatio-temporal applications in an efficient and effective manner, utilizing modern concepts and technologies. We test and validate the feasibility of our proposed framework by designing, implementing and deploying a traffic mobile application in the Amazon AWS environment and conducting a set of experiments, as will be described in more detail in Chapter 6.

The first contribution of our work is the introduction of an adaptivity-enabling architecture for mobile spatio-temporal applications. Our architecture is unique in that it includes autonomic components with concrete functionality on multiple levels in a modular design. This architecture is three-tier providing a high degree of modifiability, as any tier of the application can be independently altered or replaced without critical implications on the other tiers. In adition, this architecture allows to test, configure, deploy and manage each of the tiers separately, whereas the application and data tiers can

easily be deployed on elastic infrastructures such as the cloud. Furthermore, we emphasize sensor management and local storage capabilities on the mobile web client so as to exploit the features of modern smart devices and promote dependable sampling and transmission of data.

The second contribution of our work regards the introduction of adaptive methods for mobile spatio-temporal applications, which are compatible with our proposed architecture. We provide an analysis of the adaptations that can take place given the specific features of spatio-temporal applications and challenges during their deployment. Thus, we propose adaptations both at the server and at the client side. On the server side our methods support the adaptive scaling and configuration of distributed server clusters according to continuously changing distribution of requests. On the client side, they facilitate the routing of requests according to user's current location and the optimal client configurations such as setting of sampling and transmission rates, according to availability. In addition, the clients and the servers share a common state about runtime metrics and can both implement adaptation decisions in a different scope, as will be thoroughly explained in Chapter 5. The key advantage of our approach is offering context-aware adaptivity and cost efficiency while maintaining important QoS characteristics, as we showcase in Chapter 6 by conducting a relevant set of experiments.

A third contribution of our work derives from the fact that apart from evaluating the feasibility of our framework, we also conduct specific experiments which allow us to evaluate alternative settings and configurations of our application and acquire an

understanding of our problem space by observing concrete results. In this respect, we evaluate the use of two different database systems for our web application data tier, namely MySql and MongoDb, producing results for our application that could be generalized for mobile real-time applications running on Amazon AWS [37] instances. We generate different read/write mix workloads to simulate client requests and we measure performance characteristics such as latency, throughput and resource utilization. To our knowledge, such experiments for this class of applications have not been published yet and our work sheds some light on the advantages and disadvantages of each case.

1.4 Thesis Overview

This thesis is organized as follows. Chapter 2 presents the background for the main areas related with our research field; Chapter 3 presents the related work and provides an evaluation of approaches with regards to our work. Chapter 4 describes our proposed adaptation-enabling architecture for mobile spatio-temporal applications. Chapter 5 describes our adaptive methods and related algorithms both for server and client adaptation.Chapter 6 describes our experimental settings and presents our obtained results. Conclusions and Further Work are presented in Chapter 7.

Chapter 2
Background

In this chapter we provide a brief overview of the main areas of this research, namely cloud computing, spatio-temporal applications and autonomic computing. The aim of this chapter is to provide readers with the relative background for the current work.

2.1 Cloud Computing

Cloud Computing (CC) has received significant attention and popularity over the past years, as a promising realization of the long-held dream of computing as a utility [1]. The predominant goal of Cloud Computing is to provide on-demand computing services with high reliability, scalability and availability in distributed environments. Former technologies such as Cluster and Grid have also aimed at allowing access to large amounts of computing power in a fully virtualized manner [2]. However, what presently make CC so attractive, are its characteristics such as pay-per-use, elastic capacity and illusion of infinite resources, self-service interface and virtualized resources, combined with the maturity of several technologies which have made CC viable [2].

CC has been described in many different ways [1][3] and no standard definition has been adopted until now. Recently, the Information Technology Laboratory at the National Institute of Standards and Technology (NIST) [4] has posted a working definition of cloud computing: “Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g.,

networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction". According to NIST the essential features of the cloud model are Rapid Elasticity, Measured Service, On-Demand Self-Service, Ubiquitous Network Access and Location-Independent Resource Pooling. In addition, based on the different level of services that a cloud provider can offer, there are mainly three delivery models for cloud (Software as a Service, Platform as a Service, and Infrastructure as a Service). Finally, based on who owns, manages and operates the cloud appliances, cloud can be found in four deployment models (Public Cloud, Private Cloud, Community Cloud and Hybrid Cloud). Unfortunately, the exact definition and use of this technology is still causing confusion among practitioners in the commercial and academic spheres [2][1].

Conceptually, in CC everything is assumed as a service (XaaS), such as IaaS (Infrastructure as a Service), SaaS (Software as a Service), PaaS (Platform as a Service), HaaS (Hardware as a Service), delivered over a network, such as the Internet. To this end, a large number of cloud service providers and middleware suits have emerged, each providing different CC services. This evolution has led to a prospect of CC in which cloud users can choose from a diverse set of different services and providers, while being able to act as service providers themselves.

2.2 Spatio-temporal applications

Space and Time are the most important dimensions related with events in real-life activities. People's behaviours and decisions are usually influenced by geographical space and time parameters. Spatio-temporal objects are those that have spatial and temporal attributes and their spatial attributes change over time [5]. Many location technologies such as GPS, TA (Timing Advance), AOA (Angle of Arrival) etc. are currently available, enabling for the development of applications based on data about moving objects, for example traffic control, weather monitoring, military affairs and so on.

Throughout the past two decades there has been extensive research regarding spatial and temporal data models, database management systems (DBMS) and database applications [6]. The terms spatial database and temporal database have been extensively used in the scientific literature to refer to databases where the description of the data of interest to users includes, whenever relevant, some description of how the real-world phenomena represented in the database are positioned in a given spatial and temporal framework [7]. Spatio-temporal database models emerged from the integration of spatial and temporal database models which were developed separately [8].

Spatio-temporal data handling is not an obvious task due to the complexity of the data structures requiring careful analysis in structuring the dimensions, together with the representation and manipulation of the data involved [8]. Thus, specific management systems are required to describe the extent of objects in space as well as pervasive

phenomena, lifecycles and validity periods for time-varying information and trajectories or movement of mobile objects. These systems are called Spatio-Temporal database management systems and support explicit modeling and manipulation of data with spatial and temporal characteristics, facilitating efficient querying [7].

Spatio-temporal applications are real world applications, where spatial changes occur over the time line. Existing work on such applications examines them as traditional applications but with a special focus on data representation and management requirements, as explained above. In this respect, Pelekis et al. [8] provide a comprehensive review of the literature on different types of spatio-temporal data models that have been proposed as well as theories and concepts that have emerged. According to Pelekis et al. [8] spatio-temporal applications requirements fall in four categories, as follows.

- Temporal Semantics: This category deals just with the nature of time including the basic features that are used to describe it. Main aspects include time granularity and density, time order and duration and time representation.
- Spatial Semantics: This category handles the pure spatial aspects of the existing approaches. It mainly concerns structure of space (raster vs. vector representation), support of orientation, measurement capabilities and topological relationships.

- Spatio-Temporal Semantics: This category deals with the unified spatio-temporal semantics, such as datatypes, change and evolution in time and space and the degree of dimensionality.
- Query Capabilities: The final category considers the query capabilities of the models. In specific it regards the degree to which spatio-temporal systems can support queries about spatial, temporal and interrelated spatio-temporal behaviors and relationships and the expressiveness of these queries.

2.3 Autonomic computing

Autonomic computing (AC) is a relatively new concept, which was introduced as an answer to dealing with the management and maintenance of complex, heterogeneous and distributed computer systems [9][10]. Inspired by the sophisticated mechanisms of the human body in order to respond and adapt to changing conditions, it was firstly referenced by IBM [11]. Autonomic Computing Systems (ACS) are systems that can effectively manage themselves throughout their lifecycle.

There are four major self-* properties that can characterize a system as autonomic. These major characteristics are self-Configuration, self-Healing, self-Optimization, and self-Protection, usually called self-CHOP [9]. Most applications implement some of these properties to acquire some form of autonomic behavior [13]. The description of these properties is as follows.

- Self-Configuration is the ability of the system to perform configurations according to predefined high-level policies and seamlessly adapt to change caused by automatic configurations [12]. An ACS must dynamically configure and reconfigure itself under changing conditions [9].
- Self-Healing is the ability of the system to automatically detect, diagnose and repair faults [12]. An ACS must perform failed components detection and eliminate or replace it with another component without affecting the operation of the rest of the system. In addition, it must predict problems and prevent failures [9].
- Self-Optimization is the ability of the system to continuously monitor and control resources to improve performance and efficiency [12]. Thus, it is the capability of maximizing resource allocation and utilization for satisfying user requests. Resource utilization and workload management are two significant issues in self-optimization. An ACS must identify and detect attacks and cover all aspects of system security at different levels such as the platform, operating system, applications, etc. [9].
- Self-Protection is the ability of the system to proactively identify and protect itself from malicious attacks or cascading failures that are not corrected by self-healing measures [12]. It must predict problems based on sensor reports and attempt to avoid them [9].

Apart from the above-mentioned characteristics, which are considered major, there are also a number of minor characteristics that define an autonomous system. An ACS needs to be self-aware, i.e. to be knowledgeable about its components, its current status and available resources. In addition, it needs to know all relevant information about the resources and their usability. An ACS must also be aware of the environment in which it is executed, enabling it to respond to any changes in requirements and conditions, which is called context-awareness. Openness means that an ACS must operate in a heterogeneous environment and must be portable across multiple platforms. Finally, an ACS is characterized by information hiding, meaning that end users are not aware of its inner complexity and adaptivity implementation. A reference model of autonomic element, which is an element of a system with autonomic characteristics as described above, is depicted in Fig. 1.

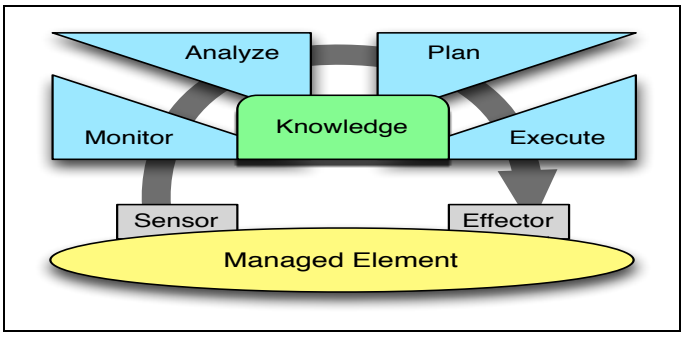


Figure 1. The reference model of autonomic element (MAPE-K Loop).

Other important aspects of AC are the extent to which it is applied to a system and the different ways in which it can appear. The first distinction is usually found in literature as

weak adaptation versus strong adaptation [14]. Weak adaptation (or self-adaptation) refers to the modification of parameters, selection of proper behaviours and interaction protocols at the component level. On the other hand, strong adaptation (or self-expression) refers to the bottom-up modification of the structure of components and possibly the emergence of new, unplanned characteristics during runtime.

The second distinction regards the level at which adaptation takes place. Either identified as tight-coupling versus loose-coupling [9], externalization versus internalization [12] or individual versus collective adaptation [14], this property addresses whether adaptation in a system is centralized, inherent to the system itself or distributed to each of its components, analogously to orchestration versus choreography. The level of adaptation indicates the selection of the appropriate techniques and methodologies that should be utilized.

2.4 Summary

To sum up, in this chapter there was a synopsis of the main concepts and notions regarding the three pillars of research relevant to our work. Thus, we presented cloud computing as an emerging paradigm, which has revolutionized the way computing resources are delivered. We explained the advantages of cloud computing delivering resources as services over the internet and we provided the most commonly accepted definition of this model.

Subsequently, we analyzed the specific features of spatio-temporal applications and stressed their importance for the management of real-life circumstances. We identified

important challenges within the development of such applications and discussed about the existing research in the field which covers mainly the data modeling aspect.

Finally, we introduced autonomic computing and analysed its properties. We explained the self-* characteristics of an autonomic element and depicted its model. In addition, we discussed about different implementations of this paradigm and identified the points of distinction among them.

Chapter 3
Related Work

In this chapter we present existing work in solutions to vehicular sensing for traffic monitoring and road quality assessment as well as in autonomic mobile applications, in an attempt to relate our research contribution to existing approaches in our field of research. In Section 3.1 we present related work for traffic monitoring and road quality assessment while explaining how our architecture differs and what advantages it offers. In Section 3.2 we present related work on adaptive methods in mobile applications, aiming to point out their relationship to our work and how our adaptive methods for mobile spatio-temporal applications extend existing ideas and frameworks. Finally in Section 3.3 there is a summary of this chapter.

3.1 Application Architectures for assessing traffic and road quality

Several solutions have been proposed as vehicular sensing systems for traffic monitoring and road quality assessment. In this section there is a short review of the closest approaches to our traffic application use case, most of which are intended for pothole detection. These solutions are based on the use of sensing hardware such as accelerometers, which can be found in almost every modern smartphone and their functionality can easily be extended beyond the detection of road anomalies. The first systems that paved the way towards effectively facing this problem used specific

hardware and software platforms designed to collect acceleration data in addition to GPS-sensed geolocation data, such as the BusNet system [15] and the Pothole Patrol system [16]. However, it was soon realized that the deployment of such systems on the increasingly popular smartphones would have many advantages such us leveraging the potential of crowdsourcing and high scalability. The following systems are using smartphones to sense the condition of streets similarly to our solution.

Mohan et al. [17] introduce the Nericell system, which addresses the challenge of monitoring road and traffic conditions using accelerometer, microphone, GSM Radio and GPS sensors available in smartphones. They also refer to the need for an aggregation server that collects the information acquired by mobile users but defer it for future work. Their approach assumes users holding their smartphones in arbitrary, continuously changing orientation and they define an analytical algorithmic methodology to virtually reorient the axes of the phone according to the vehicle's orbit. They use two interchangeable threshold-based heuristics to detect bumps, z-peak or z-sus depending on the vehicle's speed. In addition, they use "trigger sensing" for energy efficiency, which is the activation of resource intensive sensors only when certain conditions are met, as measured by the constantly active energy efficient sensors. This framework might have been one of the first to suggest mobile phones as sensors to detect road quality instead of specialized hardware but it does not explicitly describe a complete system architecture. What is more, the methodology for virtual reorientation is rather complex and possibly

prone to noise errors and the system has only been tested on a windows mobile OS, failing to address the need for cross-platform interoperability.

Bhoraskar et al. [18] introduce Wolverine, a system similar to Nericell [17] but with different algorithm based on smartphone's magnetometer to virtually reorient the coordinate axes of a disoriented phone and Machine Learning techniques to identify bumps and breaking events. They develop a mobile application for the Android Platform to test their system and use mean and standard deviations in the three coordinate axes (μX,μY ,μZ,σX,σY and σZ) over one-second windows of the collected acceleration data, after manually labeling them. In the same fashion as Mohan et al. [17], they do not provide architectural or infrastructural information about their proposed solution and mention the development of an application only for one specific platform. They also use machine-learning techniques for assessment, which makes seamless migration to other platforms even more challenging.

Mendis et al. [19] also identify the benefits of participatory sensing to ensure road surface quality. They envision their system as an added layer to existing navigation systems for advanced real time event detection with simple resources such as a modern smartphone. They implement their system on an Android platform and try three different heuristics for acceleration-based pothole detection: Z-THRESH which is similar to Nericell [17] z-peak, Z-DIFF which detects fast changes in vertical acceleration data and standard deviation of vertical axis acceleration (STDEV(Z)). Through testing and based on prior manual detection of potholes, they define the optimal configuration parameters for

successful pothole detection for each of the above-mentioned algorithms and indicate Z-DIFF as the most effective one. Like the above-mentioned approaches, the authors do not provide any reference to a proposed architecture for their system. Moreover, they do not mention wireless transmission of data to a server, leaving many open issues about data gathering.

Aksamit et al. [20] propose an approach to use mobile phones of large number of individual, anonymous car drivers and transmit sensed acceleration data wirelessly over the Internet. They use HTC Desire S mobile phone for testing and they keep the phone in the pocket and on the dashboard while being in a moving car. They use the power of the total acceleration signal to assess road quality and they transmit 12B frames of just the users' real-time coordinates and the power of acceleration signal to the server. Their approach not only eliminates the effects of device rotation but is also resistant to noise and false positives due to the large amount of independent sources of data. Despite the fact that this approach is one of the first to explicitly recognize the power of crowdsourcing to assess the condition of roads, it uses a rather simplistic approach to minimize resource utilization, which might be unnecessary considering modern smartphone's capabilities. Furthermore, like the above solutions it suffers from platform lock-in as it has only been tested using one specific platform.

Ghose et al. [21] also present a system collecting data from multiple mobile users over the Internet. They claim that their solution, which is based on open standards, is improved compared to previous attempts regarding energy efficiency, privacy and

usability. It is phone-orientation-agnostic, it uses in-phone analytics and pre-processing in order to minimize the data volume that needs to be sent to the server and as a crowdsourcing application, it provides more accurate results. Regarding the heuristics, it calculates the derivatives of the measures constantly coming from device accelerometer and estimates an event's confident score for specific location that is sent to the server through a REST-ful API. The system also includes a geo-spatial database and asynchronous backend aggregation processing for data fusion. Thus, data is uploaded to the server in a user-transparent manner and user is notified when approaching road anomalies. The authors claim that their solution can be easily deployed as an "app" on popular Smartphone platforms such as iPhone and Android but do not clearly present the relationship between the application and the data tier. In addition, the authors might claim low battery consumption because of pre-processing and lower transmission rate but pre-processing itself could cause high levels of battery consumption and CPU utilization.

To our knowledge, our application consists of unique features compared to other proposed solutions. Main advantages include the capabilities provided by our proposed three-tier architecture such as adaptiveness and robustness. Our architecture allows us to test, configure, deploy and manage each of the tiers separately and enables a seamless deployment of both data and logic tiers on scalable infrastructures. Moreover, our solution includes location-awareness components which can manage the transmission of user data to specific servers according to their location and smart monitoring metrics, providing more control and reliability to such inherently distributed systems.

Furthermore, it is a true cross-platform solution using open source libraries and PhoneGap to compile pure JavaScript and HTML5 code into native mobile applications. Thus, it can be developed centrally and deployed as a usable native application to any popular platform without the need of any supplementary hardware or software components. Using adaptive methods combined with configurable queuing and local storage on the device, users can send real time data reliably to the server-side, which according to the features mentioned can support a large number of concurrent users.

3.2 Autonomic Mobile Applications

The importance of adaptivity in highly dynamic mobile applications was early realized [24][25]. The improvements it can bring to performance, user experience and modifiability are apparent and have been experimentally verified [24][25][26].

Neophytou et al. [24] advocate the importance of adaptivity in the context of mobile wireless networks. Focusing on service reliability and maintenance of quality of service throughout environmental changes, they propose a QoS adaptation framework which can adapt to changes in network topology or application requirements. The idea is to use methodologies such as QoS degradation and application reconfiguration in order to avoid termination or blocking of service, while maintaining QoS characteristics according to the availability of resources. They simulate the use of their framework for different types of applications on a UMTS network and show that a certain level of QoS can be maintained while probability of service block or termination can be significantly reduced. As will be explained later in this document, in our solution we also adopted the idea of

maintaining QoS requirements while satisfying the goal for service availability. Thus, we consider context awareness as a soft goal while focusing on maintaining low end-to-end response time.

Similarly, Paspallis et al. [25] encourage the use of adaptivity in mobile applications, which can result in improved experience as perceived by users. They argue that separating the concerns of functional requirements implementation and enabling of adaptive behavior, can foster not only simplicity in development but also efficient resource utilization in distributed heterogeneous environments. They introduce and test a component framework which uses Java annotations to dynamically create links between various components with different roles and adapt to changes in the environment. Their approach is very interesting, defining functional and qualitative roles for the different components. However, the current popularity of paradigms such as Cloud Computing has shown that more focus should be on scalability and responsiveness rather than the ability of the application to restructure its components.

Maciel et al. [26] also promote the separation of concerns in mobile applications adaptivity and they propose a middleware which abstracts control functions in a centralized fashion. They take under consideration the distinct characteristics and constraints of mobile applications and follow an aspect-oriented approach to deal with crosscutting concerns. The modular architecture that they propose is capable of providing adaptivity for different important attributes for mobile applications, such as connectivity, power and memory utilization, and security. An adaptivity manager is responsible for

monitoring, analysing and propagating adaptation while implementing preferred policies. This is very similar to our approach, where we use autonomic managers to provide adaptation such as scaling and localization. Nonetheless, our solution is not implemented as a middleware but using integral features of the deployment environment, which can result in smaller overhead in terms of running time and memory.

A recent work by Pascual et al. [27] provides a more mature perspective of the application of Autonomic Computing principles to mobile applications. They define a Context Monitoring Service and a Dynamic Reconfiguration Service to implement MAPE-K loops on a middleware for mobile applications with the innovative feature that the adaptation planning can dynamically change at runtime. Analogously to the above-mentioned works, they identify variability in requirements and propose the use of *Feature models* to define different significance of interacting features. Their plan generator uses a genetic algorithm to dynamically reconfigure the application in order to adapt to changes in context such as battery, network, location, memory e.t.c.. This work was reviewed after the implementation of our solution and despite the fact that it focuses on client-side adaptation, it seems to identify many of the important concerns in this field as we did, such as limited resources, conflicting requirements and the need for dynamic autonomic management which can be modified at runtime.

3.2.1 Context aware Mobile Applications

The following two approaches concentrate on the context awareness aspect of mobile applications, enabling them to adapt to changes in context such as location, resource utilization and activity.

Mowafi and Zhang [28] present a conceptual framework for user-centered context awareness in mobile applications. Using user's transactions as implicit contextual input, their approach reduces the dependencies on environmental context acquisition which can insert significant ambiguity and data intensity. Thus, users can have their personalized context profile based on their self-defined environment (location, activity e.t.c.) which reduces the problem space per user. This attractive idea simplifies the procedure of context acquisition, but requires users to willingly share their behavioral information, which might not be obvious in practice. Nevertheless, the authors provide a conceptual model for context-aware mobile artifacts, identifying among others the importance of location-based adaptivity, similarly to our work.

Likewise, David et al. [29] recognize the importance of context-awareness for mobile applications and introduce a context-aware middleware with multi-modal user interface. Instead of architectural concerns, they emphasize on the need for easy development and maintenance of configurable mobile applications. Their solution enables the runtime configuration of applications automatically or programmatically, selecting the most suitable among available alternative implementations of the same context-based functionality. Through a use case they show that, from a development perspective their

middleware simplifies the implementation of context-awareness as opposed to native applications alone. However, the authors do not specify which exactly is the context for which their methodology can be applied, leaving many open questions about their framework use.

3.2.2 Energy Efficiency

Our research on mobile application adaptivity revealed that among other resource utilization concerns, energy efficiency is one of the dominant aspects that drives research in this field. In this subsection there is a review of approaches on using adaptive methods to deal with power consumption of mobile applications.

Mizouni et al. [30] propose an architecture where both the client and the server collaborate, using battery status as feedback, to adapt and increase the lifespan of the battery. By prioritizing tasks and adapting to power levels they span the service from high user experience to energy saving mode, providing different features at the application transparently to the user. They also introduce the notion of distributed adaptation decisions, where server and client have some shared knowledge about the status of the monitored element, which is something that we have also used for our solution where we have both server and client adaptation as will be explained in more detail in Chapter 5.

In a similar fashion, Misra and Lim [31] introduce the ACQUA framework, which optimizes event based data acquisition from mobile devices for energy efficiency. They use a sequencing algorithm with impressive results, which filters sensed or other type of

event-based data prior to transmitting them to the server thus reducing the overall volume of wirelessly transmitted data. In our work for our use-case mobile application, we also recognized the high energy cost of wireless transmission and therefore tried to keep it as low as possible by batch transmission and by not collecting data when users are not moving. However, in the future we could integrate the idea of dynamic reconfiguration of the cost functions based on continuous data retrieval and evaluation.

In the same direction, Alnawaiseh and Abdelghany [32] propose an adaptive algorithm for keeping energy consumption to desired levels while at the same time maintaining high performance in tracking applications. Their framework is based on ad-hoc networks at which one node has knowledge and can make predictions of the position and speed of all other nodes within its communication range. Instead of passive sensors which collect data and transmit them to a centralized server, they focus on the organizing of sensors in groups, obtaining discrete roles and through simulation they show significant benefits in energy consumption. This approach can also be considered for future implementations of our application, with the possible drawback that it heavily depends on users' sense of community.

3.3 Summary

In summary, in this chapter we reviewed work related to our main contributions: adaptable architectures for mobile applications for traffic and road condition monitoring, and adaptation mechanisms for mobile applications. Thus, we presented the main aspects of existing architectures and we identified similar ideas as well as the differences of our

architecture. We highlighted the main benefits of our approach including the multiple layers adaptivity capabilities that it offers. For the adaptation mechanisms part, we reviewed existing approaches towards adaptivity, context-awareness and energy efficiency for mobile applications. We explained how our approach was inspired by related work and we recognized the ideas that we adopted and the ability of our proposed architecture to support the adaptations mentioned, as will be better shown in the following chapters.

Chapter 4
An Adaptation Enabling Architecture for Mobile Spatio-Temporal Applications

This chapter presents the architecture that we propose for mobile spatio-temporal applications. The goal of this architecture is to take under consideration the specific nature of such applications and provide adaptation capabilities in a dynamic context. Thus, the three-tier architecture proposed in this chapter offers a high degree of modifiability, enabling the application of various methods in order to efficiently maintain desired QoS properties.

The next sections are organized as follows: first we introduce the requirements for the above architecture in section 4.1; in section 4.2 we present our proposed architecture and explain its advantages and how it meets the requirements; sections 4.3 and 4.4 present our case study and empirical results that we obtained about its performance, respectively; finally, we present a summary in section 4.5

4.1 Requirements

The central idea behind an architecture specific to mobile spatio-temporal systems is to capture their distinct characteristics and support a high degree of modifiability and (self-) management.

The requirements that our application architecture should satisfy are the following:

- Data should be transferred easily and reliably over different application components in a real-time basis. This requirement derives from the fact that the class of applications that we are examining are responsible for managing a large amount of complex spatio-temporal data.
- The application should be agile, elastic and resilient to varying client traffic arriving from different physical locations. That is because an additional aspect of dynamicity is added to these already dynamic applications due to the fact that clients are mobile. Hence, application components should have the capability of being added, removed, replaced and modified according to demand. This aspect also dictates the use of location-awareness keeping under consideration not only improvements in performance but also the user experience.
- Our architecture should support the distribution of functionalities with infrastructures facilitating heavy computation on the server side. The reason is that primal information derives from mobile devices with possibly limited capabilities.
- Our application should be designed focusing on efficient resource utilization and service localization aiming at cost efficiency. Techniques such as on-demand acquisition of resources and context-aware adaptivity should play a major role in our design. To this end, our architecture should make feasible the realization of the application using emerging technologies and frameworks which provide opportunities towards that direction.

- The application should be robust and secure. Spatio-temporal applications are particularly sensitive to rapid changes in time and space but remaining available is one of their crucial requirements. Our application should be able to handle unexpected events and be fault-tolerant, considering that it is intended for a large number of independent and diverse users. For the same reason, it should be able to provide secure transactions and prevent unauthorized access to user-generated data.

4.2 The Architecture

This section describes the three-tier architecture that we propose for mobile spatio-temporal applications. The main reason for using a three-tier architecture is that the application consists of concrete components with concrete functionality and can easily be considered in a modular design.

Apart from conceptual integrity, this architecture allows adaptivity throughout all angles of our framework. It also provides high degree of modifiability, as any tier of the application can be independently altered or replaced without critical implications on the other tiers. For example, the Logic Tier can be upgraded by moving the servers to an elastic cloud infrastructure and the Client Tier can be adjusted with different user interfaces of varying granularity adapting to users' needs and platform diversity. Furthermore, this architecture allows to test, configure, deploy and manage each of the tiers separately.

In addition, the need for processing of an enormous amount of real-time data and the diversity of the hardware participating in this application dictate the use of a centralized architecture.

Fig. 2 shows the architecture – tiers, components and connectors – of the application, as will be explained below.

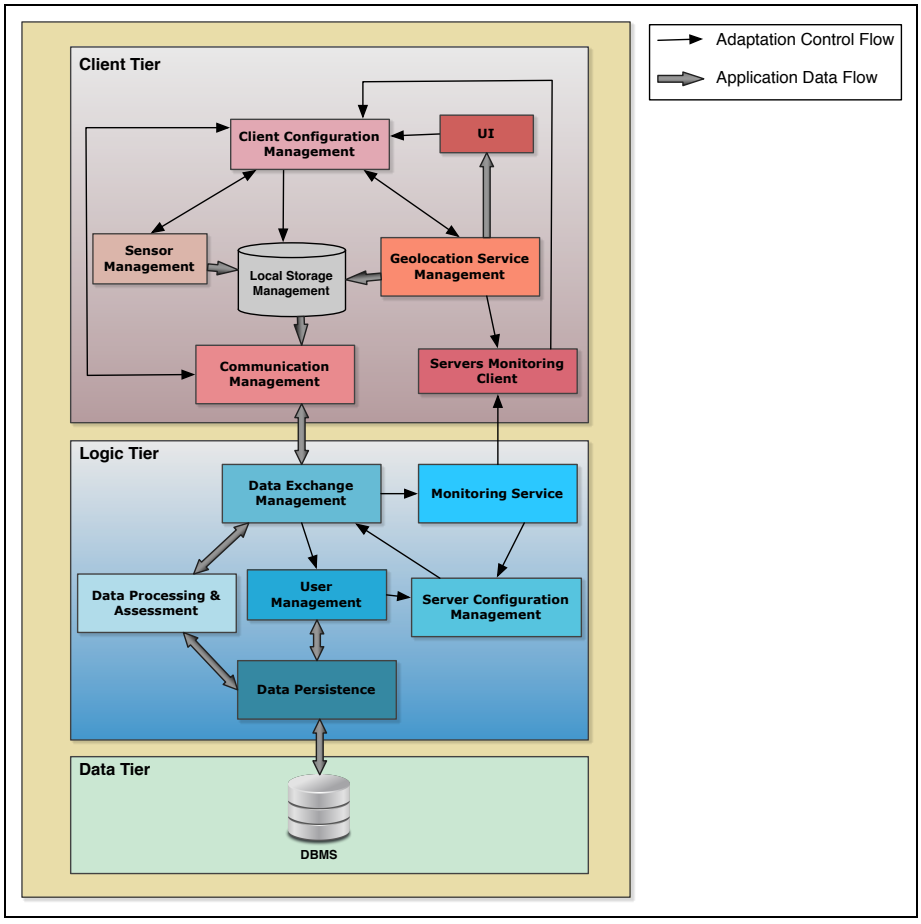


Figure 2. Application three-tier architecture.

The Client Tier consists of the users' view and controls. In our case, users are the mobile web clients.

Mobile web clients are the thin clients of our application, who are transmitting through mobile devices among other data, information about their current location and other sensed information such as speed and 6-axis acceleration. These clients are mobile and integrated to the application using wireless connection. This tier can be enriched with more functionality on the mobile application, providing services such as routing and annotated information on maps on the mobile screen. Main Components of Mobile Clients include:

- UI: User Interface is responsible for user configurations and application interaction. Therefore it sends user-inserted configuration options to the Client Configuration Management component. It also displays information about user's current position using map graphics and thus it receives such information during runtime from Geolocation Service Management component.
- Client Configuration Management: This is the most important component for the implementation of runtime adaptation on the client side. It receives feedback from:
    - UI component, about user-inserted configuration options,
    - Sensor Management component, about device resources utilization,
    - Geolocation Service Management component, about user's current location,
    - Communication Management component, about transmission details such as response time,

- Servers Monitoring Client component, about monitoring details for servers such as CPU utilization and area coverage.

It constantly assesses this information, plans and controls the dynamic configuration parameters for the various components. Thus, it controls the configuration parameters for the Communication Management, the Sensor Management, the Local Storage Management and the Geolocation Service Management components, as is described in each component below.

- Geolocation Service Management: Manages geolocation services and facilitates the use of algorithms on top of GPS and device sensors in order to improve accuracy and to ensure optimal mobile device tracking. It periodically sends geolocation information to the Local Storage Management component for storage. It also sends runtime geolocation data to the UI component for display of current position on the screen, as well as to the Servers Monitoring Client and to the Client Configuration Management components to update with user's current position. Its configuration parameters such as the sampling and storage rate are controlled by the Client Configuration Management component.
- Local Storage Management: Manages local storage for caching and for storing data even while connectivity is not available. It is used to implement a queuing data flow, which is described below. It receives data both from the Sensor Management and the Geolocation Service Management components and it is configured by the Client Configuration Management component. It periodically

sends data to the Communication Management component, as will be described in more detail when describing the queuing data flow below.

- Sensor Management: Manages sampling and collecting information from sensors such as accelerometer, gyroscope, camera etc. It is also responsible for collecting information about device resources utilization, such as battery and CPU utilization, which it sends as feedback to the Client Configuration Management component. Its configuration parameters are controlled by the Client Configuration Management component and the information it gathers from device sensors is sent to the Local Storage Management component for storage.
- Communication Management: Receives data from Local Storage Management component and manages bi-directional communication between client and server(s). Configuration parameters such as the transmission rate are controlled by the Client Configuration Management component. The Client Configuration Management component also plans to which server data is transmitted and applies that configuration to the Communication Management component. This functionality is very important for the implementation of location-based adaptivity, which is part of the context-aware adaptivity that we propose for client adaptivity, as will be described in detail in Chapter 5.
- Servers Monitoring Client: Receives monitoring data about running servers from the Logic Tier Monitoring Service. It sends this data as feedback to the Client Configuration Management component and it receives information from

Geolocation Service Management component about current position, in order to scope the received information from the Monitoring Service, for example only about nearby servers.

To this end, mobile devices' local storage plays a very important role not only in coping with unreliable Internet connectivity but also in enabling data collection and data transmission to operate independently. In order to achieve this goal, a producer-consumer pattern is implemented, using a Queuing Flow for data storage and transmission as shown in Fig. 3. Data is retrieved from mobile device sensors and packaged by a Device Sensing Module. At every 'Step' (which may or may not be constant, according to application configuration), one record is added to the device Local Storage. In a FIFO fashion, data is moved from Local Storage to the Message Queue at every 'queueStep'. Every 'sendStep', the oldest item from the Message Queue is transmitted to the server. If the server receives an acknowledgement message, next item is prepared for transmission, otherwise same item is resent. It should be noted that both Message Queue and Local Storage are preconfigured with a maximum capacity to prevent memory leakage. Consequently, in situations where connectivity is lost, data is sent to the Message Queue until it reaches maximum capacity. From there on, data is accumulated at the local storage and is available when connectivity is restored and even in cases where application is unexpectedly terminated.

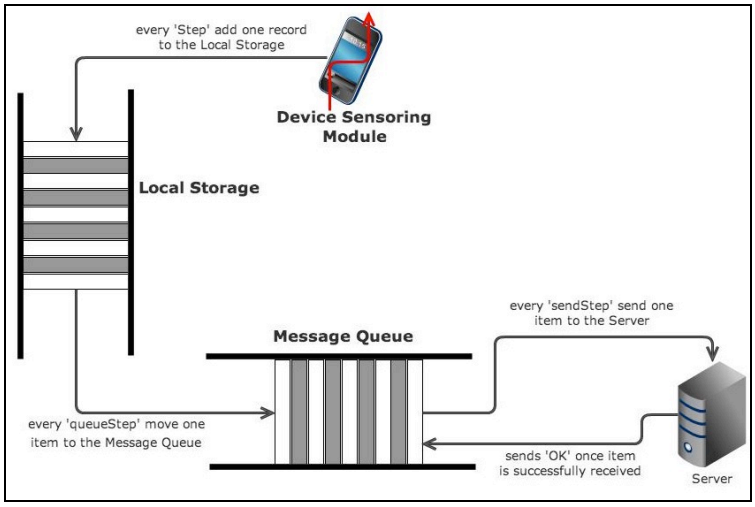


Figure 3. Queuing flow for data storage & transmission.

The Logic Tier includes all the application logic and core processing of the application. It manages the interactions between different tiers, handling the incoming requests from the users and collecting all necessary data. It is also responsible for the processing and assessment of the data and for the monitoring and configuration of running services. Thus, main components are:

- Data Exchange Management: Handles bi-directional transactions with Client Tier Communication Management component. It sends received data from clients to the Data Processing & Assessment component and updates information about users' activity by communicating with the User Management component. It also sends information about its runtime performance to the Monitoring Service. Its configuration parameters such as the geographical areas covered by specific servers, are controlled by the Server Configuration Management component.

- Server Configuration Management: This components is responsible for runtime adaptation on the server side. However, it should be stressed that unlike the corresponding Configuration Management component on the Client Tier, this component does not offer all the server adaptivity capabilities, as we explain in Chapter 5. The reason is that in contradiction to the client adaptation which takes place internally within the application, the important server adaptations such as scaling happen externally outside the application. Thus, this component could be regarded as a controller of partial server adaptation and also as an interface to the external Autonomic Manager, as will be presented in Chapter 5. It receives feedback information about runtime performance of servers from the Monitoring Service component and about user activities and profiles from the User Management component, assesses this information and controls configuration parameters such as geographical area coverage for the Data Exchange Management component.
- Monitoring Service: Gathers runtime information about servers' performance and resource utilization and makes that information available to the Server Configuration Management component and to the Servers Monitoring Client component on the Client Tier, after required data processing and transformation.
- User Management: Manages user information and profiles. It communicates with Data Persistence component to store such information and it also shares user

information with the Server Configuration Management component, providing input for adjustment of services per-user.

- Data Processing and Assessment: Processes raw data and extracts useful results, depending on the application functionality. It is also responsible for data transformation when exchanging data with Data Exchange Management and Data Persistence components.
- Data Persistence: Manages persistence of data for optimal data staging and transformation. It receives data from the Data Processing & Assessment and from the User Management components and communicates with the DBMS for storage. In order to offer the capability of seamlessly switching between different database technologies, this component includes a data layer which abstracts data modelling and functions for interaction with the DBMS.

Finally, the Data Tier is the lowest tier of the application. It is responsible for the representation and storage of the collected information, as well as for the optimal data transaction and archiving. This tier is particularly important for the application, since a vast amount of real-time data has to constantly be not only gathered, but also available for use.

The proposed architecture is suitable for mobile spatio-temporal applications because of the modifiability and adaptation capabilities that it offers. Its modular design allows for decoupling of different functionalities as well as independent management of its various components. Therefore, the application can be configurable and adjustable to different

levels of traffic load coming from mobile users. Furthermore, the different tiers can be managed individually and transparently to each other offering among others the capability of test-based selection of the most suitable technologies for the different tiers of a specific application, for example the best-performing database technology.

In addition, the Mobile Web Client includes the Sensor Management, Communication Management and Local Storage Management components which can be orchestrated in order to ensure reliable sampling and transmission of real-time data. Suitable mechanisms can be applied using these components in order to promote service robustness by modifying configuration parameters on the client device during runtime. This offers advantages both at the client and the server side. The client side can benefit from the optimal utilization and tuning of its limited resources while at the same time the server can avoid situations of overloading or malicious traffic.

Additionally, the logic and data tiers can easily be deployed on elastic infrastructures such as cloud. This fact, combined with location awareness management available on the client tier, enables the application of methods to facilitate server scaling based on demand as well as context-based adaptivity. Such an autonomic adjustment of the application according to traffic distribution can cause significant cost reduction and can accommodate localized services.

Finally, despite the fact that security concerns are beyond the scope of this thesis, our proposed architecture can support availability not only because of its adaptation capabilities but also because it can easily support redundancy. Furthermore, an obvious

solution for secure transactions would be encryption and other authentication methods which can easily be implemented by the communication management components.

4.3 Case Study

In this section we present the application that we implemented in order to validate the feasibility of our proposed architecture. We refer to this application as Traffic Monitoring and Assessment of Road Quality (TMARQ) application. First there is a description of the application functionality and consequently we present the application's implementation details and use.

The application can be implemented using a smartphone that collects data about each vehicle position, speed and 6-axis acceleration and sends the users information about the traffic congestion and road quality. A graphical representation of such an application can be seen in Fig. 4.

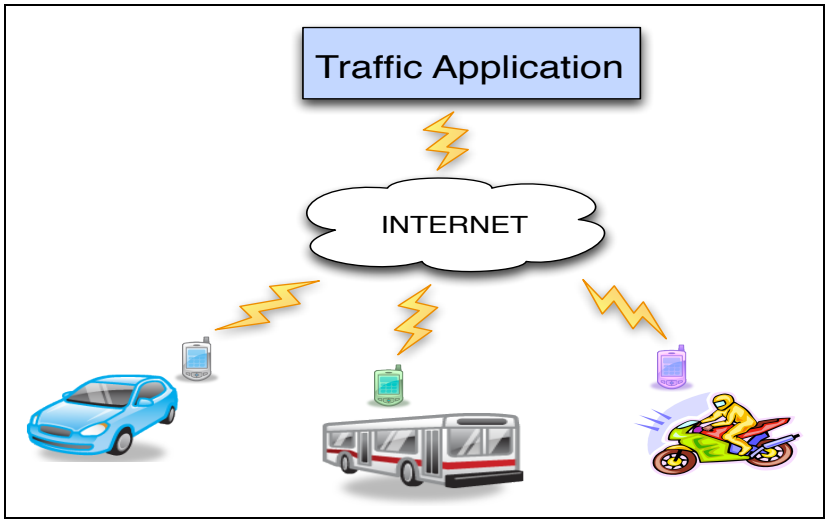


Figure 4. Traffic monitoring & assessment of road quality app (TMARQ).

As can be seen from Fig. 4. the application consists of mobile clients, who are drivers and public transit passengers moving around a city and wirelessly transmitting sensed data through their smartphones over the internet. Data is aggregated and processed in a real-time manner and useful results are constantly being extracted regarding road quality and traffic condition.

It is obvious that such a dynamic application is heavily affected by time and space variables and it requires careful management in order to optimally utilize available resources in a smart fashion. The number of connected users is also a variable, which can potentially reach values of up to hundreds of thousands, if we consider the number of vehicles on the street in a city such as Toronto. It is clear that the volume of real time events that needs to be processed is extremely large. This computationally intensive application also needs to be highly modifiable and adjustable to continuously changing environments.

4.3.1 Development and Implementation

Following is a description of the implementation of our solution; the main concerns and the technologies used. One major issue about the implementation is the real-time collection of data from an enormous number of devices. This in turn requires interoperability between different devices and the server. The design choice to face this challenge was the use of PhoneGap [33] , which is a free and open source framework that allows for the translation of HTML, CSS and JavaScript into native mobile apps for

popular platforms using standardized web APIs. The mobile client application was developed in JavaScript and HTML5 [34], which is a standard, core technology of the Internet, compatible with almost all current mobile devices. Subsequently, it was compiled into native applications for Android, iOS, Blackberry and other platforms using PhoneGap. PhoneGap platform is HTML5 compatible, it is easy to use and provides a powerful set of APIs such as Geolocation and Acceleration. It also provides the capability of advanced local storage on the client side, enhancing the functionality of traditional methodologies such as cookies. That is temporarily stored data are transmitted from the client only when requested. One more interesting feature is that it exposes device APIs, which would not respond from the level of device browsers because most devices have not yet implemented all HTML5 features despite their claims.

Another important consideration is the storage of a large amount of data. Storing information such as the location and acceleration of thousands or million users for a large period, using a traditional, relational database would impose multiple possible threats. Not only would the performance of the queries possibly decrease dramatically with the increasing number of “monitored” devices but also the flexibility of such a database would be fairly restrictive. This solution examines the use of NoSQL storage as an answer to these issues. In specific, the technology tried for the implementation was MongoDB [35], which is a very light, scalable, high-performance and open source NoSQL database. MongoDB provides high performance and modifiability with the apparent trade-off in ACIDity. It facilitates the storage of a large amount of data and

allows for any number of changes in the database schema, thus enabling migration to the cloud, extensions and multiple versioning. Furthermore, there are multiple libraries and drivers for MongoDB, making possible its use through diverse environments and frameworks, such as the one used for this implementation. However, in order to have a complete picture, the application was also tested using a corresponding MySQL DBMS. New technologies enable the migration of both Database types to scalable infrastructures, which is something that will be tested by the authors in the future.

The environment, which was chosen for the development of the Logic Tier of the application, was Play Framework [36] . Play Framework is a Java Web framework which provides a wide set of capabilities on top of traditional Java web development. The main advantage of this framework is maintainability, as it offers the ability to dynamically modify an application without having to bring it down or re-deploy it. In addition, the requirement for additional software or servers such as Tomcat is optional, as it provides a powerful web server which can run autonomously on any node, listening to any port (9000 by default). Traditional J2EE technologies such as Java and JavaScript can be used as is but enriched with powerful scripting capabilities with only as many as few lines of code.

When it comes to the use of the application from mobile devices, a map was considered as the most suitable means of representing information. In this respect, Google Maps API was used, but the use of other map services can also be examined and even a framework

for the optimal selection of map services in a service composition has been considered as a future task.

Regarding data transmission, JSON was used as the simplest model to offer the necessary parsing and editing capabilities. JSON is a popular format and it is supported by most frameworks and programming languages either with the use of existing libraries, such as GSON for Java, or even natively, such as in JavaScript.

With regards to the application logic, it is according to the practices dictated by the platform used, i.e. PlayFramework. The design pattern used for this platform is Model-View-Controller. The Model Component includes the definition of the Data Schemes for the interaction between the Database and the rest of the application and the persistence and exchange of information between different components. The View concerns the client side code, which is written in HTML5 and JavaScript. It has been used for the Administrator View of the Desktop Web Client for the monitoring of the currently connected devices, which is shown on a map, the extraction of user data on '.csv' files and animation of stored routes on a map . Furthermore, the Controllers are in charge of managing the application, including the View Controllers for the rendering of the necessary information to the Views, the Communication Controllers for the implementation of the bidirectional communication with the clients and the File Controllers for the file exchange between servers and clients and communication with the File System. Finally, Database Controllers perform all the queries to the database, based

on the formalizations defined in the Model Component. The Development View described above can be seen in Fig. 5.

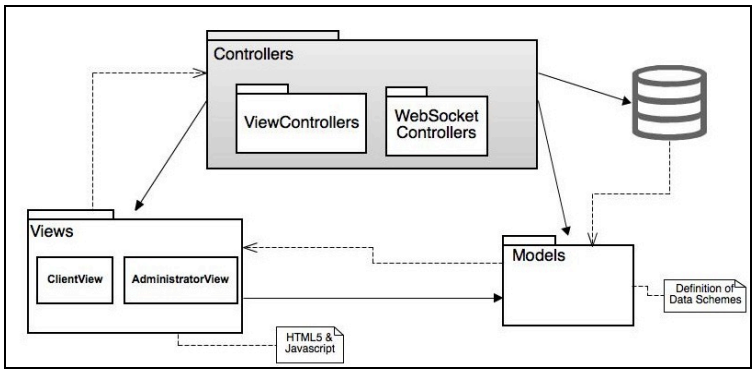


Figure 5. Development view of application server.

4.3.2 Infrastructure Deployment

When it comes to deployment of the application, it consists of multiple nodes. As mentioned above, one important aspect of our architecture is the ability to deploy the application servers on the cloud. For this implementation we used one of the most popular public cloud computing platforms, offered by Amazon.com, i.e. Amazon AWS [23]. Amazon AWS offers many remote computing services, the most popular being Amazon EC2 where a user can use a remote virtual machine (called instance) to deploy her application, and Amazon S3 for online web storage. The most useful features of this platform are scalability according to metrics such as CPU Utilization and monitoring.

For the deployment of our application servers we used AWS Elastic Beanstalk, which is a PaaS service offering a complete deployment environment. It offers many Amazon AWS

services such as Amazon EC2, S3, elastic load balancers and AWS Cloudwatch for monitoring, making the process of defining policies for auto-scaling a simple task. Considering mobile clients, they are deployed as native application on the mobile devices, using of course the cross-platform capabilities offered by PhoneGap. Most of our tests were conducted using Android devices but it is expected that the application would work fine on other popular platforms as well.

As for the Data Tier, current implementation includes either a remote MySQL DBMS or a remote MongoDB DBMS which is deployed on an Amazon EC2 instance, as will be described in the following experiments. The communication between the application servers and the database servers is conducted using corresponding drivers. The above-mentioned deployment is shown in Fig.6.

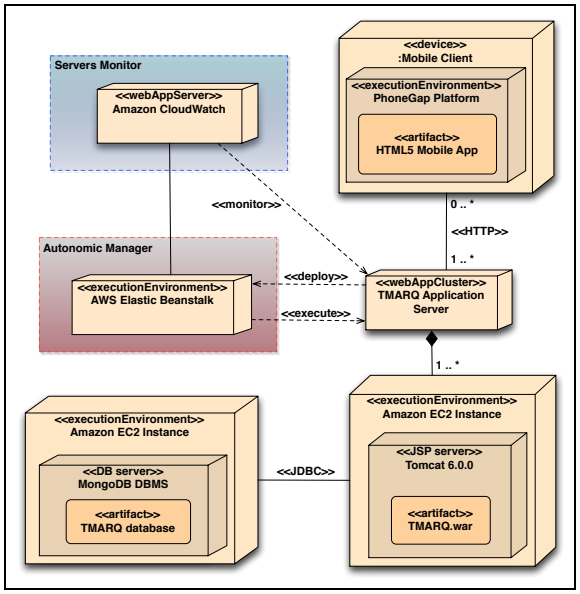


Figure 6. Deployment diagram.

As can be seen from Fig. 6, multiple Web Application Clusters, can contain a number of server instances. Thus, HTTP requests from mobile clients can be sent to either of those clusters, depending on client's context (e.g. location etc) as well as servers' state (e.g. cpu utilization, cost etc), as described in more detail throughout the client adaptation mechanism in Section 5.3 below. Another important thing to notice is the adaptation capability which is offered through this infrastructure deployment, externally to the application. Amazon CloudWatch serves as the Monitor for the managed elements, which are the application servers in our case and AWS Elastic Beanstalk serves as the Autonomic Manager who can analyze, plan and execute adaptations to the managed elements. Thus, this deployment diagram shows in practice how our proposed architecture can support server adaptation using existing technologies.

4.3.3 Application Use

At this point it would be useful to wrap up the application use and functionalities. A common use case would include a user inside a vehicle launching her mobile application and starting transmission of her location, speed, 6-axis acceleration and some other data retrieved by the smartphone's embedded sensors. This data would be received by a server instance, which would communicate with the Database and store collected data. This would last for as long as the user uses the application. Afterwards, the user could launch the Desktop Web Client from her PC and export the data of her route in a '.csv' file and watch the route being animated on a map. It should also be mentioned that the mobile

application includes a ‘Bad Button’ that a user can press to indicate bad road quality for parts of her route. However this is only for identification and verification of the most proper evaluation function to identify bad road quality and in future implementations bad road parts will be detected automatically. A use case diagram of the application can be seen in Fig. 7.

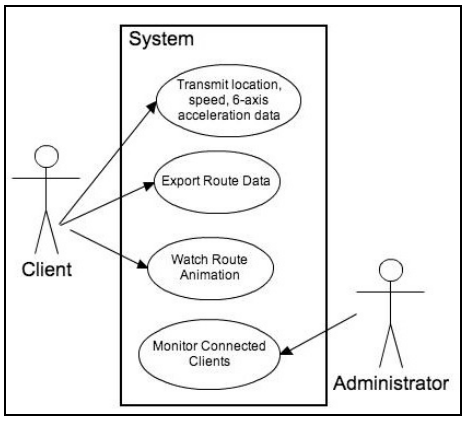


Figure 7. Use case diagram.

4.3.4 Data Collection

In this subsection there is a discussion of the process of collecting and assessing data using our application. First there is a description of the algorithms we use to collect data and determine whether the quality of a road segment is good or not. Subsequently there is a presentation of sample results gathered through our application.

Our testing agreed with the fact that the most "painful" functionality of a mobile application from a power consumption perspective is frequent wireless transmission, as noticed by other researchers (see Chapter 3). In addition, keeping transmitted data to a minimum also has apparent implications on performance and cost, keeping the load of the application as a whole as low as possible. However, it is important for transmitted data to contain enough information in order to provide adequate input for assessment. Thus, there is an obvious trade-off in performance and quality of transmitted data.

Our implementation choice to tackle this issue was the use of a regression algorithm at the client side. In this respect, we sampled data for every new sensed value but we only kept statistical indicators (mean value, standard deviation and so on) for specified time intervals. We stored those values at the device local storage and we transmitted a set of such values to the server at predefined intervals. In addition, we paused the sampling of data at situations when user's position did not change, avoiding the collection of useless information.

When it comes to road quality assessment, we noticed that acceleration was not a sufficient indicator for road anomalies. Instead, our testing showed that whenever there were many bumps on the road there was big and sudden change in acceleration. Therefore, we used an acceleration forward operator (jerk) to indicate bad quality of road segments with satisfactory results. It is also important to notice that the quality of our results heavily depended on the physical direction of the mobile device and for that

reason we conducted our testing trying to keep the device on the same direction. Sample gathered data from mobile devices can be seen in the following graphs in Fig. 8.

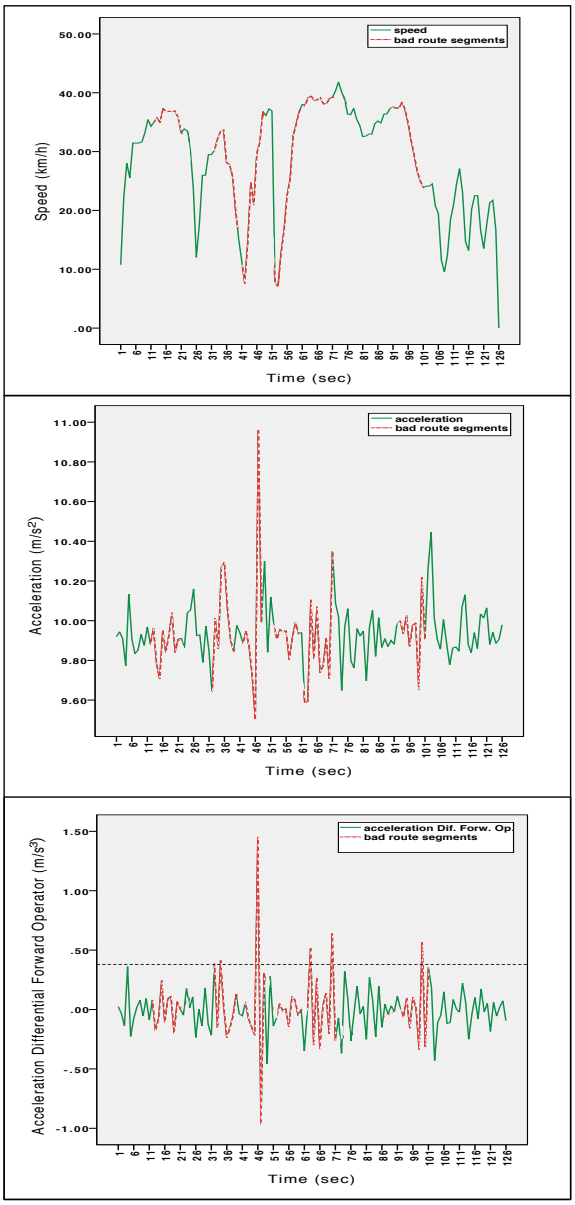


Figure 8. Speed, acceleration and jerk over time.

In these graphs we can see speed over time as measured by the application, acceleration over time and acceleration differential operator over time. The values for acceleration are calculated as the square root of the sum of squares of each dimensional acceleration as such:

- $a = \sqrt{a_x^2 + a_y^2 + a_z^2}$,

  where a_x, a_y and a_z are the measured by the application accelerations for the x, y and z axis correspondingly.

When it comes to acceleration differential forward operator for each point, we define and calculate it as such:

- $a_{diff} = \frac{a_{i+1} - a_i}{t_{i+1} - t_i}$,

  where :

  - *a_{i+1} and a_i are the next and the current estimated acceleration respectively and*
  - *t_{i+1} and t_i are times to which these acceleration values correspond.*

In Fig. 8 the red portions of the graphs represent segments during which the "bad route" button of the mobile application was pressed, which was the input of the tester indicating that the quality of the road at the corresponding segment was bad. As we can see from the graphs, by setting a threshold to the appropriate level, we were able to identify 4 of the 6 bad road segments, using the acceleration differential forward operator with 4 true

positives and 2 false negative, providing us with accuracy of 60% and precision of 100% for the specific example.

It would be safe to assume that for this application, for practical reasons precision is more important than accuracy, as it is adequate to indicate the road segments with the worst quality which require immediate attention. However, it should be noticed again that the purpose of these experiments was not to come up with the best possible algorithm for this application but instead to show the feasibility of our open architecture to facilitate the implementation of such applications. In this respect, more experiments should be conducted in the future to determine the optimal threshold level and methodologies towards this goal.

4.4 Summary

To sum up, in this chapter we described the architecture that we propose for mobile spatio-temporal applications and the proof of concept of its implementation. As we explained, this 3-tier architecture supports adaptations in multiple layers and offers a high degree of modifiability.

First we presented the requirements for this architecture and the rationale behind them. We identified adaptivity, agility, reliability and cost-efficiency among the most important requirements, based on the nature of this class of applications and we highlighted the opportunity of using emerging technologies to satisfy these requirements.

Subsequently we described the proposed 3-tier architecture and its various components, stressing the main advantages that it offers. In this respect, we attempted to show how

this architecture satisfies the stated requirements and we noted how its modular design can support adaptivity at many layers. We also explained how proper orchestration of its various components can promote robustness and resiliency and we stressed the cost-efficiency it can offer due to the scaling and resource management capabilities that it offers.

Next we introduced our case study application that uses common smartphones to assess the traffic and the condition of roads. After explaining its use and the technologies that we used to build and deploy this application, we presented some graphs from field observation by testing the application and we proved the feasibility of successfully implementing this application using our open architecture, which offers the benefits mentioned in this chapter.

Chapter 5
A Two-Levels Adaptations Architecture

As mentioned in the previous chapters, one of the key foci of our proposed architecture is adaptiveness. In this chapter there is a description of the adaptation mechanisms employed by our architecture.

Our proposed framework supports adaptation both on the server and on the client side. On the server side it supports the adaptive scaling and configuration of distributed server clusters according to changing demand. On the client side, it facilitates the routing of requests according to user's current location and the optimal client configurations according to availability. Both of these adaptation mechanisms are described more thoroughly in the following sections.

In Section 5.1 we provide an overview of our approach towards adaptiveness, which is the implementation of an adaptation mechanism that dictates the use of both server and client adaptations concurrently. In Sections 5.2 and 5.3 we present more details and the algorithms for each of the server and the client adaptation respectively. Finally in Section 5.4 we summarize this chapter and we restate the contribution of the presented mechanism.

5.1 Overview of Adaptation Mechanism

The practice so far regarding adaptivity in mobile applications has been either to implement adaptive methods on the server side, for infrastructure performance and

robustness, which is usually realized externally to the application, or less commonly on the client side mainly for energy efficiency. We propose an adaptation mechanism based on the 3-tier architecture that was presented in Chapter 4, which uses both server and client adaptivity during application runtime. This adaptation mechanism dictates the use of two MAPE-K loops – one on the server and one on the client, as shown in Fig. 9.

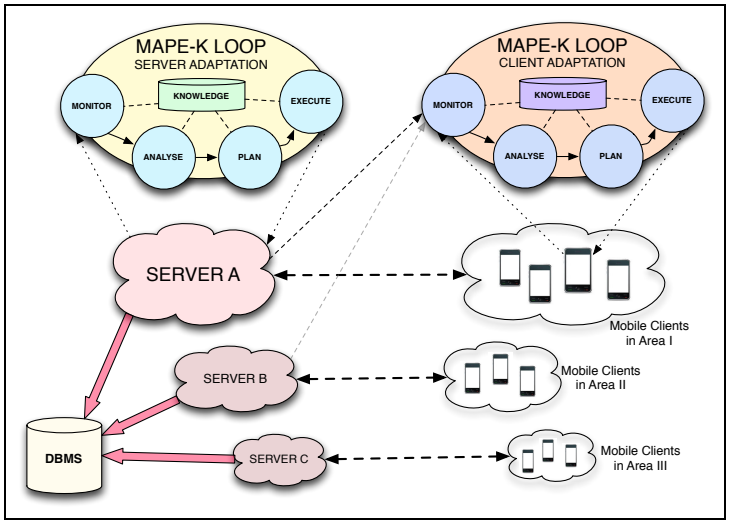


Figure 9. Adaptation mechanism for 3-tier architecture.

The way that this adaptation mechanism works is that the client and the server adaptation run independently and concurrently. The server adaptive loop constantly monitors the state of a server and executes required runtime adaptations about scaling and modification of configurations according to defined policies and rules. Scaling and other infrastructural adjustments can be realized externally to the application, using existing tools as they will be explored in Chapter 6. The client adaptive loop does the same for

every mobile client in order to adjust configuration parameters such as sampling and transmission rates, but it also receives information about the state of running servers as feedback in order to analyze and plan the routing of requests from that client, based on her current location. It is important to notice that unlike server adaptation, client adaptation can be realised exclusively internally to the application, using the architectural components of the Client Tier as they were introduced in Section 4.2.

As an illustrative example, let us consider the case of a spatio-temporal application where one mobile client, client_A directs her requests to server_A, because that is the geographically closest server. Client_A samples and transmits spatio-temporal information to server_A at rates that are configurable based for instance on her device resource utilization and response time from the server. At some point let us assume that client_A changes her location and is now closer to another application server – server_B. The adaptive loop on this cient would monitor this change and the result would be the redirection of requests from that point on to server_B. At the same time, the adaptive loop on the client constantly receives partial state information about both server_A and server_B for instance about their resource utilization.

Subsequently, let us consider that at some point requests at server_B arrive at higher rates, causing the adaptive loop on server_B to execute its scaling in order to cope with the increase in traffic. In practice, scaling does not happen instantaneously, so for some time the adaptive loop on client_A would sense that utilization on server_B is high, and

that could cause the redirection of her requests back to server_A until utilization on server_B would drop.

The advantages that this mechanism offers, as compared to using only one of either server or client adaptivity are apparent. To begin with, as it was illustrated through the previous example, despite the fact that the two adaptations run independently, their combination is synchronized through shared state information about the servers, and this can foster stability for the system. In addition, it allows for flexibility and prioritization of goals, since the requirement for context-aware adaptivity can be configured to take place only when the satisfaction of more important QoS characteristics is not threatened for instance due to unmanageable traffic. Furthermore, this conditional satisfaction of non-functional requirements offers the ability for optimal utilization of resources and therefore cost-efficiency.

The deployment diagram of the TMARQ application in Fig. 6, as presented in Subsection 4.3.2 is one example of the deployment of our architecture enhanced with the above-mentioned adaptation capabilities. To clarify where adaptation takes place in this example, client adaptation is realized internally to the application at the mobile clients by components of the Client Tier while server adaptation takes place both at the web application servers and externally using Amazon CloudWatch service as a Monitor and AWS Elastic Beanstalk as an Autonomic Manager. In specific, AWS Elastic Beanstalk is responsible for the server-side scaling and internal components of the Logic Tier are responsible for the application of configuration parameters including the geographical

area covered by each cluster. In the following sections we have more detailed description of the server and the client adaptation separately.

5.2 Server Adaptation

An integral aspect of our proposed architecture is the ability to dynamically adjust to changing conditions, while maintaining important QoS characteristics such as availability and cost-effectiveness. On the server side, this is ensured by employing the adaptation architecture depicted in Fig. 10, inspired by the work of Woodside et al. [22].

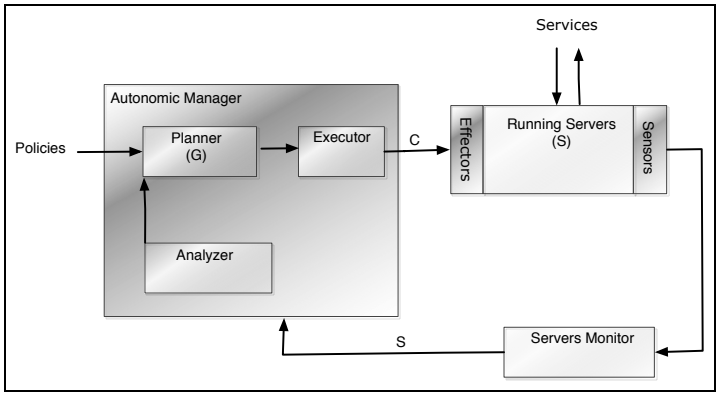


Figure 10. Server-side adaptation architecture.

As can be seen from Fig. 10 , the system is composed of three main components: Running Servers, Servers Monitor and Autonomic Manager.

Running Servers are the managed elements of this architecture and the recipients of requests from mobile users from different geographic locations. At every given moment the Running Servers have a specific state (S). S contains information such as the number of server instances per web application cluster, the geographical area covered by each

cluster, the resource utilization at each cluster and performance metrics such as average latency and throughput. Through sensors this information is provided to the Servers Monitor which is constantly alert, either polling at specific intervals or listening for specific events.

The Servers Monitor passes on this information to the Autonomic Manager, which is responsible for analyzing the current state and acting accordingly to ensure efficiency and quality of service. The Autonomic Manager consists of the Analyzer, the Planner and the Executor. The Planner takes as input the current state (S), the current cost and defined Policies and plans the general configuration and rules for all the controlled servers. It then communicates the decision to the Executor, which propagates the appropriate control (C) to the Effectors of the managed element. An example would be an increase in service requests at a specific area at a specific time, which would cause the increase of resource utilization and thus decrease in the performance of the Cluster responsible for that area. The autonomic manager would then decide for instance to increase the number of server instances of that cluster, improving the service performance. In Table 1 and Table 2 we can see the notation and the algorithm of the adaptation described in this section respectively.

Table : Notation		
SC		Set of server clusters
RI_i		Set of running instances for i^{th} server cluster (SC_i)
N_i		Number of running instances for i^{th} server cluster
A_i		Geographical area covered by i^{th} server cluster
Monitored Data	**L_i**	Average latency of running instances for i^{th} server cluster
	U_{ij}	j^{th} resource utilization metric of running instances for i^{th} server cluster
	T_i	Average throughput of running instances for i^{th} server cluster
C_i		Control over i^{th} server cluster (e.g. "increase running instances by one")
G		Set of general configuration rules/parameters for all server clusters (e.g. "CPUUtilization threshold for all server clusters is 70%")

Table 1. Notation for server adaptation.

The Server Adaptation Algorithm is constantly executed while the application is running, as can be seen from Step 1 and Step 19 in Table 2, implementing the adaptive loop on the server side. Steps 2 - 12 implement the monitoring and the analyzing part of the adaptive loop, gathering and updating information about the monitored data as these are presented in Table 1, iterating through all managed elements and aggregating collected data by using the aggregation strategy that is relevant for each monitored element. Steps 13 and 14 implement the planning part of the adaptive loop by receiving as an input the

monitored data, checking whether adaptation actions are required and planning accordingly.

Algorithm 1 Server Adaptation Algorithm

```
1: repeat
2: //iterate through all server clusters
3: for all server clusters SC_i ∈ SC do
4: //iterate through all running instances of cluster
5:    for all running_instances ri ∈ RI_i do
6: //iterate through all monitored resources for instance
7:          for all monitored_resources j ∈ U do
8:          aggregate(U_ij, <aggregation_type_j>)
9:          end for
10:         aggregate(Ti, average)
11:         aggregate(Li, average)
12:  end for
13:  if action_required(U_ij, T_i, L_i)
14:         C ← plan_control(U_ij, T_i, L_i)
15:         execute(C)
16:  end if
17: end for
18: update(G, current_state(U,T))
19: while application_is_running
```

Table 2. Server adaptation algorithm.

Steps 15 and 18 implement the execution part of the adaptive loop by enforcing required adaptations for each managed server and for general configuration parameters correspondingly.

As mentioned above, server adaptation related to infrastructural adjustments is realized externally to the application. For example in our deployment diagram in Fig. 6 the role of the Autonomic Manager is assigned to Amazon Elastic Beanstalk and the role of the

Servers Monitoring is assigned to AWS CloudWatch. However, adaptations can also happen internally to the application, for example updates of the servers' configuration parameters. These adaptations are managed by the Server Configuration Management component on the Logic Tier, as we described in Section 4.2. This component can also provide the required interfaces for the enforcement of adaptations from sources external to the application.

5.3 Client Adaptation

Adaptation is also possible at the mobile client level. Mobile clients are wirelessly transmitting data to the server at specific rates. There are three incentives that dictate the application of autonomic management at this level.

Firstly, client adaptation can contribute to better application performance by complementing server adaptation and preventing application unresponsiveness caused by resource over-allocation. With a suitable mechanism clients' requests can be redirected to less "busy" servers than the ones they were originally assigned to, which is particularly valuable during transition periods. Secondly, the location, time and activity contexts of the mobile clients are constantly changing. At different times clients might find themselves in different environments, with different connectivity and possibly different rules. Lastly, there has to be a careful consideration of the available resources of mobile clients. Mobile devices nowadays offer limited processing capabilities as compared to

PCs and their battery life is restricted, especially when they perform network-intensive tasks such as real-time transmission using 3G and GPS technologies.

To address the above-mentioned issues we propose an adaptation mechanism which constantly monitors the state of the mobile client. State information at this level includes geographic location, resources utilization such as battery consumption, and service performance and availability metrics as perceived by the client such as response time. According to our mechanism, an integral part of the state information which is available to the client is also the resource utilization of the server clusters which are the potential recipients of client's requests. For example, the client is aware of CPU utilization of every server cluster that is closer to her location. The way this information becomes available to the client is through the Monitoring Service component on the Logic Tier, as was explained in Section 4.2.

Based on the current state, an autonomic manager plans and executes decisions such as configuration of data collection and transmission rate and routing of service requests to suitable application cluster.

Following is the algorithm of the client adaptation mechanism. It is important to notice that this algorithm depends on various functions running asynchronously. Some of them run periodically with possibly different periods and others are triggered by specific events, as shown in Table 4.

Table : Notation		
S_URL		Selected server URL
TR		Transmission rate
SR		Sampling rate
P		Set of defined policies
Monitored Data	**GL**	Geographic location
	CRU	Client resource utilization
	SRU	Servers resource utilization
	R	Average response time

Table 3. Notation for client adaptation.

In Table 4 we can see the asynchronous functions running, as well as their occurrence type and the updated output that they produce. Following in Table 5 we can see the Algorithm of the Client Adaptation as described in this section, expressed in pseudocode.

Asynchronous Function	Occurrence Type	Updated Output
updateLocation	periodical	GL
updateClientResourceUtilization	periodical	CRU
updateServersResourceUtilization	periodical	SRU
updateAverageResponseTime	event-triggered	R
updatePolicies	event-triggered	P

Table 4. Asynchronous functions during client adaptation.

Algorithm 2 Client Adaptation Algorithm		
Input:	Geographic location	GL
	Client resource utilization	CRU
	Servers resource utilization	SRU
	Average response time	R
	Current transmission rate	TR_t
	Current sampling rate	SR_t
	Set of defined policies	P
Output:	updates transmission rate	TR_{t+1}
	updates sampling rate	SR_{t+1}
	updates Selected Server URL	S_URL

1. $(SR_{t+1}, TR_{t+1}) \leftarrow (SR_0, TR_0)$
2. **while** application_is_running **do**
3. $SR_t \leftarrow SR_{t+1}$
4. $TR_t \leftarrow TR_{t+1}$
5. State1 ← analyze(GL, R, SRU)
6. **if** (action_required(State1, P))
7. S_URL ← select_server(State1, P)
8. **end if**
9. State2 ← analyze(CRU, TR_t, SR_t, R)
10. **if** (action_required(State2, P))
11. SR_{t+1} ← update_SR(State2, P)
12. TR_{t+1} ← update_TR(State2, SR_{t+1}, P)
13. **end if**
14. **end while**

Table 5. Algorithm for client adaptation.

The first 4 asynchronous functions from Table 4 implement the Monitoring part of the adaptive loop on the client side. This monitored data is provided as feedback to the autonomic manager for analyzing. At this point we should notice that we have two different subclasses of client adaptation:

- server selection according to geographic location, servers resource utilization, response time and defined policies and

- updating of sampling and transmission rates according to their current values, client resource utilization and response time.

Analyzing for these adaptations is implemented at Step 5 and Step 9 of the algorithm in Table 5. The outcome is two different states, which are used for planning and execution of adaptations at Steps 6-8 and Steps 10-13. After initialization at Step 1, this algorithm is constantly executed as long as application is running, implementing the adaptive loop on the client side.

This client adaptation is realized internally to the application by components on the Client Tier. Specifically, the Client Configuration Management component as was presented in Section 4.2 plays the role of the Autonomic Manager for the client adaptive loop. The monitoring part is implemented by all the components on the Client Tier from which the Client Configuration Management component receives feedback. After analyzing this feedback the Client Configuration Management component executes required adaptations by enforcing them to the relevant components. We should notice at this point that information about Servers Monitoring becomes available to the client from the Monitoring Service of the Logic Tier. At the design of our architecture we decided to have a separate communication channel for the transmission of this information to the client, other than the one for application data transmission, for decoupling and separation of data and control plane.

5.4 Summary

To summarize, in this chapter we presented perhaps the most important contribution of our work, which is the introduction of a two-layer adaptation mechanism for mobile spatio-temporal applications. We explained how our mechanism works by the concurrent application of two adaptive loops; one loop for server adaptation for server scaling and reconfiguration and one loop for client adaptation for routing of requests and updating of device function parameters.

After illustrating through an example what would be a common implementation of our mechanism, we described the advantages of stability, flexibility and cost-efficiency that this mechanism offers and we presented a possible deployment of a three-tier architecture implementing this mechanism. Subsequently, we explained each of the server and client adaptivity seperately and we presented their algorithms in pseudocode.

The advantages of our proposed mechanism will be better illustrated through the adaptation experiments following in Chapter 6.

Chapter 6
Experimental Results

In this chapter we present the experiments that were conducted in order to validate our architecture and proposed methodologies and to produce valuable results about alternative configurations of our application. First there is a description of the experimental environment and the custom workload generator that we developed for the purpose of these experiments. Subsequently, we present the experiments that were conducted with alternative database systems, i.e. MongoDB and MySQL, and we evaluate their use based on obtained results. Lastly, we present the experiments that were realized using different types of adaptation in order to demonstrate the capability of our proposed methodology to implement context-aware adaptation and maintain QoS characteristics such as low latency while reducing resource utilization and thus cost.

The purposes of the experiments presented in this chapter can be summarized as follows:

- Evaluate the feasibility of our proposed architecture.
- Showcase the capability of using and comparing alternative settings and configurations of our application.
- Showcase the value of using our proposed adaptation methodologies.
- Acquire an understanding of our problem space by observing concrete results.

In Section 6.1 we describe the experimental environment and provide concrete details about the cloud technologies used and the functionalities that they offer. In addition, we

describe our approach for the simulation of clients and present our custom Workload Generator and its architecture.

In Section 6.2 we present the first set of experiments according to which we used two alternative database systems; MySql and MongoDB. We performed these experiments in an attempt to showcase the capability of our architecture to seamlessly switch between different technologies for its different tiers. In this respect, we measured some of our application performance characteristics for these two systems and we performed their comparison for our specific use case. In specific, we generated a workload of up to 100 clients and we tested the use of the two database systems when clients send only read requests, only write requests or both. Our results show better results for MongoDB in terms of response time and network traffic.

In Section 6.3 we present the second set of experiments which regards the use of adaptation mechanisms for our mobile spatio-temporal application. We generate a workload of hundreds of concurrent users and show the inability of a static architecture to handle these requests and the need for adaptive mechanisms. In this respect, we present results using specializations of our adaptation mechanisms as presented in Chapter 5. In specific, we present results from experiments using alternative adaptation configurations using:

i. only server adaptation for scaling out application servers,
ii. only client adaptation for directing requests to alternative clusters according to location and server utilization and

iii. both adaptations.

Through these experiments we intend to showcase the capability of implementing adaptive mechanisms on our architecture and to evaluate the use of those mechanisms. Our results show that using both server and client adaptation we can achieve location-aware adaptation while maintaining low response time and reduced number of launched VMs, as compared to using only one of the adaptations.

6.1 Experimental Environment

In order to conduct experiments to showcase the use of our proposed architecture and methodologies, we used the traffic monitoring & road quality assessment app that we built, as presented in Section 4.3. We deployed the application servers on the Amazon AWS [23] platform. This cloud computing platform provides additional components for scaling, monitoring and management which allowed us to perform our different experiments. Specifically, we used PlayFramework [36] to develop our Java-based application servers, which allows to extract standalone applications in a WAR format. We deployed the WAR files of our application server on Apache Tomcat [37] environments running on Amazon EC2 Linux instances.

For the monitoring part, we developed a monitoring client component that reuses the Amazon Cloudwatch monitoring service [38] which is available on Amazon AWS. For our experiments, the monitoring was set at 1-minute frequency, which is the most detailed monitoring that Amazon Cloudwatch is currently offering. Some of the metrics that this service provides include CPU Utilization of the running VMs, network

throughput for incoming and outgoing traffic and latency. Throughout our experiments we also monitored end-to-end response time from a client perspective, using our custom Workload generator, as will be explained in more detail later in this Section.

All components that were necessary for our experiments, including the Database servers, were running on Amazon EC2 instances. However, this was not a hard requirement, as they were communicating with each other over HTTP and could thus be running at different environments as well. Hence, the Database servers were running on remote VMs which gave us the capability not only to configure them independently and seamlessly switch between alternative ones but also to monitor them separately.

In order to generate workload and simulate the mobile clients for the experiments we developed a custom workload generator. This Java-based generator is similar to Apache JMeter [39] but it was developed for convenience. Not only is it extremely customizable making it possible to implement our adaptation mechanisms but also it can be easily integrated with other system components, such as the topology server for monitoring. Nonetheless, in order to verify its functionality we conducted tests which showed that it causes similar results to JMeter. The architecture of this generator is depicted in Fig. 11.

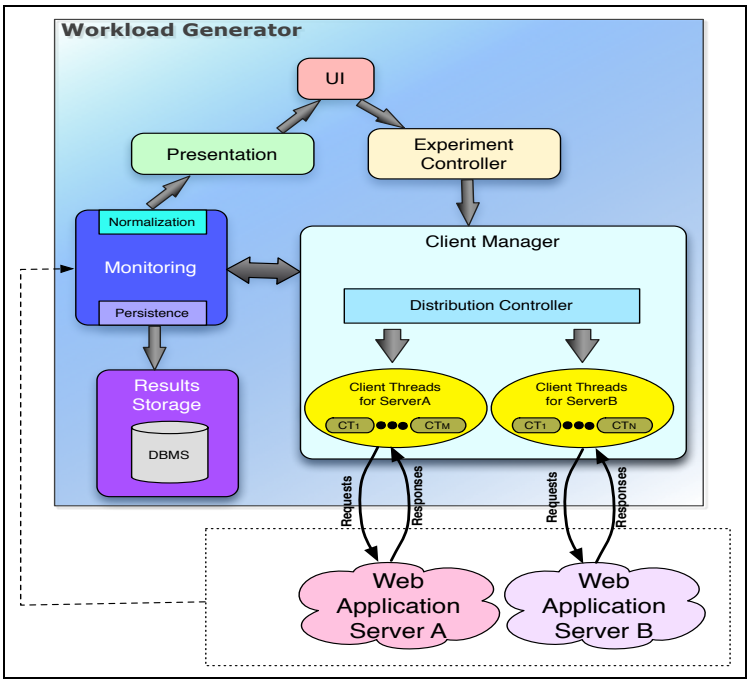


Figure 11. Workload generator architecture.

As can be seen from Fig. 11, the main components of the Workload Generator are the following:

- Client Manager: Manages the lifecycle and distribution of the simulated clients, according to commands and policies it receives from the Experiment Controller. By distribution, we imply direction of clients' requests to specific servers (URLs). For every request sent the clients wait for a response and for every pair, metrics about the time elapsed are sent to the Monitoring component. It also receives runtime information from the Monitoring component which are required to implement policies and adaptation, mimicking application logic.

- Monitoring: Receives runtime information about the application components and response times from simulated clients. After relevant processing it stores this information in an appropriate format and also makes it available for real-time or offline presentation during the experiments.
- Experiment Controller: Initiates and terminates experiments and controls their runtime configuration and parameters. It is responsible for enforcing defined policies to the Client Manager and handling events such as failures.
- UI: User Interface for user configurations and system interaction during experiments, interacting with Experiment Controller. It also offers the functionality of presenting real-time runtime information about the experiment by receiving information from the Presentation component, as can be seen in Fig. 12.
- Presentation: Receives information from Monitoring components and handles formatting and transmission of that information to the UI.
- Results Storage: Handles storing of experimental results for persistence and future reference.

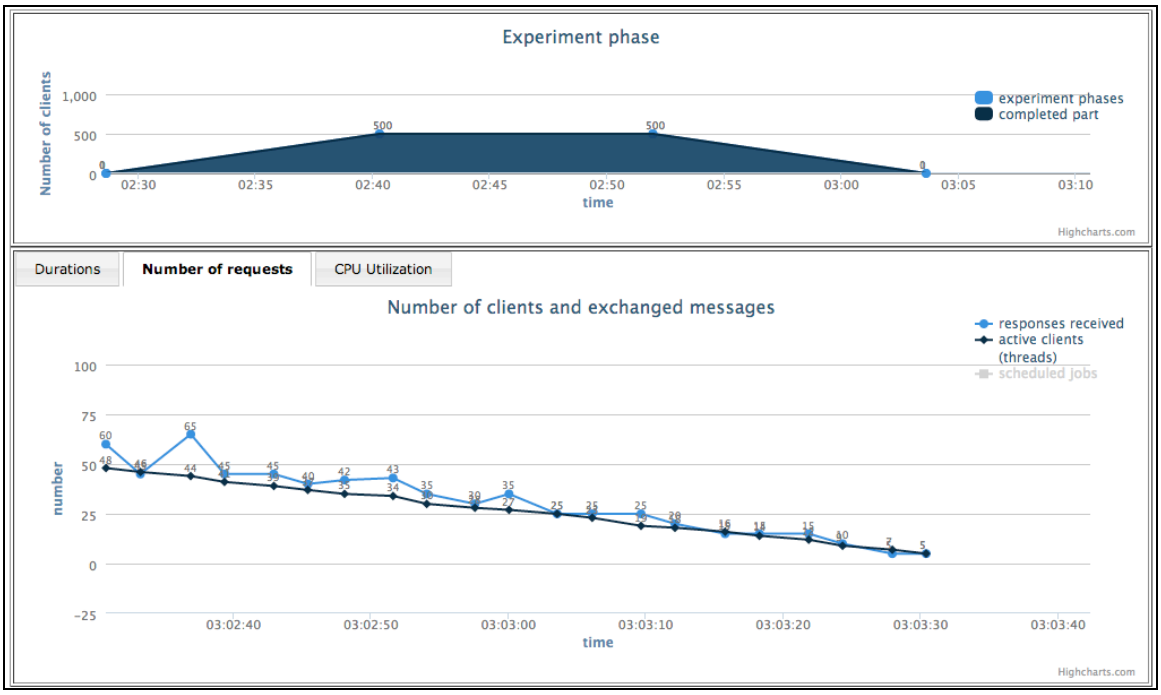


Figure 12. Screenshot of UI runtime monitoring.

Our framework simulates each client by creating a different thread. Each client-thread periodically sends over HTTP a custom request to the application server and waits for a response. The server accepts the request and according to the message content it can decide what is the required action to be taken. If the message contains real-time spatio-temporal information about the client, the server parses and persists this information writing corresponding records to the database. If there is no problem with storage it sends an "ok" response back to the client. Similarly, if the message contains a data acquisition query, the server translates the request to a database specific query, retrieves and parses required data and renders an equivalent response back to the client. A sequence diagram of data transmission from a client is shown in Fig.13 below.

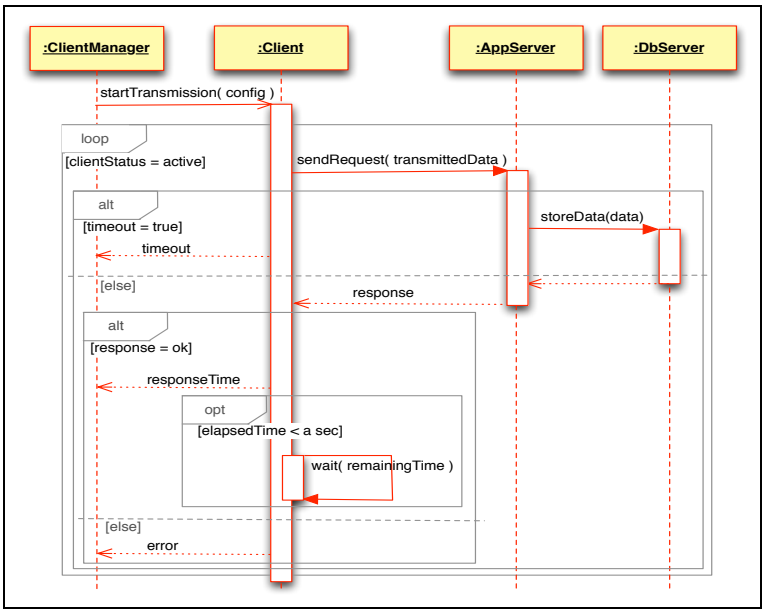


Figure 13. Sequence diagram of data transmission.

6.2 Database Experiments

In this section we present the experiments that were conducted in order to measure performance characteristics, using alternative system configurations. To this extent, the experiments described in this section showcase the feasibility of our architecture and the ability to switch between different implementations in one tier independently of the other tiers. For these experiments we deploy the architecture that we proposed in Chapter 4 using two alternative database technologies for the Data Tier. In specific, we examined the end-to-end response time as perceived by the mobile clients and we used both Sql and

NoSql DBMS for storage, to measure transaction durations and possible performance differences between these two systems, under generated workload.

As mentioned above in Chapter 4, in order to offer the capability of seamlessly switching between different database technologies, the Data Persistence component on the Logic Tier includes a data layer which abstracts data modelling and functions for interaction with the DBMS. As a result, when changing the database at the data tier, no other changes need to be performed for the system, provided that the relevant APIs and connectivity drivers are available, as described below.

In the first set of experiments we used MySql and MongoDB running on Linux EC2 64-bit instances. The Linux distribution for all instances was Ubuntu 10.04 (Lucid Lynx), the version of MySql was 5.1.66-0ubuntu0.10.04.3 and the version of MongoDB was 2.4.1. The application server communicated with the remote database server using each time a relevant driver. For persistence to the MySql server we used Java Persistence API (JPA) and for connectivity we used a JDBC driver. Correspondingly for the MongoDB server we used Morphia for persistence and a Mongo Java Driver for connectivity. For both databases we tried to keep the independent variables as similar as possible and therefore we used both similar data modelling and same number of connections throughout the experiments.

The number of maximum connections proved to be one very important parameter through our testing and it was very often found to be a bottleneck. For our experiments we set the number of maximum connections per node to 100, which means that each application

server (only one for database experiments) could reach up to 100 connections to the database server. For data modelling our approach was to use only one big Table for the MySql database including all data attributes, in correspondence to one collection with all fields in the MongoDB database. The modelling for both databases can be seen in Appendix A.

In Fig. 14 we can see a depiction of the experimental environment for the database experiments, showing the deployment of the various components as described above. The Workload Generator simulates clients which make requests and sends them to the TMARQ application running on an EC2 Instance. The application then stores the information it received to the database or queries the database to obtain required information, according to whether the request was a write or read request respectively, and then replies back to the client. A module within the Monitoring Service is an Amazon Cloudwatch client which collects monitoring information throughout the experiment and sends this information to the Workload Generator.

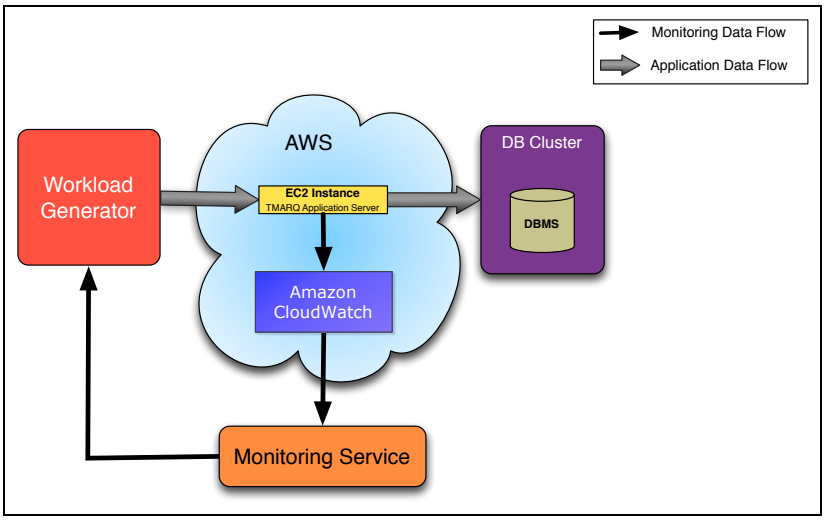


Figure 14. Experimental environment for alternative database experiments.

A very interesting aspect of our application's functionality is performance under varying workload. In specific we were interested in monitoring performance when workload is increasing, decreasing or remaining stable. Therefore the workload we chose to generate for these experiments consists of three phases; increasing number of clients (Phase I), stable number of clients (Phase II) and decreasing number of clients (Phase III), as can be seen in Fig. 15.

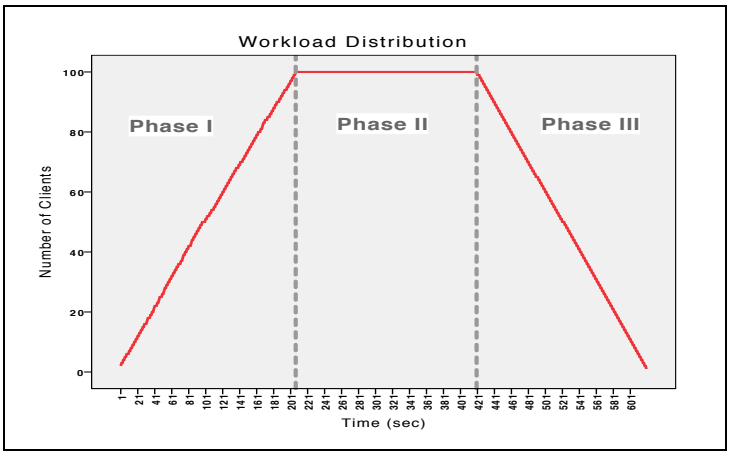


Figure 15. Workload distribution for database experiments.

During these experiments there were two different types of requests that a client could make to the application – read and write requests. During a read request a client would send information to the server that would be subsequently stored to the database. During a read request a client would request information about a specific username, which would cause a 'select' query to the database. The Experiment Controller would enforce to the Client Manager the defined ratio of read-to-write requests for each simulated client. Every client thread would thus send either a read or a write request in a sequence of requests as long as it would be active, according to that ratio.

In order to assure that the experiments would simulate the application running in real-life scenario, we loaded the databases with 1,000,000 custom records (or documents for the MongoDB) before the beginning of the experiments. We also inserted the custom records in such a way that whenever there was a read request (i.e.'select') there would be

minimum influence on the results caused by caching. All the inserted data differed from each other for the field ‘username’ and each time there was a read request, it was about a different username. In addition, to make the scenario realistic, both databases were indexed, which is a feature that can increase the speed of read requests in extremely impressive rates with an insignificant overhead for the write requests, as we noticed from testing.

We conducted three different experiments for each database with different ratio of read to write requests. Each client was selected to send a request every 1 sec. The end-to-end response time from client perspective, the CPU Utilization of the application server and the incoming and outgoing traffic to and from the application server over time for each use case can be seen in the following graphs.

6.2.1 Results for ‘write’ requests

In this subsection we present experimental results for clients sending only write requests, meaning that they only transmit data to the application server. As explained in Section 6.1, a client sends a request in JSON format to the application, containing information about her location, speed and 6-axis acceleration. The server parses the received information and stores it to the database. The following diagrams show results for end-to-end response time as perceived by the client, using MySql and MongoDB respectively.

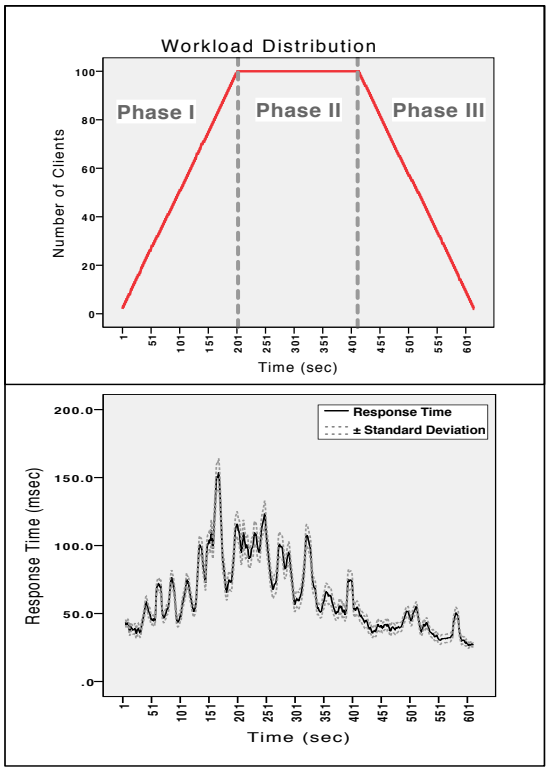


Figure 16. Response time using MySql.

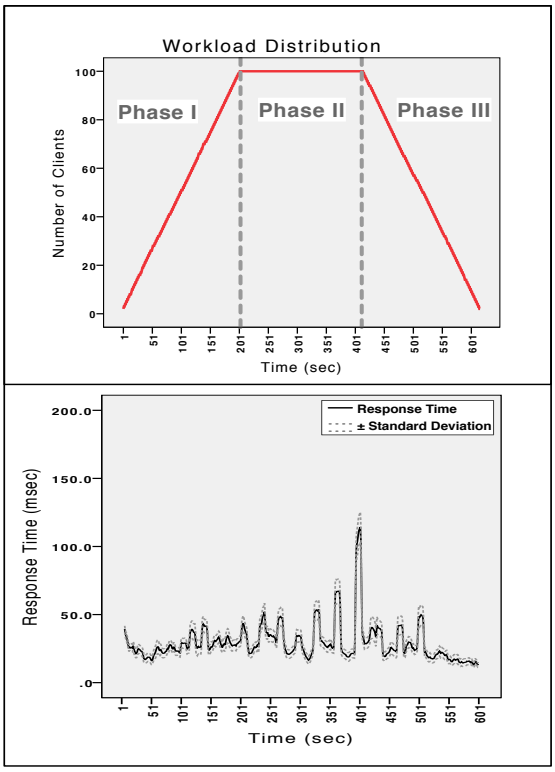


Figure 17. Response time using MongoDB.

Intuitively, we would expect the system's performance to be similar using the two different databases, since they perform a similar function, which is simply the storage of plain datatypes. Perhaps we would even expect better performance using MySql, since it has been the predominant system used over the past decades. However, as we can see from the results, response time is much smaller for MongoDB. This agrees with the performance superiority claims found on the MongoDB website [35], reporting that MongoDB can be about 2.5 times faster than MySql.

One obvious explanation of the results could be the fact that NoSql databases such as MongoDB provide no guarantees about ACIDity. Thus, for the same transaction they can perform much less functions than a relational database, therefore improving performance perhaps by sucrificing robustness and resilience. Some typical functionalities that are not offered by NoSql databases are locks and rollbacks. In addition, the concept of storing large amount of data fast – something that NoSql databases embrace – implies that there are minimal checks for data validation during their insertion to the database, as opposed to the full transactional model.

Following in Fig. 18 and Fig. 19 we present results about CPU Utilization and network traffic using MySql and MongoDB respectively.

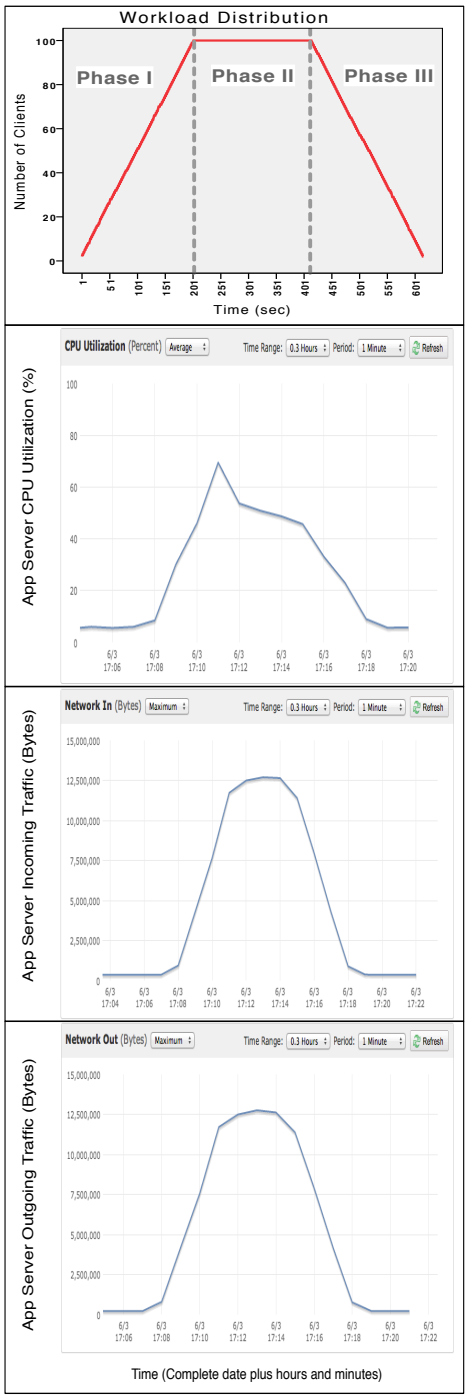


Figure 18. CPU and throughput using MySql.

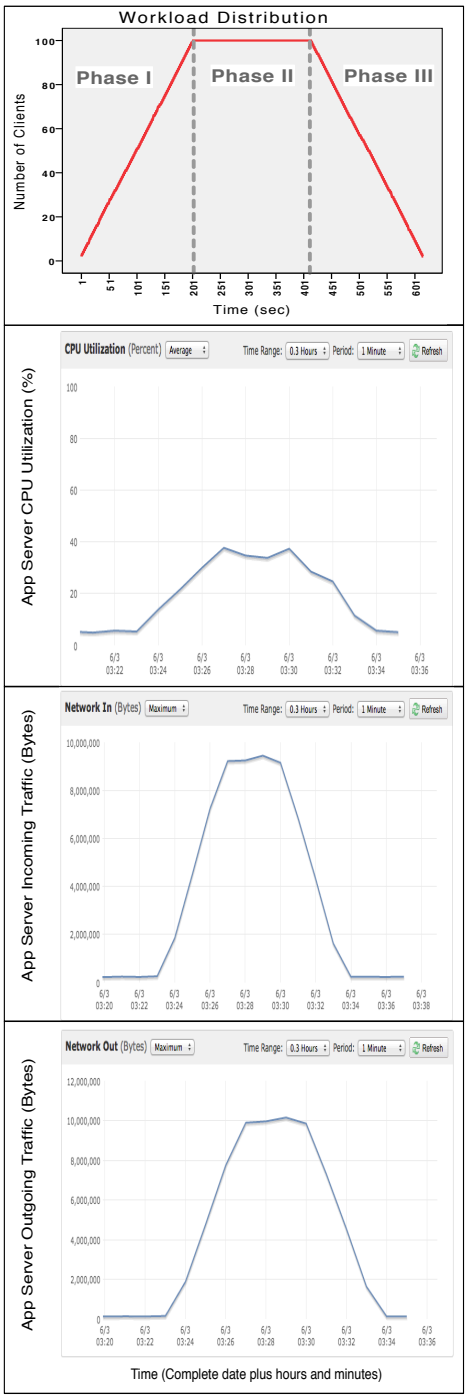


Figure 19. CPU and throughput using MongoDB.

Again from these results we can see that using MongoDB database results in better performance as compared to MySql, this time with regards to CPU Utilization and network traffic. For both these metrics there can be an obvious explanation, and that regards the size of transmitted objects and data representation.

The advantage of MongoDB for this experiment is that it uses the JSON format for storage, which is the same representation as we use for data transmission. As a result, there are minimum requirements for data transformation and minimum overhead for data processing, which would explain the reduced CPU Utilization.

Considering the network traffic, the overall size of the data exchanged between the application server and the database server is expected to be smaller for MongoDB, since it contains less information, for example about datatypes. In addition, the MySql system is possibly performing additional tasks such as checks and has a more heavy communications protocol.

6.2.2 Results for 'read' requests

This subsection shows our experimental results for clients sending only read requests, meaning that they only request from the application server information which exists in the database. As explained in Section 6.1, a client sends a request to the application asking information about a specific userName. The server then performs a 'select' query to the database, retrieves relevant information and passes it on to the client in a JSON

format. The following diagrams show results for end-to-end response time as perceived by the client, using MySql and MongoDB alternatively.

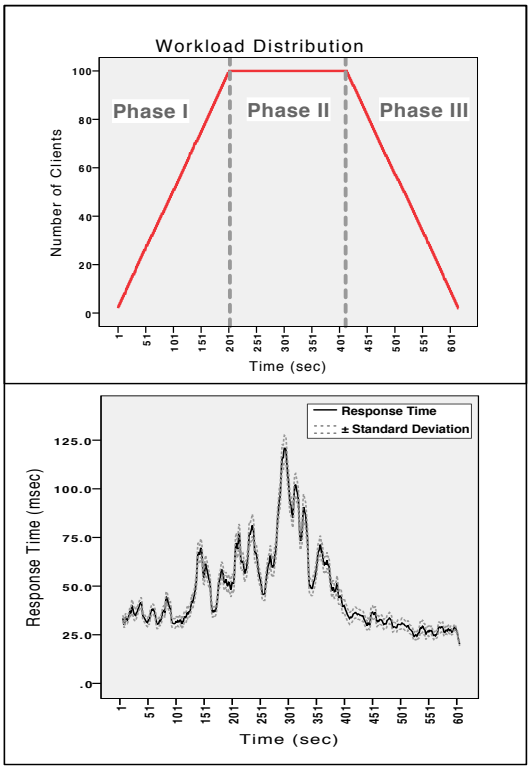


Figure 20. Response time using MySql.

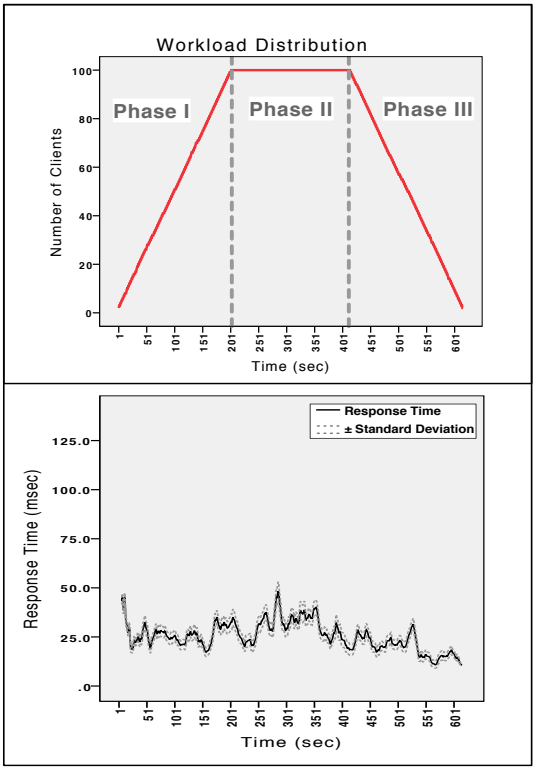


Figure 21. Response time using MongoDB.

In a similar fashion to the 'write' request experiments, we can see that MongoDB performs better in terms of speed. The reasons can be similar as explained for the 'write'

request experiments, namely the ACIDity related tasks performed only by the MySql database and not the MongoDB.

What seems to be particular about these results is that when using MySql the response time appears to increase dramatically when the number of users increases, in contrast with using MongoDB where response time does not appear to change significantly. This leads us to assume that there is possibly a significant overhead for the MySql case, regarding networking functions such as maintaining state of open connections and avoiding overlapping transactions. In any case, the conclusion we can draw from these results is that MongoDB can perform better under a large number of concurrent users.

Following are the results for CPU Utilization and network traffic for the two database systems in Fig. 22 and Fig. 23.

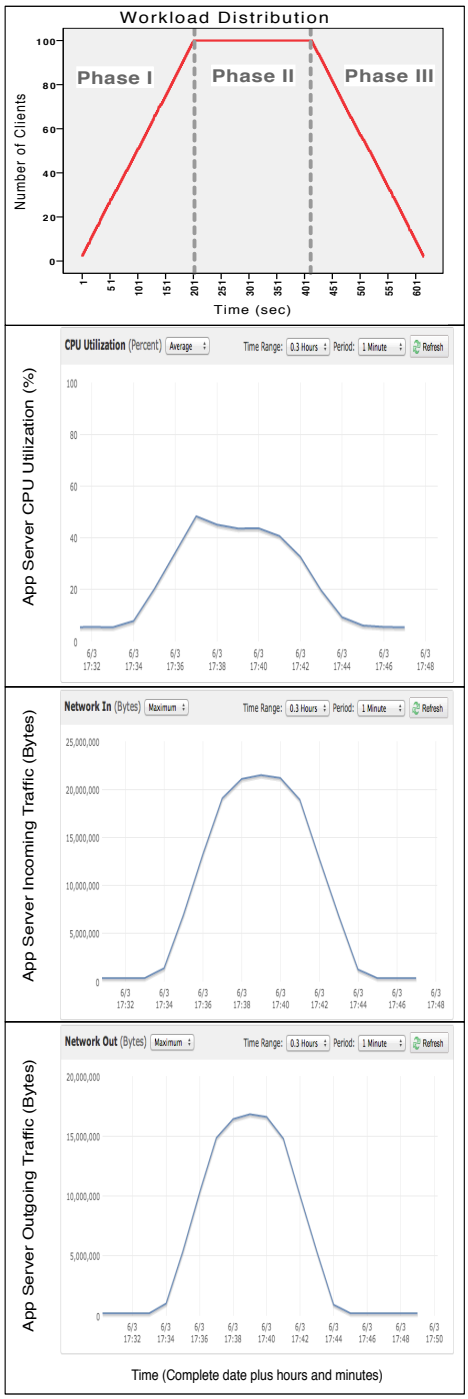


Figure 22. CPU and throughput using MySql.

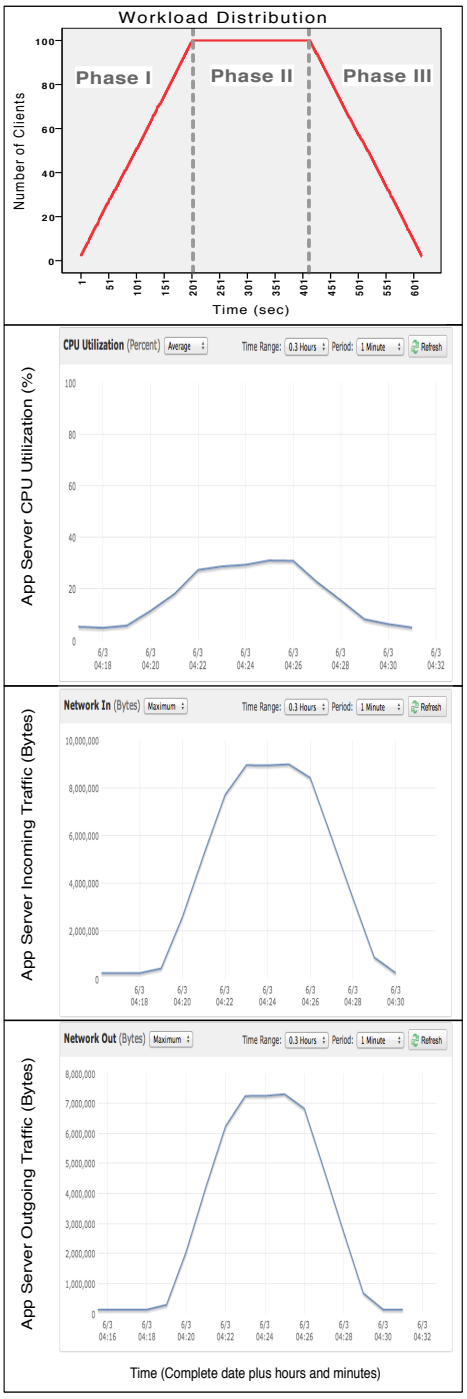


Figure 23. CPU and throughput using MongoDB.

As in the ‘write’ request experiments, we see a big difference in CPU Utilization and network performance using MongoDB. The reasons are the same as explained in the ‘write’ request experiments. Moreover, the fact that the difference in network performance is greater for these experiments, confirms that data representation can play an important role. We can assume that the size of data being sent from the database to the application server is significantly greater for the case of MySql.

6.2.3 Results for 50% ‘read’ – 50% ‘write’ requests

In this subsection there are results for clients sending both ‘read’ and ‘write’ requests,in an equal rate. Each simulated client is sending successively one ‘read’ request and then one ‘write’ request for as long as it is active. As it would be expected, results show performance which takes values intermediate to the values in ‘read’ and only ‘write’ experiments with the only exception that response time for MongoDB appears to be greater than response time for only ‘write’ requests which was shown in the previous experiments. Despite the fact that the difference is not great, it could be an indication that for the MongoDB system there are more resources being shared for both the read and the write requests, which could cause their exhaustion at situations with a large number of both ‘read’ and ‘write’ requests.

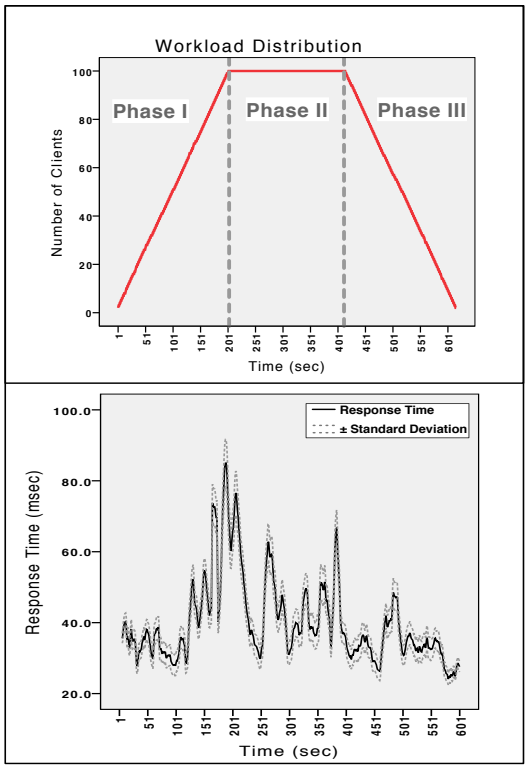


Figure 24. Response time using MySql.

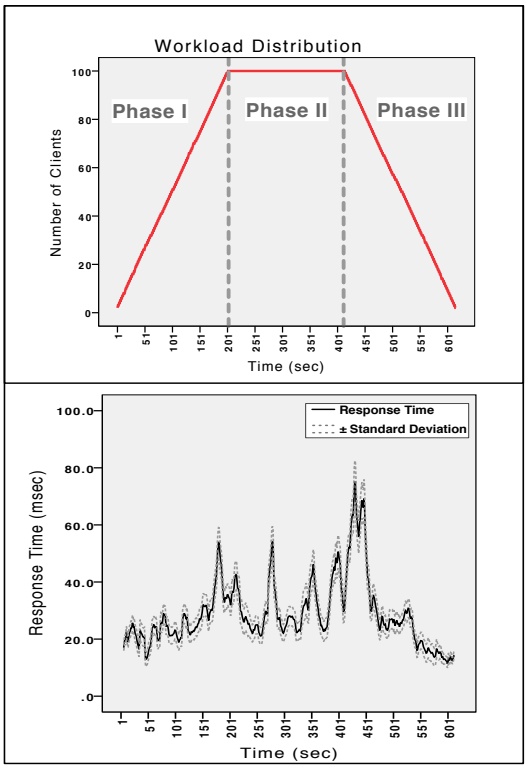


Figure 25. Response time using MongoDB.

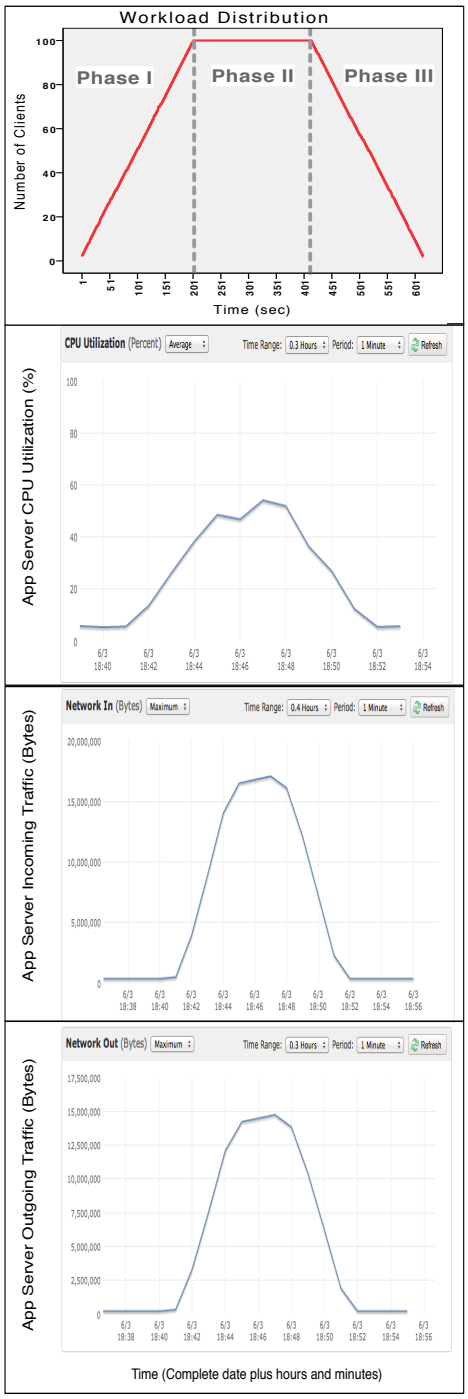


Figure 26. CPU and throughput using MySql.

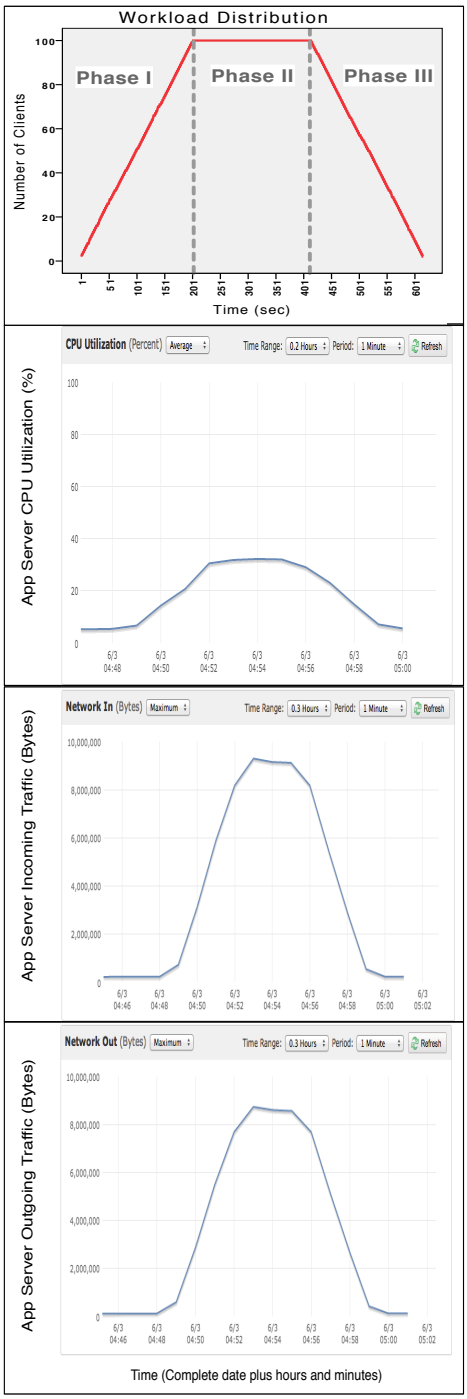


Figure 27. CPU and throughput using MongoDB.

6.2.4 Results discussion

The bargraphs below present the results in a more dense way for the extraction of useful conclusions about each database. In the first bargraph in Fig. 28 we can see the average response time using the two databases for the period during which the workload was maximum, i.e.100 concurrent clients. In Fig. 29 and Fig. 30 we show bargraphs about the maximum CPU Utilization of the application server and the maximum incoming and outgoing network traffic respectively.

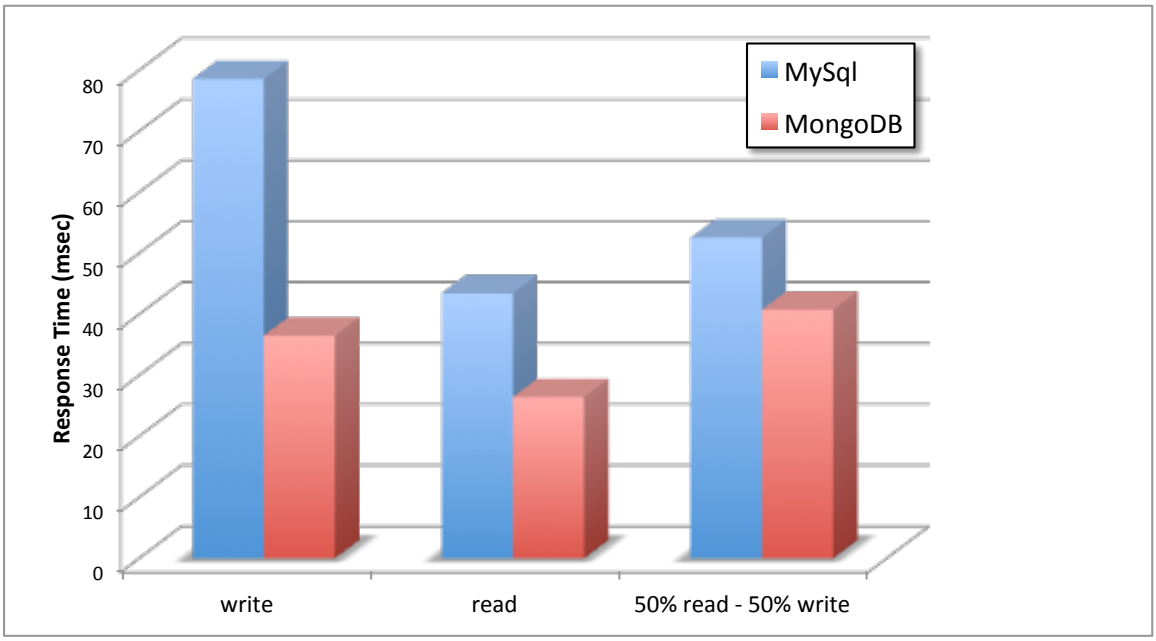


Figure 28. Average response time during peak workload.

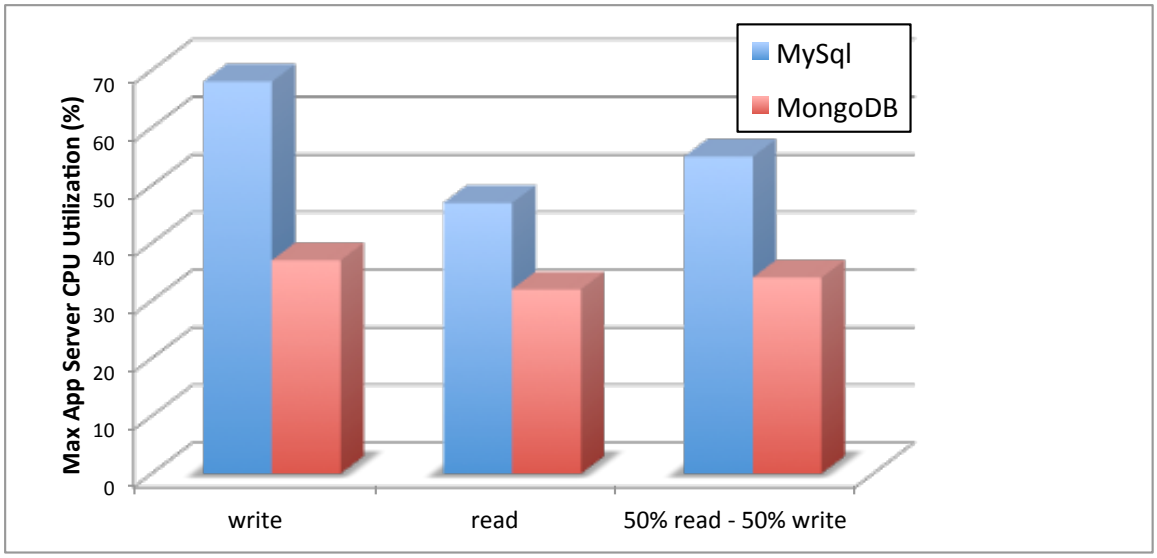


Figure 29. Maximum app server CPU utilization.

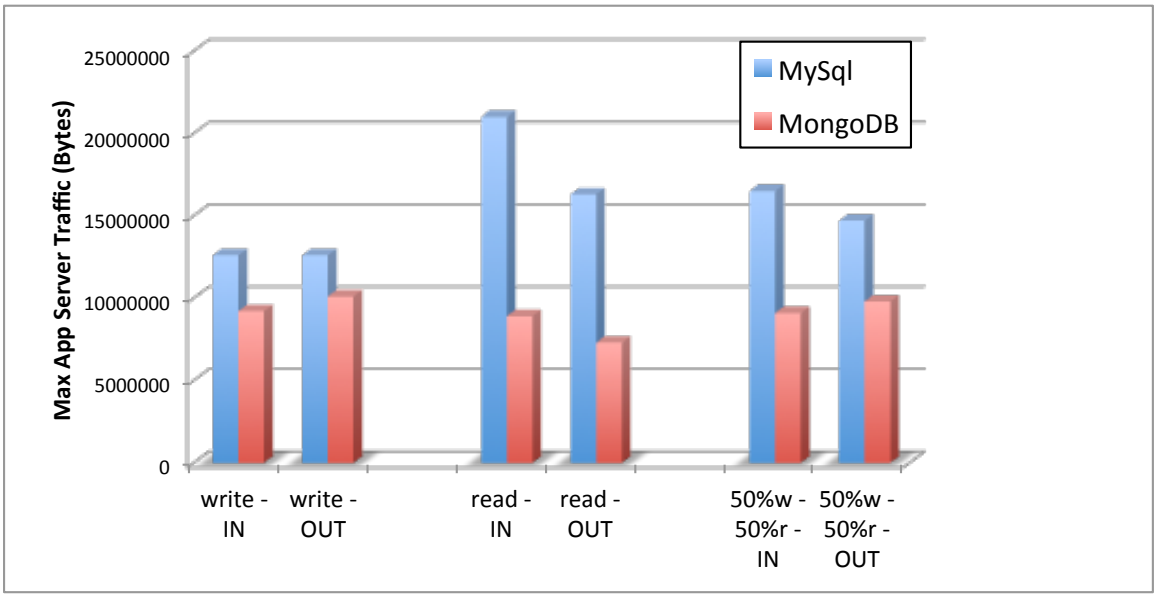


Figure 30. Maximum app server incoming (IN) and outgoing (OUT) network traffic.

To summarize, the results show that MongoDB performs better than MySql in almost every aspect. Using MongoDB we appear to achieve smaller response time, smaller CPU utilization at the application server and smaller incoming and outgoing network traffic for the same amount of information. The CPU and network utilization metrics observed are not very often taken under consideration when performing database performance testing but for our use case they are particularly important for two reasons. The first reason is that based on our proposed 3-tier architecture we have a remote database which needs to frequently interact with the application server over a specific network and therefore network performance characteristics are particularly important. The second reason is that based on the cloud paradigm, such resources are being used remotely and users are charged according to their utilization.

As already explained, the main explanation of better performance results obtained for MongoDB concerning response time and resource utilization is the nature of NoSql databases. In this respect, NoSql databases focus on speed and easiness of data transmission, enabling the efficient storage of large, loosely structured data with no complex relationships. In an abstract way, we could consider NoSql databases as systems that receive blocks of bytes and simply drop them to memory. In contrast, relational databases such as MySql perform many more actions in order to guarantee high degrees of consistency, robustness and reliability. Furthermore, one more notable advantage of NoSql databases is the ability to modify the models of data during runtime due to their schemaless nature, as we verified during our testing.

More specifically for our experiments we used MongoDB, which is currently one of the leading NoSql databases in terms of performance. In addition, MongoDB also bears the advantage of storing data in a similar representation to our transmitted data, which is JSON format. This fact makes MongoDB particularly compatible with our architecture by minimizing any possible parsing and validation overheads, which is one of the main reasons for the observed lower CPU utilization using MongoDB.

Of course the above-mentioned performance advantages for NoSql databases come at the cost of ACIDity. Despite the fact that our experiments were not designed to expose such vulnerabilities, these characteristics can be pivotal for many classes of applications which require reliable, safe and secure transactions such as e-banking. Another possible disadvantage of NoSql databases can be the lack of experience from their use, since they are a relatively new paradigm unlike Sql databases, for which we possess vast accumulated knowledge.

In fact we detected some limitations of using NoSql databases through our experiments. One of them was average stored object size which for MongoDB was approximately 736 Bytes while the equivalent record size for MySql was only 288 Bytes, leading to almost 2.5 times smaller database size for the same amount of data. In addition, specifically for MongoDB we noticed that a database system running on a 32bit machine cannot exceed 2GB of size, which is a serious limitation if only one database is used.

It is important to notice that the intention of these experiments was not to cover all aspects of MongoDB and MySql and a thorough evaluation and comparison of the two

database technologies extends beyond the scope of the current thesis. Our main objective was to showcase that our architecture can support the implementation of different technologies as well as the testing and evaluation of their performance. Generalizing the results of these experiments to assess the use of the database technologies tested would not be accurate for two main reasons. The first reason is that MySql is not intended to be used with only one big table; without the use of more complex database schemas, we are denuding this relational database of many of its features and disallowing the exposure of its powerful characteristics such as query optimization. The second reason is scalability; the most advertised advantage of nosql databases is the ability to operate at scale and using only one VM for the Database server is not adequate to expose this functionality.

Nonetheless, our experiments can reveal some important performance characteristics about these database technologies and compare their performance under similar conditions and during requests from concurrent users. Thus, we successfully showed that it is possible through our architecture to evaluate alternative database technologies for real-time applications. In particular, we showed that for our three-tier, real-time application and under certain specified conditions, as explained in this chapter, MongoDB which is a NoSql database can have significantly better performance in terms of speed, network and CPU utilization, compared with MySql which is the most popular relational database.

6.3 Adaptation Experiments

This Section presents our experiments about using adapation mechanisms for our mobile application. Our intention is to showcase the capability of implementing adaptive mechanisms on our architecture and to evaluate the use of those mechanisms and different adaptation configurations with regards to performance and cost. These experiments also serve as a use case for our proposed adaptation algorithms for mobile spatio-temporal applications as presented in Chapter 5, in an attempt to show in a realistic environment how the use of both server and client adaptation can produce better results as opposed to the use of each one separately. As a reminder, the client adaptation that we introduced in Chapter 5 facilitates the routing of requests according to user's current location and the optimal client configurations according to availability. The server adaptation supports the adaptive scaling and configuration of distributed server clusters according to changing demand. Through these experiments we aim at showing the feasibility and efficiency of these two adaptations and the benefits from using them both at the same time in the adaptation mechanism that we propose.

Thus, we produce a workload of 500 clients and after showing that the application is unable to handle such a traffic in a static manner, we perform experiments for two different configurations of the server adaptation, two different configurations of the client adaptation and then for both adaptations. For the purpose of these experiments, by 'server adaptation' we refer to the ability of the application servers to scale in and out according to CPU utilization as it is affected by decreasing or increasing workload. In contrast, by

‘client adaptation’ we refer to the ability of the mobile clients to direct their requests to alternate server clusters according to their location and the server clusters’ average CPU utilization. Both these adaptation functionalities are subsets of the adaptation mechanisms as were presented in Section 5. With our results we prove that using both server and client adaptation we can achieve location-aware adaptation, which is not achieved using only server adaptation, while maintaining low response time and reduced number of launched VMs and therefore reduced cost, as compared to using only client adaptation.

The experimental settings were almost the same as for the database experiments, as described in the previous subsection. The difference was that for these experiments, instead of using only one Amazon EC2 instance, we used server clusters for the deployment of application servers. In fact, the deployment architecture that we used for these experiments is identical to the one depicted in the deployment diagram in Fig.6 with the only difference that instead of actual mobile devices, we used our Workload Generator to simulate Mobile Clients. Thus, we used Amazon CloudWatch for monitoring the servers and AWS Elastic Beanstalk for analysis, planning and execution of the server-side infrastructural adaptations. We implemented the adaptive loop on the client side by applying corresponding management logic on the Workload Generator for distributing client’s requests to the different servers according to the feedback that it receives from monitoring services.

Regarding the database, we used the same EC2 instance running MySql for all of the experiments, in an attempt to keep the experimental environment consistent and the

independent variables constant for as valid as possible comparison of results. Concerning the workload we generated for these experiments, it was similar to the one we used for the database experiments with three phases (increasing, stable and decreasing number of clients) with the difference that clients reached up to 500 and the duration of the experiments was longer. The requests that these clients made were only 'write' requests and the transmission period was again set to one second.

At this point it is also important to notice that for the purpose of our experiments we consider the quality of service of our application to be heavily dependent on the end-to-end response time. Therefore, we consider a hard goal of the application runtime to maintain an average response time smaller than transmission period of the mobile clients. The reason is that for greater values of response time the application becomes 'unresponsive' by which we mean that data is either lost or accumulated in high rates causing the application to crash. During these experiments there is also one additional quality of service characteristic and that is location-aware adaptivity. The approach we follow is a 'best effort' one, meaning that we treat 'keeping average response time lower than transmission period' as a strict SLO which should be maintained at all times whereas 'directing requests to servers closer to client's location' is considered one desired but not mandatory requirement, thus realizing prioritization of requirements.

6.3.1 No Adaptation

In this first experiment we show how the application is unable to handle traffic of hundreds of users without the use of dynamic methodologies. Thus, we use the same

static architecture that was used for the database experiments in Section 6.2, with only one small EC2 instance for the application server and we measure average response time while attempting to produce a workload of 500 simulated clients. The results can be seen in Fig. 31.

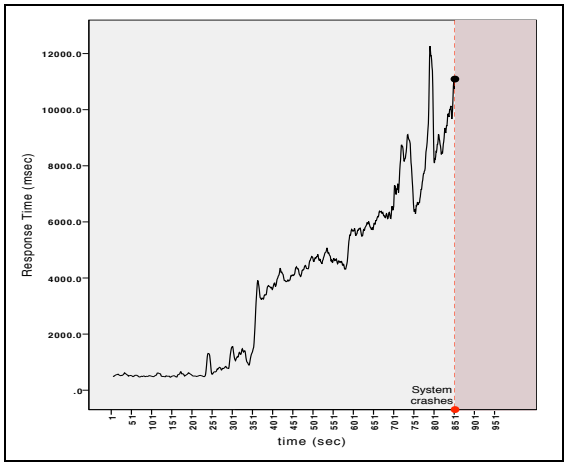


Figure 31. Response time for experiment without any adaptation.

As we can see from the results the application is not able to handle incoming traffic and response time keeps increasing in an unmanageable fashion. What is more, the experiment cannot even be completed as the application crashes at some point, possibly because of code vulnerabilities and we can obtain no more results for response time after that point.

6.3.2 Only Server Adaptation

In this subsection we present the results that we obtained using only server adaptation with two alternative configurations. As explained in Section 5.1, server adaptation refers to the adaptive scaling and configuration of distributed server clusters according to changing demand. Examples of such adaptation could be the increase of server instances of a cluster because of increase in the number of requests and the change of the borders of the geographical area a cluster 'covers' for example because of the positioning of an additional cluster at that area.

For our experiments we consider as service adaptivity the special case of scaling in and scaling out of identical server instances in a cluster according to demand, which is the most popular adaptivity mechanism currently used for web applications. In this respect we deploy our application servers inside one Web App Cluster, as can be seen in Fig. 32. For the Analysis, Planning and Execution of adaptation we use AWS Beanstalk and for Monitoring we use Amazon CloudWatch. This architecture provides the capability of monitoring running instances and auto-scaling according to predefined policies.

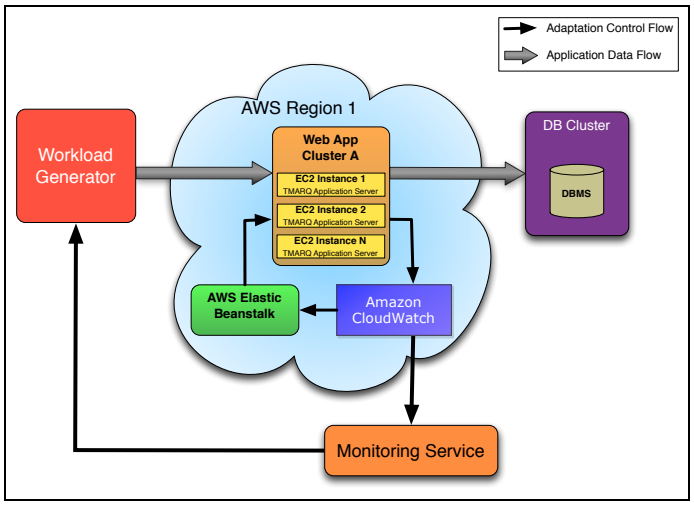


Figure 32. Architecture for server adaptation experiments.

Thus, with reference to our Server-side Adaptation Architecture from Section 5.1 the Running Servers are the servers inside the cluster, the Servers Monitor is the Amazon CloudWatch service and the Autonomic Manager is the AWS Elastic Beanstalk environment. Using this architecture, we perform two different experiments:

- **sad_20-40**: scale out for average CPU>40%, scale down for average CPU<20% and
- **sad_30-60**: scale out for average CPU>60%, scale down for average CPU<30% .

For these experiments, the Elastic Beanstalk environment receives monitoring information about the running instances in an interval of 1 minute. If the average CPU utilization of all running instances rises above the defined upper threshold it adds one EC2 instance to the cluster and if it drops below the defined lower threshold, it removes

one instance if that is possible, leaving always at least one instance running. For these experiments the number of running instances can take values from 1 to 5. In Fig. 33 and Fig. 34 we can see the response time, average CPU utilization and number of running instances in the cluster for the two experiments respectively.

As we can see for the results, for both experiments the number of instances starts at 1, scales out to 5 instances for the maximum workload which is 500 clients, and then scales in back to 1 instance as the workload gradually decreases. However, from the results we can imply that for sad_30-60 the servers scale out too late and scale in too fast, which causes bad performance in terms of response time for the transition periods. For sad_20-40 we can observe satisfactory results as response time is generally lower than transmission period, except for some very short exceptions. This could be comprehended as such: If we know this specific workload behavior in advance, we can safely assume that configuring the application according to sad_20-40 would guarantee the maintenance of performance whereas according to sad_30-60 it would not.

Despite the fact that this adaptation can handle high rates of changing traffic, it does not provide location-based adaptivity, since all requests are directed to the same server cluster, among others creating the issue of ‘one point of failure’.

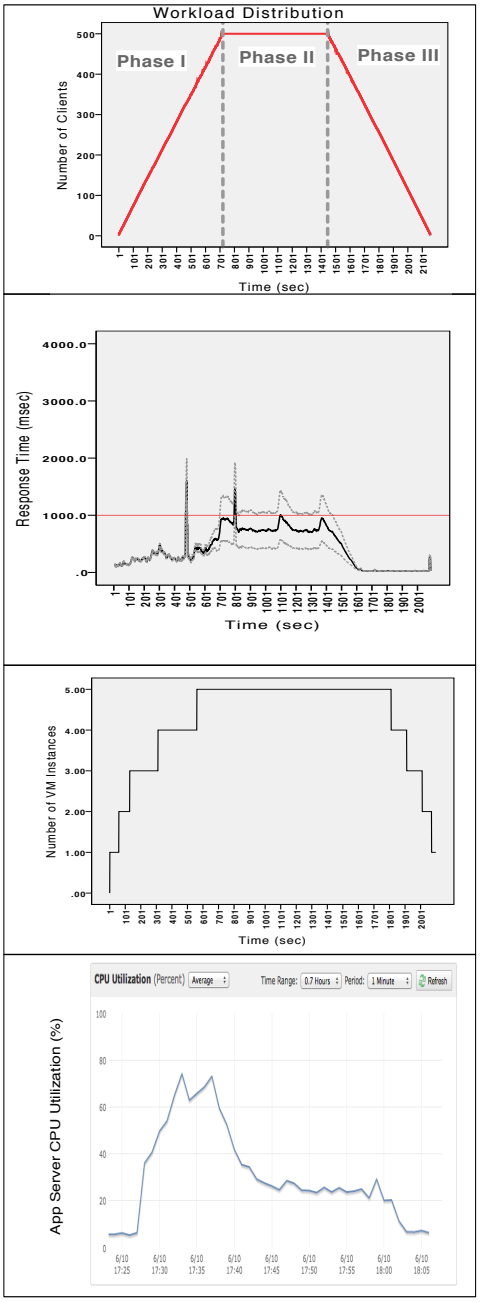


Figure 33. Response time, number of VMs and CPU for sad_20-40.

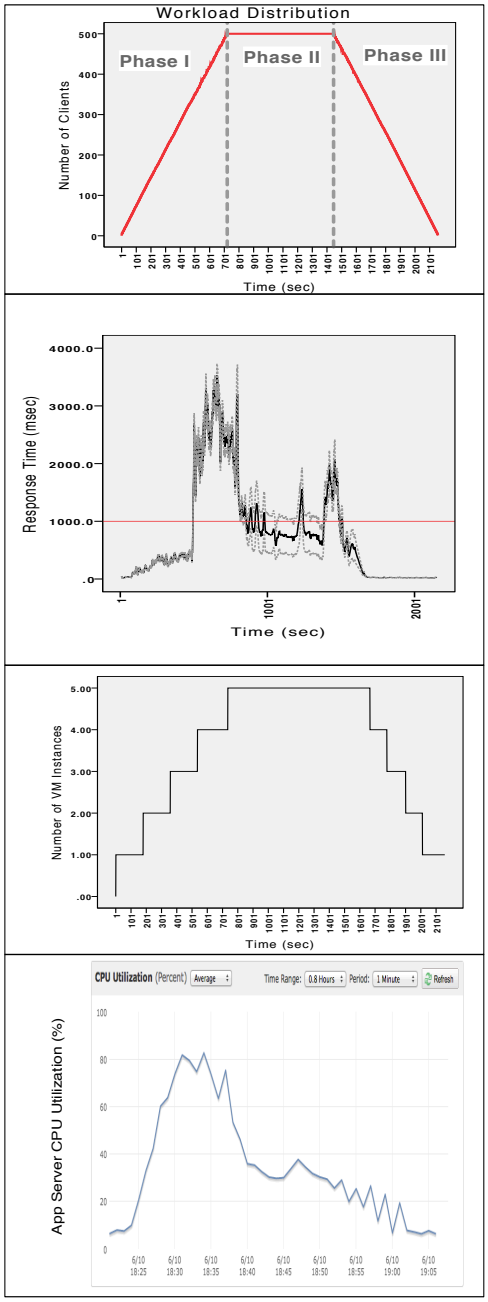


Figure 34. Response time, number of VMs and CPU for sad_30-60.

6.3.3 Only Client Adaptation

This subsection presents the results that we acquired using only client adaptation with two alternative configurations. In Section 5.2 we explicitly explained our notion of client adaptation, according to which the mobile clients also monitor state and implement adaptations about the sampling and transmission rate and the selected server to which they direct their requests. For these experiments we specifically test the direction of requests to the appropriate server when there are more than one available, according to resource utilization on the server. The resource we refer to is CPU and thus we test the performance of our application when requests are redirected from one server cluster to another based on the average CPU of their running VMs.

Again for these experiments we use a web application cluster for our servers but this time we have two different clusters where requests can be sent, which represents the geographic distribution of different servers for the same application, as is depicted in Fig. 35. In addition, in order to test only client adaptation the environments are not scalable and so the number of running instances for each cluster is preset and constant; 2 server instances for Cluster A and 3 server instances for Cluster B.

With reference to the algorithm presented in Section 5.2, the monitored parameters are assumed to be the geographic location of the simulated clients and the Server Resource Utilization, and specifically the average CPU utilization of each cluster which is provided from Amazon Cloudwatch to the Monitoring Service and then to the Workload Generator. This state information influences the adaptation that takes place at the

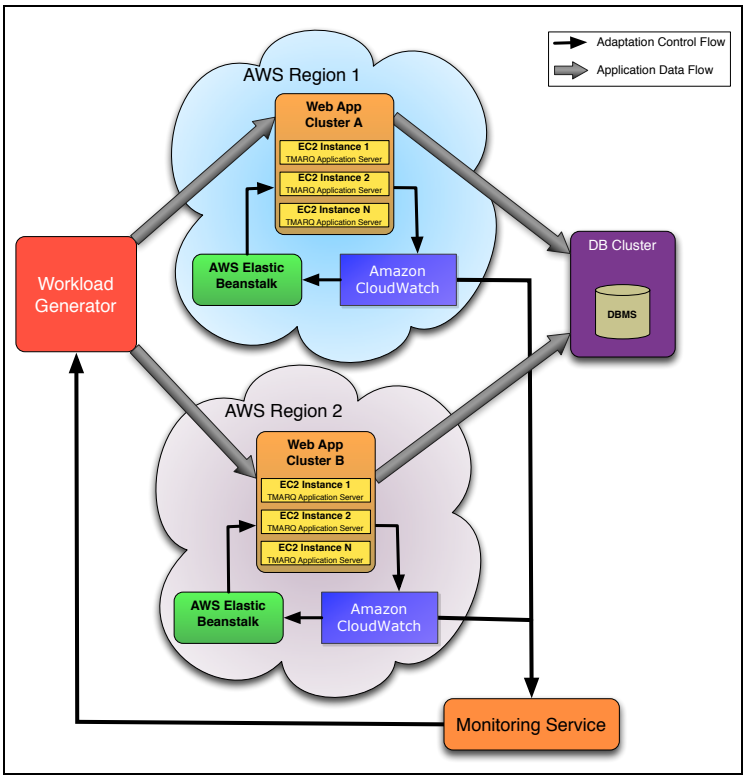


Figure 35. Architecture for client adaptation experiments.

Workload Generator which has as an output the distribution of simulated clients to the two different clusters. This way we simulate the ability of each mobile client to direct requests according to feedback it receives about available servers' utilization.

Again for these experiments we measure response time as a QoS characteristic of the application runtime. In addition, we show how the prioritization of requirements is being implemented, since the goal is to enable the sending of requests to different servers based on location but only when this is possible while maintaining response time lower than transmission period.

Using the described architecture, we perform two different experiments:

- **cad_20**: start sending traffic to second server cluster when CPU Utilization goes over 20%, start receiving back extra traffic when CPU Utilization drops under 20%.
- **cad_30**: start sending traffic to second server cluster when CPU Utilization goes over 30%, start receiving back extra traffic when CPU Utilization drops under 30%.

For both experiments we assume that all requests are supposed to be received by Cluster A because of the clients' geographical location. However, as described we constantly monitor average CPU utilization on both clusters and when it receives values over a specific threshold for Cluster A caused by increased traffic, we redirect all new traffic to Cluster B. When the value drops below the same threshold, we start receiving traffic back to Cluster B with a specific rate. In Fig. 36 and Fig. 37 we can see the response time and average CPU utilization of both clusters for the two experiments respectively.

As we can see, with both configurations the results are satisfactory, since average response time generally remains under 1 sec. However, with cad_30 we observe more smooth changes and lower values for CPU utilization, which indicates that this configuration is more suitable based on this certain traffic load behaviour.

Despite the fact that with Client adaptation we achieved location awareness and good performance, it should be noticed that this architecture is the most costly, with 5 VMs running at all time for the application servers.

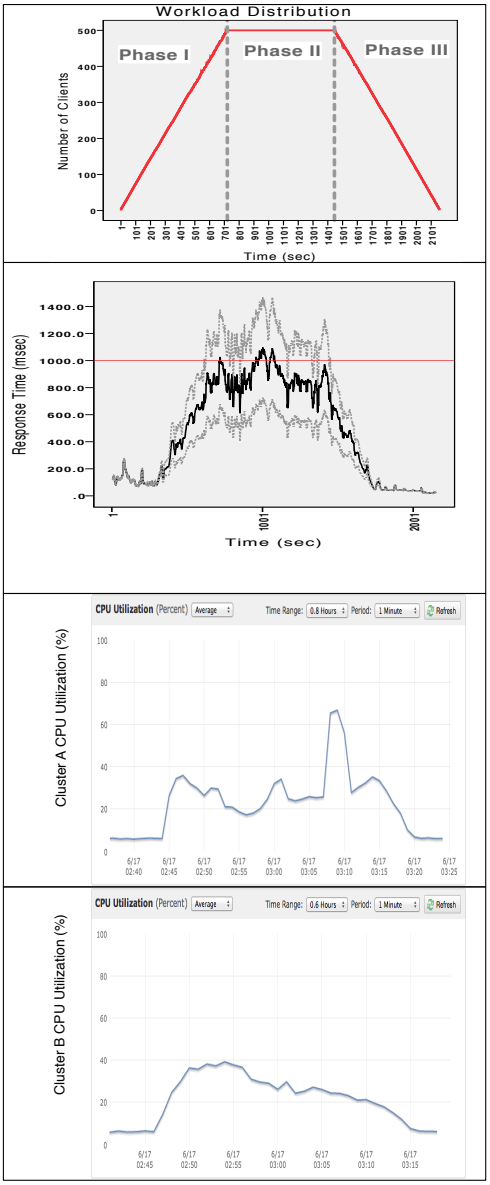


Figure 36. Response time and CPU utilization for cad_20.

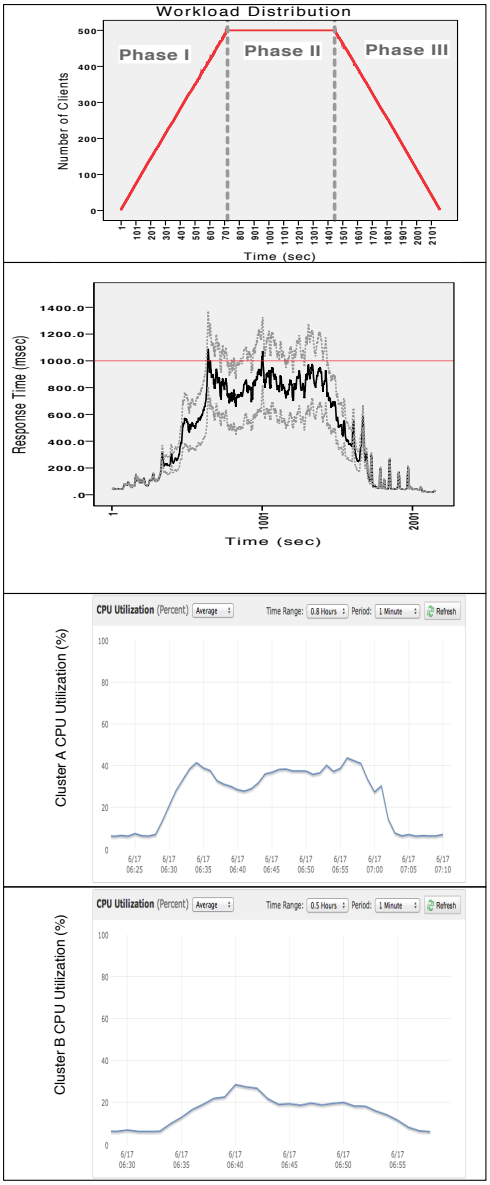


Figure 37. Response time and CPU utilization for cad_30.

6.3.4 Both Client and Server Adaptation

In the last experiment we test both client and server adaptation, as described above. To achieve that we use the exact same architecture with two clusters (Elastic Beanstalk environments) as in the client adaptation experiments with the only difference that the servers are now scalable and the number of server instances for each cluster can increase or decrease according to demand. For the configuration parameters, we combine the configurations from client adaptation and server adaptation experiments which produced best results, namely *sad_20-40* and *cad_30*.

The way that this adaptation works is that clients direct all their requests to Cluster A, assuming that this is the responsible cluster for these clients, based on their location. This cluster can scale from 1 to 2 server instances according to demand. When the average CPU utilization of its instances exceeds the predefined threshold (30%) all additional arriving requests are redirected to Cluster B, which can also scale from 1 to 3 server instances according to demand. When average CPU utilization on Cluster A drops under the threshold (30%), if there is traffic on Cluster B it starts receiving it back.

The results for workload distribution per cluster, response time, average CPU utilization and number of running VMs over time for both clusters can be seen in Fig. 38 and Fig. 39 and we refer to this experiment as *cad_sad* as it presents the use of both client and server adaptation at the same time.

As we notice, application performance when combining both adaptations appears to be satisfactory. What is more, with this architecture we achieved location awareness and low

response time with a reduced number of running VMs over time and therefore reduced cost.

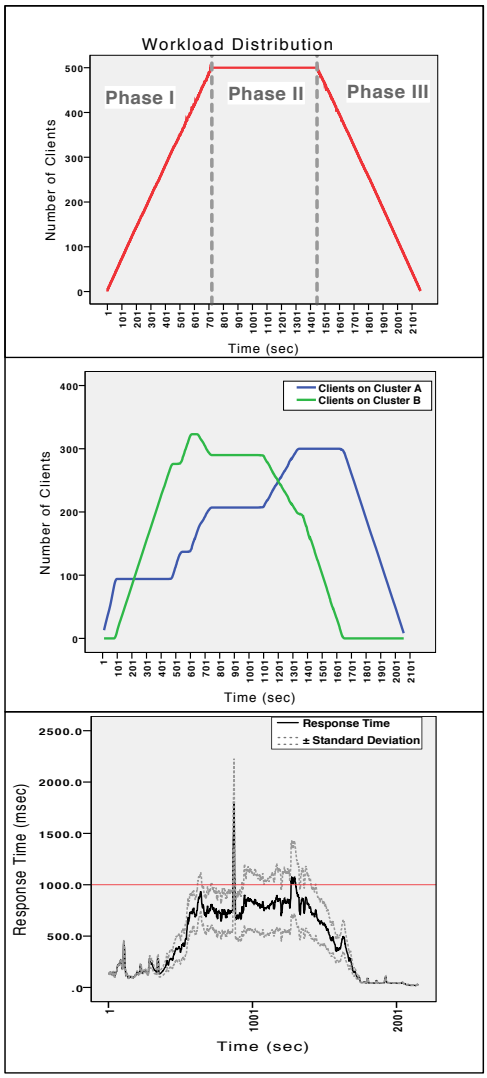


Figure 38. Workload distribution and response time for cad_sad.

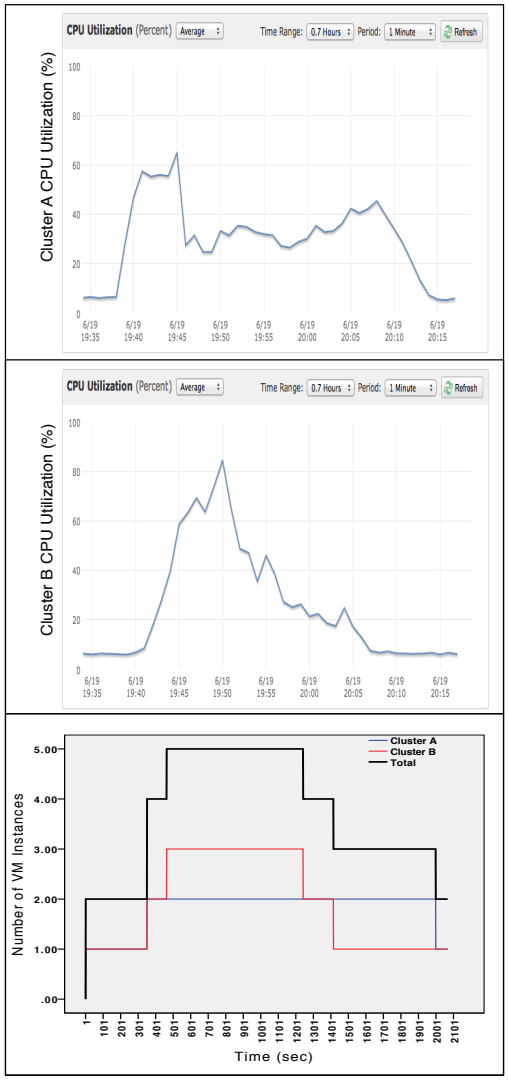


Figure 39. CPU and number of VMs of Cluster A and Cluster B for cad_sad.

6.3.5 Discussion

In the following bargraphs we can see compact results for all adaptation experiments presented. From Fig. 40 we can compare the different adaptations in terms of average response time as perceived by clients. For a more detailed view, we show results for the different phases of the workload distribution; Phase I during which number of clients increases from 0 to 500, Phase II during which number of clients remains constant at 500 and Phase III where number of clients decreases from 500 to 0.

As we can see from these results, all adaptations except from *sad_30-60* for the reasons explained in the corresponding subsection, perform at satisfactory levels. This proves that with the use of any of these adaptations and the appropriate configuration we can achieve a sustainable application performance that can handle increasing and decreasing workload traffic as well as a large number of clients. We also notice that the differences between working adaptations are not great and could possibly be caused by the cloud variation, as will be explained in Subsection 6.3.6. In addition, for the maximum workload (Phase II), *sad_20-40* appears to have the lowest response time, followed by *cad_sad*, then the two client adaptations and finally *sad_30-60*.

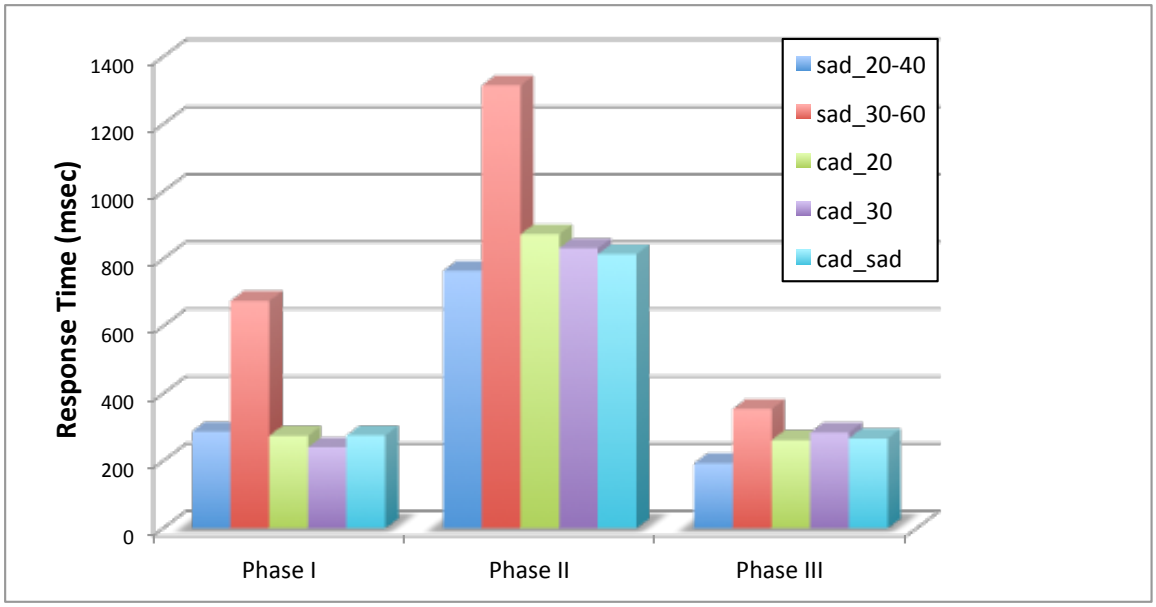


Figure 40. Average response time for Phase I (increasing clients), Phase II (constant clients) and Phase III (decreasing clients).

As we mentioned in the beginning of this Section, what we consider as a strict SLO is keeping average response time below transmission period. Thus, for our application we can define *SLO adherence* as the percentage of time during which average response time remains under the defined threshold (1 sec), over the complete time of the experiment. According to this definition, we can see results about each adaptation in the bargraph below in Fig. 42.

In addition, we can define *relative cost* as the number of running server instances multiplied by the time period by which they are running aggregated throughout the complete time of the experiments. Our granularity is set to 1 sec and therefore time period is measured by number of seconds. For instance, if we consider 3 server instances

running for 10 seconds and then 2 server instances running for 20 seconds we would have a relative cost of:

- *relative_cost = 3 * 10 + 2 * 20 = 70*

Using this definition we calculate the complete relative cost of total running server instances for each experiment and we receive the results as depicted in Fig. 41.

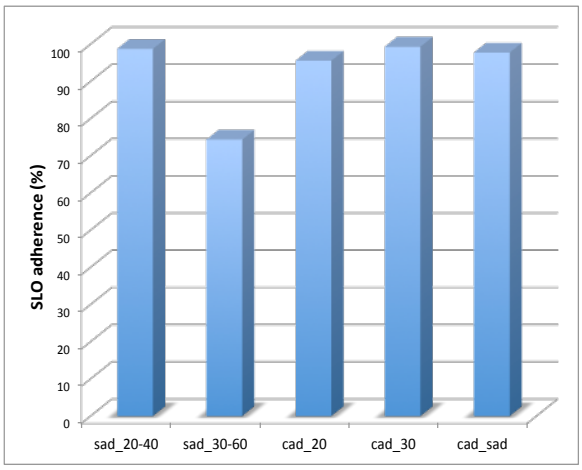


Figure 41. SLO adherence for different adaptations.

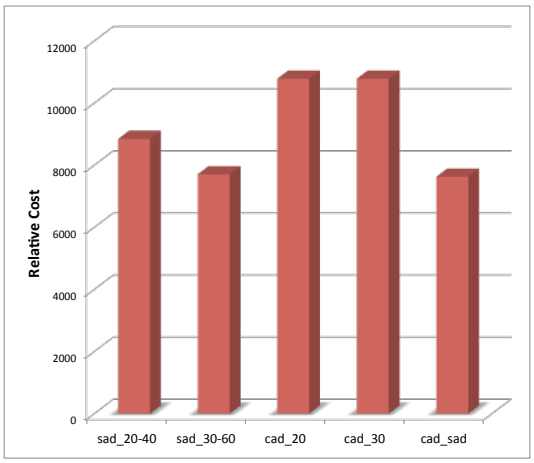


Figure 42. Relative cost for different adaptations.

As we can see from Fig. 41, SLO adherence appears to be adequately high for all cases of adaptation except from *sad_30-60*, similarly to results for response time. For the rest of the cases, we can consider that they could all be qualified as successful, as they offer SLO adherence over 95% and the differences between them do not appear to be particularly significant. Thus, we could verify again from these results that the

application using any of these adaptations would be robust and capable of managing the assumed workload.

From Fig. 42 we can see what is the most important advantage of using both adaptations – cost. As we can observe, using both adaptations has the lowest cost, even compared to the unsuccessful configuration of *sad_30-60* which does not offer the required QoS. The relative cost of the two client adaptations has the largest value, as expected, since the full capacity of the application, 5 server instances, are constantly running throughout the complete experiment. Both adaptations have the lowest relative cost and *sad_20-40* is somewhere in between.

In Table 6 below we can see the results that were presented in this chapter, more compact for easier comparison.

Adaptation	sad_20-40	sad_30-60	cad_20	cad_30	cad_sad
Average Response Time - Phase I (msec)	287.4	675.6	274.9	240.4	275.8
Average Response Time - Phase II (msec)	764.3	1315.3	873.3	831.4	813.8
Average Response Time - Phase III (msec)	193.4	354.6	262.3	284.8	267.3
SLO adherence (%)	98.7	74.3	95.5	99.3	97.7
Location-based adaptivity support	✗	✗	✓	✓	✓
Relative cost	8,842	7,705	10,780	10,780	7,625

Table 6. Collective results for all adaptation experiments.

As we can see from Table 6, it is obvious that the case of both adaptations offers the best results altogether. As compared to the cases of only client adaptation, it offers 29% reduced cost, in terms of the number of launched virtual machines. In comparison with *sad_20-40*, despite the fact that this adaptation seems to offer slightly (1%) higher SLO

adherence, again the cost is reduced by approximately 14%. In addition, this adaptation has the crucial disadvantage of not offering the desired support for location-based adaptivity, as opposed to *cad_20*, *cad_30* and *cad_sad*. In fact what we mean by location-based adaptivity support is the apparent easiness by which one application server can be configured independently of any other servers and therefore can have a specific innate behaviour towards incoming requests from clients belonging to its 'area of responsibility'. Thus claiming for instance that server adaptation alone does not offer this support, does not strictly exclude any type of context-aware adaptivity but rather the simplicity of its implementation according to that definition.

Finally, compared to *sad_30-60*, *cad_sad* offers much higher level of SLO adherence and lower cost, setting this case simply as an example of the wrong configuration of a partially successful solution.

6.4 Threats to validity

This section describes threats to the validity of our experiments, in terms of how they were alleviated as well as how they should be taken under consideration throughout evaluating our conclusions and generalising our results.

6.4.1 Construct Validity

The Construct validity refers to the degree to which experimental variables accurately measure the concepts they purport to measure. One important aspect of our experiments was the definition of QoS characteristics for the class of spatio-temporal applications that

we examine. In this respect, what we regarded as the most significant quality metric was the end-to-end response time as perceived by the clients. The reason was not only that time is one of the two fundamental dimensions of these applications, but also because we intended to test our application in situations of increased workload and thus potential accumulation of unfinished tasks, which could lead to application unresponsiveness because of unmanageable rates of incoming requests, as proven in Subsection 6.3.1.

The reason that we used end-to-end response time was that we aimed at testing the application performance as a whole. Due to the modularity of our architecture and the fact that it consists of multiple layers, we configured the internal method calls to be synchronous in order to expose the delays throughout all participating components. One additional rationale behind this configuration was that during our testing period we noticed that asynchronous configuration produced transparent behaviour to the end clients on one hand, but difficulties controlling the experiment and exposing useful results on the other.

Another QoS dimension that we examine throughout our experiments is location-based adaptivity, as a specialization of context-aware adaptivity. Although location is not the only important context information that could be taken under consideration for a client and other context information such as activity and content should not be ignored, our claim is that location is the most important case for mobile spatio-temporal applications. An additional importance of this characteristic is that the redirection of requests to the appropriate server according to client's location can also lead to cost savings, considering

for example different pricing policies depending on the physical geographical region at which the server hardware is located.

More specifically about cost, which is also one of the important characteristics that we consider, it should be mentioned that different cloud providers currently offer different pricing policies for their services. Our choice of using relative cost as a metric, reflecting the utilization of number of server instances over time, implies a pricing policy of "pay per number of launched instances", which is currently the case for AWS services but not for all service providers. Another issue that should be considered is the granularity at which cost is estimated; for example our definition of relative cost using a granularity of seconds could not currently have a reflection to the model used by AWS which calculates number of instances launched on an hourly basis. However, we believe that in the future, as cloud will mature pricing policies will become more granular and "pay-per-use" will be applied more accurately.

6.4.2 Internal Validity

Internal validity refers to the degree to which conclusions can be drawn about the causal effect of independent variables on dependent variables. Our first step in order to ensure internal validity was the conducting of thorough testing prior to the experiments, which enabled for the identification and mitigation of bottlenecks, such as the number of connections to the database and the type of the server instances. Through our testing we also acquired an understanding of the field which facilitated us in selecting the appropriate experimental setting concerning parameters such as the duration of the

experiment and maximum number of simulated clients and also in verifying that our experimental results showed reasonable behavior.

In addition, in order to avoid instrumentation threats, we built our custom testing framework – the Workload Generator – that provided us with more control over the experiments and the monitoring granularity, which was kept the same for all experiments. When it comes to selection of parameters, such as the different adaptation configurations, they were selected randomly, of course within a range which would produce reasonable results. The same argument is valid for the selection of workload traffic behavior and duration of the experiments, which were selected randomly but carefully in order to be adequate in exposing application's features, for example giving scaling enough time to take place. Specifically about the workload, we decided that there was no benefit in randomizing its change over the time of the experiment, as it was selected to show responsiveness of the application to increasing, decreasing and stable traffic. When it comes to transmission period, it was arbitrarily set to 1 second, but this value is a valid rate of data sampling for road quality assessment mobile applications, as observed by other researchers' work which was reviewed in Section 3.1.

Another point that should be mentioned is that we assumed CPU utilization as an adequate metric of server utilization which would affect response time. Through our testing our assumption proved to be valid, despite the fact that there should be no apparent reason why other metrics could not be used alternately, such as number of incoming requests over time or incoming/outgoing traffic. One advantage of CPU

utilization as a metric is its ability to be directly priced in a 'processing power as a service' manner, as is currently the case for some cloud providers, which price CPU cycles.

Finally, concerning the database experiments in specific, we used not only the same configurations for the two different database systems but also corresponding data modelling, as explained in the relative section. This was necessary in order to evaluate and compare the different systems under as similar conditions as possible.

6.4.3 External Validity

External validity refers to the degree to which the results can be generalized to real-world settings. The most significant external threat to our experiments is the use of specific technologies. Our approach was to use the currently most popular technologies as representative examples of each paradigm. Thus, we used Java for the development of our web applications, we used Amazon AWS for our cloud services, which is currently the leading IaaS provider in terms of market share, we used MySql as a representative relational database system and MongoDB as a NoSql database system, which are currently the most popular open-source Sql and NoSql systems respectively.

Despite using the most perdominant technologies as representative examples, it should be highlighted that there is great variability with regards to cloud services among different providers. Different clouds might have different characteristics depending on many factors such as proprietary hardware and software appliances, architectures and business

processes. In addition, different vendors usually offer different services, SLAs, pricing policies and management capabilities.

Moreover, great variance exists even for the cloud services within the same provider. As we noticed throughout our testing, different types of VMs can have significant differences in performance. However, as we noticed even two VMs of the same type, for example two Amazon EC2 small instances can differ considerably. This would suggest that repeating our experiments might not produce exactly the same results; for instance ranking of different adaptations based on performance might change since the differences are already small.

Another threat to our external validity is the use of specific thresholds and configuration parameters. These configuration parameters should not be taken as panacea as they were selected randomly simply for the experimental purposes. With regards to our constraint that average response time should remain below transmission period, which we also expressed as an SLO, it was generated through observation of the testing results and perhaps more strict or more loose SLO definitions would be more appropriate in different cases, especially when transmission period changes significantly.

Finally, as explained in Subsection 6.2.4, in no case should our conclusions derived from the database experiments be generalized for the use of Sql versus NoSql databases. In an attempt to compare these databases under similar settings, the way we used them was anorthodox, using MongoDB running on only one VM and using MySql with only one big table and no relationships. Thus, these results should only be comprehended as

general findings on concrete characteristics of the two database systems and with regards to our specific use case application architecture.

6.5 Summary

In this chapter we presented our empirical results from the implementation and experimental evaluation of our proposed architecture and adaptation mechanism.

First we described our experimental environment and presented the architecture of the Workload Generator that we built to simulate clients and monitor performance metrics. Subsequently, showing the results we obtained using two alternative database systems we proved that our architecture provides the capability of using different technologies in one tier without affecting the others and also the ability to test the alternative configuration of our application and draw useful performance conclusions. One such conclusion was that using MongoDB led to better results than MySql in terms of overall performance and through our results we also made some interesting observations about general characteristics of these two different systems and their comparison under similar conditions. However as we stressed out these results should not be generalised for these systems in any context and more detailed and complex testing would be required to showcase and compare their full potential.

Concerning our adaptation mechanism, we presented results comparing only server adaptation, only client adaptation and both adaptations, where points of adaptation were a specialization of the adaptations that can be applied to our architecture, namely scalability, location-based adaptivity and routing of requests to the right server according

to servers' resource utilization. Our results prove that the application of both server and client adaptivity, as suggested by our proposed mechanism, produces better overall results in terms of combined performance and cost-efficiency while offering location-based adaptivity support.

Finally we described in detail threats to validity of our research results, aiming at explaining what could be possible differences if our experiments were repeated and setting the expectations for generalizing our results. Thus we mentioned among others the threats of cloud variability and the use of particular technologies and we explicated how our results should be interpreted.

Chapter 7
Conclusions

In this thesis we proposed a 3-tier architecture and a comprehensive two-layer adaptation mechanism for mobile spatio-temporal applications, taking under consideration the particular nature of these dynamic applications and the available emerging paradigms such as cloud and NoSql databases. In order to validate the feasibility of our proposed framework, we considered, implemented and tested the use case of a mobile application for the collection of data to monitor traffic and assess the quality of roads. Our collected empirical data, after conducting a series of experiments, proves that the use of our architecture and mechanism can result in high performance, robust and cost-efficient applications with capabilities of context-aware adaptation.

For the conduct of this thesis we studied the existing research work on Cloud Computing, Spatio-temporal applications and Autonomic Computing, in an attempt to acquire valuable background knowledge on these areas. We also reviewed related work on traffic monitoring and road assessment using mobile devices as well as autonomic in mobile applications, in order to adopt useful ideas and identify opportunities for improvement in these fields of research. Taking this under consideration we believe that the contribution of our work can be summarized as follows:

- We introduced and validated the use of an adaptivity-enabling architecture for mobile spatio-temporal applications. This 3-tier modular architecture allows to

test, deploy, replace and manage each of its tiers independently and allows for the application of adaptive methods at various angles, while exploiting mobile device features.

- We proposed a two-layer adaptation mechanism for spatio-temporal mobile applications. This mechanism is compatible with our proposed architecture and it offers benefits in combined performance and cost while supporting context-aware adaptivity, as proven by our experimental results.
- We conducted a series of experiments which allowed us to evaluate alternative settings and configurations of our application and acquire an understanding of our problem space by observing concrete results. In this regard, we tested MySql and MongoDB as alternative databases for our application and drew useful conclusions about their performance.

Regarding our 3-tier architecture we explained how it meets some of the most important requirements for mobile spatio-temporal applications, such as the ability to handle time-varying, geographically distributed traffic. We justified how its modular design fosters dynamicity and the application of adaptivity at multiple angles. In addition, we explicated how our architecture addresses the importance of mobile device capabilities using sensing, location management and local storage components to promote mechanisms for reliable data transmission.

Our work on the adaptivity part was based on concepts from Autonomic Computing and we proposed the use of two adaptive loops that operate concurrently; one at the server

and one at the client side. We showed the algorithm for each adaptation and how these two adaptations can cooperate and result to application stability on one hand and flexibility on the other. In this respect, we explained how this mechanism can support the prioritization of requirements and the strict satisfaction of SLOs while at the same time facilitating additional desired features such as context-aware adaptivity in a best-effort manner.

In an effort to empirically support our assumptions about our proposed architecture and mechanism and showcase the benefits from their use, we conducted a series of experiments. Our experimental results validate all three hypotheses as we defined them in our work:

1) *Our proposed architecture can support the functionalities of a spatio-temporal application and be implemented in alternative configurations.*

This hypothesis was confirmed by implementing our use case application, showing that it can produce useful results and testing it under different adaptation configurations and with two alternative database systems.

2) *In the context of three-tier, real-time applications, a NoSql Database System has significantly better performance than Sql Database System.*

We verified this assumption by testing two different database systems, MySql and MongoDB and showed that the latter can produce smaller network traffic, while performing faster.

3) *Location-awareness and elastic scaling can be supported for mobile spatio-temporal applications, while maintaining response time lower than transmission period and infrastructural cost to a minimum.*

This hypothesis was validated by applying a specialization of our adaptation mechanism on our implemented use case application and showing that response time can sufficiently remain under transmission period (set at 1 sec for our experiments), with support of scalability and location-based redirection of requests which leads to cost reduction.

7.1 Future Work

Our work can be considered as a first step towards the definition of a holistic approach for the application of adaptive methods specific to the class of mobile spatio-temporal applications. In this regard, during our research we identified many aspects that should be addressed through further work.

One of the points that should be carefully examined in future research is the improvement of these applications deriving from data tier scalability. As we mentioned in this thesis, our 3-tier architectures enable the deployment of both application and data tiers on scalable infrastructures, but through our experiments we only showed application tier scalability. Complex database design and thorough evaluation of different database architectures extended beyond the scope of the current thesis; however in light of new database technologies such as MongoDB which favour horizontal scalability, it would be

very interesting to design experiments that expose this feature and how it can support persistence in context of data intensive mobile spatio-temporal applications.

Regarding the data tier, it would also be valuable to examine the case of more complex relationships between data objects. Considering existing research about modelling spatio-temporal data it would be of great interest to test the performance of our architecture using more complex database schemas. Our intuition is that relational databases would perform better under these circumstances but one compelling area of research would be the support of such complex data relationships using NoSql databases. In the same spirit, the support for geo-spatial indexes which exists in MongoDB in specific, should also be tested systematically.

When it comes to adaptivity, there should be further work on the application of optimal algorithms and configuration parameters for specific use cases of applications. Moreover, further research on additional context features, other than location, that can be managed by similar architectures to ours would be very useful. In this direction, the deployment of such applications on different clouds should be evaluated, in terms of the different capabilities that they offer which can vary significantly due to cloud variability and early evolution phase.

Furthermore, the implementation details of our proposed adaptation mechanism should further be examined and analyzed, as it is important to clarify some main concepts such as the communication between different layers of adaptation. Throughout our work we assume some shared state available to both server and client adaptations but it is

definitely a challenging task to assess efficient ways in which this state information can be delivered to different layers as well as persisted.

Finally, serious attention should be paid to security issues of mobile spatio-temporal applications, since large amount of data are derived from users, much of which could be sensitive. Authorization and authentication processes should be examined in future work so that users can transmit data over secure networks. In addition, availability is one of the key features of such applications and thus they should be shielded from security attacks which is one more area that should further be researched.

Bibliography

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski et al., “Above the Clouds: A Berkeley view of cloud computing,” UC Berkeley Reliable Adaptive Distributed Systems Laboratory White Paper, 2009.

[2] R. Buyya, J. Broberg, A. Goscinski, “Cloud computing principles and paradigms,” John Wiley & Sons, INC., Publication, March 2011.

[3] Vaquero, L.M., Rodero-Merino, L., Caceres, J., Lindner, M., “A break in the clouds towards a cloud definition,” ACM SIGCOMM Comput. Commun. Rev. 39[37], pp.50–55, 2009.

[4] Mell, P., Grance, T., “Perspectives on Cloud Computing and Standards,” National Institute of Standards and Technology (NIST), Information Technology Laboratory, 2009.

[5] Han Liang; Wu Jie; Xie Kunqing; Ma Xiujun; Xu Dan; Zhang Huibin; Chen Zhuo; , "A spatio-temporal database prototype for managing moving objects in GIS," Geoscience and Remote Sensing Symposium, 2005. IGARSS '05. Proceedings. 2005 IEEE International , vol.2, no., pp. 4 pp., 25-29 July 2005.

[6] Stojanovic, D.; Djordjevic-Kjan, S.; Stojanovic, Z.; , "Modeling and management of spatio-temporal objects within temporal GIS application framework," Database Engineering and Applications, 2001 International Symposium on. , vol., no., pp.249-254, 2001.

[7] Christine Parent, Stefano Spaccapietra, Esteban Zimányi, “Conceptual Modeling for Traditional and Spatio-Temporal Applications: The MADS Approach” Lecture Notes in Computer Science, Vol. 3534, Springer, 2006.

[8] Nikos Pelekis, Babis Theodoulidis, Ioannis Kopanakis, and Yannis Theodoridis, “Literature review of spatio-temporal database models” Knowl. Eng. Rev. 19, 3 (September 2004), 235-274.

[9] Nami, Mohammad Reza; Sharifi, Mohsen; , "Autonomic Computing: A New Approach," Modelling & Simulation, 2007. AMS '07. First Asia International Conference on , vol., no., pp.352-357, 27-30 March 2007.

[10] M . Parashar, S. Hariri, “Autonomic Computing: Concepts, Infrastructure, and Applications,” Taylor & Francis Group, LLC, Publication, 2007.

[11] J. Appavoo and et al. Enabling autonomic behavior in systems software with hot swapping. In IBM Systems Journal, volume 42, pages 60–76, January 2003.

[12] Khalid, A.; Haye, M.A.; Khan, M.J.; Shamail, S.; , "Survey of Frameworks, Architectures and Techniques in Autonomic Computing," Autonomic and Autonomous Systems, 2009. ICAS '09. Fifth International Conference on , vol., no., pp.220-225, 20-25 April 2009.

[13] Zhenxing Zhao; Congying Gao; Fu Duan; , "A survey on autonomic computing research," Computational Intelligence and Industrial Applications, 2009. PACIIA 2009. Asia-Pacific Conference on , vol.2, no., pp.288-291, 28-29 Nov. 2009.

[14] Zambonelli, F.; Bicocchi, N.; Cabri, G.; Leonardi, L.; Puviani, M.; , "On Self-Adaptation, Self-Expression, and Self-Awareness in Autonomic Service Component Ensembles," Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2011 Fifth IEEE Conference on , vol., no., pp.108-113, 3-7 Oct. 2011.

[15] K. De Zoysa, C. Keppitiyagama, G.P. Seneviratne, and W.W.A.T. Shihan, “A public transport system based sensor network for road surface condition monitoring,” in Proceedings of the 2007 workshop on Networked systems for developing regions, ser. NSDR ’07. New York, NY, USA: ACM, 2007, pp. 9:1–9:6.

[16] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan, “The pothole patrol: using a mobile sensor network for road surface monitoring,” in Proceeding of the 6th international conference on Mobile

systems, applications, and services, ser. MobiSys ’08. New York, NY, USA: ACM, 2008, pp. 29–39.

[17] P. Mohan, V. N. Padmanabhan, and R. Ramjee, “Nericell: using mobile smartphones for rich monitoring of road and traffic conditions,” in Proceedings of the 6th ACM conference on Embedded network sensor systems, ser. SenSys ’08. New York, NY, USA: ACM, 2008, pp. 357– 358.

[18] Bhoraskar, R.; Vankadhara, N.; Raman, B.; Kulkarni, P.; , "Wolverine: Traffic and road condition estimation using smartphone sensors," Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on , vol., no., pp.1-6, 3-7 Jan. 2012.

[19] Mednis, A.; Strazdins, G.; Zviedris, R.; Kanonirs, G.; Selavo, L.; , "Real time pothole detection using Android smartphones with accelerometers,"Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on , vol., no., pp.1-6, 27-29 June 2011.

[20] Aksamit, P.; Szmechta, M.; , "Distributed, mobile, social system for road surface defects detection," Computational Intelligence and Intelligent Informatics (ISCIII), 2011 5th International Symposium on , vol., no., pp.37-40, 15-17 Sept. 2011.

[21] Ghose, A.; Biswas, P.; Bhaumik, C.; Sharma, M.; Pal, A.; Jha, A.; , "Road condition monitoring and alert application: Using in-vehicle Smartphone as Internet-connected sensor," Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on , vol., no., pp.489-491, 19-23 March 2012.

[22] Woodside, M.; Tao Zheng; Litoiu, M.; , "Service System Resource Management Based on a Tracked Layered Performance Model," Autonomic Computing, 2006. ICAC '06. IEEE International Conference on , vol., no., pp. 175- 184, 13-16 June 2006.

[23] Amazon Web Services. Available: http://aws.amazon.com/.(URL)

[24] Neophytou, M.; Stavrou, K.; Vassiliou, V.; Pitsillides, A., "The Importance of Adaptive Applications in Mobile Wireless Networks," Software in Telecommunications and Computer Networks, 2006. SoftCOM 2006. International Conference on , vol., no., pp.96,101, Sept. 29 2006-Oct. 1 2006.

[25] Paspallis, N.; Papadopoulos, G.A., "An Approach for Developing Adaptive, Mobile Applications with Separation of Concerns,"Computer Software and Applications Conference, 2006. COMPSAC '06. 30th Annual International , vol.1, no., pp.299,306, 17-21 Sept. 2006.

[26] Maciel da Costa, C.; da Silva Strzykalski, M.; Bernard, G., "An Aspect Oriented Middleware Architecture for Adaptive Mobile Computing Applications," Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International , vol.2, no., pp.81,86, 24-27 July 2007.

[27] Gustavo G. Pascual, Monica Pinto, Lidia Fuentes, “Run-Time Adaptation of Mobile Applications using Genetic Algorithms,” SEAMS 2013.

[28] Mowafi, Y.; Dongsong Zhang, "A User-centered Approach to Context-awareness in Mobile Computing," Mobile and Ubiquitous Systems: Networking & Services, 2007. MobiQuitous 2007. Fourth Annual International Conference on , vol., no., pp.1,3, 6-10 Aug. 2007.

[29] David, L.; Endler, M.; Barbosa, S.D.J.; Filho, J.V., "Middleware Support for Context-Aware Mobile Applications with Adaptive Multimodal User Interfaces," Ubi-Media Computing (U-Media), 2011 4th International Conference on , vol., no., pp.106,111, 3-4 July 2011.

[30] Mizouni, R.; Serhani, M.A.; Benharref, A.; Al-Abassi, O., "Towards Battery-Aware Self-Adaptive Mobile Applications,"Services Computing (SCC), 2012 IEEE Ninth International Conference on , vol., no., pp.439,445, 24-29 June 2012.

[31] Misra, A.; Lipyeow Lim, "Optimizing Sensor Data Acquisition for Energy-Efficient Smartphone-Based Continuous Event Processing," Mobile Data Management (MDM), 2011 12th IEEE International Conference on , vol.1, no., pp.88,97, 6-9 June 2011.

[32] Alnawaiseh, A.; Abdelghany, K., "TeamSense: Energy-efficient autonomous mobile wireless sensor networks for object tracking," Wireless Communications and Mobile Computing Conference (IWCMC), 2012 8th International, vol., no., pp.660, 665, 27-31 Aug. 2012.

[33] PhoneGap. Available: http://phonegap.com/.(URL)

[34] HTML5specification. Available:http://www.w3.org/html/wg/drafts/html/master/.(URL)

[35] MongoDB. Available: http://www.mongodb.org/.(URL)

[36] Play Framework. Available: http://www.playframework.org/.(URL)

[37] Apache Tomcat. Available: http://tomcat.apache.org/.(URL)

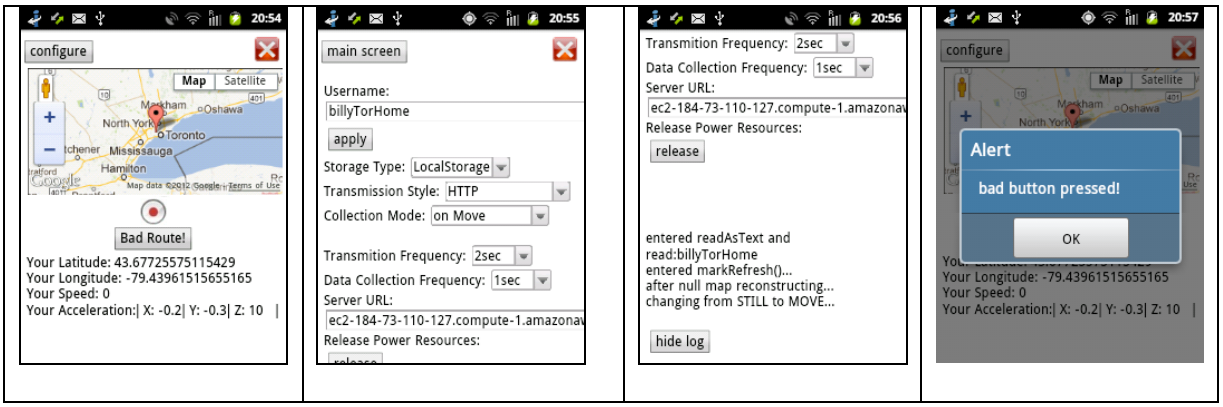
[38] Amazon CloudWatch. Available: http://aws.amazon.com/cloudwatch/.(URL)

[39] Apache JMeter. Available: http://jmeter.apache.org/.(URL)

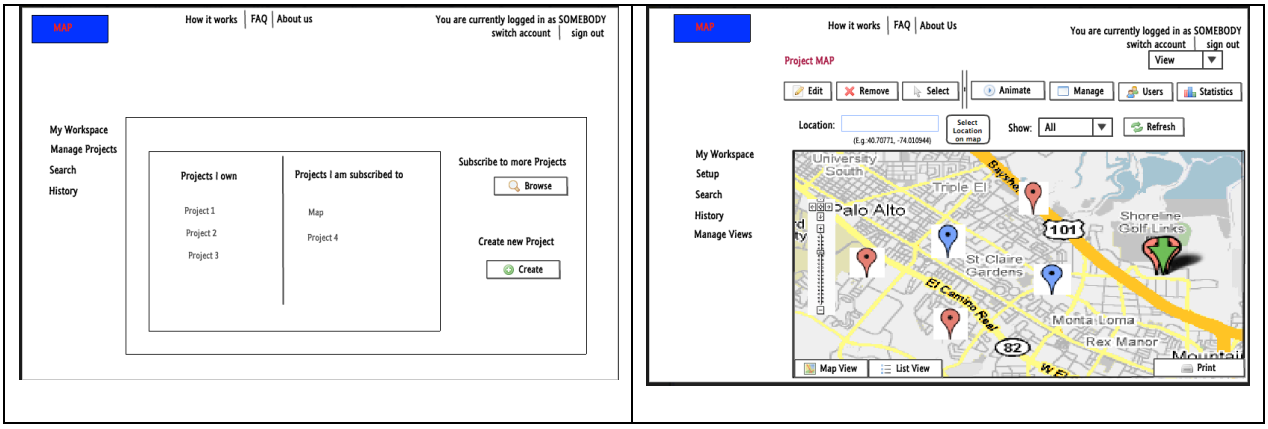
[40] Solomon, B.; Ionescu, D.; Litoiu, M.; Mihaescu, M., "Towards a Real-Time Reference Architecture for Autonomic Systems," Software Engineering for Adaptive and Self-Managing Systems, 2007. ICSE Workshops SEAMS '07. International Workshop on , vol., no., pp.10,10, 20-26 May 2007.

[41] Ionescu, D.; Solomon, B.; Litoiu, M.; Mihaescu, M., "A Robust Autonomic Computing Architecture for Server Virtualization," Intelligent Engineering Systems, 2008. INES 2008. International Conference on , vol., no., pp.173,180, 25-29 Feb. 2008.

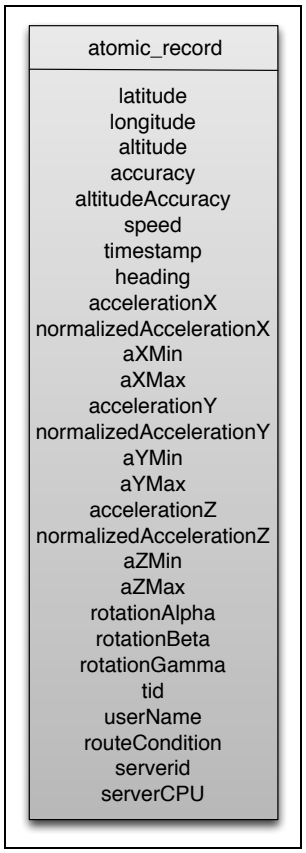
Appendix A: Implementation Details



Mobile application screenshots



Administrator view screenshot



Database model