Crowd-sensing for Smart City Applications

Towards solving crowd-sensing data challenges by introducing edge and cloud services

Aseel T. Alkhelaiwi
MSc, BSc. (Hons) Computer Science
113224229

Thesis submitted for the degree of Doctor of Philosophy

National University of Ireland, Cork

Faculty of Science

Department of Computer Science

January 2019

Head of Department: Professor Cormac J. Sreenan

Supervisor: Dr. Dan Grigoras

Research supported by King Saud University in Saudi Arabia

Contents

List of Figures

List of Tables

List of Algorithms

I, Aseel T. Alkhelaiwi, certify that this thesis is my own work and I have not obtained a degree in this university or elsewhere on the basis of the work submitted in this thesis.

Aseel T. Alkhelaiwi

Abstract

Crowd-sensing is the ability of a crowd to utilize sensors embedded in mobile devices to sense the surroundings and then send data to a centralized server or the cloud. With crowd-sensing, a wide range of applications have been empowered, such as smart city, healthcare and marketing, of which the smart city is the domain of interest in this research. However, sending a large amount of data to the cloud has introduced several challenges, such as data truthfulness, redundancy, transfer cost, bandwidth consumption and the way data are stored and managed in the cloud.

This thesis presents a crowd-sensing architecture for smart city applications. This architecture contains several services that play a key role in solving a number of the challenges listed earlier. Services are distributed between the cloud and public local servers. The local servers are distributed around a city to improve citizens' quality of life. Services located on public local servers are called edge services and are concerned with trust, the scheduler and compression. Services located in the cloud are known as cloud services and contain a partitioning method along with two reduction techniques: optimization and context extraction.

The trust service calculates trust using different factors. Then, if the trust value is above a predefined threshold, data are trusted; otherwise, they are discarded. The scheduler removes redundant data and schedules sending data to the cloud depending on their priority. The compression service compresses singleprecision floating-point data using two lossless compression algorithms. The partitioning method in the cloud highlights the importance of data en- tries using time, access rate and singularity factors. Then, based on the output of this method, users can apply optimization and context extraction to optimize data entries and extract important information, respectively. The order in which these services are performed and how they work and communicate are presented.

Evaluations and use cases are performed on the mobile, local server and the cloud using Android-based mobile devices and the Amazon EC2 cloud. The results show the effectiveness of the proposed work by meeting predefined requirements, such as reducing the amount of the data transferred.

Acknowledgements

This thesis becomes a realty with the great support of many individuals. I would love to express my sincere thanks to all of them.

First of all, I would like to extend my special thanks to my husband, Nasser, my beautiful daughter, Haya, my father, Turki, my mother, Mariam and my siblings, May, Lama and Sulaiman. This dream will not be achieved without your sacrifices, support and patience throughout my PhD journey. I love you all.

I would like to express my special thanks to my supervisor Dr. Dan Grigoras. Thank you for your guidance and valuable comments throughout the last few years. Thank you for your wisdom, enthusiasm, patience and pushing me farther than I thought I could go. I really can't thank you enough.

I also want to thank Dr. John Herbert and Professor Dana Petcu, internal and external examiners of this thesis respectively. I am extremely grateful for your suggestions and observations during my Viva Voce examination.

I wish to thank all my colleagues in the Mobile and Internet Systems Laboratory (MISL) for their helpful comments during my reading group presentations. A special thanks goes to Dr. Hazzaa Alshareef and Dr. Michael O'Sullivan (my previous two mobile-cloud PhD colleagues) for their great support and encourage especially on the first two years. I also want to thank Mary Noonan in MISL for her great help, kindness and beautiful soul. I want to thank all of my friends in Cork who supported me emotionally and gave me such a wonderful memories.

Finally, I would like to dedicate this work to my father, mother, husband and daughter for all of the support you gave me to fulfill my dream.

Aseel Alkhelaiwi

Chapter 1

Introduction

1.1 Mobile Cloud Computing and Crowd-sensing

In recent years, a cloud computing model has been widely used due to the reliable, fast and practical services clouds can offer through the Internet. The adoption of cloud computing has seen rapid growth in different domains, such as business, government and academia. Cloud computing offers powerful computational ability and storage, which has attracted different technologies such as mobile devices. Today, mobile devices play an important role in people's lives in terms of socializing, gaming, conducting business and commerce. However, mobile devices suffer from low storage capacity and computational capability and energy constraints. Therefore, the emergence of cloud computing and mobile devices has introduced the notion of mobile cloud computing (MCC), enabling mobiles to use cloud capabilities. This means that mobile devices do not need powerful computations or storage in the device itself, as they can utilize different resources in the cloud. MCC is a motivating technology for crowd-sensing, which utilizes the different sensors in the mobiles and can upload a large amount of data to the cloud to be analysed, managed and stored.

Crowd-sensing can enable a broad range of applications, such as smart city (environmental, planning, traffic, etc.), healthcare, social and advertising, which are powered by a crowd's abilities to sense using mobile devices. The smart city is the domain of interest in this thesis. As an example of a crowd-sensing application and the type of data produced by it, consider a pothole application that is implemented for citizens to send locations of bad or dangerous potholes in the roads in a particular city. Citizens can start sensing by taking pictures of the potholes and tagging them with the location, time and an optional description. Alternatively, a mobile device can perform the detection automatically by sending a voice note of the noise when a pothole is detected using the accelerometer. Therefore, data produced in this application are: GPS readings, photographs, voice notes and text descriptions (if any). The data produced from crowd-sensing can be large or, sometimes, extremely large depending on the number of users contributing data through crowd-sensing applications.

1.2 Motivation

With the increased utilization of mobile devices in recent years, citizens can now exploit crowd-sensing applications as a communication tool between themselves and the local government (i.e. city council or government initiatives). Furthermore, local government can measure citizens' satisfaction and receive their community-oriented problems (e.g., traffic congestion, potholes, environmental issues, etc.) by issuing crowd-sensing applications. Thus, citizens can reflect their opinion about a particular issue in order to improve the quality of life in the city.

However, these applications have several challenges in terms of the battery life, cost and bandwidth consumption involved in the huge data transfer of mobile devices and the reliability of the data transferred. The way in which crowd-sensing applications are heavily dependent on cloud computing to store these large amounts of data also causes other challenges to arise. These challenges reside in the way data are stored and managed in the cloud and the amount of time these data need to be there. This is due to the cloud pay-as-you-use model, whereby the cost increases as time passes for continuously storing new data and keeping these data in the cloud without filtration (i.e. the removal of unwanted or useless data).

From this perspective, crowd-sensing and a cloud architecture that can tackle these challenges (or at least some of them) is very much needed. This architecture needs to introduce a new model of computation that can make the data and the transfer of that data from the crowd to the cloud as trustworthy and as low in cost and bandwidth as possible. Furthermore, this architecture needs efficient managing criteria for data stored in the cloud in order to benefit different applications (particularly smart city applications).

1.3 Challenges

Crowd-sensing has a number of challenges that occur either on the user's side or on the cloud's. User-side challenges are all the difficulties that the user faces during the sensing task and while the crowd-sensed data are sent to the cloud. Cloud-side challenges are all the complications that occur after the crowd-sensed data are received in the cloud.

1.3.1 Challenges Related to the User End

1. Users might have mobile preferences and privacy concerns that are not supported by crowd-sensing applications. Some users will have concerns about disclosing their personal information, such as name, date of birth and location.

2. Battery life: the energy consumption in terms of mobile battery life is another challenge for crowd-sensing applications due to the huge battery consumption when using the sensors located in a mobile device.

3. Costs and user incentives: there are costs in terms of price and users' efforts when they utilize their mobile device for sensing tasks. The costs occur when users send crowd-sensed data using their mobile operator (3G/4G). Therefore, convincing users to take part in sensing tasks and avoiding higher energy consumption as well as costs are major challenges for the implementation of crowd-sensing applications. Therefore, for a crowd-sensing application to succeed, there must be a proper incentive mechanism to engage users in participating in the sensing task.

1.3.2 Challenges Related to the Cloud End

1. Trustworthiness: different user applications, particularly smart city applications, depend on users' involvement and the sensed data received from them. However, data should be trustworthy and truthful in order to benefit and improve the quality of life for citizens in the smart city context.

2. Data origin: tracing the origin of data without affecting users' privacy is an important aspect. The cloud receives a large amount of data and stores these data for a long time. Therefore, if errors occur in the data and nothing indicates the origin of those data, the applications that should benefit will generate the wrong results and incorrect information. Hence,

ideally, it should be possible to identify the source of data in the cloud when an error occurs. Furthermore, tracing the origin of data might be useful in other scenarios such as, data analysis purposes. For example, calculating percentages of damaged roads in different locations in a city can only be captured by locating data origin.

3. Large data transfer: transferring large amounts of data from a mobile device to the cloud is not ideal, since the transfer will take a long time due to the bandwidth limitations of different networks, especially 3G/4G/Long Term Evolution (LTE).

4. Data storage: with the limited resources of mobile devices, sensed data are usually offloaded to the cloud. However, in smart city applications and others, such as scientific applications, data volumes will increase at a very high speed and this will introduce the challenges of storing these data as the costs are also increased. Managing and reducing data in the cloud might cause the loss of important information. Therefore, the managing process must be efficient enough to avoid the loss of critical features and values.

In this thesis, only the cloud-side challenges are taken into consideration, since only the challenges occurred after collecting crowd-sensed data are the main concern in this thesis and ,therefore, the proposed work is based on overcoming these barriers.

1.4 Summary of Thesis Contributions

The contributions provided by this thesis to the crowd-sensing and mobile cloud computing domains are as follows:

- An analysis of related works employing the integration of crowd-sensing, mobile cloud computing and smart city applications.
- The adoption of local processing using public local servers (i.e. edge servers) for implementing different services with data produced by crowd-sensing applications before sending them to the cloud. These edge servers will include different filtration processes applied to the crowd-sensed data. The local processing adopted in this thesis will have several benefits, particularly in reducing the amount of data sent to the cloud and bandwidth consumption.

- A novel reputation system for data produced by crowd-sensing applications to assess the trustworthiness of these data. This reputation system is located in the proximity of the users (i.e. data contributors) using public local servers (i.e. edge servers). Data are assessed by different factors, as presented in chapter 4, whereby trusted data are considered for sending to the cloud and untrusted data are simply discarded. The history of the trust values for all the previous contributions of every user is combined and calculated in the cloud. Therefore, every user will have a reputation history that will play a significant role in further decisions throughout the edge computation.

- Support for the utilization of traceability for data produced from crowd-sensing applications without exposing users' identities, in order to ensure anonymity. This will help in identifying the origin of data and the source of errors if any occur. With the element of traceability, different smart city applications in the cloud will benefit most from the data sensed.

- A scheduling service that is located in public local servers (i.e. edge servers) that will schedule the transmission of trusted data to the cloud depending on their priority. This service will also remove data contributions that are captured in the same location; this removal is based on the reputation values of the users who contributed to the sensing process.

- A novel lossless compression service that takes advantage of the distributive nature of local servers around a city. This service is performed on single-precision floating-point data sets and contains two compression algorithms: one for location-based data (latitude and longitude) and the other for an accelerometer's three-dimensional data.

- A novel smart city data management architecture that contains a partitioning method. This method is performed with smart city relational databases located in the cloud in order to reduce the cost and the amount of data stored. The partitioning method highlights the importance and sensitivity of certain data parts (i.e. chunks) in order to perform reduction services. Using this method, users can freely perform reduction services on unnecessary data, which will improve storage and reduce costs without losing important data.

- Two storage reduction services (optimization and context extraction) located in the cloud and part of the smart city data management architecture. These services are applied to the databases depending on the sensitivity of the data.

1.5 Thesis Structure

The remainder of this thesis is ordered as follows:

- Chapter 2, "Background and Literature Review", outlines the background to the topics related to this thesis and presents previously implemented and theoretical works in the areas of crowd-sensing and smart city applications and architectures.
- Chapter 3, "Architecture and Design", describes the challenges for implementing a crowd-sensing architecture and lists the requirements for such implementation. Furthermore, this chapter introduces the design and architecture of crowd-sensing that meets requirements and overcomes the challenges listed.
- Chapter 4, "Towards Reduction Before the Cloud", presents the local processing model that is located on the edge servers. These servers are public local servers that are distributed around a city to improve citizens' quality of life. The server contains three services: the first service is the trustworthiness evaluation of the data received; the second is the scheduling of the trusted data based on their priority; and the final one is a reduction service that includes two compression algorithms for floating-point numbers. This chapter presents the design of each of the three services and evaluations performed on these services to demonstrate their outcomes.
- Chapter 5, "Data Management in the Cloud", introduces the partitioning method for the smart city databases located in the cloud and how this method can partition data in a way that highlights the sensitivity and importance of certain data entries. Based on the partitioning method outcomes, two reduction services are presented: optimization and context extraction. This chapter demonstrates how the partitioning method is performed and on what data parts (i.e. chunks) the reduction services can be applied without losing any important features or values.
- Chapter 6, "Evaluation", presents two use cases that demonstrate the effectiveness of the approaches presented in chapters 4 and 5. The first use case presents an evaluation that demonstrates how the edge services will

work on real crowd-sensed data. The second use case will show how data management will take place in the cloud.

- Chapter 7 is the conclusion and outlines future work.

1.6 Publications

Chapters of this thesis are put together from the following publications:

[7] A. Alkhelaiwi and D. Grigoras, “The origin and trustworthiness of data in smart city applications,” *IEEE/ACM 8th International Conference on Utility and Cloud Computing*, 2015, pp. 376-382.

[6] A. Alkhelaiwi and D. Grigoras, "Scheduling crowd sensing data to smart city applications in the cloud," *2016 IEEE 12th International Conference on Intelligent Computer Communication and Processing (ICCP)*, Cluj-Napoca, 2016, pp. 395-401.

[5] A. Alkhelaiwi and D. Grigoras, "Data reduction as a service in smart city architecture," *2017 IEEE 3rd International Conference on Big Data Computing Service and Applications (BigDataService)*, San Francisco, CA, 2017, pp. 172-178.

[3] A. Alkhelaiwi and D. Grigoras, "Smart City Data Storage Optimization in the Cloud," *2018 IEEE Fourth International Conference on Big Data Computing Service and Applications (BigDataService)*, Bamberg, 2018, pp. 153-160.

[4] A. Alkhelaiwi and D. Grigoras Challenges of Crowd Sensing for Cost-Effective Data Management in the Cloud. In: Zbakh M., Essaaidi M., Manneback P., Rong C. (eds) Cloud Computing and Big Data: Technologies, Applications and Security. CloudTech 2017. Lecture Notes in Networks and Systems, 2019, vol 49. Springer, Cham

The end of each chapter will indicate from which, if any, of the above publications the materials were taken. Parts of this chapter were published in the Springer book chapter [4].

Chapter 2

Literature Review

2.1 Introduction

The proliferation of mobile devices with sensing capabilities (e.g., the Global Positioning System GPS, accelerometers, cameras and microphones) has made them a rich source of sensing data [88]. People-centric sensing can be used to develop a wide range of applications to improve citizens' quality of life, including environmental, traffic monitoring, smart city and healthcare apps. However, due to the limited resources of the mobile devices themselves, sensed data are usually offloaded and processed in the cloud.

Cloud computing provides resources to clients on an on-demand basis using the Internet. Mobile cloud computing (MCC) has the added benefit of being a motivating technology for context awareness and crowd-sensing by uploading a large amount of sensed data (from mobile phones) to the cloud to be processed and to feed different applications. Furthermore, by using clouds, mobile devices will, in most cases, avoid short battery life and memory constraints. Thus, cloud computing for mobile devices is a very attractive and productive trend.

The cloud not only offers sensor data collection and scalable storage, it can also be used as a hub that is accessible to users or third parties who have permission to use the data and can then produce useful applications that can serve a city in different ways. From this perspective, the cloud has unlimited potential for smart cities. In this chapter, an investigation of previous work takes place in the area of mobile cloud crowd-sensing in the context of smart city applications.

This chapter is organized as follow. Section 2 introduces the background. Section 3 presents the characteristics of and previous studies on context awareness, crowd-sensing and crowdsourcing. Section 4 presents smart city applications and previous mobile cloud platforms that benefit these applications. Section 5 is the chapter summary.

2.2 Background

This section introduces and defines the main concepts considered in this thesis. The concepts are cloud computing, mobile cloud computing and mobile crowd-sensing.

2.2.1 Cloud Computing

Cloud computing is a smart technology that offers on-demand computing services to users while ensuring scalable and reliable storage. The National Institute of Standards and Technology (NIST) [106] defines cloud computing as: *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. (p.6)* [106] Cloud computing has several benefits, such as reliability, being easy to access, availability and robustness. However, there are also challenges, such as privacy and security [14].

Cloud computing offers different deployment services and users can choose whichever suits their needs. These deployment services are [121]:

1. Infrastructure as a Service (IaaS): delivers the cloud infrastructure (servers, storage, etc.) in a pay-as-you-go model.
2. Platform as a Service (PaaS): delivers everything needed for application development, without worrying about the infrastructure setup.
3. Software as a Service (SaaS): delivers applications over the Internet.

2.2.2 Mobile Cloud Computing

MCC is a combination of cloud computing and mobile systems, whereby mobile users can overcome the limitations of mobiles (e.g., limited storage and processing) and benefit from cloud computing capabilities. The MCC Forum defines MCC as follows [113]:

Mobile cloud computing at its simplest, refers to an infrastructure where both the data storage and data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from mobile phones and into the cloud, bringing applications and MC to not just smartphone users but a much broader range of mobile subscribers.

MCC is useful in extending the battery life of mobile devices, since the processing and computations takes place in the cloud. MCC increases the data storage capacity and improves processing by utilizing the cloud.

2.2.3 Crowd-sensing

With the large sensing capabilities of mobile devices, users can use their mobile devices to sense different aspects of their environment, which introduces the mobile crowd-sensing area. Therefore, instead of deploying immutable wireless sensor networks around a city, crowd-sensing takes advantage of user mobility and the powerful mobile devices users have with them to sense around the city using different mobile applications. This approach makes crowd-sensing more cost-effective than immutable wireless sensor networks, as well as more scalable if users' sensing data are reliable and truthful.

There are numerous application areas that can benefit from crowd-sensing, some of which are [166]:

- Smart city applications:

  With the large populations in cities, there is a major need for effective city management in order to make cities smarter and improve the quality of life for their citizens. Smart cities contain several domain applications, such as those for the environment (e.g., weather and water quality), traffic, roadworks and road surveillance. Crowd-sensing engages citizens in the management of their city to make it smarter and safer.

- Healthcare applications:

  Mobile crowd-sensing is a promising paradigm for the healthcare area. It is a useful tool for gathering day-to-day information from large numbers of patients at a lower cost than medical body sensors [129] [130].

- Advertising applications:

  With users sharing their locations and other data using mobile crowd-sensing, advertising companies can use this information to promote their products [112].

- Social applications:

  Mobile crowd-sensing can be used as a tool for users to socialize with people and friends and share different information with them, such as their exercise data. One example is a mobile app called Swarm [164], developed by Foursquare [48]. Using Swarm, users can share their locations and experiences with their friends.

The smart city is the domain of interest in this thesis. Crowd-sensing in the context of the smart city is defined as a subset of users, called a crowd, located in the same area in a particular city who use mobile apps to utilize the sensors of their mobile devices and then send the data to a centralized server or cloud.

2.3 Mobile Cloud Sensing: Related Work

Mobile devices and the sensors [12] available on them (see Table 2.1) have led to sophisticated context-aware applications and systems. This area has captured a large amount of interest with regard to how mobile phone users can contribute sensor data to enabling awareness of the environment (e.g., air pollution and transportation), community, healthcare, etc.

Context awareness is considered the first generation of sensing and is defined in [124] as the ability of devices to sense, infer and interact with a user's local environment. Today, the second generation of sensing is also drawing a great deal of attention and is called crowd-sensing. Crowd-sensing is the acquisition of sensor data from multiple (not only one) mobile devices. Crowd-sensing is the capability to collect sensor data from a large number of mobile users upon request to be used towards a specific goal (e.g., locating events or road works) [133]. It enhances the efficiency of information collection in order to facilitate collective intelligence [167].

Crowd-sensing might go beyond collecting processed sensor data from mobile users to tasking them with human intelligence tasks, such as image recognition, tagging or translation. This is called a crowdsourcing system, an example of which is Amazon Mechanical Turk (AMT [9]).AMT provides infrastructure that enables hundreds of thousands of people to perform paid work on the Internet. AMT is used for a number of simple tasks, such as labelling or tagging content, and for complex tasks, such as translating or proofreading [9].

Mobile crowd-sensing applications have a number of characteristics that distinguish them from other sensing classes [52] as follow:

- The processing, communication, storage resources and sensing capabilities allow different and flexible capabilities for sensing the user's surroundings.
- Mobile devices do not need to be deployed; users carry their mobiles wherever they go.
- The different conditions of mobile devices due to device mobility, energy consumptions, communication, and users' preferences. It is, therefore, a difficult operation to guarantee the required quality.
- Different applications can benefit from the same sensor data.
- Users can be involved in different sensing tasks, since they carry their devices everywhere they go. However, some users have privacy and personal preferences that might not be compatible with some applications.
- There must be proper incentives to attract and involve users. Since user devices may suffer energy and monetary costs, this can lead to users being unwilling to participate unless there is some incentive.

2.3.1 Crowdsourcing

Mobile crowdsourcing applications can be classified into either stand-alone applications or extensions of Web-based applications [27]. Web-based applications target users who do not have access to an organization and offer them the ability to add their real-time location-based information to a service, such as Gigwalk [57], Jana [72], and work done in [92]. Stand-alone applications offer functionalities via crowdsourcing, such as collecting and searching, road, traffic and location-based applications, as well as applications that assign tasks to users.

Table 2.1: Mobile sensors

Sensor	**Format**	**Usage**
Accelerometer	Floating-point numbers for all three axes	Used for detecting acceleration and movement.
GPS	Longitude and Latitude as floating-point numbers	Used for determining the mobile phone user's location.
Camera	Images (BMP, GIF, JPEG, etc.)	Used for taking photographs.
Microphone	AAC, MP3, FLAC, etc.	Used for dealing with recordings.
Light sensor	Floating-point number (e.g. lx)	Used for detecting light levels.
Gyroscope	Floating-point numbers for all three axes	Used for revealing complex orientation features.
Proximity	Floating-point number	Used for detecting the location of a phone near a human ear.
Fingerprint	Different formats (Images, electrical charges, etc.) depending on the detection method used (optical, capacitive, etc.)	Used for authentication and verification.
Magnetometer	Floating-point numbers for all three axes	Used for detecting North and usually used with GPS.

2.3.1.1 Crowdsourcing by Collecting and Searching Data

Several works have been based on crowdsourcing. Authors of [49] argue that there are queries that cannot be answered by database schemes and search engines. Therefore, they propose CrowdDB, which utilizes individuals' entries through crowdsourcing to process queries. Although CrowdDB has many of the characteristics of traditional database systems, the conventional closed-world assumption [111] of query processing does not apply to individuals' entries. This important change makes the system more accurate and robust, since some crowdsourced data need to be merged and removed.

CrowdSearch is a crowdsourcing image search system for mobile devices presented in [188]. It integrates an automated image search implemented both locally and on remote servers with real-time human support for search outputs using AMT [9]. Search outputs are sent to the user after validation. With Crowd-

Search, there are some tradeoffs between battery consumption, efficiency, delay and validation costs. CrowdSearch tackles these tradeoff issues by creating a predictive algorithm that helps decide what to validate and how then to distribute the image search tasks between mobile devices and servers. Another crowdsourcing framework is the Crowdsourcing Data Analytics System (CDAS) [97]. Different applications with high user involvement for verification tasks can be performed by the CDAS system to ensure a high degree of accuracy. The CDAS system takes a two-phase approach: the given task is first implemented by a high-performance remote server; the result is then split into chunks and sent to AMT for verification and the result is achieved by combining the different chunks of the results.

Turning now to the area of disasters, today's disaster response protocols do not consider citizen collaboration [40]; they are still based on a centralized system that originated with the military [104]. In [60], authors propose a crowdsourcing system that allows people to share data for the area they are in, using their devices (equipped with GPS) to sense and guide them to safety. Other systems for disasters are Usahidi [172] and Sahana [148], which are crowdsourcing systems for the management of aid attempts deployed in several countries with the use of maps. Furthermore, a mobile application that observes the spread of dengue fever using GPS is proposed in [139]. The patient inputs the symptoms using the mobile app and waits for a response and guidance on how to avoid dengue fever or to call a doctor. The user's information and location (captured using GPS) are stored in a database for medical use. A map that displays the zones with a high incidence of ill people is also created.

Finally, Portolan [59] is a crowdsourcing system that relies on mobile devices to structure an annotated graph of the Internet. After the mobile devices observe the measures of the network around them, the information is gathered in a central server.

2.3.1.2 Crowdsourcing with Roads and Traffic

CrowdSC [19] is a crowdsourcing platform that utilizes citizen contributions in the context of a city. It converts queries such as "What roads are in need for repair?" into easy tasks: gather, filter the data contributed by citizens, and send the outcomes back to the citizens. After the requester posts a query, such as roads in need of repair, a number of images are received with different evaluations for the same location. To make the right evaluation, authors suggest three phases: a *data collection* phase that gathers images from contributing citizens; a *data selection* phase that requests other citizens to choose the picture that best describes the problem; and, finally, a *data assessment* phase that requires citizens to evaluate each picture nominated from the previous phase. Furthermore, the platform offers three general policies to manage the process: buffer, deadline, and first in, first out (FIFO). Policy selection is based on the user and the degree of urgency.

CrowdOut [15] is a crowdsourcing system for ensuring street safety in cities by asking citizens to record dangerous traffic situations (e.g., speeding) in real time and to apply them on a city map. The CrowdOut system has been used in an experiment in Nancy in France. The promising architecture can benefit future systems that utilize crowdsourcing and visualize the data sensed into a map. Another work is presented in [44] and is called "The Pothole Patrol", as it uses vehicles with installed sensors (vibrations and GPS) to collect data that are used to evaluate the road status. Authors installed their system in seven taxis in Boston, in the US, and show that their system identified a number of potholes. However, by using mobile devices to sense the road during a journey instead of deploying sensors in the vehicles, they might have avoided wasting the time and cost of deployment.

Other systems that address traffic and exploit mobile devices in their research work are VTrack [168] and SignalGuru [84]. VTrack's goal is to estimate journey time, while SignalGuru's goal is to anticipate a traffic light schedule.

2.3.1.3 Crowdsourcing and Location-based Services

There is a number of crowdsourcing projects for searching, routing and map creation in outdoor environments. However, recent statistics show that people allocate the majority of their time (80-90%) to indoor environments [163], such as shopping centres and libraries. Therefore, more research is being directed towards indoor location-based services (LBS), such as in-building assistance and navigation. For example, CINA [142] is a crowdsourcing system that creates an indoor navigation adviser using relative position. CINA utilizes mobile devices to provide path information and navigation for pedestrians inside buildings. The interesting aspect of CINA is that it does not need floor designs (e.g., [16] [31]) or building sensors (e.g., [171] [87]). CINA also uses gamification to raise user involvement. Similar to CINA is Airplace [89], another indoor localization system, which allows users to locate themselves and view a floor design map using the received signal strength (RSS) of Wi-Fi access points.

Further systems that are based on crowdsourcing for map construction but for outdoor environments are OpenStreetMap (OSM) [62] and Waze [152]. In OSM, users provide data using a map system or sensors on a mobile device. Waze utilizes traffic metadata to improve road navigation. In Waze, only mobile devices placed on vehicles are utilized for sensing.

Furthermore, a Crowdsourced Linked Open Data Architecture (CLODA) is presented in [90]. Using mobile devices, CLODA integrates crowdsourcing, localization and location-based facilities to provide useful information and public datasets. One of the benefits of CLODA is that it can be run indoors by installing it on recent mapping systems. Ear-Phone [136] and NoiseTube [162] are two other systems that work on building maps using the microphone sensor in mobile devices.

Another location-based service is presented in [27].Authors present three applications that improve location-based searches based on the data gathered: SmartTrace+, which supports matching paths; Crowdcast, which records user surroundings; and SmartP2P, which works on improving the energy and time spent on search tasks using proximity features. Authors implement and evaluate the applications using SmartLab. SmartLab is their programming cloud and consists of more than 40 Android smartphones and a number of emulated devices. Their work shows that energy consumption, privacy preservation, and application performance are new spots for future mobile applications. Another work is CityExplorer [103], which uses location-based games to collect location data to benefit other applications (other than games). CityExplorer works as a location-based service.

Finally, in [98], authors investigate the idea of using strangers as sensors through social media. They use MoboQ, a location-based real-time system in the form of questions and answers. By exploiting the idea of crowdsourcing via MoboQ, users can request location-oriented questions and obtain responses from different users in real time. The authors improve system accuracy by evaluating the answers received using a microblogging system called Sina Weibo to detect users who are currently at the location that is embedded in a question.

2.3.1.4 Crowdsourcing and Assigning Tasks

In terms of assigning tasks to users through crowdsourcing, authors of [85] introduce Turkomatic, a system that binds crowds to task requesters in order to help perform complicated queries. They first propose a set of simple tasks, select a set of interesting ones and, finally, replace a complex task with a set of simpler ones. This approach represents a tradeoff between delay and accuracy, since the system requires observation during the runtime to ensure the level of quality desired. Another crowdsourcing system is Soylent [20], which is a word processing system that supports authors by asking AMT [9] workers to reduce, proofread, and correct their documents. They offer the Find-Fix-Verify approach, which divides authors' requests into a set of phases to ensure quality progress.

In addition, authors of [80] propose new techniques for allocating the maximum number of tasks to users. Users forward their locations to a centralized server that allocates a task to an adjacent user. The experimental evaluations demonstrate the effectiveness of the proposed techniques. However, verifying the correctness and trustworthiness of users' outcomes is a challenge with spatial crowdsourcing. To overcome this issue, authors provide a reputation score for every user and a confidence score for each task. The reputation score reports how well a user completes a task. The confidence score decides if the outcome of a requested task is accepted, and this is achieved when the confidence score is greater than a predefined threshold [81]

Crowdsourcing is also used in creating news, by motivating users (i.e. volunteers) to provide data that are used to make news. In [154], authors offer location-based crowdsourcing, in which a news company refers a task based on users' mobile device location. However, some users believe that their location is personal information that should not be sent or shared and this is one of the main challenges in this work.

2.3.2 Crowd-sensing

Mobile crowd-sensing applications can be classified into three categories: environmental, situational and social. Environmental applications are those concerned with the natural environment and city features such as noise. Situational applications relate to public surroundings and transportation, such as road and building conditions, parking availability and traffic. Social applications are those that allow users to share sensed data about themselves, such as their exercise routine information [52].

2.3.2.1 Environmental

One environmental application is Common Sense [39], which measures air pollution (e.g., CO_2) using special sensing devices that send the data sensed to mobile phones via Bluetooth. Common Sense is inspired by “citizen science” [70] and “street science” [33], which enable citizen collaboration in collecting data and making decisions. Another application is CitiSense [199], a location-based system that measures air pollution using special body-worn sensors that collect data and send them to the user’s mobile device. The data are displayed to the user on the mobile device and sent to a back-end server for different purposes, such as inferring a detailed regional air-quality map. CitiSense captures more detailed information at a regional scale than any other air pollution monitoring system, as it exploits crowd-sensing to provide regional information.

Another system is AirSense [38], an air-quality monitoring system that gathers sensor data from around a city to observe air pollution. Users can gather data using mobile devices, a proposed air-quality monitoring device (AQMD) or both to collect data opportunistically and offload them to the cloud. After that, the data received in the cloud from mobile and AQMD devices are analysed to provide an air pollution map of the city. When evaluating the system, users can successfully ascertain the air quality in the neighbourhood and an air-quality index map (AQImap) of the city. Other work is presented in [47], in which authors develop a mobile application that enables users to benefit in real time from an air pollution (e.g., $PM_{2.5}$, which is particulate matter less than 2.5 μm in diameter) monitoring service and share their images. The authors utilize the cloud in their platform to gather historical data from different public places, while users send and share pictures at the same time. Their experiment shows high evaluation accuracy for $PM_{2.5}$ estimation.

A mobile crowd-sensing architecture based on publish/subscribe middleware is presented in [128]. The middleware makes sure data are aggregated and filtered in a real-time manner to mobile crowd-sensing applications. Authors demonstrate that their middleware is energy efficient by conducting an air-quality application experiment, as it avoids unnecessary data transmission and controls the sensing process. OptiMoS [189] is a mobile sensing system that tries to reach optimal mobile sensing, taking into account the cost of sensing and coverage. Another air-quality system is the Open Sense project [2], which monitors air pollution by deploying sensors on buses and maintaining sufficient sensing coverage. Finally, CreekWatch [34], an iPhone application developed by the IBM Research Center, empowers citizens to observe water quality in creeks and send reports (e.g., pictures and messages). These reports are sent to local water authorities to locate and manage pollution.

Authors of [195] propose a mobile crowd-sensing platform that creates large-scale noise maps in order to reduce noise around a city and uses microphones in mobile devices to do so. Their system received encouraging feedback when it was planned, established and tested in Southern Italy. Another work is NoiseSense [132], a real-time noise-mapping service. Since mobile devices have limited noise measurement sensors, authors propose a noise standardization algorithm that gathers noise levels from different locations (e.g., road networks and points of interest) with minimal user involvement. Their results confirm the effectiveness of the proposed algorithm in deducing noise levels with only a few measurements from mobile users.

2.3.2.2 Situational

CarTel [68] is a situational application that uses special sensor devices placed in cars to determine the speed and location of the vehicles. The sensed data are sent to a central server that will provide information about the routes with the least delay and other traffic queries. The Nericell [114] application uses mobile devices to sense roads, which will lower the cost and time of installing special sensors on vehicles to define speed, traffic delays, potholes, etc. The data sensed using Nericell are sent to a server for aggregation. A similar system to CarTel is proposed in [144], the difference being that the system in [144] uses a cyber-physical system to gather traffic, behavioural and environmental data, while the data in CarTel are gathered using GPS and On-Board Diagnostics (OBD).

In [29], authors provide design guidelines for a mobile crowd-sensing system that combines parking guidance with a navigation system called the COntribution-based Incentive Design (COIN) [30]. The authors' approach is different from others as it avoids manual entries in the road navigation system while driving, in order to ensure safety. This is unlike other applications that do not take the safety issue into account, such as Open Spot [83], which requires drivers to perform a manual task while driving in order to search for parking places. Another parking application is ParkNet [102], which uses small sensing devices connected to vehicles along with mobile devices to gather information on available parking places while driving.

A crowd-sensing system is introduced in [193], based on the argument that GPS is sometimes not sufficient in cities with large buildings. Therefore, authors deliver a mobile location search service that enables users to capture a short video clip or take photographs of nearby buildings and send them to the cloud. The cloud processes the data, extracts important features, and then sends them to a large database for the matching process. After that, the cloud returns an image with the location and the facilities attached to it [37].

With regard to road conditions and traffic, authors of [76] have developed a mobile crowd-sensing application called CRATER. CRATER estimates road conditions by using the accelerator to measure the acceleration readings in the road to detect the locations of speed bumps and potholes and then sends them to the cloud. In the cloud, the data are received and processed to provide a clear estimation of the road conditions for users. The processed information is issued on a map to benefit both citizens and municipal authorities in locating potholes, speed bumps and roads that need repair around the city. The evaluation in [76] shows that CRATER successfully detected a high percentage of potholes and speed bumps. Another work is presented in [53], in which authors gathered GPS data from users using a quality-aware sensing data collection system for road and traffic condition monitoring. They based their work on a scenario in which, when a user arrives, the system will use a previous quality method to estimate the amount of high quality data received by that participant. The system then chooses whether to select this participant from the estimated result, which is then merged with a requested reward. To estimate the amount of high-quality data sent by a participant, the authors use binomial-Poisson distribution (BPD) and a two-level iterative algorithm (expectation maximization [EM]). They compare their work with [127] and show, using the simulation results, that their research could provide higher-quality data using a real data set. Furthermore, [95] propose SafeRNet, which uses mobile cloud computing and crowd-sensing to collect and analyse traffic and historical data in order to deduce safe routes (using a Bayesian network) and send them back to users in real time. They present a case study with real traffic data that demonstrates the effectiveness of their approach in inferring safe routes and increasing the chance of safe driving.

Finally, authors of [73] present a Mobile Sensor Data EngiNe (MOSDEN), a mobile sensing system that is used by mobile devices to take sensed data and share them between several users and applications. A common and important feature of the proposed system in this thesis and MOSDEN is that the application-oriented processing is detached from collecting and sharing the sensed data. MOSDEN is considered a well-structured framework in terms of its simplicity and reusability due to the least development potential needed.

2.3.2.3 Social

A number of social applications have been created for crowd-sensing , one of which is “iSee” [120], a crowd-sensing system that senses and identifies events in outdoor public surroundings. Authors use their approach for two events: (1) identifying smoking in surrounding areas; and (2) identifying graffiti in cities. For example, if an “iSee” user observes someone smoking in a smoking-prohibited area, the user only needs to swipe on the mobile screen to detect the event. The data produced (i.e. GPS location, compass direction, user angle and time) are transferred to a central server. With the data collected from “iSee” users, more events can be discovered more accurately. Another localization system, but one which takes place indoors, is given in [143]. This system is based on Gaussian processes [137] for estimating the spatial field of a quantity and utilizing users’ information once they cross some area of interest. In the evaluation, the proposed method showed good results for estimating user positions and, when the number of measurements increased, the computational cost did not.

Another crowd-sensing system is presented in [91]. Authors create a public human sensor web using a sensor data request application. In their system, anyone can register and post a task and then receive sensor data for that task. Furthermore, every user has to provide an individual profile that includes information about the user’s data and location, so that tasks can be automatically sent to the most appropriate user. One benefit of this approach is the evolution of manageable geodata. One major challenge in a social network is the number of registered users, as the larger the number of users, the more valuable the network becomes.

Furthermore, mobile users can use clouds to share photographs and videos and tag their friends through social networks, such as Twitter. In [93], authors introduce Mobile/Multimedia Experience bLOGging (Melog), a mobile cloud system that utilizes automatic blogging to share real-time experiences (e.g., travel) by analysing the GPS data and photographs captured on the journey [37]. Similarly, DietSense [141] is an application that allows users to share pictures publicly of what they eat in order to compare different eating habits.

Middleware for social crowd-sensing applications, called mobile edge capture and analysis (MECA), is presented in [192]. MECA provides efficient sensed data gathering from mobile devices to be used by different applications. MECA's architecture has three layers: the first is the data layer (mobile phones); second is the edge layer, which selects devices to collect data; and the third is the application layer, which processes and responds to application requests.

CenceMe [109] is a crowd-sensing system under the MetroSense project [107] that develops different mobile-sensing projects. CenceMe allows users of social networks to share their sensing data with their friends securely. The CenceMe application on mobile phones observes user activity (e.g., running), mood (e.g., happy, sad), daily places (e.g., at the office) and contexts (e.g., cold), then sends them to social networks, such as Facebook.

Bikers can also benefit from using crowd-sensing to sense new routes. For example, Biketastic [140] is a system on which bikers can send information to other bikers and show them routes and directions. While using new roads, bikers can employ the Biketastic application on a mobile phone and capture the location, road conditions and noise level of the road using GPS, an accelerometer and a microphone, respectively. Furthermore, bikers can take images and video clips of points of interest during the ride. These data are sent to a back-end server to present these data on a map to ease information sharing between bikers. One major problem with Biketastic is the battery life of the phone, since a biker's phone continues to sense during the route and the data are processed in the phone before being transferred to the back-end server. However, authors can reduce battery usage by processing the data in the back-end server.

BikeNet [42] is another system for bikers and represents a sensing system that collects personal (e.g., heart rate), bicycle (e.g., speed), and context (e.g., noise level) information. BikeNet uses sensors installed on bikes (Moteiv Tmote Invent motes) and mobile phones. The system sends data using sensor access points (i.e. static or mobile) to a back-end server in real time. With BikeNet, bikers can send their sensor data at a different time during the day. After uploading data, BikeNet enables the biker community to share the data from the back-end server using a Web portal called "BikeView". BikeNet is a full sensing system for bikers and the results of the authors' experiments are encouraging. However, installing sensors on every bike and the need for sensor access points along the route make it a little harder and more costly and time consuming to use. This would not be the case if every biker only used his/her phone as a sensor and uploaded data whenever Wi-Fi is detected or 3G is used.

CrowdWatch is proposed in [175] and takes advantage of mobile crowd-sensing to identify momentary problems and create alerts for unfocused walkers. After studying walkers' activities, authors use Dempster-Shafer evidence theory to learn their behaviour and surroundings, and then calculate the likelihood of a problem existing. They also trigger alerts for unsafe areas by tracing walkers. With these experiments, the authors demonstrate the effectiveness of CrowdWatch in detecting surrounding problems.

Finally, in [179], authors propose PublicSense, a crowd-sensing system for public facility management in a smart city (i.e. information reporting, issuing intelligent classifications and intelligent tagging). Users can report issues using the camera in their mobile device alongside GPS to tag the location. The sensor data are sent to a back-end server, where the data management will take place using a proposed image classification technique that is based on a Fourier descriptor algorithm. Their experiments show that PublicSense can identify the type of issue using sensor data.

2.3.2.4 Offering Incentives and Offloading Data

Many crowd-sensing applications recruit users to contribute sensor data for different crowd-sensing tasks. There is a large number of studies on how to motivate users to contribute to different sensing tasks. One of the common methods used for successful crowd-sensing is offering incentives to users to increase involvement and the quality of the sensing task. In [198], authors propose an incentive mechanism that is based on a reverse auction and a Vickrey auction. Their mechanism overcomes an important problem in the existing incentive mechanisms, which is bad competition. They based their work on a scenario in which the payment is calculated before a given task and users are paid after they complete it, taking into account the quality of the sensed data. With the simulation results, the authors show the effectiveness of their incentive system by improving the bid mechanism and the quality of the data received from users.

Another incentive system is proposed in [185]. Authors propose PIE, a personalized incentive for location-aware mobile crowd-sensing that can be online or offline. PIE is not based on auctions or server-dominant incentives. Overall payment for all the users depends on their participation as a group, while every user is rewarded based on their individual input. The authors use a Voronoi diagram and Shannon entropy to calculate the contributions of every user and the participation level of all the users. Their experimental work demonstrates that PIE can successfully take participation level and different users' contributions into consideration. Although there are several incentive mechanisms for mobile crowd-sensing, none of them addresses the idea of users' cooperative compatibility in joint tasks. Therefore, in [187], an incentive mechanism is proposed that reduces the social cost of joint tasks. Authors define two bid types and a number of compatibility models, in which the Social Optimization Compatible User Selection (SOCUS) issue is framed for each bid type and users' connections are used for the compatibility models. They propose two reverse auction-based incentive mechanisms that contain two steps: compatible user grouping and a reverse auction. The authors demonstrate the efficiency of their work using analysis and simulation

Authors of [184] offer MagiCrowd, an incentive mechanism for mobile crowd-sensing that is based on negotiated bargain theory and motivates users to be part of the sensing tasks. In MagiCrowd, users are sorted into numerous groups to deal with the sensing tasks' initiators. Authors present K-least in order to group the users and accomplish K-anonymity location privacy. In the evaluation, the authors show that MagiCrowd can attain a large number of users involved in sensing tasks with the location privacy guarantee.

Finally, [75] presents an object tracking system, called CrowdTracker. CrowdTracker tracks and predicts movement by assigning users to take photographs of an object. The main goal of CrowdTracker is to accomplish a tracking job in real time at lower cost. The CrowdTracker incentive is measured by the number of participants in a given task and the distance needed to finish that task. Furthermore, authors propose a prediction model for object movement (MPRE) and two task distribution algorithms (T-centric and P-centric) that are used by CrowdTracker. T-centric picks the participants and P-centric assigns tasks to the participants. According to the experimental results, CrowdTracker demonstrates effectiveness in tracking objects at a low cost.

In mobile crowd-sensing, data offloading usually takes place either in real time or whenever Wi-Fi is available (after the task is over). Authors of [174] propose a data-offloading system called effSense, which addresses the challenges of energy consumption and data-transfer cost. effSense uses adaptive offloading patterns that allow users to select a suitable timing and network to offload data. The users in effSense can either be NDP (non-data-plan) or DP (dataplan) users. For NDP users, they suggest eliminating data cost while offloading by using the least costly connection (e.g., Bluetooth or Wi-Fi); DP users can select suitable events (e.g., a voice call) that need less energy while offloading data instead of using a 3G network. The experiment results show how effective it is to use effSense when offloading data in mobile crowd-sensing.

Offloading data to the cloud has a number of challenges, such as energy consumption and data transfer cost. Therefore, in [[28], authors propose a framework that takes into consideration the energy consumption when uploading data to the cloud using only a Wi-Fi network. They upload data to the cloud when Wi-Fi is available and is not being used by any other application, by using Wi-Fi Ready Conditions (WRCs). Their framework will choose optimal WRCs in order to minimize energy consumption. According to their evaluation results, 30% of the energy is saved when uploading data compared with a greedy-based technique.

Finally, in [94], authors present a Quality-of-Information (QoI) satisfaction ratio metric, a user sampling behaviour model and a QoI-aware energy-efficient user selection method. The metric is used to measure the sensor data needed for different tasks, whereby the behaviour model will measure the connection between the initial energy and the user contribution. The user selection method offers a solution to the optimization issue described. After conducting evaluations, the results validate the effectiveness and strength of the approach.

2.3.2.5 Crowd-sensing Architectures

This section presents a number of current-state crowd-sensing architectures. These architectures highlight the different building blocks of any crowd-sensing architecture and their functionalities.

In [61], authors propose a general crowd-sensing and computing architecture, depicted in Figure 2.1. Their architecture contains five layers: crowd-sensing, data transmission, data collection, crowd data processing, and applications. These layers are distributed between three different locations: sensing devices (e.g., mobile devices and vehicles), a back-end server (i.e. the cloud) and application interfaces.

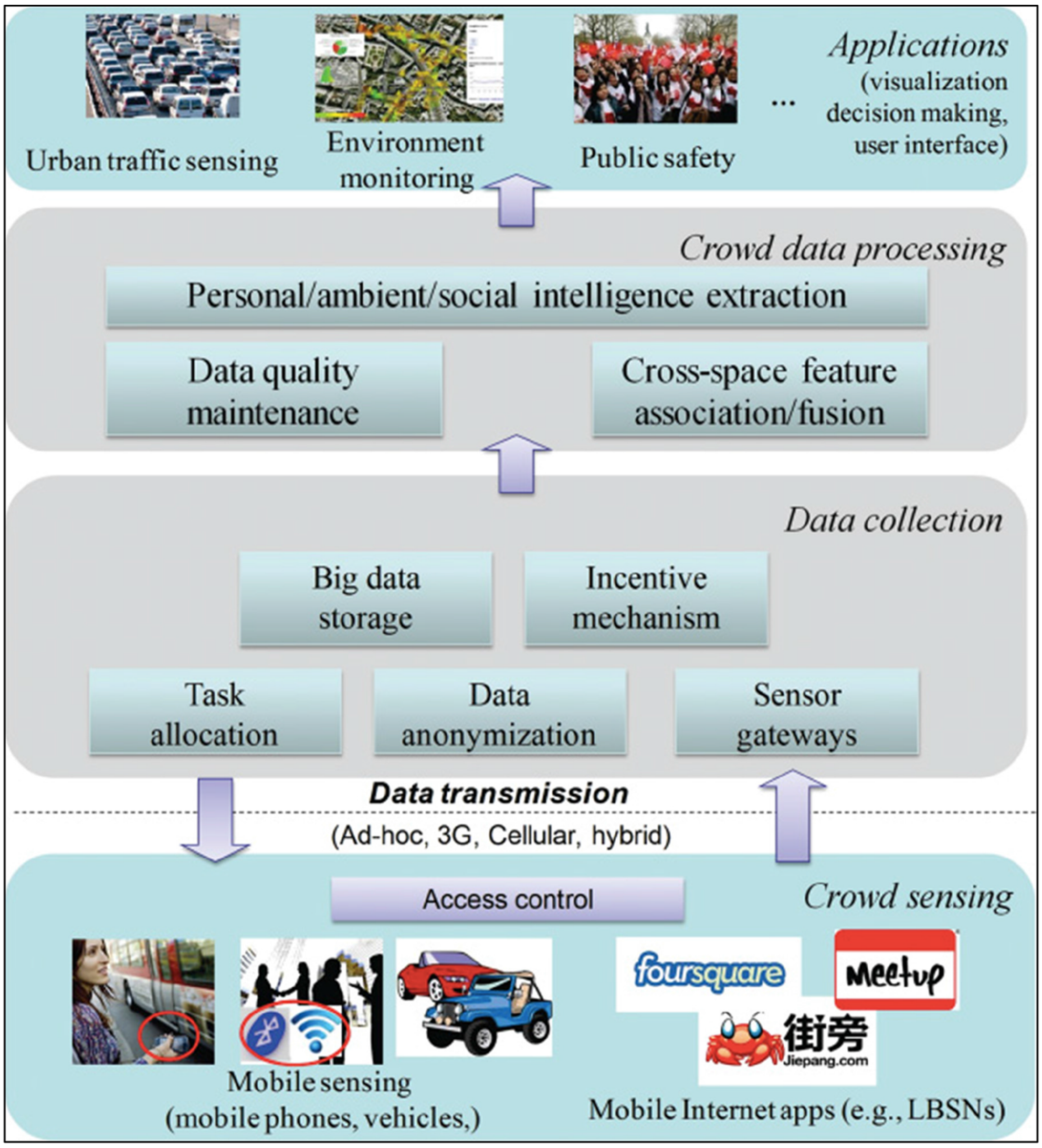


Figure 2.1: Generic crowd-sensing architecture [61]

The crowd-sensing layer in Figure 2.1 consists of data gathered from mobile devices, vehicles, etc. The data transmission layer is where sensed data are sent to a back-end server or the cloud. The data collection layer is located in the back-end server or cloud. This layer consists of different functionalities regarding the sensed data, such as data storage, filtration, anonymization to ensure user privacy, and incentivizing users to become involved in the sensing process. The crowd data processing layer contains extraction services, data quality measurement services, etc. Last is the application layer, in which the results from the previous layer are visualized and presented in application interfaces.

Another crowd-sensing architecture is DTRF [54] (dynamic-trust-based recruitment framework). However, DTRF is different from the previous generic architecture as it selects trustworthy users to perform a crowd-sensing task, as depicted in Figure 2.2. After the service requesters publish a task, the DTRF platform chooses an appropriate number of users, depending on their locations and trust values. The users who accept the task start the sensing process and send the sensed data to the DTRF platform, which is located in a cloud or a centralized server. The quality of the data is calculated in the DTRF platform and the sensed data are then sent to the service requesters. Finally, the service requesters send their feedback regarding the sensed data to the platform and, depending on the feedback, users receive rewards or penalties.

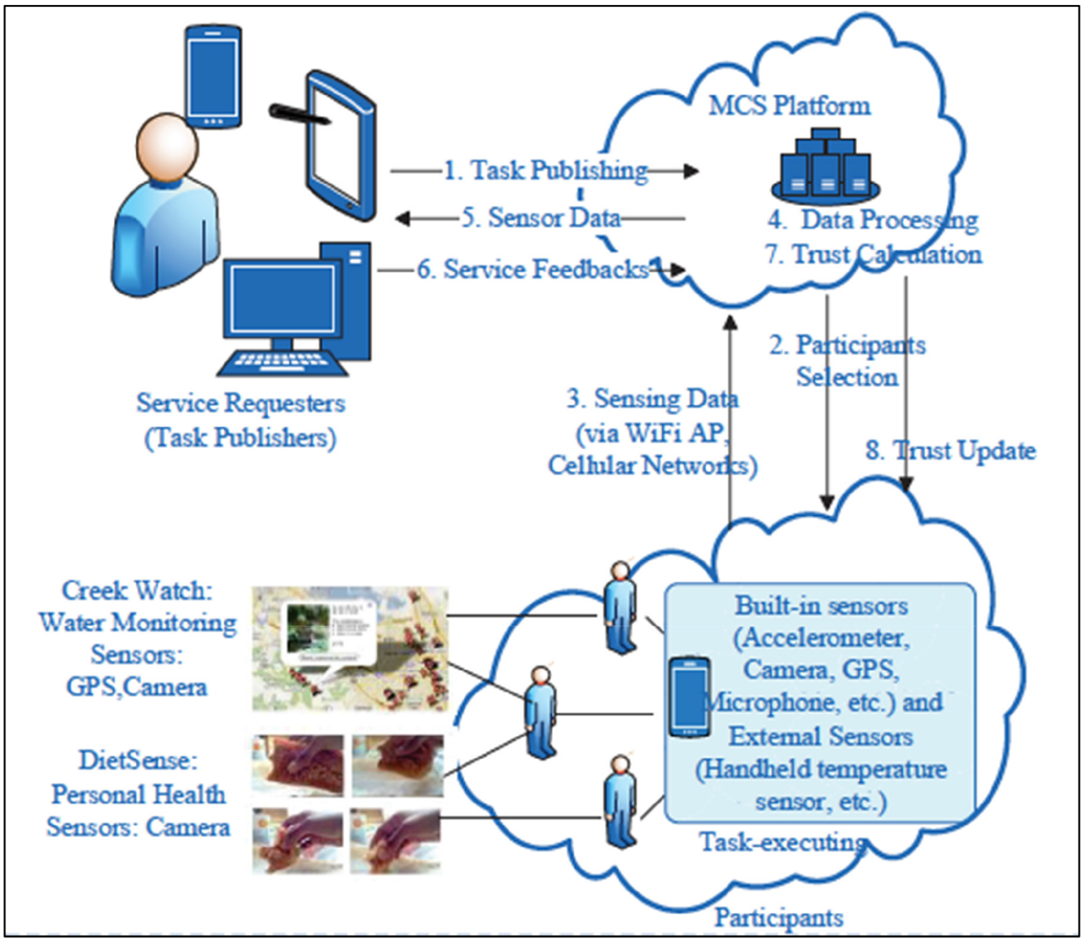

MCS Platform
1. Task Publishing
5. Sensor Data
6. Service Feedbacks
4. Data Processing
7. Trust Calculation
2. Participants Selection
3. Sensing Data (via WiFi AP, Cellular Networks)
8. Trust Update
Service Requesters (Task Publishers)
Creek Watch: Water Monitoring Sensors: GPS,Camera
DietSense: Personal Health Sensors: Camera
Built-in sensors (Accelerometer, Camera, GPS, Microphone, etc.) and External Sensors (Handheld temperature sensor, etc.)
Task-executing Participants

Figure 2.2: Dynamic-trust-based recruitment framework [54]

The majority of crowd-sensing systems and applications presented in the literature above (sections 2.3.2.1, 2.3.2.2, 2.3.2.3 and 2.3.2.4) follow the generic architecture shown in Figure 2.1, or part of it. In the following chapter, the architecture offered in this thesis will be presented and will introduce a new layer of processing located between the crowd and the cloud.

2.3.3 Context Awareness

There are several context-awareness systems that help individuals, such as blind people, to deal with their environment by using their mobile phones to sense their surroundings. A system to identify traffic lights for blind navigation is presented in [13] The system contains two main parts: the mobile, which can be any mobile device with sensors (e.g., a camera with GPS); and the cloud, which offers context awareness via a set of servers. Mobile devices are utilized to take videos and capture location using sensors and send them to the cloud. The cloud, on the other hand, manages the data received and sends the existence and status of traffic lights back to the mobile device. The system shows an acceptable response time and is promising in terms of gaining accurate results in real-time problems. However, delivering real-time responses about the status of traffic lights has some challenges. One of the main challenges is the time taken from sending the video to the cloud until getting a response. Utilizing data preprocessing services prior to the cloud could solve this. Another major challenge is the battery life of the mobile device, since it continues to record videos. To overcome this problem, videos could be taken using prior information of the location of traffic lights using GPS.

Another context-awareness service is proposed in [159], where authors have developed a service, called MobileMiner, that is located on a mobile device. MobileMiner service educes co-occurrence patterns using limited resources to highlight context cases that happen at the same time. MobileMiner occurs completely on the mobile device and thus guarantees privacy for users, which is considered a challenge with cloud computing.

In [149], a context-awareness system that provides the proper context to users is proposed. This system takes into account different users' situations when trying to obtain the right context. Authors use Case-Based Reasoning and Nearest Neighbour algorithms in modelling the system. Furthermore, a soundawareness application, called Sound Chat, is implemented and proposed in [100]. This application concentrates on the interaction between and convenience of visually impaired individuals and collaborative Web applications, such as Google Docs.

The results of Sound Chat indicate the necessity for such applications and that users were sufficiently accepting of the interaction.

In the area of healthcare, [125] proposes MECA, a system that utilizes context awareness to help health providers in Brazilian society. MECA supports health providers in a context combination task that provides agents with useful information. This information is systematized by different contexts, such as location and time.

2.3.3.1 Middleware for Provisioning Context-aware Mobile Cloud Services

A key feature of mobile devices today is their ability to sense users' contexts, such as location, acceleration and movement. A key technical challenge is to monitor user contexts and provide the right services for every context change. Consider a mobile phone that runs an application that subscribes to appropriate cloud services. Once the user moves, the context may change, so the application must be able to invoke another service to adapt to the changed context. This activity is called *context-aware service provisioning*.

Several works have been conducted on context-aware service provisioning in mobile cloud services. One is VOLARE [122], which is middleware that dynamically adjusts cloud services while observing the context of a mobile device connected to the cloud to ensure the quality of service (QoS). VOLARE observes the context of a mobile device, such as battery consumption, central processing unit (CPU) usage and network bandwidth, while receiving service discovery requests from applications. It compares the existing QoS with the thresholds for dealing with each service request. If the QoS level and the mobile device context change at any time, it will search for a service that fulfils the requirements and reconnect to it.

Similarly, in [86], authors propose an architecture for context-awareness service provisioning. This architecture prompts the mobile user to invoke a cloud service and select the most suitable based on the context information. The authors identify a model consisting of four layers of context elements: the first is *monitored context*, which refers to the device context (e.g., environmental and device settings) and user preference; second, the *type of gaps* that happen when content changes; third is the *type of causes* of the gaps; and last, the *adapters*, which refer to the actions taken to remove the causes of the gap.

Authors of [196] present a context-awareness middleware platform called Senz. They exploit context recognition algorithms and merge the data coming from users with data that already exist online. Senz successfully identified different patterns in experiments, the system obtaining around 83% accuracy in pattern recognition. Authors believe that using Senz with a cloud platform will produce interesting mobile applications in context recognition.

2.3.4 Discussion

From the works stated earlier relating to crowd-sensing and crowdsourcing, it can be seen that they have several points in common. First, they usually use a backend server, which can be in the cloud, for processing the sensed data. Second, the battery life in the context of mobile sensing is always considered a challenge. Therefore, this must be taken into consideration when using crowd-sensing applications.

One important concern shown in the context of crowd-sensing is that of incentivizing citizens to participate in the sensing process. The literature showed different works that used different ways to incentivize citizens. However, the citizens' motives for contributing in the sensing process as a consequence of being offered incentives are not the main concerns of this thesis; the main considerations of this thesis are data trustworthiness, scheduling, management and reduction that well be shown in details in the next chapters.

Therefore, with the large amount of data sent to the cloud (or servers) and the open nature of crowd-sensing applications, a number of challenges appear. The first challenge is the trustworthiness of the sensor data contributed. For example, some users might contribute incorrect data unintentionally or their mobile device might be in the wrong position. A second challenge is the origin of the data contributed, taking into consideration the privacy of the user. This is really important in identifying the source of errors in the data received. Another challenge is the size of the data sent to the cloud (or server). The fewer the data the better, in terms of network traffic and bandwidth. The last important challenge is the way big sensor data are managed, stored and processed in the cloud. Since the cloud offers a pay-as-you-go model, the data in the cloud need to be managed efficiently in order to be cost effective. This can be achieved by numerous services, such as removing redundant data or non-used data without losing important features or values.

The above challenges and gaps are taken into consideration in this research and, therefore, a number of methods and algorithms are proposed. These methods will help improve the way data are trusted, sensed and managed in the cloud.

2.3.5 Participatory and Opportunistic Contributions

A key characteristic of mobile crowdsourcing and mobile crowd-sensing is whether the crowd's contribution is participatory or opportunistic. Participatory contributions include users in sensing and generating data, while the input for opportunistic sensing is data produced automatically from the users' devices without user involvement. In general, participatory sensing usually relates to the crowd services on the Web because they need users' involvement. Opportunistic sensing, on the other hand, is transparent to users [27]. Tables 2.2 and 2.3 show the mobile crowdsourcing and mobile crowd-sensing applications listed above, whether they use participatory or opportunistic sensing (if any) and the types of sensors they use.

Table 2.2: Crowd sourcing applications

Application	**Sensing**	**Sensors**	**Application**	**Sensing**	**Sensors**
CrowdOut [15]	Participatory	Camera / GPS	Travelling through disasters [60]	Participatory	Camera / GPS
CrowdSearch [188]	Participatory	Camera	Airplace [89]	Opportunistic	GPS
CrowdSC [19]	Participatory	Camera / GPS	OSM [62]	Opportunistic	Camera/GPS
PotHole [44]	Opportunistic	Vibration / GPS	Waze [152]	Both	Camera / GPS
VTrack [168]	Opportunistic	GPS	CLODA [90]	Opportunistic	GPS
SignalGuru [84]	Opportunistic	Camera / GPS	CityExplorer [103]	Participatory	Camera / GPS
SmartTrace/ Crowdcast/ SmartP2P [27]	Opportunistic	GPS	Ear-Phone [136] / NoiseTube [162]	Opportunistic	Microphone / GPS

2.4 Smart City Applications

The concept of the *Smart City* has been introduced as the application of pervasive computing models in urban spaces focusing on developing city network infrastructures, optimizing traffic and transportation and improving the citizens' quality of life. Emerging technologies, such as mobile devices, wireless

Table 2.3: Crowd-sensing applications

Application	**Sensing**	**Sensors**	**Application**	**Sensing**	**Sensors**
iSee [120]	Participatory	Camera/GPS/ Compass	Biketastic [140]	Participatory	Microphone/ GPS / Accelero-meter
[193]	Participatory	Camera	GreekWatch [34]	Participatory	Camera/GPS
[93]	Participatory	Camera/GPS	CarTel [68]	Opportunistic	Special sensors
CommonSense [39]	Opportunistic	Special sensors	ParkNet [102]	Opportunistic	Special sensors/GPS
CitiSense [199]	Participatory	Special sensors	[29]	Opportunistic	GPS
PublicSense	Participatory	Camera/GPS	CenceMe [109]	Both	Accelero-meter/ camera/ microphone/ virtual software sensors
MOSDEN [73]	Opportunistic	Internal and external mobile sensors/virtual sensors	AirSense [38]	Opportunistic	Internal mobile sensors/ special sensor
Open Sense [2],	Opportunistic	Special sensors	DietSense [141]	Participatory	Camera
BikeNet [42]	Both	Microphone/ GPS/special sensors	Nericell [114]	Opportunistic	Accelero-meter/GPS/ microphone/ radio

networks, cloud computing and vehicular networking, promote the development of urban computing within a smart city by enabling a way to track and sense people's use of mobile devices [21]. Therefore, people's interaction is required in the Smart City concept. This interaction makes people an important and integral part of the system, which is sometimes called a living lab [69].

There are a number of smart city experiments and projects that have been run successfully. Some of these works cover smart city applications (which is the interest of this study), while others are plans for an entire city. One example of the latter is *Masdar City* [64] in Abu Dhabi, which has an area of 6 square kilometres and about 50,000 citizens. This city has an integrated public transport system to minimize the need for private vehicles. Another example is Zaragoza [64] in Spain, which has large sensor networks over approximately 90% of its routes to monitor the traffic in real time in order to improve the road network through effective policies [22].

Furthermore, there is a smart city campaign in Seoul, in Korea, called "Smart Seoul 2015" [157]. This campaign concerns delivering healthcare services for disabled and elderly citizens and providing them with smartphones and tablets to ensure they obtain the required and suitable medical care. Another is the "Smart City Wien" [156] project in Vienna. This project is aimed at achieving a smart vision for the city's energy in the long term (2050). This project contains three settings: "Smart Energy Vision 2050", "Roadmap for 2020 and beyond", which allows the city to start achieving its energy goals by 2020; and finally, the "Action Plan for 2012-2015" for executing the processes for the roadmap.

Amsterdam is one of the European cities that are making major efforts towards innovation. Amsterdam has introduced two effective programmes: StartupDelta [161] and StartupAmsterdam [160]. The first programme's goal is to combine the Dutch startup ecosystem with government and corporations in order to have one connected hub. The second programme is to provide Amsterdam's startup ecosystem. The city works hard to maintain innovation and be energy efficient by using electric rubbish collection lorries, solar panels, energy-efficient roofing, smart meters, etc. London also aimed to increase citizen satisfaction by launching the largest European free Wi-Fi network by the O2 operator [118].

"Smart Town" [158] is a smart town in Fujisawa City in Kanagawa in Japan, which was built with the help of Tokyo and large companies such as Panasonic and Tokyo Gas. This town has zero carbon emissions, is supplied with renewables and contains 1,000 households. Furthermore, Tokyo is considered a "green island" after planting one million trees in 2015 and providing smart parking and large Wi-Fi hotspots. On the other hand, San Francisco [155] is one of the cities that have considered smart parking the most. It also has a high number of charging stations for electric vehicles. San Francisco is helps visually impaired citizens on public transportation by offering different apps that help them navigate the city.

Finally, there are a number of European-funded projects in the context of the smart city. One of these projects is CITYOPT [32], which was established in Helsinki in Finland, Vienna in Austria and Nice Cote d'Azur in France. The main goal of CITYOPT is to have more energy-efficient environments. The design of CITYOPT is user-centred and users are engaged in the application and connection of all the different stakeholders in the decision making of these energy-efficient neighbourhoods. Another European-funded project is e-SAVE [41], for an energy-efficient supply chain. e-SAVE's goal is to assist different companies in providing and monitoring energy use to help in the supply chain design decision process.

2.4.1 Special-purpose Systems

A number of systems have been designed to serve specific purposes, such as monitoring environmental changes in cities, locating objects of interest, sensing by vehicles and more. First, in [101], a framework that empowers the gathering and distribution of city data is introduced, called CityWatch (CW). Two different methods of sensing are utilized by CityWatch to collect data: fixed sensing and crowd-sensing (i.e. participatory or opportunistic sensing). CityWatch allows citizens to sense the environment in a city by using crowd-sensing. Furthermore, authors use an interesting approach, gamification, to encourage citizens to sense data. The main components in the CityWatch framework are the CW middleware, the CW application server and the mobile application. The CW middleware provides an interface for data gathering and distribution. The CW application server runs the CW application located on the Web and mobile phones, the Web side is used only to observe data, and the mobile device side is used to both observe and sense data. A similar but smaller approach is presented in [165]. Authors present a general architecture for mobile crowd-sensing that is developed on top of the Extensible Messaging and Presence Protocol (XMPP) [146] and uses a publish-subscribe communication model [108] to feed smart city applications.

Another system is introduced in [22], whereby authors present a location-based smart city service designed to help citizens collect information relating to interesting nearby objects. The clients are users who have a client application on mobile devices. This application is called “around me” and is a case study to consider the challenges in developing such an application and delivering knowledge to users regarding their surroundings. This application helps citizens to locate useful objects ordered by distance around their current location and distances are updated with the user’s movements. The data queries are performed using a complex On-Line Transaction Processing (OLTP) server platform. Their work has focused on many issues in the improvement of smart city applications and has been adopted by the Italian city of Cesena to make it a smart city.

In the context of vehicular computing, MobEyes [55] is middleware that supports urban watching applications using vehicles. A number of sensors are installed on vehicles, such as a wireless connection, video cameras and other sensors, to sense surroundings and events, then to process and categorize the data produced. MobEyes stores sense data in mobile node storage instead of uploading the sensed data directly to the Internet. The sensed data are managed by on-board processors to extract important features, e.g., car number plates. This knowledge is then sent regularly to the cloud and can then be found on the Internet for different sensed events. A different vehicle-sensing application is presented in [178], where authors exploit a mobile device for sensing (using accelerometers and gyroscopes) the movements of a vehicle to define the driver's mobile device location (e.g. is it going left or right?). They believe their work could ease the development of safety applications. They refer to this as a low infrastructure approach, as it does not use a built-in Bluetooth system. Their experiments also show that their approach is highly accurate.

Furthermore, a system called MineFleet [79] was designed for a commercial fleet to measure and analyse different vehicles' features in order to reduce operational costs. In MineFleet, the vehicle data are gathered and processed using custom devices that run MineFleet software, whereby these devices are located on fleet lorries. The software observes the current status of the vehicle, such as driver behaviour. The stored data are transferred to an external server for more processing when a network connection is detected.

2.4.1.1 Traffic in Smart Cities

Traffic is considered one of the major issues in city management and is a widely explored topic. Authors of [105] propose the CityWatcher application, which is used by citizens to send their video recordings about road accidents or any other city problems, e.g., potholes. CityWatcher contains three key components: Internet-connected objects (ICOs), middleware, and user software. ICOs utilize an application developed for recording videos, which are tagged with a time stamp and location. These ICOs can be mobile devices, vehicle-mounted surveillance cameras (VMSCs), which have become widely used in recent years [25], special cameras, etc. The middleware is created using an open source middleware system for intelligent Internet of Things (IoT) applications, called OpenIoT [119]. Finally, the user software is simply a Web application for users to send requests for videos and watch them. In other work in [182], authors present how in-vehicle sensors that are used to determine accidents [26] can be replaced by mobile devices that are equipped with sensors, such as accelerometers, cameras and microphones. They have developed a model that combines mobile sensors and the surroundings in order to decrease the time between an accident occurring and the emergency response.

2.4.2 General-purpose Systems

There are a number of frameworks that are not designed for a specific application. One is [8], which presents a smart city architecture that exploits different sensing devices (e.g., mobile devices and sensors) to serve a city. The framework permits users to collect surrounding data, and then the data are managed and analysed in the cloud to obtain information about the surroundings. With this information, a number of functions can be implemented to improve urban management, such as healthcare and smart traffic. As stated above, this cloud-based framework is not designed for a specific application; instead, it provides cloud services for urban management. It allows handy extensibility of different components, since it contains a group of components.

Another system is introduced in [123], where authors use cloud computing techniques to process a large amount of data for a ubiquitous city (U-City) system in order for citizens to be able to use different services at different times and places. They propose a system called “SOUL”, which supports Web services in mobile devices with Android operating systems. SOUL interacts with virtual machines automatically without the need to have knowledge of cloud infrastructure. The authors believe that SOUL is the first mobile cloud portal for a UCity. However, they also believe they need to improve their work to include more operating systems.

A new flexible sensor cloud architecture that is used for smart cities is provided in [131]. This architecture would enable users to provide different forms of data from different sensing infrastructures and is based on the Sensor Web Enablement (SWE) standard defined in [150]. An integrated sensing architecture for different sensing applications is proposed in [45]. The architecture is located in the cloud, which makes it robust and reliable. Authors consider their architecture as a basis for a Mobile Application as a Service (MAaaS), as they provide a foundation for developing different applications with different sensing models. With their emergency response system case study, the authors reduce the energy consumption of the mobile devices used in sensing. A similar system in [63] uses wireless sensor networks instead of mobile devices to connect to the cloud to serve different city applications. In [110], authors propose sensing architecture for mobile devices. Their architecture is intended to process metadata instead of sensor data. The authors argue that keeping metadata is more useful than maintaining sensor data, as the latter are not well-structured and are not easy to process [74]. Their evaluation results show high effectiveness with processing metadata; however, the privacy of the mobile devices is still a problem.

Finally, in [82],authors propose a cloud-based service that helps in managing smart cities. To manage the data generated from smart cities, they use information and communications technology (ICT) tools and a software service, since ICT has a major effect on smart cities' management in terms of better communications services and helps citizens to exploit their surroundings by delivering essential information. Cloud computing has the potential to manage, store, and process city-oriented data; however, different tools are required to process these data efficiently.

2.4.3 Discussion

From the work referred to above, it can be stated that data availability and user involvement are important keys to the success of smart city architectures. Thus, involving users by crowd-sensing can play a major role in providing sensor data in the context of a city for achieving improvements in their cities (education, transportation, etc.). The modelling and organizing of data once received in the cloud (or servers) are essential, as they can have a significant effect on the way the data are analysed and knowledge extracted.

For the success of a smart city, important features need to be tackled, such as user privacy and trust. User identity and information should come with a high level of privacy and security and the data contributed for the sake of the city should be trusted and truthful. Furthermore, with the major usage of mobile devices, low-cost communication is required and this can easily be achieved with the use of wireless communication. The challenges of crowd-sensing in the context of a smart city are illustrated in Table 2.4.

The architectures in Figures 2.1 and 2.2, show that sensed data are all offloaded to a cloud or a back-end server in order to perform several services on these data. However, this will consume resources (e.g. bandwidth and cloud storage) and, therefore, increase the necessity to adopt some kind of preprocessing before data transmission to the cloud. This processing must consist of important services such as data filtration, redundancy removal, data reduction, etc.

Therefore, an architecture that can overcome all (or some) of these challenges and the challenges in table 2.4 is required. This architecture needs to introduce a new layer of processing located between the crowd devices and the cloud. The processing of data in the middle layer before the cloud has several advantages such as, data filtration. The architecture and the benefits behind it will be discussed in detail in the following chapters.

Table 2.4: Smart city application challenges

Challenge	**Description**	**Requirement**
Bandwidth consumption	Sending large amounts of data will consume bandwidth and thus increase cost and cause major network traffic.	Efficient data reduction services.
Mobile power consumption	Exploiting the built-in sensors of mobile devices for crowd-sensing applications will reduce battery life.	Improvements and new designs in mobile devices to increase the battery life.
Data trustworthiness	With the explosion of data, smart city applications are exposed to untrustworthy data that might affect the city in terms of decisions.	Trust evaluation criteria that calculate how truthful data are.
User privacy	When using different crowd-sensing applications, users will be worried about the exposure of their identities.	Either provide strong encryption and cryptographic tools or ensure anonymity for user identities.
Data management	With the notion of big data that are transferred and stored in the cloud, these data need to be managed efficiently to increase storage effectiveness and reduce storage cost.	Effective data management techniques need to be adopted in the cloud platform.
Low-cost communication	Different connection protocols can be used in crowd-sensing applications, such as Wi-Fi, 3G and Long Term Evolution (LTE).	Users can either use a Wi-Fi connection to avoid any costs or use the other protocols intelligently to manage the costs as much as possible.

2.5 Chapter Summary

Previous work offered in mobile cloud sensing shows a promising area that can be used by local authorities and by public and private organizations. This chapter presented different methods of mobile cloud sensing for different applications that increase the quality of life for citizens and provide comprehensive understanding of how crowd-sensing is a good method for making cities smarter. The difference between crowd sourcing and crowd-sensing was presented, whereby crowd sourcing requires human intelligence, while crowd-sensing does not. Crowd sourcing applications are organized into either standalone applications or extensions of Web-based applications. On the other hand, crowd-sensing applications are classified into environmental, situational and social. Smart city applications that can be either special purpose or more general were also presented.

A number of challenges in the area of crowd-sensing and smart city applications were stated, as these increase the importance of providing architecture that can tackle some if not all of these challenges.

Through this thesis an architecture that includes methods and algorithms to overcome some of the challenges is given. In the next chapter, the design of the system components and the architecture that include different middleware services are presented.

Chapter 3

Design and Architecture

3.1 Introduction

The works in the literature review in the previous chapter demonstrate that the amount of data drawn from crowd-sensing and smart city applications is large and difficult to manage. Since mobile devices have limited resources, a number of studies have considered offloading data to be stored and processed in the cloud. However, offloading data to the cloud introduces a number of challenges, such as the trustworthiness of the data, which is an important element that plays a large role in the success and improvement of a city. Furthermore, the size of data that are transferred to the cloud and stored must be taken into consideration, as the challenges of mobile energy and bandwidth consumption then arise. Therefore, a mobile cloud architecture that can articulate the challenges in the literature is needed. This chapter first presents the challenges that this system is designed to address. A set of requirements is then introduced that shapes the components and properties of the proposed system, which is intended to make data management and the transfer of crowd-sensing data more efficient. The design and architecture of the proposed system are also shown.

3.1.1 What are Crowd-sensed Data?

Before introducing the design of the system, crowd-sensed data need to be clearly defined. Mobile devices have a variety of sensors that produce heterogeneous data types. For example, data could be latitude/longitude values, time, photographs, voice notes, accelerometer readings and/or GPS coordinates, the last two being floating-point numbers. Therefore, in this thesis, crowd-sensed data are all the different types of data produced from the various kind of sensors of mobile devices (see Table 2.1).

In this thesis, an Android mobile sensing app, called "SenseAll", is developed (see Appendix A). In this mobile app, the user will choose from different aspects of the city, such as weather, traffic and road conditions, and then start to sense depending on the data required for his/her selection. After that, data are sent as a packet that has the following structure: application user ID, annotation about the type of data (data description) in the packet, and after that will come the data produced by the mobile sensors. For example:

1. If the user chooses to send simple traffic data, the packet will contain the following:

   (User Application ID-Traffic-Latitude-Longitude-Time-Accelerometer reading).

2. If the user chooses to send road condition data, the packet will contain the following:

   (User Application ID-Road-Latitude-Longitude-Time-Accelerometer reading-Photo- voice note).

3.2 Challenges

In this section, and after reviewing the literature chapter 2, the challenges that can face crowd-sensed data and the ones that this research took into consideration are:

- Data trustworthiness

  The success of smart city applications depends on user involvement. However, with the openness of crowd-sensing and high user contributions, smart city applications can be exposed to untrustworthy and malicious data. For example, someone could contribute data that serve his/her own interests or might unintentionally pass on incorrect data because the mobile device has become decalibrated.

- Bandwidth consumption

With crowd-sensing and utilizing cloud computing for processing and analysing, a large amount of data is sent to the cloud, which requires high bandwidth utilization, energy consumption and network traffic.

- Data usability

  Useful crowd-sensed data are those that are vital and not redundant. Therefore, there must be some kind of usability evaluation of crowdsensed data, since sending these data without filtering and prioritizing them will introduce high bandwidth consumption. Crowd-sensed data that are useful will have high priority when data are sent to the cloud.

- Data management in the cloud

  With the limited resources of mobile devices, crowd-sensed data are usually offloaded to the cloud. However, in smart city applications or scientific applications, data volumes will increase at a very high speed and this will introduce the challenge of storing these data as the costs are also increased.

- Losing important data features

  The previous challenge introduces a significant consideration, which is the loss of important data. Therefore, the managing process must be efficient enough to avoid the loss of critical features and values.

3.3 Requirements

Having defined the research challenges in the previous section, the requirements can now be identified into two sets of requirements: first, the general requirements that should be part of any smart city architecture and second, the specific requirements that can be identified for designing the system proposed in this thesis. These requirements can improve the way in which crowd-sensed data are trusted, scheduled, sent and organized in the cloud.

3.3.1 General Requirements

Cities usually have different characteristics due to their specific financial and environmental constraints. However, there are essential requirements that need to be met when designing and implementing the architecture for any smart city application. These key requirements are [151]:

1. **Sensing objects**: Including sensor objects that are either fixed or mobile is one of the important requirements of smart city architecture. These objects are important when sensing and collecting data in a city.
2. **Real-time sensing**: Continuous sensing and monitoring is another important requirement for collecting data in order to use them to predict useful information.
3. **Sustainable policies**: Every architecture must have clear and sustainable policies regarding each domain (economic, environmental, etc.). For example, the architecture needs to guarantee a minimal negative impact on the city in terms of community, safety, environment, economy, etc.
4. **Storing data**: Architectures must include storage for the data collected in order to use them in analysing and mining.
5. **Availability**: The availability of a centralizing middleware (cloud infrastructure) is highly important, since the middleware must continue to obtain, store and analyse data.
6. **Privacy**: Privacy is a highly important and sensitive requirement. Privacy policies must be clear on exactly the type of data that are going to be captured and what will be done with them. This requirement is a challenge for every smart city architecture.
7. **User involvement**: Users must be part of the smart city architecture and the deployment process of the system, since the purpose of smart city applications is to increase the quality of life of the citizens.
8. **Extensibility**: The architecture should be flexible when adding new services and new types of sensors.

3.3.2 Specific Requirements

The question now is how to achieve an architecture that can manage and reduce the large amount of crowd-sensed data contributed by different users. To achieve this goal, a set of core requirements that can optimize the data transfer and storage of crowd-sensed data in smart city architecture is listed below.

1. *The crowed-sensed data must be filtered before they are sent to the cloud and this filtration needs be located in the right spot.*

With the increasing volume of crowd-sensed data that is sent to the cloud, the necessity to filter these data is increased. This filtering process must be located in the right place in order to reduce the amount of data as much as possible i.e. by locating the filtering process in the proximity of the users (edge servers/public local servers).

2. *Users and their data must be trusted.*

   Crowd-sensing and smart city applications include high levels of user contributions and these applications could be exposed to untrustworthy and malicious data. However, the success of these applications depends on the originality and trustworthiness of the crowd-sensed data. Therefore, ensuring data trustworthiness is essential in satisfying successful crowd-sensing and smart city applications.

3. *The origin of the crowd-sensed data needs to be traced.*

   In the cloud, large amounts of data are stored and used by different smart city applications. If data are wrong and there is no information about their origin, not only will applications return incorrect results, but there is also no way of identifying the source(s) of the errors. Therefore, data traceability is another important requirement when processing crowd-sensed data.

4. *The privacy of the user information must be ensured.*

   Tracing the origin of crowd-sensed data is essential (see requirement number 3). However, users' identities and location could be exposed. Therefore, ensuring anonymity is another important requirement in such a system.

5. *The approach has to minimize the bandwidth and network traffic.*

   Crowd-sensing applications upload a large amount of data from mobile phones and other sensors to the cloud for processing. However, sending a large amount of data to the cloud requires a high bandwidth and network traffic and consumes a lot of energy. Therefore, the proposed system needs to overcome these challenges.

6. *The data should be managed and reduced efficiently in the cloud.*

   As increasing amounts of data are sent to the cloud, the size of the data that need to be managed also increases. Sometimes, the free space in the cloud is limited; for example, Google Drive provides 5 GB. The crowd-sensed data in the cloud then need to be managed and reduced periodically, as all cloud providers offer users pay-as-you-go storage.

3.3.3 Non-functional Requirement

- **Availability:** The system should function and be accessible at any time.
- **Scalability:** The system should be able to accommodate a large amount of data at any time.
- **Privacy:** The system should protect users' personal information from being exposed anywhere.
- **Security:** With the openness of crowd-sensing applications, the system should provide some level of security in order to prevent attackers from accessing users' mobile devices and intentionally generating false data.
- **Extensibility:** The system should be able to add new services and features without affecting or changing the current functionality.
- **Cost-effectiveness:** The system should efficiently minimize the costs related to data transfer and data storage.
- **Data quality:** The system should ensure the quality of the data received in order to be beneficial in the context of a smart city.

3.3.4 Performance Requirements

- **Bandwidth utilization:** With the large amount of crowd-sensed data generated in the smart city context, the system should reduce the amount of data transferred in order to decrease bandwidth utilization and network traffic.
- **Storage saving:** Although the cloud offers data storage, there is a price for this. Therefore, the system should manage the crowed-sensed data received by the cloud efficiently in order to reduce the storage cost.
- **Compression ratio:** Since the system offers a compression service before sending the crowd-sensed data to the cloud, this service should achieve a reasonable compression ratio to decrease the size of the data transferred.

- **Robustness:** The system should successfully cope with errors. In this research, there are two kinds of errors that might occur: the first is on the user side and the other is on the server/cloud side. The system should have the right actions for the following questions:
  - What if the sensors of the mobile device become decalibrated and start to send false data?
  - What if the server were to go down?
- **Security and privacy:** The system should provide a reasonable level of protection and privacy for user accounts and prevent any personal information exposure.

The following section presents a mobile cloud architecture design that can meet the above requirements.

3.4 Design

This section discusses the system architecture, design overview, proposed services and the users who can interact with the system. The architecture proposed in this thesis aims to:

- Check the trustworthiness of the crowd-sensed data before they are sent to the cloud.
- Manage the traffic in transferring crowd-sensed data to the cloud.
- Reduce the size of crowd-sensed data before they are sent to the cloud.
- Locate the services above as close to the crowd as possible.
- Reduce the amount of storage allocated to crowd-sensed data in the cloud.

The system is divided into four main components, as depicted in Figure 3.1 and described in more details below.

1. **Users**

   Users can be a subset of citizens, from two to what we call a crowd, located in a particular area of a city, who use an Android sensing application on their mobile devices and deploy data through a Wi-Fi connection. Users will need to provide user credentials (i.e. user name and password) in order to be able to send data to the cloud using their mobile devices.

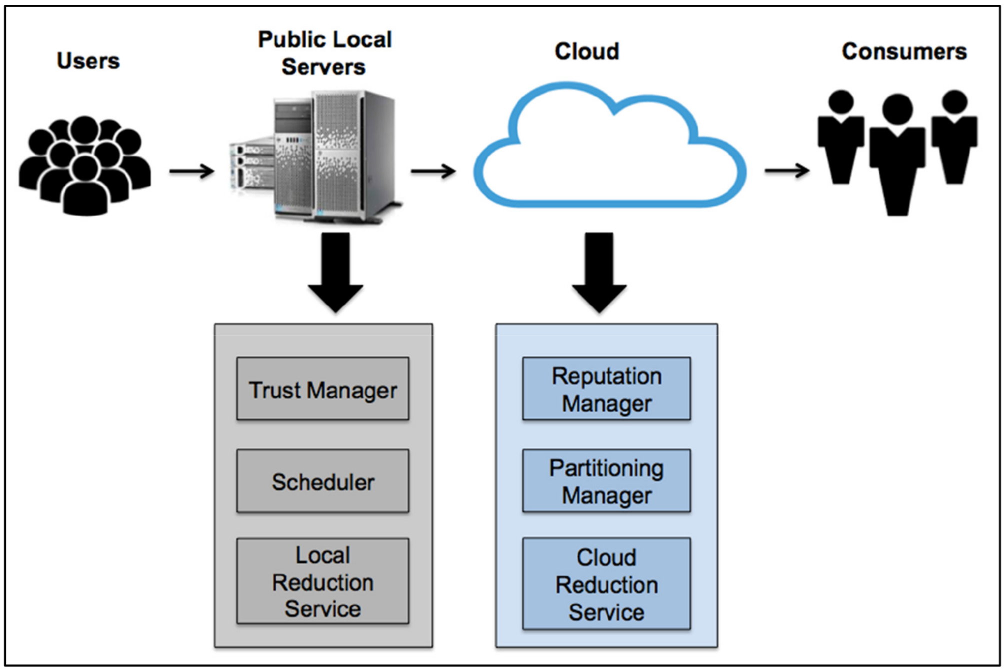


Figure 3.1: Architecture

2. **Edge Servers**

   This component consists of public local servers that are distributed around the city for the purpose of collecting crowd-sensed data, storing them for a period of time and processing them locally before they are sent to the cloud. These public servers also run other applications, such as traffic or environmental monitoring. These local servers contain three subcomponents, as shown below in Figure 3.2. First, every user's contribution (user ID and data) will have a data receiver instance that is managing a buffer. The buffers are organized in FIFO (First In First Out) form. Then, a general buffer will take users' contributions from the individual buffers, one at a time, and insert them to the local database for a limited period of time, e.g., one day. The period of time is variable and might be short or long depending on the application used and the amount of users' contributions received. The server will also record the meta-data, such as user ID, sensor data types, etc., in a log file.

   - *Trust Manager*

     The trust manager will calculate the level of data trustworthiness. The trust manager performs the trust calculation using four factors: user status, data variety, loyalty and similarity. When considering a specific user contribution, if the trust calculation is above a defined

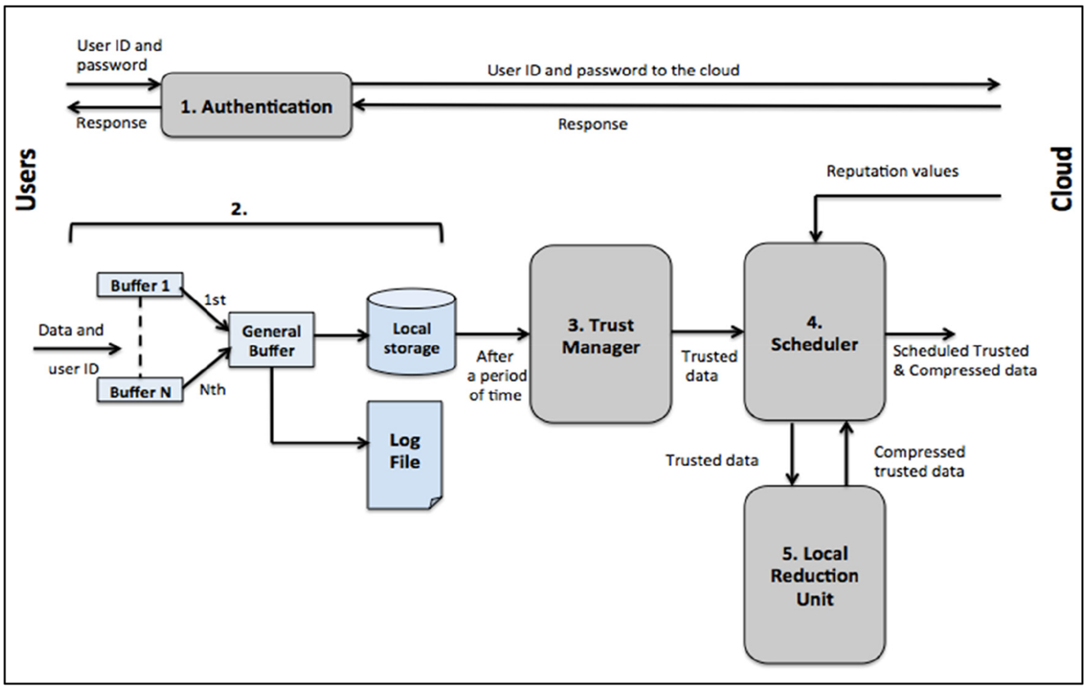


Figure 3.2: Edge server components and services

threshold, the data are trusted and sent to the cloud; otherwise, the data are discarded. The details of this component are contained in chapter 4 and parts of this work were published in this conference paper [7].

- *Scheduler*

  After calculating the level of trust, the scheduler component receives the trusted data and calculates their priority depending on a number of factors that vary depending on the application in use. After a period of time e.g., one day, trusted crowd-sensed data of higher priority are sent first and all the metadata for all the contributions are removed from the log file. The details of this component are presented in chapter 4 and parts of this work were published in this conference paper [6].

- *Local Reduction Unit*

  The reduction component focuses on single-precision floating-point data and takes advantage of the distribution of local servers in smart cities. There are two compression techniques in this component: one for location-based data (latitude and longitude) and the other considers three-dimensional accelerometer data. The details of this component are presented in chapter 4 and parts of this work were published in this conference paper [5].

A task flow diagram that shows the order of the services in the public local servers is shown in Figure 3.3.

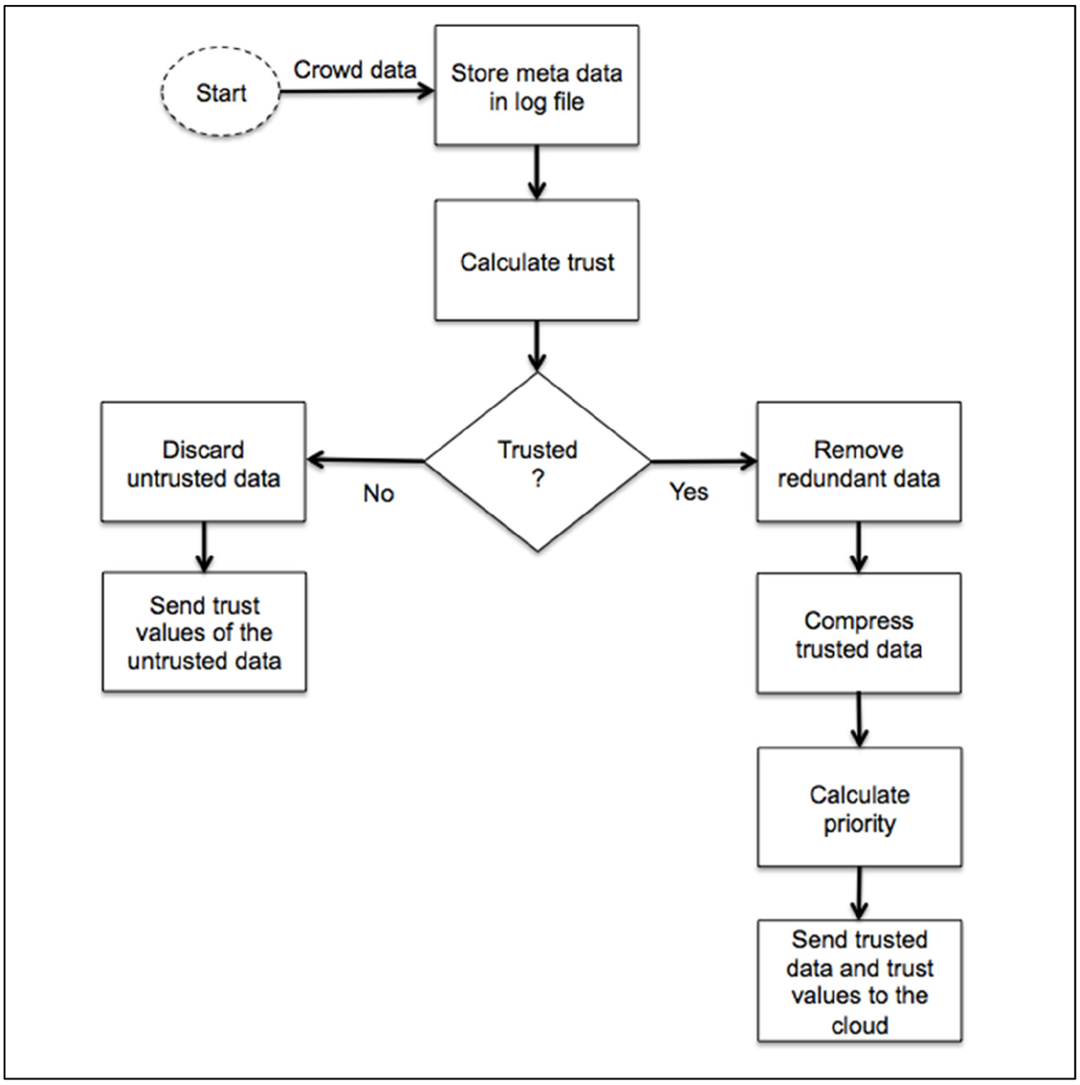


Figure 3.3: Task flow diagram for the services on the edge servers

Edge servers distribution mechanism

In this thesis, only one server is taken into account in the design and evaluation phases. However, in this section, a number of considerations are listed regarding the distribution of the servers around the city and the communication between them, if needed. These considerations are:

- The servers are distributed evenly around the city and the distribution is based upon GPS coordinates. GPS coordinates for any two nearest servers need to be different in decimal numbers in order to benefit most from the services located on these servers. However, areas with a high and active population can have two servers to avoid server overload.
- The cloud performs the server selection task. This event is triggered whenever users attempt to send their sensed data. The cloud locates the nearest local server and sends its location to the user mobile application. If two servers are considered appropriate for one user, the user can choose one of them.
- Every server must have the physical address of the two nearest servers. Then, when one of the servers is down, the other will be in charge of the down server's area.
- If one user senses data and then tries to send these data at a different time and location (whenever Wi-Fi access is available), the cloud then chooses the server nearest to the user's current location. However, the data will not receive the expected benefit from the services in the server since the GPS coordinates for the sensed data are different from those in the server. The importance of having the same GPS coordinates for both the sensed data and the server will be highlighted in chapter 4.

3. **Cloud**

   The cloud component in this architecture contains databases in which users' information, history and trusted crowd-sensed data are stored. With the large amount of crowd-sensed data stored in the cloud, there is a necessity to manage these data efficiently, since the data storage in the cloud has a pay-as-you-go model. Therefore, the cloud component contains three sub-components, outlined below

   - *Reputation Manager*

     Reputation management takes place in the cloud and updates users' reputation values regularly. Therefore, trust is a value calculated in the local server over a period of time. On the other hand, reputation is a value calculated in the cloud, whereby the cloud takes a new trust value received from the local server for a specific user and adds it to the previous reputation value to update it for that particular user. The details of this component are presented in chapter 5.

- *Partitioning Manager*

  The partitioning manager first partitions data depending on variable parameters that can be defined by the user, the application or both, such as time and access rate. These parameters logically partition the database in order to the consumers (e.g. cloud administrators, city council) to have a clear vision of what the limit of reduction might be when applying reduction services to the data, since every smart city database (as well as the databases of other domains) contains important and often sensitive entries at specific times. The details of this component are presented in chapter 5 and parts of this work were published in this conference paper [3].

- *Cloud Reduction Service*

  This service contains two different data reduction techniques: data optimization and context extraction. These services are applied depending on the sensitivity of the data, whereby sensitive data are those that contain important values, for example, within the weather database, data entries corresponding to days with flooding in a particular city are considered sensitive data. This service is strongly related to the partition manager output, whereby every technique will make use of the partitioning method to reduce data stored in the cloud efficiently and, therefore, decrease the cost of storing these data. Details of this component and the partition manager are presented in chapter 5.

4. **Consumers**

   Consumers are citizens, cloud administrators, and private or public authorities, who have access to the crowd-sensed data stored in the cloud. There could be one, two or a large number of consumers. Consumers need to register in the cloud in order to benefit from the data stored there.

3.5 System Architecture Layers

Following the system architecture presented in the previous section in detail, the five different layers contained in the system are now presented: Application, Management, Service, Data and Servers. These layers are shown in Figure 3.4.

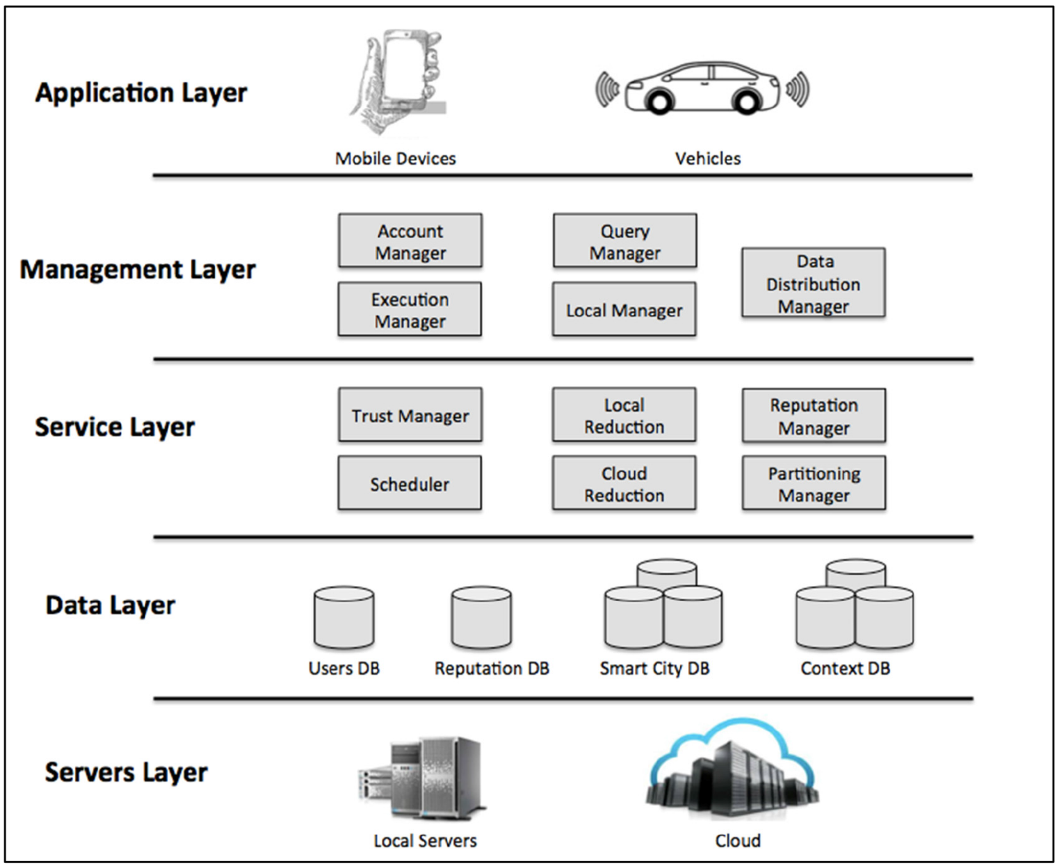


Figure 3.4: System layers

The **Application Layer** contains users and their mobile devices or vehicles that are used to sense different aspects of a city, such as its environment. Users then use Wi-Fi connections to send the crowd-sensed data. The Wi-Fi connections are distributed around the city to improve the quality of life of the citizens. The **Management Layer** contains different components that are located in either the cloud or the edge servers. These components manage the way data are managed, sent, received and stored, as follows.

- *Account Manager*

  The Account Manager, which is responsible for storing users' application IDs and passwords (credentials), is located in the cloud. In this thesis, the application IDs that are assigned by the mobile apps are used instead of real names in order to ensure anonymity.

- *Data Distribution Manager*

  The Data Distribution Manager receives the crowd-sensed data and stores them in the corresponding database in the cloud. For example, if the crowd-sensed data received by the cloud are for water quality, the Data Distribution Manager will store these data in the Water Database.

- *Local Manager*

  The Local Manager activity takes place in the public local server and is responsible for dealing with data once received in the server in terms of trust calculation, compression and scheduling. In more detail, and as shown in Figure 3.2, when crowd-sensed data are received in the local server, the Local Manager will first store the data in a local database for a period of time, which is defined depending on the application in use. It will then input the data to the Trust Manager. After that, passing through Scheduler, it will send the trusted data to the Local Reduction Unit, which will then pass trusted compressed data back to the Scheduler. The Local Manager is also responsible for sending the trust values of the users to the Reputation Manager in the cloud.

- *Query Manager*

  The Query Manager is an interface that allows consumers to query the Smart City databases located in the cloud. Using this interface, consumers can manage data in terms of partitioning, optimizing and extracting knowledge.

- *Execution Manager*

  The Execution Manager is located in the cloud and will simply receive the queries from the Query Manager and execute them.

The **Service Layer** includes all the different services in the local servers and the cloud that were presented in the previous section (section 3.4). The **Data Layer** contains all the databases needed in this thesis, which are all stored in a relational form. The Users Database contains users' account information and the Reputation Database holds the users' reputation values. The Smart City Database contains different databases, such as Water, Weather and the Environment. Every Smart City Database corresponds to one Context Database. The

Context Databases contain the knowledge extracted from the different Smart City Databases.

Finally, the **Servers Layer** is basically the cloud and the edge servers (i.e. public local servers) in this thesis.

The next section presents system featues that shows how the proposed architecture can meet the requirements outlined above.

3.6 Comprehensive System Features

The proposed system can tackle the requirements listed above and, therefore, provide a solution to each of the challenges addressed earlier in this chapter. Although the complete evaluation will take place in detail in the next chapters, the following is a high-level evaluation:

- Instead of locating all the services in the cloud, and in an approach that is different from previous work, some of the components of the proposed system are located as close to the crowd as possible and the services will run locally in local servers. This will increase the benefits of the system components, the most important of which is reduced data size before sending to the cloud (by removing untrusted data and compressing GPS coordinates and Accelerometer readings) and therefore, reduced bandwidth requirement and use.
- Trustworthiness of the crowd-sensed data is evaluated using four factors: user status, data variety, loyalty and similarity. The calculation of these factors decides whether the crowed-sensed data are trusted or not.
- The issue of exposing user identities when using traceability is overcome by ensuring anonymity. This is because user identities are not sent to the cloud in the first place. Since the only users' IDs used are application IDs that are assigned to every contributing user, these IDs are the ones sent to the cloud.
- Depending on the application in use (Weather, Traffic, Pothole Detection, etc.), the proposed system uses a scheduler to evaluate the usability and priority of the crowd-sensed data before they are sent to the cloud.
- Crowd-sensed data size is reduced using compression. This compression process is located in the proximity of the crowd (public local servers).

- The crowed-sensed data are managed in the cloud using a partitioning method that partitions data using a set of parameters, such as time and access rate, which are introduced fully in chapter 5. After data are partitioned, two reduction services are applied to the appropriate data parts to avoid losing important data and values.

3.7 Chapter Summary

In this chapter, a set of challenges in the crowd-sensing and smart city applications domain were presented, as well as the challenges that will be addressed using the system proposed. The design of the system, showing the different components and services in both the public local servers and the cloud, was also introduced. It was demonstrated that these components could operate to complete the job required. The architecture and its five layers were also shown in more details. Finally, the proposed system was analyzed to determine how it could meet the requirements listed earlier.

The next chapter presents the implementation of the different components of the public local servers. This will show how the crowd-sensed data are processed once received to the local server. The implementation of the components located in the cloud is then introduced in chapter 5, in order to show how the crowd-sensed data are managed and reduced efficiently.

Chapter 4

Edge Services for Crowd-sensed Data

4.1 Introduction

Crowd-sensing, which involves anonymous crowd data contribution, can be used to develop a wide range of applications and systems, such as smart city applications. The success of smart city applications depends on the users' involvement. However, with the open nature of crowd-sensing and high user contributions, smart city applications can be exposed to untrustworthy and malicious data. For example, some users may unintentionally contribute incorrect data simply because the mobile device was in the wrong position or the sensor had gone out of calibration. Other users may contribute data that serve their own interest e.g., a leasing agent who sends fabricated low noise readings in order to sell a particular property. Smart city applications are useless for citizens in the community if data contributions cannot be trusted. Thus, the trustworthiness of data contributed by users/devices must be evaluated to identify a malicious contribution, i.e. ensuring data trustworthiness is essential to satisfying successful smart city applications [66].

Another key aspect is the possibility of tracing the origin of data without affecting contributors' privacy. Over time, clouds store large amounts of data that will be used by many different applications. If data are wrong and there is no information about their origin, not only will applications return incorrect results, but there is also no way of identifying the source(s) of the errors. Therefore, data traceability should be another feature of smart city applications when processing crowd-sensing data. This new feature of mobile cloud architecture is important, even in the presence of a trust system as an additional check, identify and correct mechanism.

Furthermore, there is another challenge that lies in the context of the cloud: the more data are sent to the cloud, the greater the size of data that needs to be managed. Sometimes, the free space in the cloud is limited: for example, Google Drive provides 5 GB, then users need to selectively send data to the cloud as all cloud providers gave users pay as you go storage. Thus, data sent to the cloud need to be reduced in size.

In this chapter, three different services (Trust, Scheduling, Compression) are presented where these services are in the proximity of the crowd. Moreover, the utilizing of local servers and the benefits behind it are presented in this chapter as well.

4.2 Problem Formulation

This chapter is part of the work carried out to develop a crowd-sensing architecture that provides cost-effective services in the context of smart city. The increasing volume of crowd-sensing data rises the necessity to adapt local processing in order to send and store these data efficiently in clouds. The challenges presented in chapter 3 are important and can be overcome by processing data at the edge (i.e. local processing). Therefore, in this chapter, the idea of edge servers (i.e. public local server) will be utilized in order to have what so called "local processing". The users, contributing data in the sensing task, will always send data to the nearest public local servers in order to perform local processing to the data before the cloud. All of the services presented in this chapter will take place in the public local servers.

4.3 Origin and Trustworthiness of Data

In this section, a reputation system for evaluating the trustworthiness of the crowd-sensed data contributed by users/devices is presented. This reputation system is different from previous work, discussed in section 4.3.1.

Instead of having the reputation system deployed in the cloud, its service will run locally, on the public local servers. Local processing of crowd data has several benefits, especially in reducing the amount of traffic and data filtering. This service allocation on local servers is reasonable in a city Wi-Fi network that is designed to support the community. The following are the contributions in this section:

- A data traceability service.
- A trust and reputation system that is located in the proximity of data contributors using public local servers.

Definitions

In this thesis, "trust" and "reputation" are used as separate views, following another work [176]. The term "trust" relates to building the user's trust and level of reliability in his/her current contribution accumulated by the local server, i.e. data sensing contributions performed in a specific period of time, such as one day. Reputation management takes place in the cloud to update the user reputation for all of his/her contributions regularly. Therefore, trust is a value calculated in the local server over a period of time, alongside the sensed data. On the other hand, reputation is a value calculated in the cloud, whereby the cloud takes the new trust value received from the local server for a specific user and adds it to the previous reputation to update the reputation value for this particular user. The history is considered in the reputation calculation process, not in the trust process.

Motivation

Ultimately, the trustworthiness of the contributed data and establishing the origin of those data are very important in any crowd-sensing application in order for the users (i.e. data consumers) to use the data confidently. Therefore, detecting the trustworthiness of data is a major challenge, which, when overcome, will help guarantee the success of the crowd-sensing application. The following are some scenarios in which evaluating data is very important:

1) Events Reporting: Users could report events that occurred in malls, universities and other public places and share them with others. However, malicious users could try to fabricate the appearance of a crowd at an event in order to promote it [170].
2) Road Repairs: Suppose the City Hall of city "X" launched a mobile application that helped users to send photographs and locations of the roads

that need to be repaired in their neighborhood. Malicious users could fabricate images in order to have their neighborhood repaired first [19].

3) Traffic Warning: Suppose there was an application that received traffic jam warnings from users in a specific location and distributed these alerts to other drivers. Untrustworthy users could send false warnings for some location to reduce traffic congestion for themselves [65] [182].

4.3.1 Previous Studies

A widely used approach to the efficient assessment of the trustworthiness of sensed data received from different users is to use a reputation system [66]. Researchers [67] have proposed a reputation system for participatory sensing applications that measures user trust by giving a reputation value to each user for his/her contributed data. They adopt a similar architecture to the Reputationbased Framework for Sensor Networks (RFSN) [51], a reputation framework for traditional embedded wireless sensor networks. The downside of this system is that the trust value of any user is only computed by taking into account the historical behavior of that user, unlike the approach in this thesis, in which the trust value is calculated regardless of the previous contributions.

In the proposed Wi-Fi sensing system for smart city applications, authors [183] address the trustworthiness of the data published in their system by forming endorsement links between users so that the users can submit reviews of Wi-Fi hotspots along with the data publishers. Their system is different than the proposed approach in that trustworthiness is actually calculated in the cloud by the number of submitted reviews and the number of endorsements earned, not by testing the data itself (in their case, the strength of the Wi-Fi).

Another system addressing the trustworthiness of crowdsourcing systems, called Trustworthy Sensing for Crowd Management (TSCM), has been proposed as a cloud-centric crowd management scheme [78]. TSCM adopts the MSensing auction and incentives system for smart phones [191] and improves it by presenting the reputation awareness and trustworthiness of the smart phone users. User reputation is updated regularly in the cloud in TSCM, where reputation is used as a basis to pay users and assign tasks for them. The reputation score calculation depends on the accuracy of the sensed data. The authors' system performs the auction and the calculation of the trustworthiness in the cloud, while the system in this thesis undertakes the calculation of trust locally. Furthermore, in their case, the cloud receives all the sensing data, whether trusted or not, from users, whereas for the system in this thesis only the sensed data by users with a trust value above a specific threshold are sent to the cloud.

ARTSense [177] is a framework that addresses the problem of trust and anonymity in participatory sensing systems. The trustworthiness of the sensing data is calculated using a trust assessment algorithm. Their algorithm is similar to the proposed approach as they use different weighting factors (location, time, sensor mode and traveling mode) to build trust but concentrate on anonymous user reputation levels and preserving privacy for users. Unlike the local processing in this thesis, ARTSense performs the anonymity and trust algorithm in the cloud.

CrowdSC [19] is a crowdsourcing framework that uses citizen participation in the context of a city. It transforms queries into simple tasks: collect, filter the data provided by users, and provide as well as return the results to the user. To make the right assessment of the data, authors propose a three-step process. The data collection step collects photos from participants. The data selection step asks other participants to select the photo that best represents the problem. Finally, the data assessment step asks participants to assess each photo selected from the data selection step. The proposed system performs all of these steps for assessing data contributed by users in the cloud, which means that the cloud will receive both useful and useless data during the process, which is impractical, and not the case with the approach in this thesis. All of the above systems use a cloud/server to compute the trustworthiness of the data contributed. In the proposed system in this thesis, the trustworthiness calculations take place locally in, for example, WLAN APs or a local server.

4.3.2 Design

When the sensed data are received by the local server, the data are cached and stored in a log file, along with the user ID, for a determined period of time (e.g., one day) in order to start building trust. Four important factors are considered for assessing trust with a weighting parameter for every factor:

1) The user status (i.e. not moving, walking, or moving fast) is an important factor that could affect the quality of the sensed data being sent. The weighting parameter (S) of every status is illustrated in Tab. 4.1. Every status is given

a weight according to the quality of the data produced with it. This weight reflects how important the status of the user is when he/she starts the sensing process, where some statuses have higher weight than others. The numbers in Tab. 4.1 are chosen to show the impact of the proposed trust system based on the threshold assigned later in this section. For example, the quality of a picture taken from a moving vehicle is different from that taken when the user is stationary. Pictures may be shown to be blurry when someone is moving. Someone taking a picture while walking will also have less chance of taking a blurred picture than when running. Therefore, the "not moving" status is accorded a higher weighting than "walking", "walking" has a higher weighting than "running", etc.

Table 4.1: User status weighting parameter

User Status	S
Not Moving	0.2
Walking	0.15
Running	0.1
Moving > 40 mph	0.05

2) The variety of sensed data will add more weight when building trust. Therefore, the weight parameter is higher for a user who sends a photo along with a recording and location than a user who sends a photo with a location only. The weighting parameter for the different sensors is shown in Tab. 4.2. The difference in every weight assigned to every sensing style depends on the degree of effort made by the user, where some styles have higher weight than others. Recording voice notes and taking photos need more effort from the user than location, since the location is sent by default from the user's mobile phone. In the "SenseAll" application (see Appendix A), the location needs to be turned on before sensing in order to send the sensing data (voice, photo). The numbers in Tab. 4.2 are chosen to show the impact of the proposed trust system based on the threshold assigned later in this section

3) The user who contributes more to the sensing activity will add more weight to his/her trustworthiness. If one user sends sensed data regularly during a period of time (the time before the log file is sent to the cloud), the trustworthiness of this user and the data he/she sends will be scored as 0.05 for every contribution, excluding the current contribution. Loyalty is an important factor

Table 4.2: Sensing style weighting parameter

Sensing Style	*SS*
Location	0.05
Voice	0.2
Photo	0.2
Accelerometer	0.1

in the trust system proposed. It is calculated for this parameter for user “u”:

$$N_u = \text{number of previous contributions in a predefined time} *(0.05) \tag{4.1}$$

4) Sensed data that are similar to each other are beneficial to each other. For example, if user “X” and user “Y” are in the same location and send data that are similar to each other, this will add trust for both users. On the other hand, if the two users are in the same location but their data are in conflict, the sensed data of the user with the higher reputation, from previous contributions, will be considered and the other one will be discarded. Therefore, similar data will be weighted more heavily when calculating the trust value. The similarity between two contributions varies from one application to another. For example, one application considers two contributions as similar if they are captured in the same location. Another application considers two contributions as similar if the location and photos captured are for the same incident. This factor is calculated as follows:

$$\text{Sim} = \begin{cases} 0.1 & \text{if two contributions are similar } (\forall u \text{ and } u' \in \text{LOG}) \\ 0.0 & \text{if two contributions are not similar } (\forall u \text{ and } u' \in \text{LOG}) \end{cases} \tag{4.2}$$

Where u and u' are two different users, LOG is the log file in the local server during the predefined time. The value (0.1) is chosen for similar contributions based on the trust threshold shown next.

After these factors are calculated, (4.3) is performed to compute the trust for the contribution for user “u”:

$$T_u = e^{(Su+SSu+Nu+\text{Sim}_u)} \tag{4.3}$$

This equation will produce a value T_u from (1.0) to (3.8). The value (1.0) corresponds to when the user contribution contains only the location of some incident, then *SS* is (0.5) leading to a minimum value of (1.0) for T_u. Again, if *S* is (0.5) and *SS* is (0.5), then adding these two values will make (0.1) and leading to a value of (1.1) for T_u. The value (3.8) corresponds to when the exponent is equal to (1.35), this is when *S* is (0.2), *SS* is (0.55), N is (0.3) and Sim is (0.3). In other words, T_u will have the highest value (3.8) when the contribution is captured while the user is "Not Moving", the contribution contains all Sensing Styles (*SS*), the user had (6) previous contributions $(6 * 0.05 = 0.3)$ during the predefined time and 3 similar contributions $(3*0.1 = 0.3)$ that are captured during the predefined time.

The value T_u is tested before sending it to the cloud (after the predefined period of time, e.g., one day):

- If the value is below a specific threshold (below or equal to 1.2, where this threshold is chosen based on the lowest accepted trust value when the sum of the factors is (0.2). This happens in four different situations: first, when *S* is (0.15) and *SS* is (0.5). Second, when *S* is (0.05) and *SS* is (0.15). Third, when *S* is (0.05), *SS* is (0.05) and N is (0.1). Finally, when *S* is (0.05), *SS* is (0.05) and Sim is (0.1). These cases produce the lowest trusted values. This is because in these cases the user is considered aware of the incident that needs to be captured, the incident is supported by similar contributions, or the user is considered loyal (i.e. two previous contributions during the predefined time), therefore, his/her contribution needs to be trusted), then the contribution will be discarded and the trust value T_u is re-calculated using the "penalty" equation in (4.4) and sent to the cloud along with the user ID:

$$T_u = -e^{Tu} \tag{4.4}$$

  Note that T_u in (4.4) is a negative number. The reason behind this equation is to show that one untrusted contribution will have a negative exponential impact on the reputation.

- If the value is above the threshold, then it will be sent to the cloud along with the user ID and the crowd-sensed data. The user ID and the crowd-sensed data are saved in the corresponding database

From above, it can be seen that calculating the exponential sum of all of the factors in (4.3) will give more control of the numbers in order to highlight what is accepted and what is not.

The trust value is used to update the user reputation by adding it to the previous value of the reputation:

$$\mathrm{Rep}_u = \mathrm{Rep}'_u + T_u \tag{4.5}$$

Where Rep_u is the new reputation value and Rep'_u is the previous reputation value. The reputation is in the range from (0) to (100).

The intuition behind how the trust is calculated is that consecutive high trust values will build a reputation, but one low trust value will ruin it.

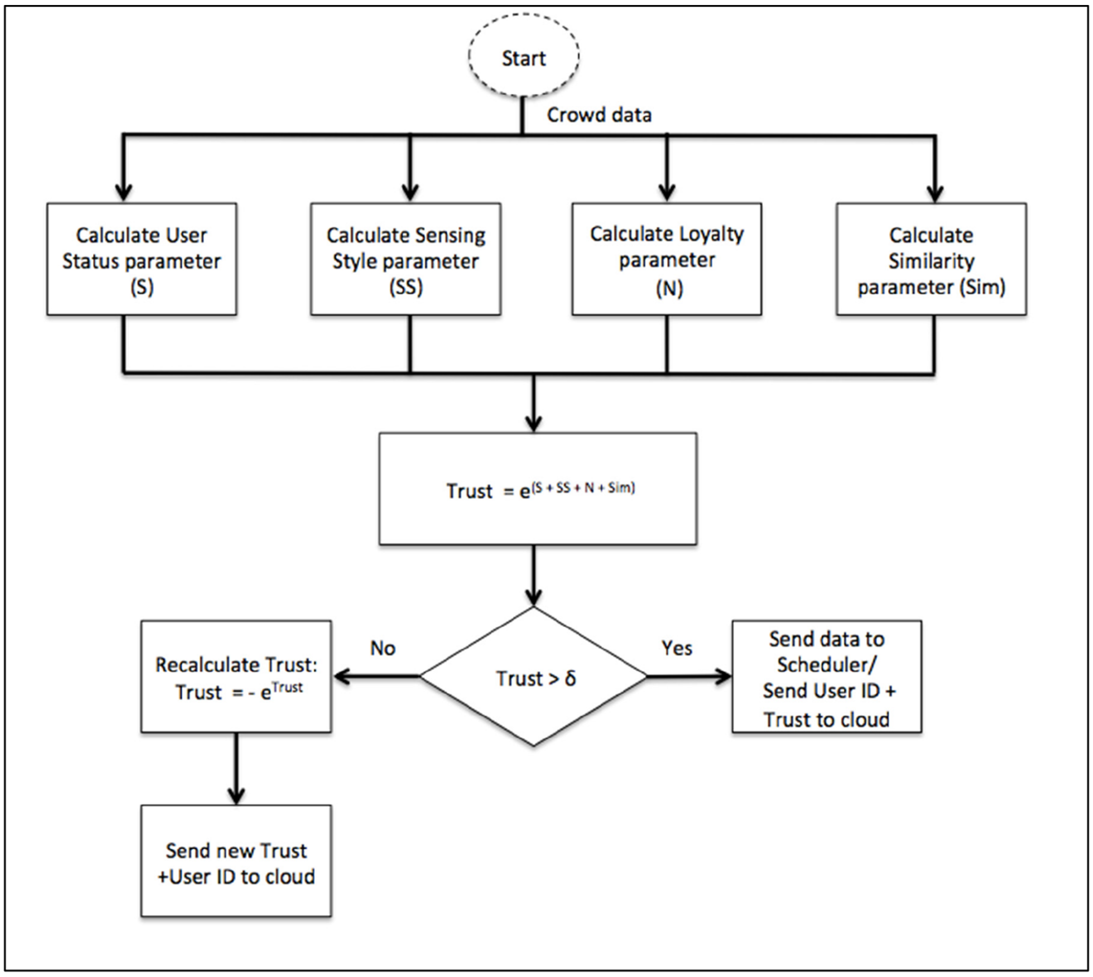

Start
Crowd data
Calculate User Status parameter (S)
Calculate Sensing Style parameter (SS)
Calculate Loyalty parameter (N)
Calculate Similarity parameter (Sim)
Trust = e^(S + SS + N + Sim)
Trust > δ
No
Yes
Recalculate Trust: Trust = - e^Trust
Send data to Scheduler/ Send User ID + Trust to cloud
Send new Trust +User ID to cloud

Figure 4.1: Task Flow Diagram in trust service. Trust threshold is denoted in δ

Example. User "X" downloads the SenseAll application, agrees to turn the location on and wants to start sensing data. He/She takes a photo while walking in an area of interest and sends it to the cloud. Before being received by the cloud, the data are evaluated in the local server. User "X" will have a weightof 0.15 for user status, 0.05 and 0.2 for location and photo, respectively, 0 for the number of contributions made during the day (since this was the first) and 0 for similarity, as no similar sensing data have been taken in that specific location. Then, $T_x = e^{0.4} = 1.4$, which means the data of user "X", along with his/her trust value $T_x = 1.4$ and user ID, are sent to the cloud, as T_x is above the threshold.

4.3.2.1 Traceability Requirement

Depending on the application, tracing the origin of the data is a key factor in the success of smart city applications since this helps in tracing the errors if any appeared in the data. In this thesis, data traceability is taking into account when users contribute in the sensing task but without affecting the user's privacy concerns. To illustrate how traceability is considered, "SenseAll" mobile app is the android mobile app that is used through out this thesis for different data sensing and gathering. "SenseAll" will have application ID assigned for every user using the app along with a user-generated password. To support traceability, users using this app will have location services turned on (by default) when sensing. However, the user will have the choice to turn it off if he/she has some privacy concerns. Full information about the app is presented in appendix A.

4.3.3 Use Cases

The use case below will demonstrate how the trust algorithm works:

- **User "X"**

User "X" downloads the SenseAll application, agrees to share his/her location and wants to start sensing the environment in location "A". During the first four days of sensing, shown in Table 4.3, he/she gains trust values that are above the threshold (1.2), which means that the reputation value is added every day (Day $1 = 1.3$, Day $2 = 1.3+2.1 = 3.4$, Day $3 = 3.4+1.9 = 5.3$, Day $4 = 5.3+1.7 = 7.0$).

However, on the fifth day, the user sends data that have a trust value equal to the threshold, as the picture he/she sends is corrupted and only the location counts.

Therefore, the trust value is calculated using (4.4):

$$T_x = -e^{1.2} = -3.3$$

Table 4.3: Trust and reputation values for user "X"

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
Trust	1.3	2.1	1.9	1.7	1.2	1.4	1.6
Reputation	1.3	3.4	5.3	7.0	3.7	5.1	6.7

And the reputation is calculated using (4.5):

$$\text{Rep}_x = 7.0 - 3.3 = 3.7$$

Note that one bad contribution will have a detrimental effect on reputation (see Fig. 4.2). On the sixth and seventh days, the user contributes good data that improve the reputation but the harm of that one bad contribution cannot be recovered completely after two days.

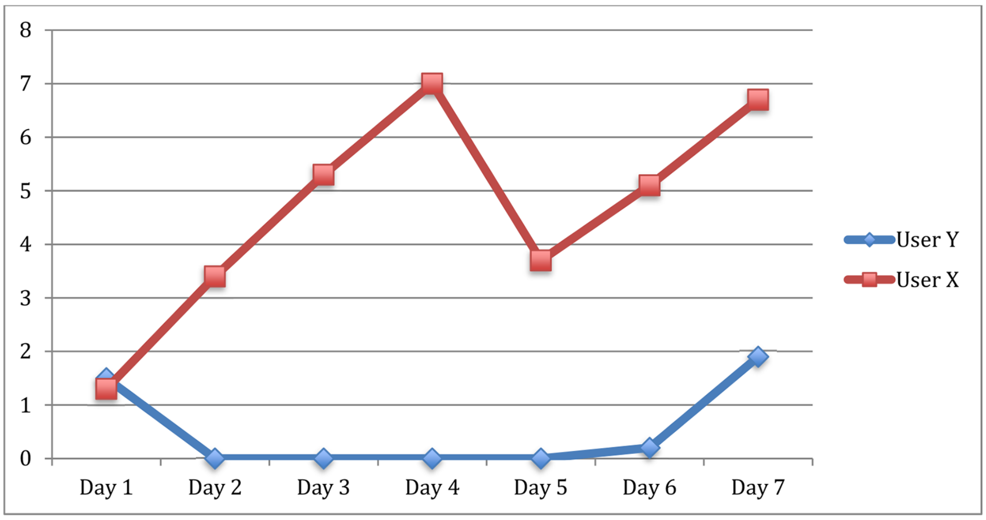


Figure 4.2: User "X" and user "Y" reputation values in one week

• User "Y"

User "Y" downloads the SenseAll application, agrees to share his/her location, and wants to start sensing the environment in location "B". During the first day, shown in Table 4.4, he/she contributes trusted data and has a reputation value of 1.5 . The data contributed on the following two days are below the threshold. This leads to negative reputation values that equate to a reputation of zero. The reputation value is treated as zero because there is no scale for negative numbers in the presented approach; These values are simply kept to make the reputation calculations. These are the calculations that occurred on the second and third days.

Table 4.4: Trust and reputation values for user "Y"

	Day 1	**Day 2**	**Day 3**	**Day 4**	**Day 5**	**Day 6**	**Day 7**
Trust	1.5	1.2	1.1	1.7	1.9	1.4	1.7
Reputation	1.5	-1.8	-4.8	-3.1	-1.2	.2	1.9

In day 2, the trust value is calculated using (4.4):

$$T_y = -e^{1.2} = -3.3$$

And the reputation is calculated using (4.5):

$$\text{Rep}_y = 1.5 - 3.3 = -1.8$$

In day 3, the trust value is calculated using (4.4):

$$T_y = -e^{1.1} = -3.0$$

And the reputation is calculated using (4.5):

$$\text{Rep}_y = (-1.8) - 3.0 = -4.8$$

Then, as shown in Table 4.4, the user took three days to recover from the two consecutive pieces of untrustworthy data contributed (see Fig. 4.2). Therefore, in this trust approach, building trust takes time but contributing malicious data ruins a reputation.

4.3.4 Discussion

This simple but effective approach will reduce the amount of untruthful data in the cloud by a large degree, since data with low scores (below a specific threshold), according to the factors presented, are discarded and not sent to the cloud. Unfortunately, existing solutions proposed in related works to solve the issue mentioned in the scenarios in section 4.3.1 and others are weak because they:

- Either require offloading all sensed data to the cloud and evaluating the trustworthiness of the data there, which is unnecessary traffic and time consuming, or

- Add significant overheads to users' mobile phones, since the processing and evaluation takes place on the phone.

In the proposed solution, the sensed data are processed and evaluated locally. Malicious data are immediately discarded when discovered, and not sent to the cloud. Thus, more time is saved and traffic overheads are avoided when sending data to the cloud. Furthermore, in this approach, that the location service in the mobile device is required to be turned ON before the user starts sensing in the "SenseAll" application in order to support traceability. With traceability, smart city applications in the cloud will benefit the most from the data sensed. In the proposed approach, the issue of exposing user identities to traceability is overcame by ensuring anonymity. This is because user identities are not sent to the cloud in the first place, since only use application IDs assigned to every contributing user are used, and these IDs are the ones sent to the cloud.

4.4 Scheduling Data

In this section, a scheduler service for crowd-sensed data is presented that is located in the public local servers. The scheduler calculates the priority factor for each user's sensor data in the local server before the data are ready to be sent to the cloud. Data with a high priority are sent first and data with low priority are scheduled for another time. Therefore, the scheduler will examine the relevance of the crowd-sensed data and prioritize sending these data to the cloud.

Data Usefulness definition in the context of the scheduler

Data usefulness is defined in terms of how important, and not redundant, crowd-sensed data offered by users are in order to send them to the cloud. Crowd-sensed data with high priority are sent first.

Motivational Example

Suppose the City Hall of city "X" launched a mobile application that let mobile phones detect the locations of roads that need to be repaired in their neighborhood and send this information to the cloud. In some cases, the cloud will receive locations of secondary roads that need to be repaired (i.e. not main or busy roads), or roads located in some area that is not of high risk e.g., no schools or playgrounds nearby [19]. Thus, the cloud might receive locations of damaged roads that are not of high importance, thus raising the question of calculating the priority of the crowd-sensed data before sending them to the cloud. Furthermore, another significant case is when a number of citizens who go along a busy route might send information for the same pothole(s) that they pass to the cloud; these replicated data will consume bandwidth. The idea of examining data usefulness (i.e. priority) is, therefore, important, meaning that it is necessary to assess how urgently a road needs to be repaired before sending the crowd-sensed data to the cloud. For that purpose, a scheduler is required to be in proximity to the crowd in order to send the crowd-sensed data to the cloud based on their priority and avoid transmitting the same data for the same pothole a number of times.

4.4.1 Previous Studies

There are a number of works related to sensing the environment and cloud computing integration. They all share the same model, which is to send all the information to the cloud to be processed there, but differ from each other in using mobile phones or wireless sensor networks. One of them is “CrowdSC” [19], that uses citizen participation in the context of a city where the citizens will respond to the queries by taking photos. Then, they are asked to select the photos that best describe the issue and assess these photoes. Another work presents CrowdOut [15], that engage users in reporting dangerous traffic situations (e.g., speeding) in real time and map it on a city. Furthermore, in [44], authors present an application called PotHole Patrol that reflects road situation by using data generated from the sensors deployed in vehicles.

In terms of examining data usability, other authors [197] have proposed time- and priority-based selective data transmission (TPSDT) for a Wireless Sensor Networks gateway that examines the usability of crowd-sensed data and depended on a Point versus Time and Priority (PTP) table maintained in the cloud using the data requested by each mobile user. Their system is similar to the service presented in that the usefulness of data is examined before sending them to the cloud in order to avoid network traffic and decrease transmission bandwidth. However, their system is different from the proposed service in that they maintain their PTP table by analysing mobile users’ behaviour in the cloud during a period of time, while the service in this thesis uses users’ reputations and a scheduling algorithm (see section 4.4.3) for the selective transmission of useful data to the cloud.

4.4.2 Scheduling Service Requirement

Once the crowd-sensed data received to the local server, the trust will be evaluated first by the trust service presented in section 4.3, then the scheduling service will take place. The crowd-sensed data should be trusted (after evaluation) in order to enter the scheduling service. Untrusted data are discarded after the trust evaluation.

4.4.3 Design

Once received by the local server, data are cached. The meta-data for this contribution, such as user ID, are registered in the log file for a set period of time (i.e. one day) in order to collect more data and calculate the priority of these data and start building trust. At the end of this period of time, the local server will selectively send data to the cloud depending on their priority and the user's trust. First, the user's trust is calculated using weighting factors presented previously (section 4.3.2), whereby crowd-sensed data for trusted users are kept in order to schedule sending them to the cloud and data that are not trusted are discarded. After the trust is calculated for any particular user, the reputation value for this user is updated in the cloud. The cloud has a reputation reference for every user and that this is updated regularly and sent to all local servers. Then, in the proposed scheduling method, there is a priority weight (w) that has, by default, the value of zero for every user contribution at the beginning. Its value is calculated by the local server using different factors applied in the following order:

1. Untrusted data are discarded using a trust evaluation method; only trusted data are considered for scheduling.
2. At the end of the determined time, i.e. one day, if there are two or more trusted users who sent data for the same location, then only the crowd-sensed data from the user with the higher reputation value are considered; the other similar contributions will add weight to this user's data using an approach outlined in equation (4.6), called a similarity weight for user "u" (sim_u):

$$\text{sim}_u = n - 1 \quad (\text{if } n > 0) \tag{4.6}$$

   Where "n" represents the number of all contributions in the same location as user "u", including his/her contribution. For example, if there are three contributions from three users (including user "u") in the same location and only the contribution from user "u" is accepted (since he/she has the highest reputation value), then $n = 3$ and $\text{sim}_u = 2$.

3. Applying the scheduling method to the crowd-sensed data left by calculating the priority depends on the application used. In this thesis, "SenseAll" application that automatically detects potholes tagged with location and time. The priority is calculated by assigning a weight value for every decision question. These decision questions can be changed depending on the city's requirements or locations and are answered by the public local servers using the city map that is installed in every local server around the city. As an example, the decision questions for the application in this thesis are:

    - "Is the data received located in a children-designated area?": this decision is the highest in weight since the safety of children is the most important.
    - "Is the street considered a main/busy street?" It is important to know if the road with a pothole is a main street (e.g,, a highway), a busy street (not a highway) or neither (a sub-road), since potholes in main or busy streets might increase congestion.
    - "Is the data captured during the rush hour?" This question is important because congestion might be increased or even caused by the pothole.
    - "Is it a rainy area?" This decision question is not as important as the others but areas with a lot of rain and potholes covered by water might be dangerous due to the unexpected reactions of traffic participants, particularly cyclists.

All trusted not-redundant crowd-sensed data pass through these decision questions and acquire a weight, as depicted in 4.5. After that, these data are sent to the compression service, if floating-point numbers exist, (section 4.5). Then, the data after compression, if any, will be scheduled to be sent to the cloud depending on their weight. Crowd-sensed data with a higher weight are sent first in order to receive a fast response from the local authority, lowerweight data are scheduled for a later time and, finally, data with zero weight are discarded.

Table 4.5: Decision questions and the corresponding scheduling weight

Decision Questions	**Scheduling Weight** (w)
Rainy area?	$w = 1$
Main road?	$w = 2$
Busy road?	$w = 2$
Rush hour?	$w = 1$
Child area?	$w = 3$

The weight ranges from 0 to 7, where 7 is the highest weight and this happens if the road is designated for children, busy, data were captured in rush hour, the area is rainy and no similar data are detected; however, it can be more than 7 if similar data are detected.

Example. As an example, consider a pothole that is located in a rainy area on a main road. User "X" (who used a pothole app on his mobile device) passed by this pothole using his bicycle at 4 pm (which is considered part of the rush hour). His mobile device detected the location and sent it to the nearest local server "SERVER1", along with the time and a voice note that recorded the environmental noise at the time (the app records every period of time and the last recording when the pothole is detected is the one that will be sent).

Another user, "Y" (who used the same pothole app), detected the same pothole when passing it using his car at 4:30 pm (also rush hour). The mobile phone of user "Y" will send the data to the nearest local server, "SERVER1". Data for "SERVER1" show that user "X" and user "Y" are trusted (using trust method in section 4.3) and that user "Y" has a higher reputation value than user "X". Then, the data contribution from user "X" is discarded, and a value of $w = 1$ (similarity weight) is given to the data from user "Y", which are considered for scheduling. The scheduler computes a weight of 4 for user "Y". Therefore, the contribution of user "Y" is scheduled to be sent to the cloud with a weight of $4 + 1 = 5$. However, if there are other crowd-sensed data with a higher priority weight than 5, these data are scheduled to be sent to the cloud first.

4.4.4 Evaluation

4.4.4.1 Evaluation setup

To perform the evaluation, the effectiveness of the scheduling method is examined when scheduling important and unduplicated data regarding pothole detection. First, the proposed method will show how it deals with data when received by local servers (before sending to the cloud) and then compare it with the idea of sending all the crowd-sensed data to the cloud, as in all previous work. In the simulation, there is one cloud, one local server and 50 mobile users (i.e. a crowd), with a total of 50 contributions in one day. There are two assumptions, first, the cloud will send the updated reputation values for every user to the local server every day and the other is that all the data received in the local server are trusted. Furthermore, another assumption is that Wi-Fi services are distributed in the city to support the community.

Table 4.6: Locations and descriptions

Location	Description
LOC1	Rainy area+ Main road
LOC2	Rainy area+ Main road+ Rush Hour
LOC3	Rainy area+ Busy road+ Child area
LOC4	Rainy area+ Busy road+ Child area+ Rush hour
LOC5	Rainy area+ Sub-road
LOC6	Rainy area+ Sub-road+ Child area

The roads are categorized into three different types. First are Main roads, such as highways. Second are Busy roads that are defined as roads with a large number of vehicles passing every day. Third are Secondary roads, which are neither Main nor Busy roads. For every type, there are 2^n possibilities for road descriptions, where $n = 3$ according, since there are three conditions for every road type (Rainy area, Child area, Rush Hour). In this simulation, two cases for every road type are covered. The Rainy road condition is included in every case, since Cork is considered a rainy area. Furthermore, the road descriptions are chosen after discarding unnecessary road conditions in the city. For example, main roads are usually highways that are crowded during rush hour but there are no child areas on highways; therefore, main roads during rush hour are the ones examined. However, child area conditions are taken into consideration for both busy roads and subroads. For the sub-roads, rush hour conditions are not examined due to their stable condition during the day (they are quiet roads all day long). The data were collected from six different locations that contain potholes. The descriptions of these locations are shown in Table 4.6. There are different numbers of contributions in each location as shown in Figure 4.3. All the contributions from the same location will have the same weight value as they detected the same pothole; however, the users' reputation values are used in the final decision to determine which contribution is considered.

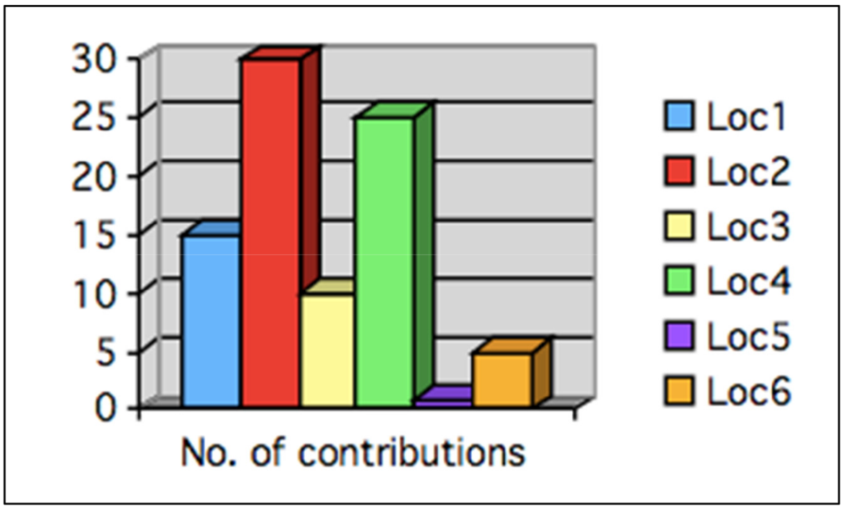


Figure 4.3: Number of contributions in every location

4.4.4.2 Evaluation result

The evaluation results with respect to the scheduling method and the usability of the data are presented in Figure 4.4. The scheduling approach is highly effective when a higher number of contributions are received from the same location. For example, in locations 1, 2, 3 and 4, the amount of data sent to the cloud is reduced by at least 90%, where only 10% or less of data are sent (only one contribution in every location). In location 6, the amount of data sent is reduced by 80% using the scheduling method. However, this approach showed the same result as the "offloading all" method in locations where only one contribution was sent due to the area being quiet (i.e. location 5).

For every location, only one contribution was chosen, depending on the reputation values. The priority weight for the one contribution selected in every location after performing the scheduling algorithm is indicated in Table 4.7. Then, the contribution from location 2 was the first to be sent to the cloud, followed by location 4, location 1, location 3, location 6 and, finally, location 5. The amount of time between contributions depends on several factors. One factor is the priority threshold that is defined by the city council or municipality (whoever is interested in the contributions). If one contribution exceeds the threshold, it will be sent immediately to the cloud. For example, if the priority weight of one contribution in this evaluation exceeds 100, this means that the pothole is really dangerous and needs to be sent to the cloud immediately in

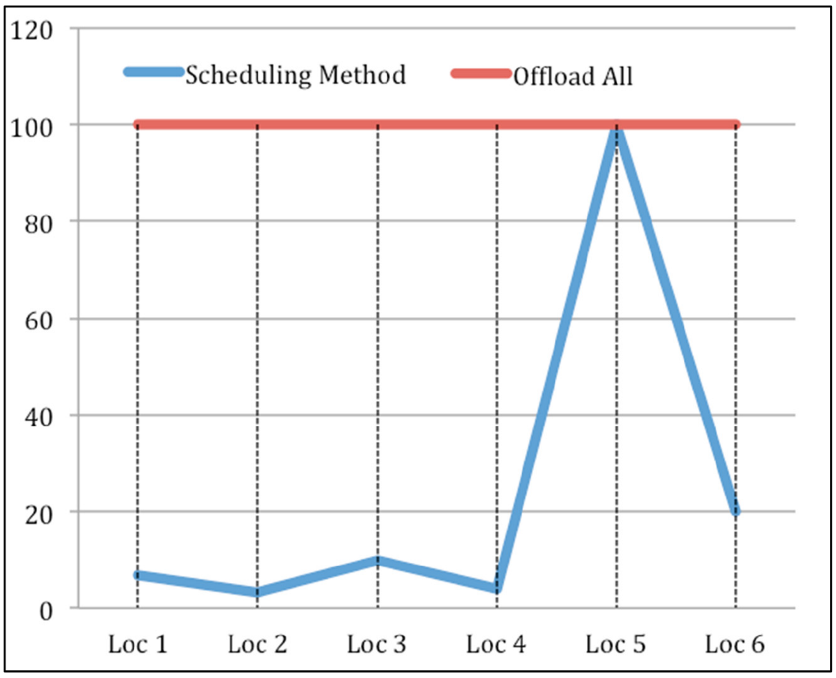


Figure 4.4: Average of data sent (Y-axis) in every location (X-axis) using the scheduling approach and offloading all the crowd-sensed data

order for the city to take a quick action. Network traffic is another factor that can postpone the transfer of contributions with lower weight. Furthermore, the number of contributions that can be sent to the cloud (bandwidth and network specifications) in a predefined time is considered another factor. However, the details of all of these factors are beyond the scope of this thesis.

Table 4.7: Priority weight (w) for every contribution considered for scheduling in every location

Location	Similarity (sim)	Priority (w)	Total
LOC1	14	3	17
LOC2	29	4	33
LOC3	9	6	15
LOC4	24	7	31
LOC5	0	1	1
LOC6	4	4	8

4.4.4.3 Discussion

The approach presented in this section is effective in reducing the amount of data sent to the cloud to only data that are relevant to the application. Instead of offloading all the crowd-sensed data to the cloud, only part of them are sent by the scheduler running on the distributed public local servers that are close to the crowd. This scheduler performs two main jobs:

1. If the local server receives a number of contributions for the same location, the scheduler will select the data that will be sent to the cloud based on the reputation values of the users. Other data are discarded but the number of contributions is considered as a value (similarity weight) that is added to the priority factor.

2. Schedules the sending of useful data to the cloud based on their priority. Data with high priority are sent first.

The scheduling approach is more effective in situations with a large number of contributions for the same location. Then, the consumptions of network bandwidth and network traffic are significantly reduced.

4.5 Single-precision Floating Point Compression

A compression method for single-precision floating-point data is the contribution presented in this section. This compression method is located in the Reduction Unit of the local server. The distribution of local servers in a smart city is taken into account and therefore two compression techniques are presented: one for location-based data (latitude and longitude) and the other for an accelerometer's three-dimensional data.

IEEE floating-point numbers have three basic components: the sign, the exponent, and the mantissa. The number of bits in single and double precision is shown in Table 4.8. In this thesis, the focus is on single-precision floating-point values (Fig. 4.5).

Table 4.8: Floating Point Representation

Precision	Sign	Exponent	Mantissa
Single	1	8	23
Double	1	11	52

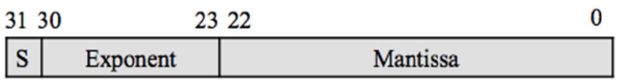


Figure 4.5: Single-precision floating-point format of IEEE 752 standard

4.5.1 Previous Studies

Although data compression is an attractive topic in the literature, only a small number of studies have focused on the floating-point compression that is needed in big data scenarios, such as crowd-sensing, smart city data and scientific data.

4.5.1.1 Scientific Data

In [96], authors propose a compression method for floating-point data streams. They used a prediction method in order to find similar numbers in a stream and use them as predictions. Then, after performing some operations on the actual number and the predictions, the residuals are compressed using multi-way compression. Furthermore, authors of [138] proposed an algorithm that compresses sequences of IEEE double-precision floating-point values as follows. First, the algorithm predicts each value in the sequence and XORs (Exclusive ORs) it with the true value. If the predicted value is close to the true value, the sign, the exponent, and the first few mantissa bits will be the same. The work in this thesis has no prediction method; the compression was performed easily without any prediction limitations (i.e. over-processing).

In [169], a floating-point compression approach (fzip) is proposed. It has two stages: the first uses the coding scheme in Burrows-Wheeler Transform (BWT) compression [21]; the second is value and prefix compression, in which fzip uses arithmetic codes to encode the prefixes. Therefore, a different pattern is compressed at each stage. Fzip achieves a good compression ratio but performs badly in terms of runtime. Other work is proposed in [58], in which authors offer a binary masking technique that partially increases the regularity of high-performance computing data sets in order to create high compressible data sets before applying it for compression. Their work shows a good improvement in compression ratio. However, their masking performs well with data of high similarity (neighbor elements in high-performance computing have close values). Finally, in [43], authors propose a delta compression algorithm to compress 64-bit floating-point values by storing the higher-order differences between values.

The majority of the works above took advantage of the similarities in some scientific data sets and their work will not be as effective if the data set values are random. In this thesis, the compression algorithms proposed will deal with both cases: the repeated values in GPS coordinates and the randomness of accelerometer readings.

4.5.1.2 Audio and Image Compression

In terms of audio and image compression, Ghido proposes a compression algorithm for floating-point audio data [56].

The algorithm transforms the floating-point values into integers, maintains the properties of the original values and creates an extra binary stream used for the decompression of the original floating-point values. In the presented approach, neither transformation made nor an extra binary stream is added; the exponent part is used for reconstruction in the cloud (in accelerometer readings only; GPS coordinates are reconstructed in the cloud without the exponent or any additional bit).

Another work that represents the floating point as an integer proposes an addition to the JPEG2000 standard in order to encode floating-point data efficiently with bit-plane coding algorithms [173]. Furthermore, in [50], changes are performed on JPEG2000 to adapt lossless floating-point compression, such as optimizations in the wavelet transformation and beforehand signaling of special numbers.

In [71], authors use a context-based arithmetic coder for single-precision floating-point coordinates. They use the exponent to shift between different arithmetic contexts, while in the presented approach, the exponent is used to represent the exponent number and classify the integer part (see section 4.5.2 for more details).

4.5.2 Compression Design

4.5.2.1 Location-based Data Compression

In this section, a compression method is proposed that takes advantage of the smart city architecture introduced in chapter 3. Public local servers are distributed around the city to serve the citizens. Therefore, the following question was asked: why do the architecture not benefit from the server distribution and apply a compression method to GPS coordinates (latitude and longitude) that are represented as single-precision floating-point numbers (32 bits)?

The servers serve an area that has a diameter not more than 130 kilometers (approximately 130 km is the distance that changes the integer value in the coordinates from x.0000 to y.0000, where x and y are two consecutive numbers). This means that all contributions that are received by a particular server will have the same sign and integer part. However, some might receive a maximum of two integers and the way this is handled will be shown.

- Case 1 – Servers cover an area that has the same integral part for both latitude and longitude (Algorithm 1).

Algorithm 1: GPS coordinate compression (Case 1)

```
Input: x.y (x is the integer part and y is the decimal part)
Output: Result (the compressed number in binary form)
if x == num then                                  //num is the integer part that
    Result = $(y_0 \ldots y_n)_b$                     //is covered by the server
else
    No compression applied, use the 32 bit IEEE float-point presentation
```

Latitude and longitude are float numbers that are represented in Figure 4.5. For the sake of simplicity, the latitude will be discussed and the same will apply to longitude. Therefore, the sign part, the exponent part and the variable number of bits in the mantissa part can be removed, since they are the same for all contributions.

The contribution is then sent to the cloud, in which there is a table that has every server and its corresponding integral number covered in order to decompress.

The number of bits in the mantissa will differ depending on how large the integer is, as shown in Table 4.9. The number of bits that are removed for compression will be from 10 to 16 (if assumed that the highest integer number is 255).

Table 4.9: The number of bits in the mantissa to compress

Integer numbers	Number of bits to compress in mantissa	Sign + exponent + bits in mantissa to compress
2, 3	1 bit	$1+8+1=10$
4-11	2 bits	$1+8+2=11$
8-15	3 bits	$1+8+3=12$
16-31	4 bits	$1+8+4=13$
32-63	5 bits	$1+8+5=14$
64-127	6 bits	$1+8+6=15$
128-255	7 bits	$1+8+7=16$

This means that the higher the integer the more bits to compress. The reason for a missing bit is because of the IEEE float number presentation.

For example, the number 13 is, in binary, 1101 but after normalization, the most significant bit is hidden and only 3 bits from the mantissa are eligible for the compression method.

- Case 2 – If the server covers an area that has two different integral parts (Algorithm 2).

  The same procedure as that for Case 1 will be applied, but with a minor difference: the server needs to add 1 bit (called a decision bit) at the beginning of the compressed value in order for the cloud to determine which integral part is considered for decompression. For example, if the server always receives GPS coordinates that can have two possible integer parts, x and y, the server will assign 0 for the integer x and 1 for integer y and add it as the most significant bit in order for the cloud to decide which integer is needed. Thus, this case is different only in the decision bit that is added as the most significant bit (see Algorithm 2).

Algorithm 2: GPS coordinate compression (Case 2)

```
Input: x.y (x is the integer part and y is the decimal part)
Output: Result (the compressed number in binary form)
if x == num1 then                                   //num1 is the first integer
    Result = $(0\,y_0 \ldots y_n)_b$                  //covered by the server
else if x== num2 then                               //num2 is the second integer
    Result = $(1\,y_0 \ldots y_n)_b$                  //covered by the server
else
    No compression applied, use the 32 bit IEEE float-point presentation
```

4.5.2.2 Accelerometer Data Compression

During a single event, an accelerometer will return data for three coordinate axes (x, y and z). These data values are single-precision floating-point numbers. In this section, a compression method is proposed that will reduce the size of these float numbers by attempting to remove some bits that can be recovered easily later in the cloud.

The integral number is considered (the integer part of the float number) to be from 0 to 39. This is based on the experiment presented in section 4.5.4 and the android accelerometer `Sensor.getMaximumRange() API` (i.e. android hardware restrictions), in which 39 was the highest integer in all axes. However, this method will also work with any number higher than 39. The sign, exponent and mantissa of the 32-bit floating-point numbers are treated as follows (Algorithm 3):

1. The sign bit is left unchanged in this method, in which 0 corresponds to a positive number and 1 corresponds to a negative number.

2. The exponent is mainly 8 bits. If the exponent is negative, the number is left uncompressed since there is no way to predict the numbers after the float point. On the other hand, if the exponent is positive, then the number of bits in the exponent is decreased to 3 bits. These 3 bits represent the number of bits moved after the float point during the normalization process. For example, in number 13.1857, the mantissa part will look like this before normalization: 1101.0011. After normalization, the number will look like this: 1.1010011 X 2^3, whereby the point is moved to the most significant 1. Since the number of bits moved after the point is three, then the exponent will only represent number 3. In the integer part range $(1 - 39)$, the exponent range is from 0 to 5. Therefore, 3 bits are reasonable to present the exponent part and there is no need to add 127 (single-precision bias). The bias addition will take place later in the cloud in the decompression process. Table 4.10 shows the numbers and their corresponding exponent values that will be stored in the exponent part.

Table 4.10: Exponent values for every integer number

Number	Exponent
0	Negative exponents
1	0 $(000)_b$
2 and 3	1$(001)_b$
4-7	2$(010)_b$
8-15	3$(011)_b$
16-31	4$(100)_b$
32-39	5$(101)_b$

3. In the mantissa, the exponent is used to categorize the integers and decrease the number of bits that represent the integer number $(1 - 39)$. The categorization of the integers is presented in Table 4.11. The reduction of the number of bits will take place in two steps:

   3.1) The most significant bit for all integers under the exponents from 1 to 5 is always 1. This bit is going to be removed all the time (the hidden bit in IEEE floating-point). Thus, all the numbers with an exponent from 1 to 5 have one bit removed.

Table 4.11: Integer categories

Exponent	Integer	Binary
0	1	1
1	2 and 3	10 and 11
2	4-7	100-111
3	8-15	1000-1111
4	16-31	10000-11111
5	32-39	100000-100111

3.2) After removing the most significant bit “1” from all the numbers, the exponents 1 and 2 will have 1 bit and 2 bits, respectively, to represent their integer numbers (Table 4.11). Therefore, these integers are left unchanged. On the other hand, the integers covered by the exponents 3, 4 and 5 can be further altered. Starting from numbers covered by exponent 5 $(32-39)$, the new most significant bit after removing “1” will always be “0”. Thus, this bit can also be removed and the numbers from 32 to 39 are represented only in 4 bits. Numbers covered by the exponent 3 are slightly different. The new most significant bit (after removing 1) can be either 0 or 1. Therefore, there need to be some indication if this bit needs to be removed. Since the exponent is represented in 3 bits and only exponents from 0 to 5 remain, the exponent 6 can be used to help indicating the bit in the decompression. The exponents 3 and 6 are used in the compression process for the integer values from 8 to 15. Exponent 3 indicates that the removal of “0”, while exponent 6 indicates the removal of “1”. Therefore, numbers from 8 to 15 can be represented in only 2 bits. The same will happen to numbers from 16 to 31, for which exponent 4 is used to indicate that the removal of “0” and exponent 7 to indicate the removal of the bit “1”. Thereafter, the numbers from 16 to 31 are represented by only 3 bits. For an illustration, see Table 4.12.

4.5.3 Decompression

In this section, the decompression processes for both location-based and Accelerometer data are presented. The decompression process will take place in

Table 4.12: Number of bits after steps 3.1 and 3.2

Number	Exponent	Number of bits after step 3.1	Number of bits after step 3.2
2-3	$1(001)_b$	1 bit	1 bit
4-7	$2(010)_b$	2 bits	2 bits
8-15	$3(011)_b$ and $6(110)_b$	3 bits	2 bits
16-31	$4(100)_b$ and $7(111)_b$	4 bits	3 bits
32-39	$5(101)_b$	5 bits	4 bits

the cloud and it will re-construct the compressed single-precision floating point numbers in a cost-less and easy way as shown below:

Algorithm 3: Accelerometer Data Compression

```
Input: x.y (x is the integer part and y is the decimal part)
Output: Result (the compressed number in binary form)
if x == 1 then
    Exponent = 000
    Mantissa=$(y_0 \dots y_n)_b$
else if x== 2 OR x==3 then
    Exponent = 001
    Mantissa=$(x_1)_b(y_0 \dots y_n)_b$                      //$x_0$ is removed
else if x >=4 AND x<= 7 then
    Exponent = 010
    Mantissa=$(x_1x_2)_b(y_0 \dots y_n)_b$                   //$x_0$ is removed
else if x >=8 AND x<= 11 then
    Exponent = 011
    Mantissa=$(x_2x_3)_b(y_0 \dots y_n)_b$                   //$x_0$ and $x_1$ are removed
else if x >=12 AND x<= 15 then
    Exponent = 110
    Mantissa=$(x_2x_3)_b(y_0 \dots y_n)_b$                   //$x_0$ and $x_1$ are removed
else if x >=16 AND x<= 23 then
    Exponent = 100
    Mantissa=$(x_2x_3x_4)_b(y_0 \dots y_n)_b$                //$x_0$ and $x_1$ are removed
else if x >=24 AND x<= 31 then
    Exponent = 111
    Mantissa=$(x_2x_3x_4)_b(y_0 \dots y_n)_b$                //$x_0$ and $x_1$ are removed
else if x >=32 AND x<= 39 then
    Exponent = 101
    Mantissa=$(x_2x_3x_4x_5)_b(y_0 \dots y_n)_b$             //$x_0$ and $x_1$ are removed
    no compression applied, use the 32 bit IEEE float-point presentation
    Exponent= $(c_0c_1c_2c_3c_4c_5c_6c_7)_b$
    Mantissa= $(x_1 \dots x_n)_b(y_0 \dots yn)_b$
End if
Result=(Signbit)(Exponent)(Mantissa)
```

4.5.3.1 Location-based Decompression

The cloud will have a table that contains the ID of every public local server around the city and the integer part(s) for both the latitude and longitude that are covered. Therefore, when the compressed positive latitude and longitude values are received to the cloud and attached with the local server ID (in the Http request), the cloud will look up the ID in the table and re-construct the latitude and longitude. The re-construction process is by returning the removed bits that corresponds to the removed integer part.

Negative values are not compressed by algorithm 3, they are received by the cloud as they are in which no processing needed.

4.5.3.2 Accelerometer Decompression

The cloud will receive the compressed accelerometer readings in a binary format then it will use algorithm 4 to reconstruct the removed bits.

Algorithm 4: Accelerometer Data Decompression

```
Input: (Sign)(Exponent) (Mantissa) (The output of algorithm 3)
Output: Result (IEEE single precision in binary format (32 bits))
if Exponent = 000 then
    Mantissa=$(1\,y_0 \ldots y_n)_b$
end if
if Exponent = 001 then
    Mantissa=$(1x_1)_b(y_0 \ldots y_n)_b$                    //$x_0$ is added
end if
if Exponent = 010 then
    Mantissa=$(1x_1x_2)_b(y_0 \ldots y_n)_b$                 //$x_0$ is added
end if
if Exponent = 011 then
    Mantissa=$(10x_2x_3)_b(y_0 \ldots y_n)_b$                //$x_0$ and $x_1$ are added
end if
if Exponent = 110 then
    Mantissa=$(11x_2x_3)_b(y_0 \ldots y_n)_b$                //$x_0$ and $x_1$ are added
end if
if Exponent = 100 then
    Mantissa=$(10x_2x_3x_4)_b(y_0 \ldots y_n)_b$             //$x_0$ and $x_1$ are added
end if
if Exponent = 111 then
    Mantissa=$(11x_2x_3x_4)_b(y_0 \ldots y_n)_b$             //$x_0$ and $x_1$ are added
end if
if Exponent = 101 then
    Mantissa=$(10x_2x_3x_4x_5)_b(y_0 \ldots y_n)_b$          //$x_0$ and $x_1$ are added
end if
Result=$(\text{Signbit})_2((\text{Exponent})_{10} + 127)_2(\text{Mantissa})_2$
```

In algorithm 4 and after returning the removed bit(s), the Sign bit, Exponent and Mantissa are put together but first some work need to be done on the

Exponent to be compatible with IEEE single- precision floating point format. First, the bias (127) is added to the Exponent decimal value then the resulted value is converted to a binary form (8 bits).

4.5.4 Evaluation

In the evaluation, the effectiveness of the compression method is assessed by first comparing the size of the data before and after performing the compression for both GPS coordinates and accelerometer readings. Then, the method is compared with some of the existing compression algorithms.

4.5.4.1 Evaluation setup

The compression ratio described above was first evaluated with a data set that contains GPS and accelerometer readings (i.e. both GPS coordinates (latitude and longitude) and three axes (x, y and z) of accelerometer readings). Then, there are five entries (latitude, longitude, x, y and z) for every sensing contribution. These entries are represented as a single-precision floating point and this is how it is received by the server. The data set is obtained by an Android app developed for this thesis (the details are in appendix A). In the evaluation, there are the cloud and one local server that received the crowd-sensed data and started performing on compression on the set. The data set consisted of 2,000 readings and was about 1.2 MB (even though the data set was not large, the intention is to examine the compression method on real crowed-sensed data): the GPS coordinates data were 0.51 MB and the accelerometer data was 0.72 MB.

4.5.4.2 Evaluation Result

The proposed compression method is implemented and ran on the data set. The location-based data reduction (step 1) was performed on the GPS coordinates and the accelerometer data reduction was applied to the accelerometer axes. The data set size was reduced to 0.82 MB, which means the method removed up to 33% of the data.

4.5.4.3 Comparison

The proposed compression method is compared in terms of compression performance with one general-purpose compression algorithm (zlib) [200] and one floating-point data compression algorithm (Szip) [194]. Zlib offers general-purpose lossless data compression and is an abstraction of the deflate algorithm. Szip is a predictive compression algorithm based on the extended-Rice algorithm that uses Golomb-Rice codes for entropy coding. Table 4.13 demonstrates that the proposed approach is more effective, as the data size after

Table 4.13: Comparison of the data size after compression

Compression algorithm	Data size (MB)
zlib	0.98
Szip	0.91
Presented approach	0.82

compression appears to be the lowest. Furthermore, the comparison is performed on the GPS coordinates and accelerometer readings separately. Table 4.14 shows the data size after performing each compression algorithm. The compression approach in this paper and Szip have a similar GPS data size after compression due to the high similarity of the GPS data received on one server. However, the approach proposed in this thesis shows a better result than the other two algorithms for the accelerometer readings, where the data are random. The proposed approach deals with every data entry as a separate entry and does not use the history of the previous entries to predict the next value. Therefore, this method shows its effectiveness for similar and random values.

Table 4.14: Detailed data size after compression

Compression algorithm	GPS data size (MB)	Accelerometer data size (MB)
zlib	0.292	0.691
Szip	0.265	0.645
Presented approach	0.261	0.562

4.6 Chapter Summary

This chapter presented the different middleware services in the smart city environment, where this middleware is in the proximity of the crowd. The middleware is offered as public local servers that are located next to the crowd and offered by the city to improve the quality of life for citizens. Theses services are:

1. Trust Service that calculates the trust for every contribution received using four factors, then if this contribution is trusted, it will be moved to the scheduler service but the trust value and the user ID will be sent to the cloud. If the contribution is not trusted, it will be discarded but the trust value will be recalculated and sent to the cloud along with the user ID.
2. The Scheduler will schedule sending the data to the cloud depending on their priority and it differs depending on the application in use. Furthermore, it removes similar contributions from the same location. Before data are scheduled to the cloud, they are sent to the reduction unit first.
3. The Reduction Unit that contains a compression unit for single precision floating point i.e. location-based numbers and accelerometer numbers.

Even though the reduction in size is technically occurred in the Reduction Unit (section 4.5), all of the other services contain reductions in data in their own way. In more details, in the Trust Service, the untrusted data are discarded and not sent to the cloud. If the Scheduler, in a specific application, receives a number of contributions for the same location, it will just pick one contribution for that location instead of sending all of these similar data to the cloud. The Scheduler will choose this contribution depending on the highest reputation values for the user contributed.

The following chapter will present the data partitioning approach and the services provided by the cloud. The data partitioning approach will logically partition the data depending on a number of parameters. The two services, then, will apply to the data depending on the result of the data partitioning approach. Chapter 6 will present a use case and experimental work for the whole services existed in this thesis.

Section 4.2 and all subsections were published in [7]. Section 4.3 and all subsections were published in [6]. Section 4.5 and all subsections (except section 4.5.3) were published in [5].

Chapter 5

Data Management in the Cloud: Cloud Services

5.1 Introduction

In recent years, the substantial increase in mobile device capabilities has led to the introduction of the crowd-sensing paradigm. Mobile crowd-sensing used Mobile cloud computing (MCC) technology in different application including Smart City applications. However, due to the limited resources of mobile devices, sensed data are usually offloaded and processed in data centres – clouds. Clouds receive a large amount of sensed data from mobile phones and other sensors to serve these smart city applications.

However, the increasing volume of crowd-sensing data in the cloud raises two challenges. The first is that sending a large amount of data to the cloud requires high bandwidth and consumes a lot of energy. Second is managing and storing these data efficiently in clouds, then optimizing data without losing important features and values.

To overcome the first challenge, middleware services are presented in chapter 4 that resides in public local servers and showed the importance of local processing in a smart city context. Furthermore, the previous chapter showed how the amount of data is reduced effectively on the edge and as close to the crowd devices as possible. With these middleware services, data sent to the cloud are reduced in numbers and size and, therefore, the amount of traffic and transmission cost are reduced as well.

Regarding the second challenge, clouds offer storage and computing resources for many applications, such as smart city applications. However, this new model needs to reconsider database management principles in order to provide scalable and efficient storage that takes into account the pay-as-you-use model. Storage optimization has been studied for years in databases using methods such as compression [147] [117] [77] [190] [46] [135]. The more data stored in the cloud, the higher the cost. Therefore, to overcome the second challenge, a relational database storage management in the cloud is proposed in this chapter using, first, a proposed method to partition the databases into two logical parts and, second, a set of proposed reduction services. The partitioning method is performed using a number of parameters defined by the user such as time, access rate etc. With the proposed partitioning method, one can identify the level of importance and sensitivity of the data entries in a specific database and this will help the user when applying different operations such as data reduction operations.

The partitioning method that is located in the cloud is presented in this chapter as well as the cloud reduction services i.e. data optimization and context extraction.

5.2 Partitioning Method

The partitioning method is performed on smart city relational databases located in the cloud in order to efficiently reduce the cost and the amount of data stored. This method highlights the importance and sensitivity of data chunks in order to perform reduction services. Using this method, users can perform reduction services on unnecessary data, which will improve storage and reduce costs without losing any important data.

The goal of using this method is to reduce the amount of stored data without loosing important or sensitive data. Upon partitioned data, new services such as optimization and context extraction are offered; these services will run in the cloud and make use of the partitioning method to reduce data stored in the cloud efficiently and therefore will decrease the cost. These services apply to data chunks that are not as important as others, where a data chunk is any number of data entries that is part of a database; for example, the first 1000 data entries in a weather database can be considered as a data chunk.

Data Storage in the cloud

Smart city data in the cloud are stored into relational databases and they consist of all types of crowd-sensed data (Longitude, Latitude, Time, Accelerometer readings, etc.) obtained from different smart city applications such as Weather, Environment etc. For simplicity, throughout this chapter, an assumption is considered that every database contains only one table, e.g., water database contains only one table. Sensitive data refer to several things depending on the database. Sensitive data defined as data that contain important values, personal information, cover critical areas or was captured at a critical time. These features vary between different databases. For example, within the weather database, data entries corresponding to days with flooding in city "X" are considered sensitive data. Immutable data are defined as data that are considered sensitive, newly added to the database or frequently accessed by users. Mutable data are data that are flexible to change, contain no sensitive data and not frequently accessed. All of these features are shown in details in section 5.2.2.

Every database "Y" maps to a log file and a context database. The log file keeps track of the number of accesses to different data chunks in "Y" (see section 5.4). The context database contains the knowledge-extracted from database "Y", such as the circumstances around a particular event. Figure 5.1 presents the proposed architecture, which works as follows:

1. The user enters a specific time period (such as 1/1/2010 to 31/12/2010) and specifies a smart city database (for simplicity, the assumption is that every smart city relational database contains only one table) to check the sensitivity of the data in which he/she wants to apply the reduction services on and sends it to the cloud using the interface.

2. In the cloud, data are checked as to whether they are immutable or mutable, according to algorithm 7. The user then will receive the response about the data.

3. The user will now have a clear idea about what services need to be applied on the data requested (see section 5.4.1 for services and when to apply them) and then can request these service(s). Depending on the decision in step 2, the user can choose to either optimize data entries in a specific database or extract context from this database. The context data are saved into separate context tables where every smart city database maps to one context database (i.e. table).

4. After the service(s) execution is complete, the cloud acknowledges the user.

Another important feature of this architecture is the **schedule** service, which will allow the user to schedule the periodic communication of either raw or context data to different users (i.e. consumers).

Motivational Example

Suppose the city council of city “X” collects air pollution data around the city using mobile phones and vehicles, and data collected are stored in a cloud. After a period of time (e.g. five years), the city council (IT department) realizes the need to optimize the air pollution database in terms of cloud storage, since it contains a huge number of entries that include unnecessary data collected during the previous five years. The council then needs to find a way to reduce the cost of storing the air pollution data and increase data storage efficiency. However, the air pollution database (i.e. table) contains data entries that are considered important since they might contain singularities or were captured during an important time frame.

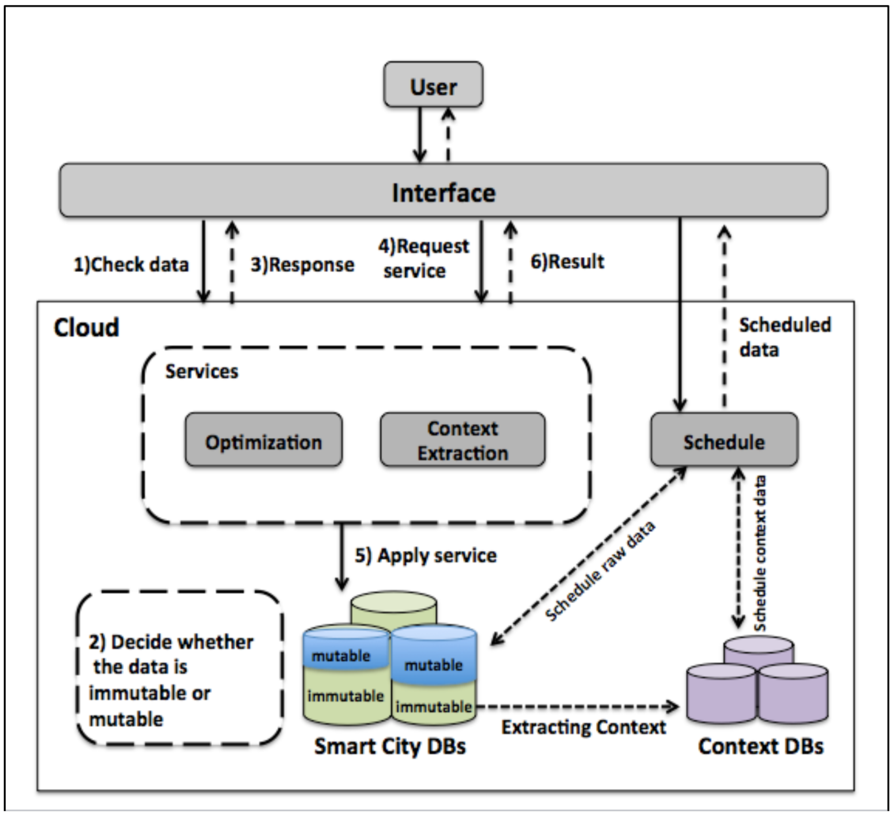


Figure 5.1: System Architecture

Therefore, the council needs to be careful when optimizing the storage to avoid removing any important data from the database. To overcome this issue, it agrees to choose a number of parameters and use them as a basis for distinguishing the important from the less important data. Section 5.4 introduces the partitioning criteria.

5.3 Previous Studies

In cloud environments and with the increasing volume of data sent to the cloud, data need to be managed efficiently in order to reduce cost and save storage. Most of the research published to date proposed to manage data in the cloud either by virtual machine (VM) utilization, or replication and parallel processing.

In [186], authors propose a data replication approach to reduce the cost of data storage and transfer for workflow applications. They categorize dataset types into three groups, fixed dataset, free-flexible dataset and constrained-flexible dataset, to build a connection between datasets and each data centre. They also developed a data replication algorithm that takes place in a build-time phase with distinct levels of data dependency, size of the data, access rate, and storage capacities of the data centres. If dataset “d1” is located in data centre “X”, but there is a task “t1” located in data centre “Y” where this task will frequently invoke “d1”, then replicating “d1” in data centre “Y” will reduce the total cost C where C = data storage cost for “d1” + data transfer cost for “d1”. Their work is similar to the work in this chapter in terms of partitioning but is different with regard to what is partitioned and why. They partitioned whole datasets in order to decide what to replicate in distributed servers, but in this chapter, the partitioning is applied for every database in order to optimize the storage in the current cloud.

Furthermore, in replication, a DARE (distributed adaptive data replication) algorithm, created by a scheduler for improved data locality measures is presented in [1]. The model uses an adaptive data replication technique that produces additional replicas for popular data and minimizes replicas for unpopular data. In dynamic data replicas, their work uses a replication plan to save extra storage and reduce overheads.

Authors of [153] define a smart city framework for data management using a fog to cloud architecture. They propose a data acquisition block where data aggregation is performed, such as removing redundant data, and use existing data compression techniques (e.g. zip format).

In [35], authors present a Software-Defined Networking (SDN-based) framework for Virtual Machine (VM) management. Their framework uses time-based network information to transfer VMs. With their framework, the communication cost is reduced and the throughput is increased since their system transferred VMs that are around 50%. Their work concentrates mainly on communication cost reduction, however, not storage reduction. Another work, presented in [99], focuses on the processing of smart meter data streams in a cloud integrated framework. The scenario states that, as the hardware utilization is increased, it could accommodate more virtual machines for real-time pricing applications. The analysis demonstrates the suitability of the cloud environment for frequently updating data streams.

In [17], the authors proposed a cloud data management system using Hadoop that analyses sensor data requirements and overcomes the limits in the traditional relational database management system by providing parallel storage and processing in the cloud. Authors proved how capable is their framework in data management tasks in cloud systems. In [24] and [23], authors propose a hierarchical Hadoop framework (H2F). This framework considers the heterogeneity of nodes and the geographic data distribution in order to overcome limitations in the original Hadoop job-scheduling algorithm.

Furthermore, a distributed storage system for high data access concurrency is proposed in [116]. Authors use a Hadoop MapReduce framework and a set of versioning algorithms. Another work is UniHadoop [18], which combines a grid middleware tool called Unicore with the Hadoop Distributed File System (HDFS) to improve data storage in terms of duration, elasticity and disaster recovery. All the previous works utilize parallel storage to manage the data, which is unlike the proposed work in this chapter.

Finally, in [145], authors propose to automate data management in cloud computing by using a storage selection system. This system chooses between different storage providers (i.e. Amazon, Azure and local clouds) based on user requirements and general features of the storage providers, such as cost or performance. Their work is interesting as it deals with data before storing them in the cloud. On the other hand, the partition method proposed examines different parameters of the data after they are stored in the cloud. The design of the partitioning method is presented in section 5.4.

5.4 Design

Data volumes increase at a high speed in smart city applications as well as in scientific applications. The challenges of storing these data and the costs are also increased. Therefore, in this chapter, the need to optimize data storage has led to a partitioning method, which will help users to apply data reduction services, such as optimization to data chunks that are not particularly important. This is advantageous, as the data storage in use is reduced without losing important data and the cost also decreases.

Using a relational database model, first, the data are partitioned depending on variable parameters that can be defined by the user, the application or both. These parameters logically partition the database into immutable and mutable part. The reason for these logical partitions is to have a clear vision of what the limit of data reduction is when applying reduction services to the data, since every smart city database (as well as the databases of other domains) contains important and often sensitive entries at specific times. Therefore, by having these immutable and mutable parts, the proposed services, such as context extraction, can be applied to the mutable part and not to the immutable one.

In this work, the mutable part can be changed, optimized or sometimes deleted if needed, whereas the immutable part cannot be changed (for a determined period of time, at least) unless there is a need to remove redundancy.

Data entries for which the parameters are examined throughout this chapter are called "$d_{i,y}$", which indicates data entries that start from entry "i" to entry "y". In the smart city domain in use, the partition parameters are defined as: time (T), access rate (AR), and singularities.

A. Time

Time simply represents the time stamp of the crowdsensed data entries in a specific database. Specifically, the time stamp is needed for the first and last entries of "$d_{i,y}$".

$T_i =$ time stamp for entry "i", which is the first entry in the data part "$d_{i,y}$".

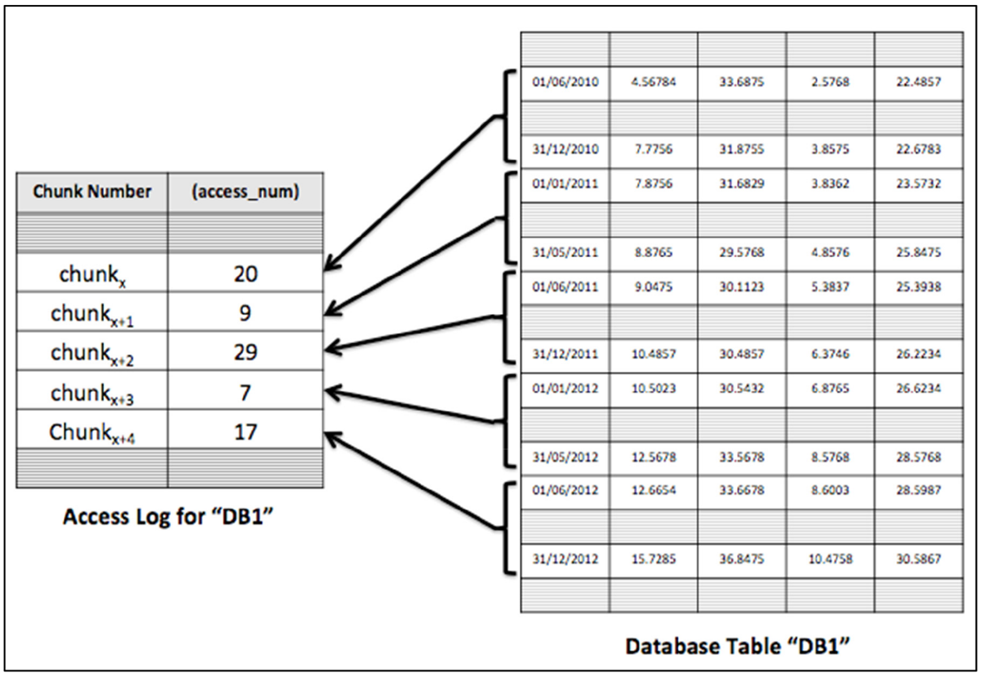
$T_y =$ time stamp for entry "y", which is the last entry in the data part "$d_{i,y}$".

B. Access Rate

Access rate shows how frequently data are accessed. The lower the values resulting from equation (5.1), the better, since having a large number of accesses will give small values using equation (5.1):

$$(\mathrm{AR})_{di,y} = (\log_2 (\mathrm{NA})_{di,y})/(\mathrm{NU})_{di,y} \tag{5.1}$$

where $(\mathrm{AR})_{di,y}$ denotes the access rate to a piece of a database called "$d_{i,y}$"; $(\mathrm{NA})_{di,y}$ is the number of accesses registered for "$d_{i,y}$" in a log file that is called "Access Log", and $(\mathrm{NU})_{di,y}$ is the number of different users who accessed "$d_{i,y}$".



Chunk Number	(access_num)
chunk$_x$	20
chunk$_{x+1}$	9
chunk$_{x+2}$	29
chunk$_{x+3}$	7
Chunk$_{x+4}$	17

Access Log for "DB1"

01/06/2010	4.56784	33.6875	2.5768	22.4857
31/12/2010	7.7756	31.8755	3.8575	22.6783
01/01/2011	7.8756	31.6829	3.8362	23.5732
31/05/2011	8.8765	29.5768	4.8576	25.8475
01/06/2011	9.0475	30.1123	5.3837	25.3938
31/12/2011	10.4857	30.4857	6.3746	26.2234
01/01/2012	10.5023	30.5432	6.8765	26.6234
31/05/2012	12.5678	33.5678	8.5768	28.5768
01/06/2012	12.6654	33.6678	8.6003	28.5987
31/12/2012	15.7285	36.8475	10.4758	30.5867

Database Table "DB1"

Figure 5.2: Database "DB1" with the associated Access Log

The logarithm function is a convenient way to handle large numbers. Therefore, it is used in equation (5.1) where the number of accesses (NA) can be a large number as there is no limit for the number of accesses every user can have. For example, there can be 20 users who accessed a particular database for 1000 times. Futhermore, the reason behind dividing the logarithm of NA by NU and not the other way around is to guarantee small values for AR.

B.1) Access Log

Every database is associated with a log file that keeps the number of accesses to every data chunk in this particular database – see Figure 5.2. In the log file, there are two columns, the first one captures the equal-sized data chunks in a particular database while the other one keeps track of the number of accesses to every chunk. For example, A chunk in the Weather database can be considered as data entries in every month, or every six months. The user defines the size of these data chunks. The size of "$d_{i,y}$" can be less, equal or larger than the size of a data chunk. The number of accesses to a specific data part "$d_{i,y}$" can be captured by checking what data chunks it belongs to and then returning the number of accesses registered in the log for these data chunks (see algorithm 5).

Algorithm 5: Number of accesses to a data chunk

```
Input: Access Log with n logical chunks (chunk$_1$, chunk$_2$,.., chunk$_n$) for database "DB1", $d_{i,y}$
(requested data part)
Output: NA$_{di,y}$(the number of accesses to $d_{i,y}$)
Assumptions: n>1, sizeof($d_{i,y}$) <= sizeof(chunk)
//which chunk $d_{i,y}$ belongs to? Start the checking process starting from the
first chunk
z=1
while (z<= n AND (z+1) <=n) do
    if ($d_{i,y} \in$ chunk$_z$ AND $d_{i,y} \in$ chunk$_{z+1}$) then        //$d_{i,y}$ in two consecutive chunks
        NA$_{di,y}$ =access_num (chunk$_z$) + access_num (chunk$_{z+1}$)
        Break                                                       //exit the loop
    end if                                                 //$d_{i,y}$ belongs to only one chunk
    if ($d_{i,y} \in$ chunk$_z$ OR equal_chunk($d_{i,y}$, chunk$_z$)) then
        NA$_{di,y}$=access_num (chunk$_z$)
        Break                                                       //exit the loop
    end if
    z++
end while
if (z==n) then                                                //checking the last chunk
    if ($d_{i,y} \in$ chunk$_z$ OR equal_chunk($d_{i,y}$, chunk$_z$)) then
        NA$_{di,y}$ =access_num (chunk$_z$)
    end if
end if
```

C. Singularities

Some entries in the database contain unique values that need to be identified and cannot be changed or deleted. From this point, it is important to check if the requested data entries "$d_{i,y}$" contain these important values. The singularities (such as minimum, maximum or sensitive values) are defined by the user depending on the smart city domain, such as, weather, traffic or urban planning. Algorithm 6 demonstrates this parameter.

After calculating the access rate and capturing the time stamp for "$d_{i,y}$", it is necessary to check if one or more singularities exist and examine the result using defined thresholds. With these thresholds, one can decide if "$d_{i,y}$" belongs

Algorithm 6: Check for important values (Exist_singular)

```
Input: $d_{i,y}$, Singular1, Singular2
Output: True if it contains singularities and False otherwise
x=i;
while (x<= y) do
    if ($d_x$ contains Singular1 OR $d_x$ contains Singular2) then
        return True
    end if
    x++
end while
return False;
```

to the immutable part of the particular database (SD) or to the mutable part (NSD). If the data time stamp is higher than the time threshold, this means that "$d_{i,y}$" is new data and not considered old. Furthermore, if the value of the access rate is lower than the access rate threshold, this means that "$d_{i,y}$" is an active part and accessed frequently. The data partitioning decisions are illustrated in algorithm 7.

In algorithm 7, different options are checked with the three defined parameters. The options are as follows:

1. Check if data are timely new (i.e. recently added to the database).
2. Check if data were captured for a long time but with a high number of accesses (time and access rate thresholds are defined by the user).
3. Check if data were captured over a long time with a low number of accesses but some important values i.e. singularities exist.

From the algorithm above, it is necessary to outline that a data chunk is considered as a mutable part (NSD) if, according to the defined thresholds, it is outdated, not frequently accessed and does not include any singularities. Otherwise, the data part is considered immutable.

The algorithm above can be applied whenever the user wants to check a specific data part (before applying the reduction services). However, this idea can be extended by applying the algorithm periodically on the data chunks that are defined in the log file. Therefore, algorithm 7 can be applied as follow:

1. Occasionally: the user needs to specify the data part to be checked. In this case, the user will decide, depending on the flexibility of the data, what are the reduction services that can be applied on the data part. **then**

Algorithm 7: Data are immutable or mutable

```
Input: $T_i$ (time stamp for entry i), $T_y$ (time stamp for entry y), AR$_{di,y}$, $\alpha$ (time threshold), $\delta$ (Access rate threshold)
Output: $d_{i,y}$ is immutable or mutable
//1//time is higher than threshold
if $T_i > \alpha$ AND $T_y > \alpha$ then
    $d_{i,y} \in$ SD                                              //immutable
end if
//2//time stamp is less than threshold but access rate is high
if $T_y <= \alpha$ AND AR$_{di,y} < \delta$ then
    $d_{i,y} \in$ SD                                              //immutable
end if
//3// time stamp is less than threshold and access rate is low,
then singularities need to be checked
if $T_y <= \alpha$ AND AR$_{di,y} >= \delta$ then
    if not (exist_singular $(d_{i,y})$) then
        $d_{i,y} \in$ NSD                                         //mutable
    else
        $d_{i,y} \in$ SD                                          //immutable
    end if
end if
//4// if the time stamp of the first entry is lower than the threshold but
the last entry is not, access rate will be examined.
If access rate is high, no need to check for singularities
if $(T_i <= \alpha)$ AND $(T_y > \alpha)$ AND (AR$_{di,y} < \delta$) then
    $d_{i,y} \in$ SD                                              //immutable
end if
//5// access rate is low (check for singularities)
if $(T_i <= \alpha)$ AND $(T_y > \alpha)$ AND (AR$_{di,y} >= \delta$) then
    if not (exist_singular $(d_{i,y})$) then
        $d_{i,y} \in$ NSD                                         //mutable
    else
        $d_{i,y} \in$ SD                                          //immutable
    end if
end if
```

2. Periodically: the algorithm is applied every period of time (specified by the user) on the data chunks that are defined in the log file. Therefore, after defining the data chunks in the log file as immutable or mutable, the optimization service is applied automatically on both the immutable and mutable chunks. On the other hand, the user needs to specify when to apply the context extraction service on themutable chunks (not automatically). The reason for that is to be more cautious when applying the context extraction service since it requires removing the whole mutable data chunk after extracting the context (see section 5.4.1). It is important to note that the classification of the data chunks (in this case) as immutable or mutable is not permanent since it can be changed when the partitioning parameters are changed after a period of time.

Changes made in the databases by the partitioning method and the reduction services are seen by all users who have access to the databases in the cloud, even when the changes are performed automatically by the system.

5.4.1 Services

After deciding in what database part "$d_{i,y}$" is included (i.e immutable or mutable), some services will be presented. These can be carried out on the immutable part, mutable part or both.

1. Data Entry Optimization

   Users can optimize a specific database in a specific period by removing the consecutive rows that have the same value for one or two columns (these are considered key columns as defined by the user). For example, if two or more rows with different time stamps have the same value for one or two key columns, then only the first row is left in the database, while the rest are removed. This service will improve storage, particularly in smart city databases, such as weather databases, since these will have the same values for different columns on different days, such as summer days. This service can be applied to both immutable and mutable partitions. However, if the data requested to be optimized "$d_{i,y}$" are considered immutable, there is a need to check whether they include singularities by using algorithm 6. If "$d_{i,y}$" includes singularities, this optimization service cannot be applied since there is a risk of losing important data, or it can applied with restrictions (e.g. checking every entry if it contains important value or not before applying the optimization service). Changes made by this service are permanent since deleted data can be easily determined. The optimization process is shown in algorithm 8 where only one key column is used. However, if more than one key columns are used, they will simply be checked as well in the "if conditions" in algorithm 8.

2. Context Extraction

   Users will select key columns they actually need (i.e. temperature in the Weather database) then, based on their selection, the context is extracted and inserted into the corresponding context database. Every database (Weather DB, Water DB, etc.) has its own context database. In this service, users will have the choice to delete the original data and keep only the extracted data in the context database, which will save a significant

Algorithm 8: Optimization using one key column

```
Input: DB (Database), key (key column), DB_length (the number of rows in the database.
Output: Optimized DB
while (the current row != DB_length) do
    if (key in the current row == key in the next row) then
        Delete the next row
    end if
    if (key in the current row != key in the next row) then
        Move to the next row
    end if
end while
```

amount of storage. This service can be applied only to mutable partitions of the database. The changes made by this service are semi-permanent. The original data are deleted from the original database but kept for some defined time in the cloud to make sure there are no requests on those deleted data. After this period of time, the data are deleted permanently.

The highest value in the column, the lowest value and average values are some examples of the context information that can be extracted from a database. The context extraction process is a linear search in the database where the extraction process used in this work is shown in algorithm 9.

Algorithm 9: Context Extraction using one key column

```
Input: DB (Database), key (key column), DB_length (the number of rows in the database,
Average= 0, Temp= key in the first row, Highest= key in the first row, Lowest= key in the
first row.
Output: time_frame= DB_timeframe, Average, Smallest, Largest
Move to the next row
while (the current row != DB_length) do
    Temp= Temp + key in current row
    if (key in the current row > Highest) then
        Highest = key in the current row
    end if
    if (key in the current row < Lowest) then
        Lowest = key in the current row
    end if
    Move to the next row
end while
Average = temp/DB_length
```

3. Scheduling

   Users can schedule sending data to themselves in any time range they specify (e.g. the data can be sent every month or every two months). The data sent can be raw data from the original databases or knowledge data from the context databases. This service can be applied to both immutable and mutable partitions.

5.4.2 Use Cases

In this section, the way the partitioning method is performed is presented using Weather database of [180]. Specifically, the partitioning method will be applied on the entries from 1/1/2010 to 31/12/2011. In these use cases, the time threshold is defined as α and it should be above 1/6/2011 and access rate threshold as $\delta = .39$ where the lower the number the better. Furthermore, the log file is presented in table 5.1 where there is an assumption that every three months is considered as a data chunk. For simplicity, the number of different users' accesses to a chunk (NU) is assumed to be equal to the number of accesses (NA) registered in the log file for that chunk. Using table 5.1, there will be four different cases according to algorithm 7:

1. The user wants to check data entries from 1/8/2011 to 31/8/2011:

   First, the time is checked using the time threshold α. The month of August in 2011 is larger than α, then this user entry is considered immutable and thus no need to check any other parameters.

2. The user wants to check data entries from 1/1/2011 to 29/2/2011:

   The time is below the threshold α, then there is a need to check the access rate. Access rate in the months of January and February equals $.314(\log_2(11)/(11)$ and this value is below the δ (this indicates large number of accesses), then this user entry is considered immutable.

3. The user wants to check data entries from 1/10/2010 to 30/11/2010:

   The time is below the threshold α and the access rate is above δ (access rate $= \log_2(2)/2 = .5$). In this situation it is necessary to check if the data entries in this time range contain important values (singularities). If not, the user entry is considered flexible and mutable, otherwise it is immutable.

4. The user wants to check data entries from 1/5/2011 to 30/6/2011:

   The user entry in this case is quite different. The time for some entries is below the time threshold α and other entries are above it. In this case, it is necessary to examine the access rate to decide. The access rate for this user entry is $.375(\log_2(8)/8)$ which is below the access rate threshold δ, then this user entry is considered immutable. If the access rate value is above the threshold δ, then it is important to check for singularities.

Table 5.1: Log File Example

Data Chunk	**Time Range**	**No. of Accesses**
Chunk1	1/1/2010- 31/3/2010	8
Chunk2	1/4/2010- 30/6/2010	7
Chunk3	1/7/2010- 30/9/2010	5
Chunk4	1/10/2010- 31/12/2010	2
Chunk5	1/1/2011- 31/3/2011	11
Chunk6	1/4/2011- 30/6/2011	8
Chunk7	1/7/2011- 30/9/2011	4
Chunk8	1/10/2011- 31/12/2011	9

The time stamps for chunks 2, 3 and 4 are below the time threshold and their access frequencies are above the threshold, but when the singularities are checked, chunk 3 found to have singularities and this makes it an immutable chunk. Therefore, 6 out of 8 chunks are considered immutable using the partitioning parameters (around 75%). Since chunks 2 and 4 have no singularities, these two chunks are the only chunks that are considered as mutable. On these chunks all the reduction services (optimization and context extraction) can be freely performed without worrying about loosing sensitive data. Therefore, if the user decides to apply the context extraction service on these chunks, then around 180 data entries are removed from the Weather database (in the case study 2010 and 2011) and the context of these entries are extracted and inserted into the context database. This means that around 24.65% of the data are removed from the Weather database in 2010 and 2011 using this service. The reduction percentage can further be increased if there are more mutable data chunks in the database. The more mutable data chunks, the more room for data reduction. So, if there is an assumption that chunk 3 contains no singularities, then the percentage after applying the context extraction service is increased to 36.98%.

If the optimization service is applied on the immutable chunks that contain no singularities, there will be further reduction. Chunks 6 and 7 are the only immutable chunks with no singularities. After applying the optimization service and using the temperature as the key column, the followings are achieved: in chunk 6 data is reduced by 15.38% (14 data entries removed from chunk 6), while chunk 7 is reduced by 30.43% (28 data entries removed from chunk 7), since the weather in summer days is almost the same. This means that around 5.75% of data are removed from the Weather database in 2010 and 2011 using the optimization service. With the two reduction services combined, data are reduced by 30.4%. From the percentages above, the effectiveness of the proposed partitioning method is seen and how this method can successfully and easily help in reducing the amount of data without loosing important features.

5.5 Evaluations

For evaluation, the Weather database is deployed using Amazon Relational Database Service [10] offered in Amazon Web Services (AWS) [11]. To be more specific, only the entries from 2010 and 2011 are deployed; the size of this Weather database is 1.7 MB.

Even though there is a lack of similar techniques where they can be compared with the functionality of the proposed approach, compression techniques can be the best fit in terms of data reduction and storage saving. Therefore, the proposed approach is compared with two compression techniques. The first is a general-purpose compression algorithm (zlib) [200]. The second is an SQL supported compression method, called page compression [36], that stores data efficiently in a row and removes redundancy. Table 5.2 shows that the petitioning approach applied with the reduction services outperforms the other two techniques; this is because this approach does not only remove repetitions, it removes mutable entries too while extracting knowledge from the removed data. Note that, storing the removed data for a period of time after applying context extraction is not considered in this evaluation since these data will be deleted permanently.

Table 5.2: Comparing database size after reduction

Reduction Method	Size After Reduction
General-purpose Compression	1.51 MB
Page Compression	1.39 MB
Presented Approach	1.18 MB

5.6 Discussion

Even though the proposed approach accomplished a large percentage in storage savings, this will introduce two new requirements:

1. Users' agreements on reduction:

There will be a tradeoff between users' agreement and reduction. As different users have access to the databases and one of them, let's call him "USER1", applied the reduction services on the Weather database, the others are not aware of that change. In this case, all the users who have access to the database need to confirm their agreement to this reduction. To overcome this issue, a database flag can be developed or notifications are sent to all users asking them to confirm their agreement to the reduction applied by "USER1". If all users confirm the reduction, then it will be applied. Some may not and decide to maintain the original copy of the database and pay for the service accordingly. This issue is solved in details in section 5.7 using a user agreement agent.

2. CPU Load:

   With the new data reduction service, the load on the CPU will increase due to partitioning and reduction services. However, utilizing the cloud with its powerful processors will drop the impact of this penalty. Furthermore, the main goal in this chapter is data reduction and storage saving, and this on the other hand will decrease I/O operations and execute the queries on the databases faster. The storage cost can be determined using equation (5.2) below. For simplicity, the data transfer cost is ignored and only the storage cost is considered.

$$\text{Cost}=\Big(\Big(\text{GB per month} * \text{data size}\Big) * \text{no. of months}\Big) \tag{5.2}$$

The cost of storing the data in the cloud for a year is .0015$. The cost savings is not that different after using the partitioning approach and reduction services since the size of the database is small. However, there can be a big saving in cost with large databases as there is more chance for data to be reduced after applying reduction services. The complete evaluation of data reduction services is presented in chapter 6.

5.7 User Agreement Agent (Notification Agent)

User agent is a notification component that can be located in the cloud or in the consumer's side (i.e. interface). This agent is triggered when a reduction event (i.e. context extraction service) occurs. All the users who have access to the data are registered in the cloud, therefore when one user attempts to apply the reduction services on the data; the user agreement agent will send email notifications to all other users. The main purpose behind this is to make sure that all the users agreed on the reduction attempt and the data removed are not necessary for all of them. With the agent, there are two scenarios:

1. At least one user refused the reduction: then the reduction attempt will be declined and the data will be left unchanged.
2. All the users agreed upon the reduction: then the reduction query is performed and the data are removed from this database. However, to ensure reliability and robustness, the data removed are kept for a period of time e.g. one month before being deleted permanently.

5.8 Chapter Summary

In this chapter, a smart city data management architecture is proposed that takes into account storage optimization in the cloud. Data are partitioned by utilizing user-defined parameters to distinguish sensitive from non-sensitive data in a specific database. First, users will check to which part belongs the data chunk they want to apply reduction services to, in order to avoid losing important data. The partitioning method and the reduction services applied to not-so-important data chunks lead to important database storage savings. A number of services are proposed that can efficiently reduce the amount of smart city data saved in the cloud. The proposed approach is an effective process of reducing the smart city data storage of the cloud without losing important data as shown in the use case earlier. Extended evaluations are conducted in the next chapter.

The following chapter will present two different use cases and their evaluations. The evaluations will be held for the whole services in the local servers and the services in the cloud. Chapter 7 concludes the thesis.

Sections 5.1, 5.2, 5.3, 5.4 (except algorithms in section 5.4.1), 5.5 and 5.6 were published in [3].

Chapter 6

Experimental Evaluations

6.1 Introduction

There are two main sets of purposes behind the work presented in this thesis: the first set involves assessing the smart city data produced from crowd-sensing in terms of truthfulness and usefulness and reducing the amount and size of the data before sending them to the cloud; the second set resides in the cloud, where different users attempt to manage the smart city data located there. In this chapter, evaluations of the performance of both edge services and cloud services are undertaken to understand the effectiveness of the solutions proposed.

Chapter 4 showed the performance of each service located on the edge as a stand-alone unit. In this chapter, the performance of all the services on the edge as a whole unit is presented in section 6.2, as well as the results of receiving and filtering the crowd-sensed data using these services. A use case study is also undertaken in section 6.3, in which different consumers utilize a user-friendly web page to exploit the reduction services in the cloud. Consumers send queries that are applied to the smart city data located in the cloud to show how efficiently these services can manage the data storage. After the evaluations are completed, the chapter outlines if the different edge and cloud services will meet the requirements outlined in chapter 3.

6.2 Use Case 1: Exploiting Edge Services

This section presents an evaluation of the services located on the edge (i.e. the trust manager, local reduction unit and scheduler) which is conducted for all the services as a whole unit. Thus, if the edge services are regarded as a black box, this section presents a use case that shows in detail the form and amount of the crowd-sensed data entered into the black box and the amount and size of the filtered data output.

6.2.1 Experiment Setup

An Android app called “SenseAll” was developed for sensing potholes around a city. This Android app was installed on 14 Android devices (two HTC and 12 Samsung Galaxy devices) in order to start the pothole-sensing activity using Wi-Fi. More details of the Android app can be found in Appendix A.

For the sake of simplicity and completion of the evaluation work, and due to the unavailability of public local servers around the city in the meantime, a Windows desktop is used as a local server and contains the three edge services (i.e. the trust manager, local reduction unit and scheduler). This local server runs a Java platform and is coded using Java Server Pages (JSP) in order to receive the crowd-sensed data and store them using the MySQL database.

The server receives the data sent from every device whenever a Wi-Fi connection is available. The server stores the data in the database and starts the services. For simplicity, the reputation calculation takes place in the server, as it contains a table that has all the reputation values for all the users. The reputation values at the beginning of the evaluation are all zero. The users build a reputation as they start to sense.

The experiment took one week. The users were asked to use their mobile devices every day to sense potholes whenever detected. Users can employ the app easily while walking, running or driving, simply by opening the app (the login process is only performed the first time) and placing the mobile device on a surface or shaking it if a pothole is detected. The user can send a photo as supporting evidence but this is an optional data entry. The data produced by the app contain the time, GPS coordinates (latitude and longitude), three-dimensional accelerometer readings, the status of the user (e.g., at a standstill, walking, etc.) and, sometimes, a photo.

The server will have a predefined time, which is one day, meaning that, every day at 11:59 pm, the filtered data are sent to the cloud. Defining the local server time as one day is not the case in a wide range of smart city applications, in which hundreds or sometimes thousands of people are sensing data during the day. However, in this evaluation, due to the small number of mobile devices, the local server schedules sending the data only once a day in order to understand the benefit of the edge services.

The data are passed to the server by sending an HTTP request and using JSON markup. The trust calculation factors work as follows:

- The "user status" factor is chosen by the user from a drop-down list, where the default value is "driving".
- The "sensing style" factor depends on the variety of crowd-sensed data. In this evaluation, the data contributed can be GPS coordinates, accelerometer readings and photos.
- The "loyalty" factor is added up if the user made previous contributions during the predefined time in this evaluation, which is one day. For example, if the user had two previous contributions during the same day and, later, the server received a third, this factor will have the value 0.1.
- The "similarity" factor checks if any of the previous contributions received had the same GPS coordinates. If yes, this factor will have the value 0.1 for this contribution and all similar contributions; otherwise, it is 0.0.

The trust service performs two different calculations: the first is the calculation of every factor for one contribution; and the other is the calculation of the trust of that contribution using equation (4.3) in chapter 4. The first calculation is performed once the contribution is received, while the second calculation is undertaken at the end of the predefined time. The trust service calculates every factor for the contribution received and saves the values of the factors in a temporary database until the predefined time (one day) is over. This is to ensure the addition of a similarity value for all previous contributions, not only those currently available. Furthermore, postponing the trust calculation (the second type of calculation) until the end of the day eliminates extra computations that occur when the trust is calculated for a specific contribution at a specific time and similar data are detected later and the trust value needs to be updated. Therefore, at the end of the day and after making sure that every contribution has a similarity factor value that it deserves, trust is calculated using equation (4.3). Untrusted data are then discarded and the reputation scores are updated. After that, the scheduler checks all the trusted contributions in the temporary database and searches for similarities using the similarity factor. If similar trusted data are detected, only the contribution with the highest user reputation score is kept, while the others are discarded.

6.2.2 Experiment Results

The results of the experiment are shown on a daily basis below and lead to the final analysis of the amount of data removed and reduced when sending these data to the cloud. Every day, the trust and reputation value calculations are shown, as well as the compression process for both location-based data and accelerometer-based data and, finally, the scheduler process. It is important to note that the trust and reputation calculations undertaken every day appear in a table that corresponds to each day. The tables do not show the flow and order of the trust calculations; they only show the final values for all the factors, as well as the trust and reputation scores that are calculated at the end of the day. Note that the reputation values are all zero before starting the experiment and then the values build up or down depending on the trust values. The number of contributions for every user on each day is shown in Table 6.1. The trust calculations and reputation scores for all users for each day are shown in detail in Appendix B.

1st Day

On the first day, the server received data from all the users in different time frames. The server starts applying the services to the data once they are received in the local database. Therefore, the server started with the data from User 3, since it was the first contribution received. However, the time stamp for when the data are received will not be taken into consideration in this evaluation except for the purpose of analysis. Trust is calculated using the parameters given in chapter 4.

After a trust value is calculated for every user, it is compared with the threshold. If the value is above this threshold, the data are trusted and taken to the scheduler; otherwise, the data are simply discarded but the trust value is still used for building reputation, as shown in Appendix B. Note that the number of contributions received for the first day is 18, as users 6 and 11 had two contributions, user 12 made three contributions and user 9 made none. The majority of the

Table 6.1: Number of contributions for every user on each day

Users	**Day 1**	**Day 2**	**Day 3**	**Day 4**	**Day 5**	**Day 6**	**Day 7**
User1	1	1	1	1	1	1	1
User2	2	1	1	2	1	1	1
User3	1	1	1	0	1	2	1
User4	1	1	1	1	1	1	1
User5	1	2	1	1	1	1	1
User6	2	2	1	2	2	1	1
User7	1	1	1	1	1	1	1
User8	1	1	1	1	1	1	1
User9	0	1	1	1	0	1	1
User10	1	1	1	1	1	1	1
User11	2	2	2	1	2	1	1
User12	3	2	1	2	2	1	1
User13	1	1	1	1	1	1	1
User14	1	1	1	1	1	1	1

users had the same score for “sensing style”, since they sent both location and accelerometer data successfully. However, users 11 and 5 had different scores. User 11’s pothole detection is supported with a photo and that increased the value of the sensing style for that specific contribution. User 5 failed to send accelerometer readings and only the default values of zero are received along with the location. Furthermore, the more a user uses the app to send data, the higher the “loyalty” factor. Therefore, the maximum number of contributions made by a single user on the first day was three and these were performed by user 12. Regarding the “similarity” factor, none of the contributions received on the first day had the same location (GPS coordinates) for potholes received by the time of this contribution; thus, this factor will have the value zero for all contributions.

Three contributions were removed from the trust service as they were considered untrustworthy and did not provide enough evidence for potholes. The trust values for these untrusted contributions were re-calculated using the penalty equation presented in chapter 4, whereby these trust values are transformed into negative exponential numbers. Therefore, the new penalty trust values will ruin the reputation scores gained thus far and, in order for the users to recover their scores, they need to improve their sensing contributions over the following days. Therefore, the trust service makes some reductions in the amount of data received from the first day when the untrusted data are discarded.

The scheduler established if there were similar contributions by checking the “similarity” factor of the 15 trusted contributions. However, on the first day, no similar data were detected. The scheduler sent the trusted data to the local reduction unit, where only the GPS coordinates and accelerometer readings are compressed. The size of the trusted data before they entered the reduction unit was 38.1 KB and, after the compression took place, the size of the data was 30 KB.

2nd Day

On the second day, the server received 18 contributions, as all the users contributed data when potholes were detected. Users 5, 6, 11 and 12 contributed twice, which added a value of 0.05 to their “loyalty” factor. User 5 showed great improvement with the data contributed as both contributions are trusted but, unfortunately, the reputation score for this user is still affected by the untrusted data received on the first day. Therefore, user 5’s reputation value was still negative and the following days will determine if this user’s reputation can be rebuilt or not. The “similarity” factor indicates that there are two contributions in the same location for users 2 and 8, which adds a value of 0.1 for both users since this similarity indicates the truthfulness of the existence of the pothole.

Two contributions were discarded on the second day, from users 7 and 13. Unfortunately, the reputation of user 7 is getting really low, since this user had two consecutive untrusted contributions on the first and second days.

At the end of the day, the trusted data are checked by the scheduler for similarities. The scheduler chooses between two similar pieces of data detected by the trust service. The scheduler chose data from user 2, since this user had a higher reputation value on the second day than user 8. This means that three contributions were discarded: two were discarded from the trust service due to low trust scores and one was discarded from the scheduler due to similarity.

After that, the compression methods were performed on the GPS coordinates and accelerometer readings in the remaining 15 contributions. The size of the trusted data before they entered the reduction unit was 40.4 KB and, after the compression took place, the size of the data was 32.3 KB.

3rd Day

On the third day, the server received 15 contributions and all the users contributed data when potholes were detected. User 11 was the only one who contributed twice on that day and, therefore, this user is eligible for receiving the value 0.5 for the "loyalty" factor that will count for the second contribution.

Only one contribution was discarded on the third day, from user 4. This contribution affected the reputation score for that user and degraded the score from 2.6 to -0.7. User 5 started to recover from the untrusted contribution made on the first day, as three trusted contributions were needed in order for that user to transform the reputation score from a negative to a positive value. However, users 7, 13 and 14 still needed more trusted contributions to recover their reputations. User 7 had the lowest reputation score thus far, due to the two consecutive untrusted contributions that user made on the first and second days.

The "similarity" factor indicated that there were two pairs of similar contributions: the data from users 2 and 8 again and from users 12 and 6. Users 2 and 8 were very likely to have been walking together on both the second and third days, as they had a similar location for the pothole detected and their contributions were captured at the same time. The scheduler chose between these similar data, taking into account the higher reputation score. Therefore, the scheduler kept the data from users 12 and 2; the contributions from users 8 and 6 were removed. This means that three contributions were discarded: one was discarded from the trust service due to a low trust score and the other two were discarded from the scheduler due to similarity.

After that, the GPS coordinates and accelerometer readings for the 12 contributions were compressed in the local reduction unit. The size of the trusted data before they entered the reduction unit was 18.4 KB and, after the compression took place, the size of the data was 12.1 KB.

4th Day

On the fourth day, the server received 16 contributions and all the users (except for user 3) contributed data when potholes were detected. All the data received were considered trusted and no data were discarded. Users 4, 13 and 14 recovered their negative scores and started to build their reputations. User 7 still had a negative score, even with two consecutive trusted contributions on days

3 and 4; again, this was due to two untrusted data contributions on the first and second days. Therefore, this shows how difficult it is to gain a reputation having made consecutive untrusted contributions.

The “similarity” factor indicated that there were two similar contributions from users 2 and 8. The scheduler then chose between these similar data, taking into account the higher reputation score. Therefore, the scheduler kept the data from user 2. This means that only one contribution was discarded and this was done by the scheduler due to similarity. After that, the GPS coordinates and accelerometer readings for the 15 contributions were compressed in the local reduction unit. The size of the trusted data before they entered the reduction unit was 23.3 KB and, after the compression took place, the size of the data was 15.4 KB.

5th Day

On the fifth day, the server received 16 contributions, where all the users contributed data (except for user 9) when potholes were detected. Users 6, 11 and 12 contributed twice on that day and were, therefore, eligible for receiving the loyalty value 0.5 that would count towards the “loyalty” factor with their second contribution.

Only one untrusted contribution was discarded on the fifth day: that from user 13. This contribution affected the reputation score for user 13 and degraded the score from 0.7 to -2.6, returning this user to a negative reputation value. Although user 7’s contribution on day 5 is trusted, the reputation score is still negative.

The “similarity” factor indicated that the contributions from users 2, 3 and 8 were similar, as well as the contributions from user 5 and the second contribution from user 11. This added the similarity value 0.1 to all five contributions. At the end of the day, the scheduler chooses from the similar contributions, taking into account the highest reputation. Therefore, the scheduler chose the contributions from users 2 and 11, as they had the highest reputations of those considered. This means that four contributions were discarded: one was discarded from the trust service due to a low trust score and the other three were discarded by the scheduler due to similarity.

After that, the GPS coordinates and accelerometer readings for the 12 contributions were compressed in the local reduction unit. The size of the trusted data before they entered the reduction unit was 19.6 KB and, after the compression took place, the size of the data was 14.2 KB.

6th Day

On the sixth day, the server received 15 contributions, where all the users contributed data once and user 3 contributed twice; this added the loyalty value 0.5 to the “loyalty” factor for the second contribution. Note that users 11 and 13 received the value 0.35 for the sensing style factor because these two sent photos alongside GPS coordinates and accelerometer readings.

Three untrusted contributions were discarded on that day: those from users 5, 6 and 9. Their reputation scores were affected accordingly.

The “similarity” factor indicated that the contributions from users 3, 8 and 10 were similar. This added the similarity value 0.1 to all three contributions. At the end of the day, the scheduler chooses from these similar contributions, taking the highest reputation into account. Therefore, the scheduler chose the contribution from user 8 and discarded the other two contributions. This means that five contributions were discarded: three were discarded from the trust service due to low trust scores, while the other two were discarded from the scheduler due to similarity.

After that, the GPS coordinates and accelerometer readings for the 10 contributions were compressed in the local reduction unit. The size of the trusted data before they entered the reduction unit was 45.8 KB and, after the compression took place, the size of the data was 41 KB.

7th Day

On the last day, the server received 14 contributions, where all the users contributed only once. Again, user 11 received the value 0.35 in the “sensing style” factor because this user sent a photo alongside the GPS coordinates and accelerometer readings.

Only one untrusted contribution was discarded on that day: that from user 6. User 7 finally overcame the two consecutive untrusted contributions made on the first two days. Therefore, it took five consecutive trusted contributions from user 7 in order to regain a positive value.

The “similarity” factor indicated that the contributions from users 2 and 8 were similar. This added the similarity value 0.1 to these two contributions. At the end of the day, the scheduler will choose from similar contributions, taking the highest reputation into account. Therefore, the scheduler chose the contribution from user 2 and discarded the other contribution. This means that two contributions were discarded: one was discarded from the trust service due to a low trust score and the other was discarded by the scheduler due to similarity.

After that, the GPS coordinates and accelerometer readings for the 12 contributions were compressed in the local reduction unit. The size of the trusted data before they entered the reduction unit was 33.2 KB and, after the compression took place, the size of the data was 27.5 KB.

6.2.3 Edge Services Analysis

In this section, the analysis and evaluation of the experiment conducted in the previous section are presented. This analysis will summarize the goals of the above experiment, which were as follows:

- To evaluate the performance of the trust and scheduler services in terms of reducing the amount of data received on the edge.
- To evaluate the performance of the local reduction service (i.e. compression) in reducing the size of the trusted data before sending them to the cloud.
- To show how the reputation scores are affected by trusted and untrusted data for every user on each day.

The main goal in this thesis is to reduce the amount and size of crowd-sensed data received by the local server to minimize bandwidth utilization and reduce network latency. In terms of data transfer, this architecture uses JSON markup, as it is more lightweight than XML, which is an important factor in terms of reducing the amount of data transferred and network delays.

The data reduction goal in this thesis is achieved at different stages of the experiment above. The first stage is the trust service, which discards untrusted data. The second stage is the scheduler, which removes similar data. The last stage is the local reduction unit, which compresses the trusted data and thus reduces the size of these data. Table 6.2 and Figure 6.1 summarize the first two stages by showing the number of contributions discarded by both the trust and scheduler services. Figure 6.2 shows the total size of the trusted data when received by the local reduction unit and the size of the data after compression on each day of the experiment. Furthermore, Figure 6.3 shows the size of the data once received in the local server before applying any service and the size of the data after applying all the edge services (i.e. trust, scheduling and compression).

Table 6.2: Number of contributions before and after applying the trust and scheduler services

Days	Number of contributions received in the local server	Number of untrusted contributions removed by the trust service	Number of similar contributions removed by the scheduler	Total number of contributions after removing untrusted and similar data
Day 1	18	3	0	15
Day 2	18	2	1	15
Day 3	15	1	2	12
Day 4	16	0	1	15
Day 5	16	1	3	12
Day 6	15	3	2	10
Day 7	14	1	1	12

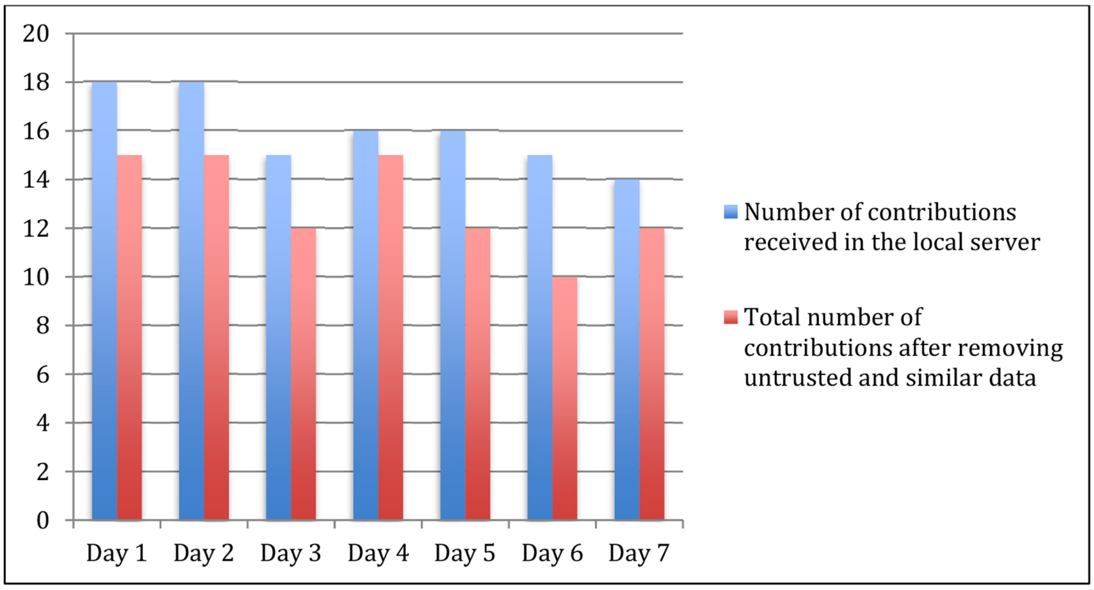


Figure 6.1: Amount of data reduction after the trust service and scheduler are applied

Figure 6.4 shows the reputation score value changes for all 14 users over the seven days of the experiment. User 7 has the lowest value, since this user made two consecutive untrusted contributions on the first and second days.

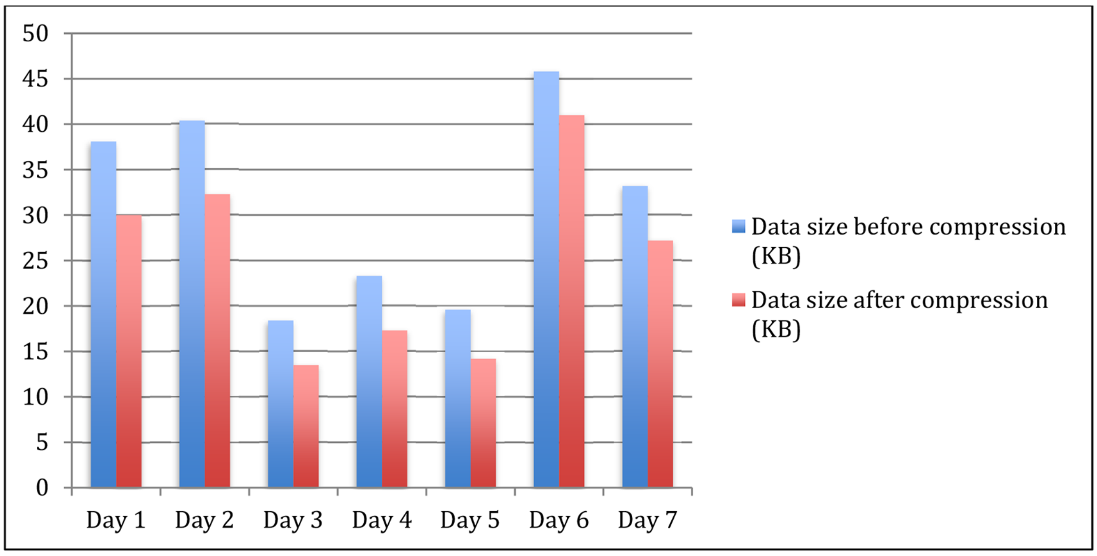


Figure 6.2: Size of trusted non-similar data before and after compression is performed

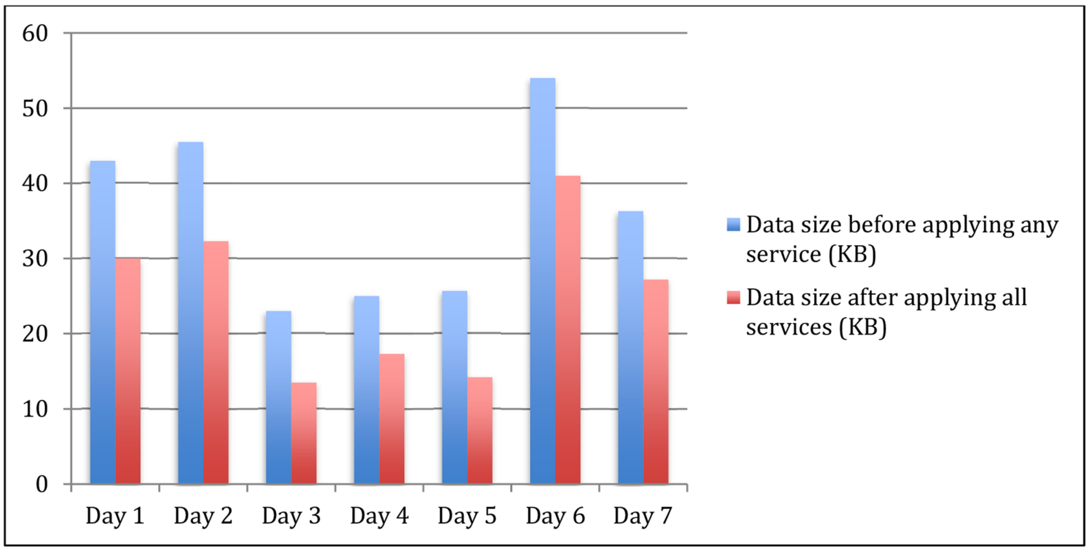


Figure 6.3: Size of the data before performing any service and after applying all the services

The proposed approach shows how difficult it is to regain reputation, as it takes five trusted contributions from user 7 in order for that user to gain a value above zero. User 6 was doing very well in the first five days but this user sent two consecutive untrusted contributions on the sixth and seventh days, which reduced the reputation score for user 6 by around 50% (from 12.5 to 6.2). All the contributions from users 1, 2, 3, 8, 10, 11 and 12 are trusted and this explains the gradual increase in their reputation scores. However, the increase differs from one user to another because some users (e.g., 11 and

12) contributed more than once on some days and used more sensing styles (e.g., photos) than others. This explains their high reputation scores compared with the other users. Furthermore, users 2 and 8 contributed similar data, which were captured in the same place at the same time. Thus, there is a high probability that these two users were walking together on some days and this added the similarity value for the similar contributions they sent.

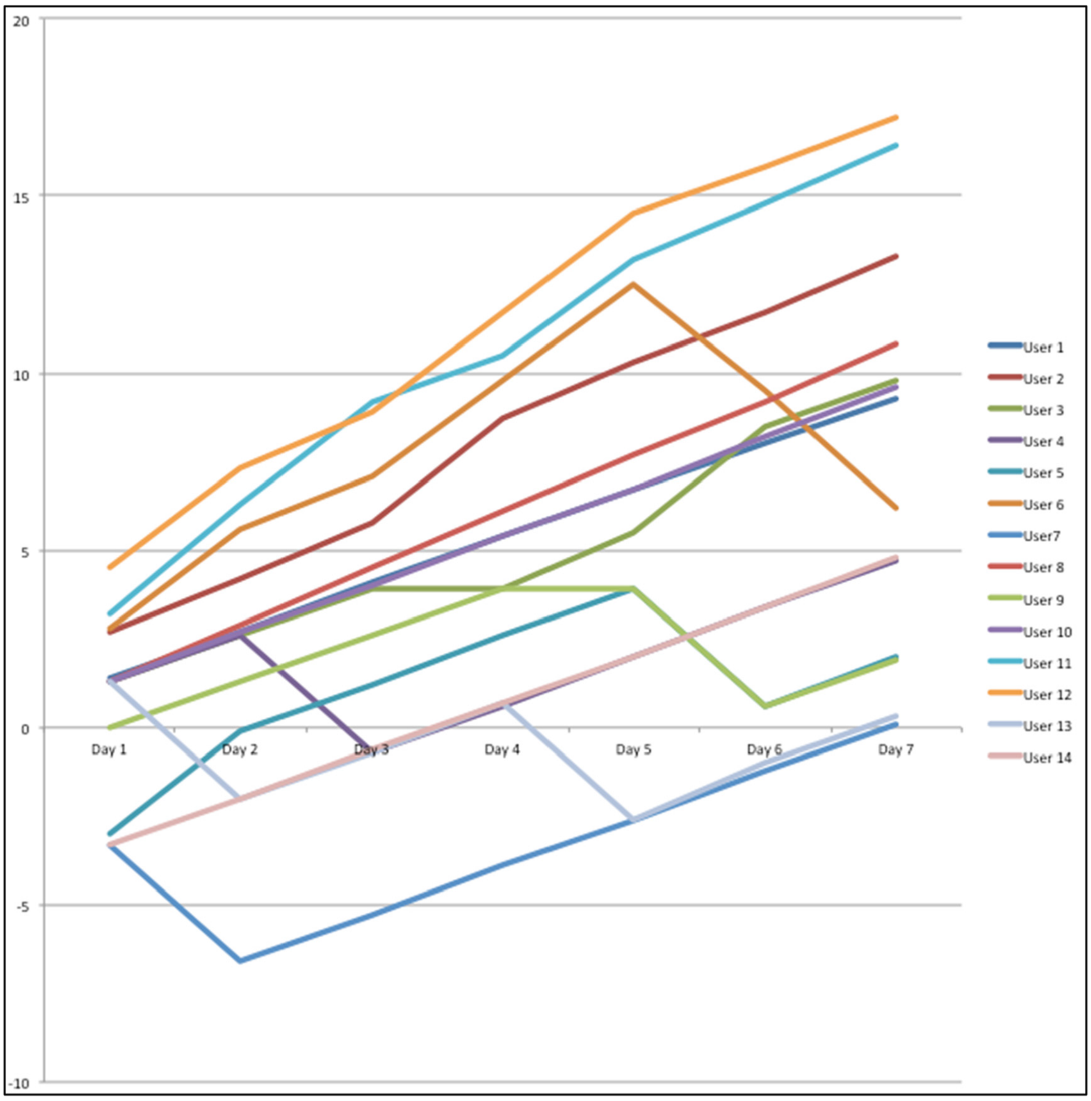


Figure 6.4: Reputation score changes for every user over seven days

Execution time

In order to understand the time delay for the services in completing a task, the execution time needs to be calculated. The execution time for the whole of the edge services is the time from the data being received to the local server and the time the data are sent to the cloud after applying the services. However, the execution time cannot be captured precisely since the services are executed in different time frames. This is because the trust service will calculate trust as soon as the data are received in the server, but the scheduler will not be performed until the end of the predefined time (one day in the experiment above). In this section, the execution time for every service is calculated separately and then the sum of the values will represent the execution time of the edge architecture. However, the time of every service needs to be defined precisely. The trust execution time (trust_runtime) is the average time this service needs to calculate trust for every contribution. The scheduler execution time (scheduler_runtime) is the time this service needs to remove similarities. The compression execution time (compression_runtime) is the time this service needs to compress all the contributions. Therefore, the approximate architecture execution time for one day can be calculated as follows, where the time values for every service correspond to Day 7:

$$\begin{aligned}\text{time} &= \text{trust\_runtime} + \text{scheduler\_runtime} + \text{compression\_runtime}\\ &= (14 * .91\,\text{s}) + 1.14\,\text{s} + 3.7\,\text{s}\\ &= 17.58\,\text{s}\end{aligned}$$

In this evaluation, one day is the predefined time and this time frame must be determined by the city. This time varies (e.g., 1 hour, 12 hours or 1 day) depending on the criticality of the data. If the crowd-sensed data received in the server are not critical (weather, traffic, etc.), the predefined time can be as long as 1 day or so. However, the predefined time might change when the number of contributions exceeds the capacity of the server. In this case, the server will minimize the predefined time value and send the data to the cloud in order to be able to receive new contributions. On the other hand, if the data received are critical (e.g., a poisonous atmosphere or pollution), the time will be as low as 30 minutes or even lower (depending on the population in the place covered by the server).

However, the runtime for some services is proportional to the amount and type of data received. Therefore, time complexity (big O notation) will also be used to describe the time for the algorithms in this architecture, since the fixed time overhead for the predefined time will not be counted.

The worst-case time complexity for the trust service is $O(n)$; this is when there are similar data and some users send data more than once during the predefined time frame. On the other hand, the best-case time complexity for trust service is $O(1)$, when there are no similar data and every user sends data only once during the predefined time frame. For the scheduler, the worst-case time-complexity is $O(n)$ and the best-case is $O(1)$, when there are no similar data detected during the predefined time frame. Therefore, the worst-case time complexity for the services all together is $O(n)$ and the best-case time complexity is $O(1)$.

Memory usage is another performance factor that captures the amount of data stored during the implementation of edge services. However, since the amount of data received in the evaluation above is very low and data are stored only for a period of time (i.e. temporarily), the memory usage in the evaluation above will not affect the performance on the edge.

6.2.4 From the Edge to the Cloud

After the cloud receives the crowd-sensed data, consumers can apply the cloud services (i.e. optimization, context extraction and scheduling) presented in chapter 5 to the data. However, the amount of data collected in the evaluation above is very small. Therefore, the reduction services (optimization and context extraction) will be applied and evaluated in the following section (6.3) using a weather database that contains a large number of data entries to demonstrate the effectiveness of these services.

Consumers can still use the schedule service to schedule sending the data in the cloud to any interested organization. In this evaluation, for example, pothole data can be sent to the municipal authorities regularly in order for them to take the required actions. Therefore, consumers can schedule sending the data from the cloud using the user interface illustrated in section 6.3.1.1. However, since no organization is involved at this stage of the research, the data entries were sent as an attached file to an email address created for evaluation purposes: .

6.3 Use Case 2: Exploiting Data Reduction Services in the Cloud

This section presents an evaluation of the partitioning method that is located in the cloud, as well as the reduction services (optimization and context extraction). This experiment is in the form of consumers who are interested in the smart city data located in the cloud and have access to those data. These consumers can partition the data and perform queries on them.

This section will not only show how the consumers interact with the data stored in the cloud, but also how any attempt to delete the data will be treated by the notification agent. The experiment has two sections: the first section shows different scenarios for partitioning the data (section 6.3.2); while the other demonstrates the output of the cloud services (optimization and context extraction) after partitioning.

6.3.1 Experiment Setup

In this experiment, there is a web interface that is used by consumers to interact with the cloud. This web interface was developed using WordPress, which is written in PHP. The interface is deployed and hosted on the AWS Amazon cloud (where the data are stored) using AWS Elastic Beanstalk and the Amazon Relational Database Service. The queries that are sent from the web page to the cloud are in HTTP format. The data are stored in the cloud using MySQL, which is provided by the Amazon Relational Database Service. The sequence of operations is outlined below:

1. The consumer will first log in.
2. The consumer will send a request to partition the data in a specific time frame and wait for a response.
3. The cloud will process this request and then send the type of data entries requested to be partitioned.
4. Depending on the response, the consumer will choose the desired service (optimization, context extraction and schedule) and send the query to the cloud.
5. If an optimization service is requested, the cloud will implement this query once it has been received.
6. If a context extraction service is requested, the notification agent will be triggered and emails sent to the other registered consumers asking their permission to attempt the deletion that is associated with this service. If only one consumer rejects the attempt, the query will be discarded. If all the consumers agree to the deletion associated with the service, the query will be applied to the data entries desired. These data entries will be

deleted but the context of these data will be saved in a context database. However, the data deleted are kept in the cloud as a backup for a period of time, such as one month, before they are deleted permanently.

6.3.1.1 Consumers Interface Setup

In this experiment, three users are registered in the cloud as consumers. The emails for the three consumers are: , , and . The three consumers are the only users who can use the web interface designed for partitioning and reduction purposes. The web interface contains three pages, as depicted in Figures 6.5, 6.6 and 6.7. The first page is the authentication page, where the consumer en-

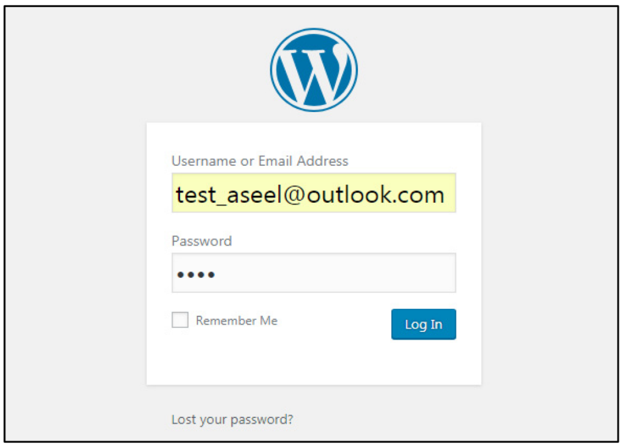
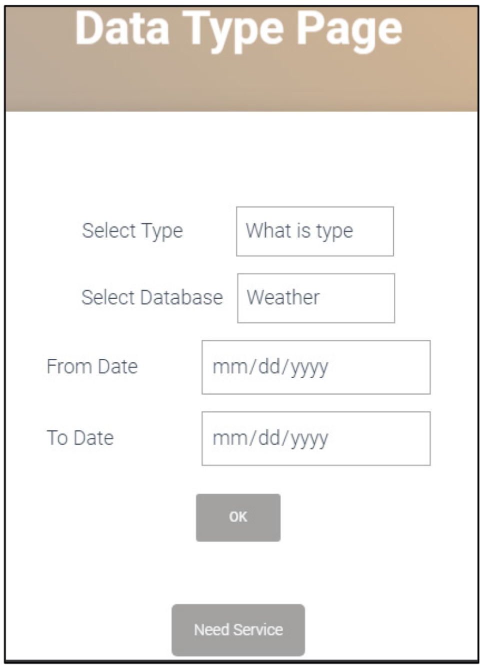

Username or Email Address
test_aseel@outlook.com
Password
••••

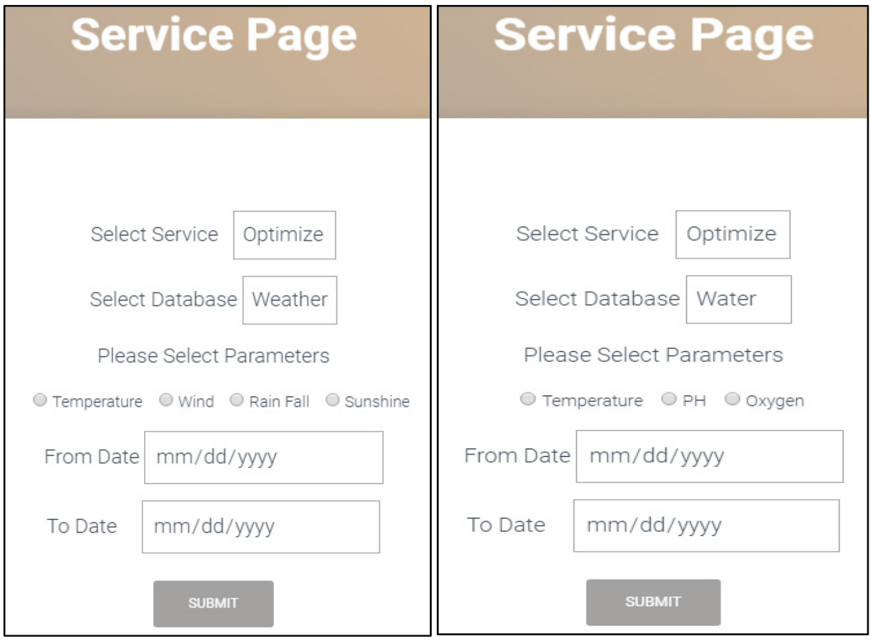
Figure 6.5: Screenshot of the first page (Login page)

ters the email and password that are registered in the cloud. After the consumer is granted access, the second page will appear. The second page contains all the information that the consumer needs for partitioning requests (i.e. database name and time frame) in a user-friendly way (i.e. drop-down lists). This will be an easy way for the consumer to send a well-written query and avoid the difficulty of typing queries. After the consumer chooses the database name and the time frame of the data entries required for partitioning, the consumer can press the “OK” button in order to send the query to the cloud. After performing the query, the cloud will return the result to the consumer. The result will be that the data entries are either immutable or mutable data. Depending on the result of the partition method, the consumer can navigate to the third page in order to perform the services (optimization, context extraction and schedule). The third page is also designed in a user-friendly way (i.e. drop-down lists) to make it easier for the consumer to build the query. In order to avoid network traffic and bandwidth utilization, the notification agent in this experiment is located on the consumer side and is triggered if a context extraction service is requested. This agent will send email notifications to the other two consumers using an SMPT plugin in WordPress and wait for their responses. If both consumers agree to apply the context extraction query, the query will be sent to the cloud. If only one user rejects the query, it will be discarded.



Select Type	What is type
Select Database	Weather
From Date	mm/dd/yyyy
To Date	mm/dd/yyyy

Figure 6.6: Screenshot of the second page (Data Type page)



Select Service	Optimize
Select Database	Weather
Please Select Parameters	Temperature, Wind, Rain Fall, Sunshine
From Date	mm/dd/yyyy
To Date	mm/dd/yyyy

Select Service	Optimize
Select Database	Water
Please Select Parameters	Temperature, PH, Oxygen
From Date	mm/dd/yyyy
To Date	mm/dd/yyyy

Figure 6.7: Screenshots of the third page (Service page). The parameters will change depending on the database.

6.3.1.2 Cloud Data Storage Setup

Weather buoy data in [181] are used in this experiment. These data are captured from February/2001 until April/2018 with 633,066 entries (size = 62.5 MB). These data are stored using the Amazon Relational Database service in a database called “weather”. The reason these data are used instead of real crowd-sensed data is the lack of a satisfying amount of crowd-sensed data that can show the performance of the partitioning method and the cloud services. A context database called “contextWeather” is associated with the “weather” database in order to save the context of the data entries removed from the “weather” database.

There are two managers in the cloud: the partition manager and the execution manager. These managers are simply Java programs that are running on a Java platform. The partitioning manager will receive requests from page two (i.e. the Data Type page) in the consumer interface. This manager will receive the data in an HTTP message, extract the request and start examining the data required. The execution manager is responsible for the request coming from page three (i.e. Service page) in the consumer interface. The execution manager will receive the data, extract the query and apply the query.

6.3.2 Partitioning Scenarios and Output

Before evaluating the partitioning method, the partition factors (time, access rate and singularities) need to be identified. Regarding the Time factor, the time threshold is assumed to be any data entry that has a time stamp that is equal to or later than 1/1/2013, which means that the data that are captured in the last five years are considered new.

For the access rate factor, the cloud will contain a log file table that contains random values for both the number of accesses (NA) to every chunk and the number of distinct users (NU) accessing that chunk (NU values are from 0 to 3 since only three consumers are registered). The partitioning manager will use this log file to capture the NA and NU values in order to calculate the access rate factor. The access rate threshold should be below 1.45 (assuming the highest number of accesses is 20) and larger than 0.49. The number of users accessing the data chunk should not be equal to 0. If it is, the data chunk will apparently be considered as not being accessed frequently.

Table 6.3: Log file in the cloud

Data chunk	Time range	No. of accesses	No. of different users
Chunk 1	6/2/2001- 30/6/2001	1	1
Chunk 2	1/7/2001- 31/12/2001	0	0
Chunk 3	1/1/2002- 30/6/2002	2	2
Chunk 4	1/7/2002- 31/12/2002	5	1
Chunk 5	1/1/2003- 30/6/2003	2	1
Chunk 6	1/7/2003- 31/12/2003	0	0
Chunk 7	1/1/2004- 30/6/2004	7	3
Chunk 8	1/7/2004- 31/12/2004	8	3
Chunk 9	1/1/2005- 30/6/2005	5	2
Chunk 10	1/7/2005- 31/12/2005	0	0
Chunk 11	1/1/2006- 30/6/2006	0	0
Chunk 12	1/7/2006- 31/12/2006	1	1
Chunk 13	1/1/2007- 30/6/2007	2	2
Chunk 14	1/7/2007- 31/12/2007	0	0
Chunk 15	1/1/2008- 30/6/2008	0	0
Chunk 16	1/7/2008- 31/12/2008	5	2
Chunk 17	1/1/2009- 30/6/2009	0	0
Chunk 18	1/7/2009- 31/12/2009	0	0
Chunk 19	1/1/2010- 30/6/2010	0	0
Chunk 20	1/7/2010- 31/12/2010	2	2
Chunk 21	1/1/2011- 30/6/2011	1	1
Chunk 22	1/7/2011- 31/12/2011	2	1
Chunk 23	1/1/2012- 30/6/2012	1	1
Chunk 24	1/7/2012- 31/12/2012	4	2
Chunk 25	1/1/2013- 30/6/2013	1	1
Chunk 26	1/7/2013- 31/12/2013	2	2
Chunk 27	1/1/2014- 30/6/2014	4	2
Chunk 28	1/7/2014- 31/12/2014	1	1
Chunk 29	1/1/2015- 30/6/2015	8	3
Chunk 30	1/7/2015- 31/12/2015	4	3
Chunk 31	1/1/2016- 30/6/2016	3	3
Chunk 32	1/7/2016- 31/12/2016	2	2
Chunk 33	1/1/2017- 30/6/2017	5	2
Chunk 34	1/7/2017- 31/12/2017	4	2
Chunk 35	1/1/2018- 30/6/2018	3	1
Chunk 36	1/7/2018- 31/12/2018	2	2

In the log file, every 6 months is considered a data chunk. The log file was created with assumed values in order to complete the experiment. This log file is defined in Table 6.3.

The Singularities factor in this experiment is only considered for one parameter (i.e. column) in the "weather" database: this column is "airTemperature". Therefore, the values that are below $6°$C or above $17°$C in the "airTemperature" column are assumed to be important (i.e. singularities).

In this experiment, there are five different requests to examine data entries in the "weather" database. These requests are sent by user . All requests' screenshots are appeared in Appendix C. The requests are:

1. Test the entries from 5/5/2015 to 5/7/2015. In this case, the only factor tested is Time. The times for these data entries are above the time threshold and, therefore, these data entries are immutable.

2. Test the entries from 7/7/2004 to 7/10/2004. In this case, two factors are tested: time and access rate. The time stamp for these entries is below the threshold and, therefore, the access rate needs to be tested. Access rate will be calculated as $\log 2(8)/(3) = 1$. The value is below the threshold, which means that the data chunk containing these data entries is frequently accessed. Therefore, the data entries are immutable.

3. Test the entries from 1/7/2009 to 1/10/2009. In this case, all the factors are tested, since the Time is below the time threshold and the number of accesses to the data chunk is equal to zero (which means the consumers had no interest in that chunk). Therefore, the system must search for singularities in order to decide whether these data entries are immutable or not. However, these data entries contain no singularities, which results in these entries being considered as mutable data.

4. Test the entries from 1/4/2001 to 1/8/2001. In this case, two chunks are included and all the factors are tested. This is because the Time for both data chunks is below the time threshold and the access rate equals zero, according to the following calculation: $\log_2(1+0)/(1+0) = \log_2(1)/(1) = 0$. Therefore, the system must search for singularities in order to decide whether these data entries are immutable or not. However, these data entries contain no singularities, which means that these entries are considered as mutable data.

5. Test the entries from 1/10/2012 to 1/2/2013. In this case, two chunks are included and only two factors are tested. Regarding the Time factor, the time for some entries is below the time threshold and the other entries are above it. Therefore, the access rate factor needs to be tested

to decide whether or not the entries are frequently accessed and there is no need to check for singularities. Access rate is calculated as follows: $\log_2(4+1)/(2+1) = \log_2(5)/(3) = 0.77$. The value is below the threshold, which indicates that these data entries are frequently accessed and are, therefore, considered immutable.

6.3.2.1 Partitioning Analysis

The aim of this evaluation is to show how the partitioning method running on cloud infrastructure works with a large database (633,066 entries) in terms of response time and the correctness of the result. The reliability of the architecture is shown in section 6.3.2, in which the factors are tested manually and the answers are compared with the response of the architecture. Each of the five cases tested above has shown the correct and expected output.

The response time can be defined as the time from the consumer sending a request until the response is received from the cloud. The average response time for each case is shown in Table 6.4, where every case has been tested five times to guarantee the same result and avoid any network latency. The average response time for each case differs because some cases require more computations and calculations than others. The table shows that case 1 has the lowest response time compared with the other cases because the only factor tested is “Time”, as the data entries are considered to have been newly added. On the other hand, case 4 has the highest response time, since all the factors are tested and two chunks are included (a 4- month period) and the search for singularities with more data entries will take some time. Table 6.4 shown that, the less parameter tested or entries involved will result in less response time.

Table 6.4: Response times for the tested cases

Cases	**Response time (seconds)**
Case 1	3.17 (lowest)
Case 2	3.51
Case 3	5.27
Case 4	6.18 (highest)
Case 5	3.90

6.3.2.2 Comprehensive Services Evaluation

More evaluations were conducted for the partitioning method. This time, every chunk in the database is tested for whether it is immutable or mutable, in order to understand the amount of unnecessary (i.e. not important) chunks in the database. Therefore, unnecessary chunks can be reduced or even removed under certain conditions (e.g., saving a backup for a period of time).

There are 11 mutable data chunks and the rest are considered immutable, as shown in Table 6.5. In the mutable data chunks, consumers are free to apply the reduction services (context extraction and optimization) without restrictions.

There might be more mutable data chunks if the size of the chunks is reduced. For example, reducing the size of a chunk to three months will give more specific results regarding the importance of that chunk. Chunk 12 is considered immutable simply because it contains an important value (i.e. a singularity). However, if this singularity exists only once in the first three months of that chunk, the second three months would be considered mutable as they contain no singularities, the time is below the threshold and the access rate is above the threshold.

6.3.3 Reduction Services Scenarios and Output

Two scenarios are taken into consideration in this experiment:

1. When the data entries are immutable, only the optimization service can be applied to these entries.
2. When the data entries are mutable, both the optimization service and the context extraction service can be applied to these entries.

However, in the second scenario, applying optimization and context extraction for the same data entries will add more time and unnecessary computations. This is because the context extraction will result in the removal of all the data entries and, therefore, there is no point in optimizing them. Therefore, when data entries are mutable, only the context extraction service will be tested.

For the first scenario, the consumer will optimize the data entries contained in the first partition request (from 5/5/2015 to 5/7/2015) in section 6.3.2. The consumer will use the Service web page to apply the optimization (see Appendix C for screenshots). The data chunk is reduced by around 19% and 1,409 data

Table 6.5: Type of every chunk

Data chunk	Immutable or Mutable
Chunk 1	Mutable
Chunk 2	Mutable
Chunk 3	Immutable
Chunk 4	Immutable
Chunk 5	Immutable
Chunk 6	Mutable
Chunk 7	Immutable
Chunk 8	Immutable
Chunk 9	Immutable
Chunk 10	Mutable
Chunk 11	Mutable
Chunk 12	Immutable
Chunk 13	Immutable
Chunk 14	Mutable
Chunk 15	Mutable
Chunk 16	Immutable
Chunk 17	Mutable
Chunk 18	Mutable
Chunk 19	Mutable
Chunk 20	Immutable
Chunk 21	Immutable
Chunk 22	Immutable
Chunk 23	Mutable
Chunk 24	Immutable
Chunk 25	Immutable
Chunk 26	Immutable
Chunk 27	Immutable
Chunk 28	Immutable
Chunk 29	Immutable
Chunk 30	Immutable
Chunk 31	Immutable
Chunk 32	Immutable
Chunk 33	Immutable
Chunk 34	Immutable
Chunk 35	Immutable
Chunk 36	Immutable

entries are removed. The whole database is reduced by .22% after applying the optimization service to that data chunk.

For the second scenario, the consumer will send a request to apply the context extraction service in the Service web page to the third request (1/7/2009 to 1/10/2009) contained in section 6.3.2 (see Appendix C for screenshots).

Once the consumer () clicks the "submit" button on the Service page, the notification agent will be triggered, since the service requested is context extraction. The notification agent will send notification emails to the other two consumers (and) notifying them of the request and asking their permission (see Appendix C for screenshots). If both users agree to the request, the context extraction request will be sent to the cloud. However, if only one of the consumers rejects the request, the context extraction request will be discarded. However, in this scenario, both users agree to the request. Then the context extraction request will be sent to the cloud and applied to the data entries by the execution manager. All the data entries in this time frame are removed from the "weather" database but inserted into a backup database for a period of time determined by the consumers. The number of data entries removed was 8,297 rows, which means that 1.31% of the data entries are removed (although not permanently) as they are considered old data, not frequently accessed and contain no singularities. However, the context of these data entries is extracted.

The context of these data entries, using the attribute "airTemperature", is inserted in the "contextWeather" database. The "contextWeather" database contains the average temperature for the time period for which the data entries are removed, as well as the highest and lowest values that occurred in that period. The attributes of any context database are different from one database (i.e. weather, water, traffic) to another. The context database can also be different when extracted from the same database if the parameters taken into account are different. For example, one consumer can choose the context database to be extracted from the "weather" database using "airTemperature"; another consumer might choose "windSpeed". The decision of which attribute to use depends on the attribute(s) that best describe the data entries removed. These kinds of decisions are made by the consumers and their departments but the criteria for these decisions are beyond the scope of this thesis. However, for the sake of simplicity and completion of the experiment, only one attribute is used in this evaluation to demonstrate the work presented.

6.3.3.1 Reduction Services Analysis

The aim of this initial evaluation (section 6.3.3) is to show the performance of the reduction services in terms of data reduction and execution time after applying the partitioning method. The evaluation shows that the amount of data reduced (after applying reduction services on only the first and third requests in section 6.3.2) is around 1.53%. This percentage shows the effectiveness of these reduction services with the partitioning method. The comprehensive evaluations for all data chunks are shown in section 6.3.3.2.

Execution time can be defined as the time from when a request is received by the cloud to when the request is fully performed in the cloud. The services execution time for both scenarios presented in section 6.3.3 is around the same, since both services need to check every entry. Thus, the first scenario (optimization service) takes 5.26 seconds and the second scenario (context extraction service) takes 5.57 seconds. The number of data entries affects the execution time when the services are applied. The more data entries there are, the more time is needed for execution.

6.3.3.2 Comprehensive Services Evaluation

More evaluations of the data chunks were performed to demonstrate the amount of data that can be reduced using the partitioning method along with the reduction services. This time, the optimization service will be applied to all

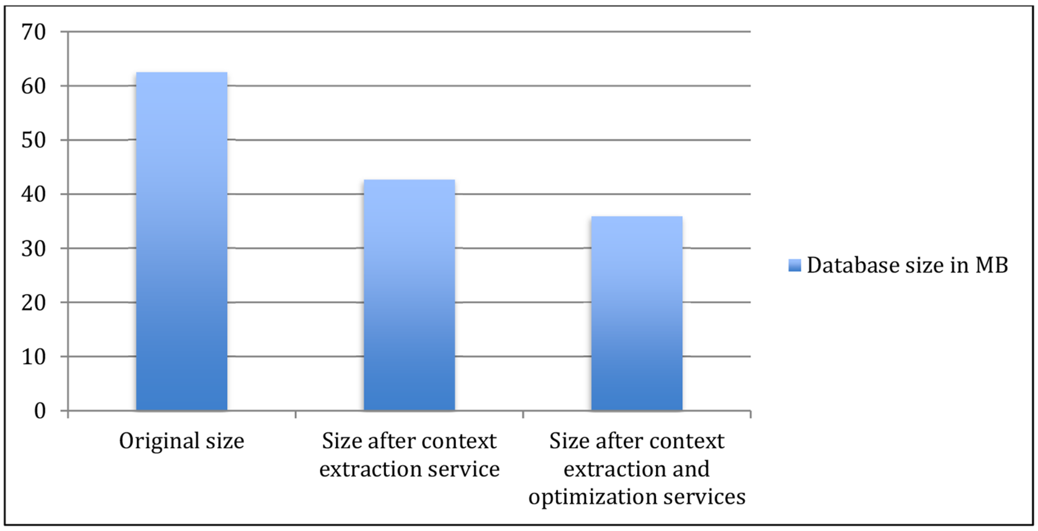


Figure 6.8: Size of the whole database after implementing the context extraction service for the mutable data chunks and the optimization service for the immutable data chunks

the immutable data chunks (25 chunks), whereas the context extraction will be applied to the mutable data chunks (11 chunks). The size of the database after applying the context extraction service to the mutable data chunks (11 chunks) is reduced from 62.5 MB to 42.7 MB, which means that the database is reduced by 31.68%. The size of the database after applying the optimization service on the remaining data chunks (25 immutable chunks) is reduced from 42.7 MB to 35.93 MB. Therefore, the size of the database after applying both services is reduced by around 42.51%, as shown in Figure 6.8. This demonstrates how effective the partitioning method and the services are at reducing the amount of smart city data stored in the cloud.

6.4 Architecture Highlights

After showing the different edge and cloud services of the architecture, there are several differences that distinguish them from those previously listed in chapter 2 (section 2.3.2.5). The first and major difference is the utilization of public local servers as an intermediate step between mobile devices and the cloud. The services allocation in this intermediate step increases the worthiness and importance of these services and thus increases the effectiveness of the architecture. Another difference is that, although previous architectures might contain similar services in terms of functionality, these are located either in a mobile device or the cloud. However, although there are no right answers regarding where to allocate crowd-sensing services, in the architecture presented in this thesis, the effectiveness of the services is increased since there are no energy or battery constraints in mobile devices, the crowd-sensed data are filtered and hence reduced before they reach the cloud. Another difference is that all the various services in the proposed architecture are distributed between local servers and the cloud. This decreases the intensity of a centralized approach in which all services are located in the same place regardless of the users' different locations.

6.4.1 Does the Architecture Meet the Performance Requirements?

Based on the evaluations and experiments presented in chapters 4 and 5 and in this chapter, this section reconsiders the performance requirements in chapter 3 to evaluate how the work in this thesis meets these requirements. The assessment is as follows.

1. Reducing the amount of data transferred

   One important goal in this thesis is to reduce the amount of data transferred from mobile devices to the cloud. This goal is achieved by introducing edge services. Even though the crowd will send data from their mobile devices to the edge with no data reduction, this transmission is local and by the use of Wi-Fi Access Points (i.e. no transmission cost). Also, choosing to have data reduction services located on the edge and not on mobile devices is due to storage and battery constraints of mobile devices where the edge servers have no such constraints. Therefore, in this thesis, the reduction of data transferred is taken into consideration from the edge to the cloud, where a large number of contributions needs to be transferred to the cloud not only a single one (from mobile device to local server). Edge services perform data reduction at different levels, as shown in the evaluation results of the first use case in section 6.2. The first step is to discard untrusted contributions and consider only trusted ones. The second step is to discard similar data that represent the same location in a predefined time frame. The third step is to apply compression algorithms to single-precision floating-point numbers that achieve a good compression ratio when compared with other techniques, as shown in chapter 4. All the different levels in reducing the data transferred will reduce bandwidth utilization and network traffic.

2. Ensuring storage saving in the cloud

   Another important goal in this thesis is to manage the crowd-sensed data in the cloud efficiently in order to remove unwanted data and save storage. This goal is achieved by introducing a partitioning method and the reduction services that are applied to data based on the output of the partitioning method. The evaluation results of the storage data management in the cloud are shown in the second use case in section 6.3. The partitioning method demonstrated its effectiveness when applying the reduction services without losing important data or values.

3. Ensuring robustness

   When utilizing local servers to act as edge servers before the cloud, there is a possibility of a server being down. Therefore, a recovery plan that can be adopted by the city is introduced in chapter 3, whereby the nearest available server can be in charge of receiving the crowd-sensed data that belong to the stopped server.

Another error might occur on the user side whereby a mobile device starts to send false data. To avoid this kind of error, the trust service located in the edge servers will examine the trust and truthfulness of the data, as shown in chapter 4. The trust service demonstrated its usefulness in reducing the amount of untrusted data by a reasonable percentage.

4. Providing a reasonable level of privacy With crowd-sensing, privacy concerns towards users' personal information are introduced. In this research, privacy concerns are taken into considerations on the user's side. The mobile app (i.e. SenseAll) was developed to use only application IDs when data are sent to the cloud in order to guarantee anonymity and ensure privacy.

6.5 Chapter Summary

This chapter has presented the results of the experiments conducted in this thesis in order to evaluate the edge and cloud services. The aim of these experiments is to understand the performance of these services in terms of data reduction before data are sent to the cloud and after data are received in the cloud.

The first evaluation was conducted on edge services in order to show the performance of each service and how those data are reduced at every stage. This evaluation ensured that the data that are sent to the cloud are trusted, distinct and compressed. The experiment was applied on real crowd-sensing data gathered from 14 different users using the "SenseAll" mobile application installed on their Android devices.

The second evaluation took place in the cloud and was applied on a weather database in order to show how consumers can partition data based upon the importance of the data. This evaluation also shows how consumers can then apply reduction services to weather data after partitioning. However, one of these reduction services (context extraction) will not be applied until the other consumers have agreed the outcome of this service.

Finally, the analysis shows that the proposed system met all the requirements mentioned during the design stage and reduced the amount and size of data in both the edge and the cloud by a reasonable percentage. The next chapter will conclude this thesis by summarizing the research work presented. Furthermore, the chapter will present the limitations in the architecture and improvements to the edge and cloud services that can be undertaken as future work.

Chapter 7

Conclusions and Future Work

This chapter presents the conclusions of this thesis by summarizing the research work conducted, showing the limitations of the work and research that needs to be undertaken in the future.

7.1 Research Summary and Benefits

This thesis presents a crowd-sensing architecture that overcomes certain challenges, such as the need to transfer a large amount of data to the cloud and manage the data there. Therefore, two sets of services are provided. The first set include the services that are located on the edge and pre-process the crowd-sensed data received in order to reduce the amount of data sent to the cloud. A smart city architecture that is located as close to the crowd as possible is presented in chapter 4. This architecture consists of three different services: the trust service, a scheduler and the reduction service (i.e. compression). The trust service calculates the trust of every contribution received using four factors (status, style, loyalty and similarity) and then updates the reputation score for every user in order to build the reputation. The scheduler will re- move similar data that are received from the same area and schedule sending distinct data to the cloud after a predefined time depending on their priority. The reduction unit service is a lossless compression step that is applied to GPS coordinates and accelerometer data. With these services, it is possible to ensure that data sent to the cloud from the edge are trusted, distinct and com- pressed. Then, the crowdsensed data that are sent to the cloud are reduced in amount and size.

The second set of services is located in the cloud and manages the data stored efficiently. These services are optimization and context extraction. Consumers of the data (e.g., a city council, cloud administrators, etc.) will first need to partition data entries to which they need to apply the services logically. Depending on the form of these data entries (immutable or mutable), consumers can choose which service to apply. The partitioning method depends on three factors to decide whether data entries are immutable or not: time, access rate and singularity. Therefore, data entries are immutable if they are newly inserted into the database, frequently accessed by consumers or contain important values. Data entries are mutable if they are old data (i.e. not newly inserted into the database), not frequently accessed by consumers and do not contain any important values.

Services are applied depending on the type of data. Thus, the optimization service can be applied to immutable and mutable parts since the data removed can be recovered. However, the context extraction service can only be applied to mutable data entries, since these will be removed from the database and the context of these data is extracted and stored. However, the removal of data is not a decision that can be made by only one consumer; this decision needs the approval of the other consumers who are involved and have access to the database. This kind of user-agreement decision is made using the notification agent. Therefore, if only one consumer rejects the removal of data entries (using a context extraction service), the service will not be applied and data will be left unchanged. Furthermore, if all the other consumers agree to the removal of data entries, these data entries are kept for a period of time as a backup in the cloud before they are permanently removed. The proposed system was evaluated and tested by deploying it on an Amazon EC2 instance, one local server and Android-based mobile devices. The system showed that it could successfully achieve the following:

- Reducing the amount of data on the edge by removing untrusted and similar data every day (i.e. a predefined period of time) from around 6.25% to 33.33%
- Reducing the size of data on the edge by compressing GPS coordinates and accelerometer readings every day (i.e. a predefined time) from around 10.48% to 34.24%.

- Improving cloud storage and reducing the cost associated with data storage by the partition method, whereby data were reduced by around 42.51% after applying both reduction services.
- Enhancing data quality so that data sent to the cloud are trusted, distinct and compressed.
- Ensuring traceability and users' privacy at the same time by tracking the origin of data and using only application IDs.

7.2 Limitations and Future Work

The results of the evaluations and experiments conducted in this thesis have successfully shown that the proposed system can reduce the amount of data transferred from mobile devices to the cloud and the amount of data stored in the cloud under certain constraints. However, the work still contains some limitations, which can be addressed and implemented in future work. These limitations are referred to briefly as follows.

7.2.1 Implementing More Services

Two types of services were implemented in this thesis. The first type took place on the edge in the form of public local servers or access points. These services act as a reduction and filtration of the crowd-sensed data in order to make them ready for the second type of services that take place in the cloud. In future work, edge services could be optimized by merging the services together in order to reduce the computations. The system could benefit further from adding more services in the cloud that could optimize the data management and reduce the cost of saving a large amount of data in the cloud. For example, the cloud could have a periodic suggestion agent that could interact with users and highlight the importance of data stored in the cloud.

7.2.2 Improving Existing Services

The proposed cloud services need to be improved to perform the same functionality with different databases and different domains. Therefore, the services need to apply different aspects when performing the reductions, such as applying more parameters (not only one) from the database. Furthermore, the notification agent showed good functionality when located on the consumer side in terms of bandwidth utilization and network latency. By allocating a notification agent on the consumer side, the query will not be sent to the cloud if only one consumer rejects the query. However, with a large number of consumers, it might be more efficient if the notification agent is located in the cloud and could exploit the cloud's powerful resources.

7.2.3 Conducting More Evaluations and Experiments

The proposed edge services were evaluated in a real-world scenario but with a small-scale crowd (14 users i.e. 14 devices). This is due to the complexity of having a large number of Android devices with which to conduct experiments (where a large number of users were using different operating systems) and the complexity of convincing some Android users (i.e. a crowd) to download the mobile app and conduct the experiment in the desired time frame (one week) and in the desired area (the city of Cork). Therefore, in future work, the system could be tested with the help of a larger crowd in order to observe how the system behaves when interacting with a large number of users.

Furthermore, cloud services were tested using available open source weather data, not data that are filtered and sent from the edge. This is due to the shortage of crowd-sensed data that could highlight the benefits of the cloud services proposed. However, in future work, cloud services could be implemented with real crowd-sensed data that are sent from the edge (i.e. local servers) in order to implement a complete chain of edge and cloud services that could show how the system would work in such situations.

7.2.4 Extending the Crowd-sensing Area

Today, the notion is that the IoT depends heavily on cloud computing to store and process data due to its unlimited storage and powerful processing. Crowd-sensing data could be extended to include any other data, such as those produced by light bulbs, wearable devices, kitchen appliances and buildings. Therefore, in future work, the data produced from crowd-sensing and IoT need to be combined and tested using the architecture proposed in this thesis to show how the system would act with heterogeneous data.

7.2.5 Using Different Operating Systems

The work in this thesis could be extended using different operating systems, such as Apple iOS and Windows. This extension would raise the number of users and thus increase the size of the crowd in which the mobile app could be installed and start the sensing activity.

7.2.6 Privacy and Security

This thesis did not focus on security and privacy features when implementing the proposed system since a smart city is by nature an open source domain. However, privacy was taken into account when users were sending their data to the local server, as the mobile app will assign unique application IDs to different users when they first register. These application IDs are the user IDs that will be encapsulated in the HTTP message and sent to the cloud. By using only application IDs, the system can ensure anonymity and hide users' identities. However, in some smart city domains, such as health, more security techniques, such as encryption, should be applied to the data saved in the cloud in order to protect those data from unauthorized users.

7.2.7 Offering Incentive Mechanisms

Persuading users to be part of the evaluation process in chapter 6 was not an easy task, which unfortunately resulted in a small crowd (i.e. 14 users). Therefore, to increase crowd involvement and the amount of crowd-sensed data received to demonstrate the architecture, there must be some kind of incentive mechanism. Such a mechanism would encourage users to install the mobile app and be part of the sensing process. Increasing user involvement will result in improving the quality of life in a city.

Bibliography

[1] Abad, Cristina L, Lu, Yi, and Campbell, Roy H. "DARE: Adaptive data replication for efficient cluster scheduling". In: *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*. Ieee. 2011, pp. 159–168.

[2] Aberer, Karl, Sathe, Saket, Chakraborty, Dipanjan, Martinoli, Alcherio, Barrenetxea, Guillermo, Faltings, Boi, and Thiele, Lothar. "OpenSense: open community driven sensing of environment". In: *Proceedings of the ACM SIGSPATIAL International Workshop on GeoStreaming*. ACM. 2010, pp. 39–42.

[3] Alkhelaiwi, A. and Grigoras, D. "Smart City Data Storage Optimization in the Cloud". In: *2018 IEEE Fourth International Conference on Big Data Computing Service and Applications (BigDataService)*. 2018, pp. 153–160. DOI: `10.1109/BigDataService.2018.00030`.

[4] Alkhelaiwi, Aseel and Grigoras, Dan. "Challenges of Crowd Sensing for Cost-Effective Data Management in the Cloud". In: *Cloud Computing and Big Data: Technologies, Applications and Security*. Ed. by Mostapha Zbakh, Mohammed Essaaidi, Pierre Manneback, and Chunming Rong. Cham: Springer International Publishing, 2019, pp. 73–88. ISBN: 978-3-319-97719-5.

[5] Alkhelaiwi, Aseel and Grigoras, Dan. "Data Reduction as a Service in Smart City Architecture". In: *Big Data Computing Service and Applications (BigDataService), 2017 IEEE Third International Conference on*. IEEE. 2017, pp. 172–178.

[6] Alkhelaiwi, Aseel and Grigoras, Dan. "Scheduling crowdsensing data to smart city applications in the cloud". In: *Intelligent Computer Communication and Processing (ICCP), 2016 IEEE 12th International Conference on*. IEEE. 2016, pp. 395–401.

[7] Alkhelaiwi, Aseel and Grigoras, Dan. "The origin and trustworthiness of data in smart city applications". In: *Utility and Cloud Computing (UCC), 2015 IEEE/ACM 8th International Conference on*. IEEE. 2015, pp. 376–382.

[8] Altomare, Albino, Cesario, Eugenio, Comito, Carmela, Marozzo, Fabrizio, and Talia, Domenico. "Using clouds for smart city applications". In: *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*. Vol. 2. IEEE. 2013, pp. 234–237.

[9] *Amazon Mechanical Turk*. `http://www.mturk.com`. Accessed: 30-10-2014.

[10] *Amazon RDS*. `https://aws.amazon.com/rds/`.

[11] *Amazon Web Services*. `http://aws.amazon.com/`.

[12] *Android Sensors*. `https://developer.android.com/guide/topics/sensors.html`.

[13] Angin, Pelin, Bhargava, Bharat, and Helal, Sumi. "A mobile-cloud collaborative traffic lights detector for blind navigation". In: *mobile data management (MDM), 2010 eleventh international conference on*. IEEE. 2010, pp. 396–401.

[14] Armbrust, Michael, Fox, Armando, Griffith, Rean, Joseph, Anthony D, Katz, Randy, Konwinski, Andy, Lee, Gunho, Patterson, David, Rabkin, Ariel, Stoica, Ion, et al. "A view of cloud computing". In: *Communications of the ACM* 53.4 (2010), pp. 50–58.

[15] Aubry, Elian, Silverston, Thomas, Lahmadi, Abdelkader, and Festor, Olivier. "CrowdOut: a mobile crowdsourcing service for road safety in digital cities". In: *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*. IEEE. 2014, pp. 86–91.

[16] Bahl, Paramvir and Padmanabhan, Venkata N. "RADAR: An in-building RF-based user location and tracking system". In: *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 2. Ieee. 2000, pp. 775–784.

[17] Bao, Yuan, Ren, Lei, Zhang, Lin, Zhang, Xuesong, and Luo, Yongliang. "Massive sensor data management framework in cloud manufacturing based on Hadoop". In: *Industrial Informatics (INDIN), 2012 10th IEEE International Conference on*. IEEE. 2012, pp. 397–401.

[18] Bari, Wasim, Memon, Ahmed Shiraz, and Schuller, Bernd. "Enhancing UNICORE storage management using Hadoop distributed file system". In: *European Conference on Parallel Processing*. Springer. 2009, pp. 345–352.

[19] Benouaret, Karim, Valliyur-Ramalingam, Raman, and Charoy, François. "CrowdSC: Building smart cities with large-scale citizen participation". In: *IEEE Internet Computing* 17.6 (2013), pp. 57–63.

[20] Bernstein, Michael S, Little, Greg, Miller, Robert C, Hartmann, Björn, Ackerman, Mark S, Karger, David R, Crowell, David, and Panovich, Katrina. "Soylent: a word processor with a crowd inside". In: *Communications of the ACM* 58.8 (2015), pp. 85–94.

[21] Burrows, Michael and Wheeler, David J. "A block-sorting lossless data compression algorithm". In: (1994).

[22] Calderoni, Luca, Maio, Dario, and Palmieri, Paolo. "Location-aware mobile services for a smart city: Design, implementation and deployment". In: *Journal of theoretical and applied electronic commerce research* 7.3 (2012), pp. 74–87.

[23] Cavallo, Marco, Cusmà, Lorenzo, Di Modica, Giuseppe, Polito, Carmelo, and Tomarchio, Orazio. "A Hadoop based Framework to Process Geo-distributed Big Data." In: *CLOSER (1)*. 2016, pp. 178–185.

[24] Cavallo, Marco, Polito, Carmelo, Modica, Giuseppe Di, and Tomarchio, Orazio. “H2F: a Hierarchical Hadoop Framework for big data processing in geo-distributed environments”. In: *Proceedings of the 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*. ACM. 2016, pp. 27–35.

[25] Chae, Kangsuk, Kim, Daihoon, Jung, Seohyun, Choi, Jaeduck, and Jung, Souhwan. “Evidence collecting system from car black boxes”. In: *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*. IEEE. 2010, pp. 1–2.

[26] Champion, Howard R, Augenstein, Jeffrey, Blatt, Alan J, Cushing, Brad, Digges, Kennerly, Siegel, John H, and Flanigan, Marie C. “Automatic crash notification and the URGENCY algorithm: Its history, value, and use”. In: *Advanced Emergency Nursing Journal* 26.2 (2004), pp. 143–156.

[27] Chatzimilioudis, Georgios, Konstantinidis, Andreas, Laoudias, Christos, and Zeinalipour-Yazti, Demetrios. “Crowdsourcing with smartphones”. In: *IEEE Internet Computing* 16.5 (2012), pp. 36–44.

[28] Chen, Longbiao, Wang, Leye, Zhang, Daqing, Li, Shijian, and Pan, Gang. “EnUp: Energy-efficient data uploading for mobile crowd sensing applications”. In: *Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), 2016 Intl IEEE Conferences*. IEEE. 2016, pp. 1074–1078.

[29] Chen, Xiao, Santos-Neto, Elizeu, and Ripeanu, Matei. “Crowdsourcing for on-street smart parking”. In: *Proceedings of the second ACM international symposium on Design and analysis of intelligent vehicular networks and applications*. ACM. 2012, pp. 1–8.

[30] Chen, Xiao, Santos-Neto, Elizeu, and Ripeanu, Matei. “Smart parking by the coin”. In: *Proceedings of the third ACM international symposium on Design and analysis of intelligent vehicular networks and applications*. ACM. 2013, pp. 109–114.

[31] Choo, Jing Hang, Cheong, Soon Nyean, Lee, Yee Lien, and Teh, Sze Hou. “I 2 navi: An indoor interactive NFC navigation system for android smartphones”. In: *Proceedings of the World Academy of Science, Engineering and Technology* 72 (2012), pp. 735–739.

[32] *CITYOPT*. `http://www.cityopt.eu`.

[33] Corburn, Jason. “Street science: community knowledge and environmental health justice (urban and industrial environments)”. In: (2005).

[34] *Creek Watch*. `http://creekwatch.researchlabs.ibm.com/`. Accessed: 10-9-2014.

[35] Cziva, Richard, Jouët, Simon, Stapleton, David, Tso, Fung Po, and Pezaros, Dimitrios P. “SDN-based virtual machine management for cloud data cen-

ters". In: *IEEE Transactions on Network and Service Management* 13.2 (2016), pp. 212–225.

[36] *Data Compression*. `https://docs.microsoft.com/enus/ sql/relational-databases/data-compression`.

[37] Dinh, Hoang T, Lee, Chonho, Niyato, Dusit, and Wang, Ping. "A survey of mobile cloud computing: architecture, applications, and approaches". In: *Wireless communications and mobile computing* 13.18 (2013), pp. 1587–1611.

[38] Dutta, Joy, Gazi, Firoj, Roy, Sarbani, and Chowdhury, Chandreyee. "Airsense: Opportunistic crowd-sensing based air quality monitoring system for smart city". In: *SENSORS, 2016 IEEE*. IEEE. 2016, pp. 1–3.

[39] Dutta, Prabal, Aoki, Paul M, Kumar, Neil, Mainwaring, Alan, Myers, Chris, Willett, Wesley, and Woodruff, Allison. "Common sense: participatory urban sensing using a network of handheld air quality monitors". In: *Proceedings of the 7th ACM conference on embedded networked sensor systems*. ACM. 2009, pp. 349–350.

[40] Dynes, Russell R. "Community emergency planning: false assumption and inappropriate analogies". In: (1990).

[41] *e-SAVE*. `http://www.e-save.eu`.

[42] Eisenman, Shane B, Miluzzo, Emiliano, Lane, Nicholas D, Peterson, Ronald A, Ahn, Gahng-Seop, and Campbell, Andrew T. "BikeNet: A mobile sensing system for cyclist experience mapping". In: *ACM Transactions on Sensor Networks (TOSN)* 6.1 (2009), p. 6.

[43] Engelson, Vadim, Fritzson, Dag, and Fritzson, Peter. "Lossless compression of high-volume numerical data from simulations". In: *Proc. Data Compression Conference*. 2000.

[44] Eriksson, Jakob, Girod, Lewis, Hull, Bret, Newton, Ryan, Madden, Samuel, and Balakrishnan, Hari. "The pothole patrol: using a mobile sensor network for road surface monitoring". In: *Proceedings of the 6th international conference on Mobile systems, applications, and services*. ACM. 2008, pp. 29–39.

[45] Fakoor, Rasool, Raj, Mayank, Nazi, Azade, Di Francesco, Mario, and Das, Sajal K. "An integrated cloud-based framework for mobile phone sensing". In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM. 2012, pp. 47–52.

[46] Feng, Bo, Wu, Chentao, and Li, Jie. "MLC: An Efficient Multi-level Log Compression Method for Cloud Backup Systems". In: *Trustcom/BigDataSE/I? SPA, 2016 IEEE*. IEEE. 2016, pp. 1358–1365.

[47] Feng, Cheng, Wang, Wendong, Tian, Ye, Que, Xirong, and Gong, Xiangyang. "Estimate air quality based on mobile crowe sensing and big data". In: *A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2017 IEEE 18th International Symposium on*. IEEE. 2017, pp. 1–9.

[48] *FOURSQUARE App*. `https://foursquare.com`.

[49] Franklin, Michael J, Kossmann, Donald, Kraska, Tim, Ramesh, Sukriti, and Xin, Reynold. “CrowdDB: answering queries with crowdsourcing”. In: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM. 2011, pp. 61–72.

[50] Gamito, Manuel Noronha and Dias, Miguel Salles. “Lossless coding of floating point data with JPEG 2000 Part 10”. In: *Applications of Digital Image Processing XXVII*. Vol. 5558. International Society for Optics and Photonics. 2004, pp. 276–288.

[51] Ganeriwal, Saurabh, Balzano, Laura K, and Srivastava, Mani B. “Reputation-based framework for high integrity sensor networks”. In: *ACM Transactions on Sensor Networks (TOSN)* 4.3 (2008), p. 15.

[52] Ganti, Raghu K, Ye, Fan, and Lei, Hui. “Mobile crowdsensing: current state and future challenges”. In: *IEEE Communications Magazine* 49.11 (2011).

[53] Gao, Hui, Liu, Chi Harold, Tian, Ye, Xi, Teng, and Wang, Wendong. “Ensuring High-Quality Data Collection for Mobile Crowd Sensing”. In: *Wireless Communications and Networking Conference (WCNC), 2017 IEEE*. IEEE. 2017, pp. 1–6.

[54] Gao, Yali, Li, Xiaoyong, Li, Jirui, and Gao, Yunquan. “DTRF: A dynamic-trust-based recruitment framework for Mobile Crowd Sensing system”. In: *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*. IEEE. 2017, pp. 632–635.

[55] Gerla, Mario. “Vehicular cloud computing”. In: *Ad Hoc Networking Workshop (Med-Hoc-Net), 2012 The 11th Annual Mediterranean*. IEEE. 2012, pp. 152–155.

[56] Ghido, Florin. “An efficient algorithm for lossless compression of IEEE float audio”. In: *Data Compression Conference, 2004. Proceedings. DCC 2004*. IEEE. 2004, pp. 429–438.

[57] *gigwalk*. `www.gigwalk.com`.

[58] Gomez, Leonardo A Bautista and Cappello, Franck. “Improving floating point compression through binary masks”. In: *Big Data, 2013 IEEE International Conference on*. IEEE. 2013, pp. 326–331.

[59] Gregori, Enrico, Lenzini, Luciano, Luconi, Valerio, and Vecchio, Alessio. “Sensing the Internet through crowdsourcing”. In: *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on*. IEEE. 2013, pp. 248–254.

[60] Gunawan, Lucy T, Fitrianie, Siska, Yang, Zhenke, Brinkman, Willem-Paul, and Neerincx, Mark. “TravelThrough: a participatory-based guidance system for traveling through disaster areas”. In: *CHI’12 Extended Abstracts on Human Factors in Computing Systems*. ACM. 2012, pp. 241–250.

[61] Guo, Bin, Wang, Zhu, Yu, Zhiwen, Wang, Yu, Yen, Neil Y, Huang, Runhe, and Zhou, Xingshe. “Mobile crowd sensing and computing: The review of

an emerging human-powered sensing paradigm". In: *ACM Computing Surveys (CSUR)* 48.1 (2015), p. 7.

[62] Haklay, Mordechai and Weber, Patrick. "Openstreetmap: User-generated street maps". In: *IEEE Pervasive Computing* 7.4 (2008), pp. 12–18.

[63] Hassan, Mohammad Mehedi, Song, Biao, and Huh, Eui-Nam. "A framework of sensor-cloud integration opportunities and challenges". In: *Proceedings of the 3rd international conference on Ubiquitous information management and communication*. ACM. 2009, pp. 618–626.

[64] Haubensak, Oliver. "Smart cities and internet of things". In: *Business Aspects of the Internet of Things, Seminar of Advanced Topics, ETH Zurich*. 2011, pp. 33–39.

[65] Herrera, Juan C, Work, Daniel B, Herring, Ryan, Ban, Xuegang Jeff, Jacobson, Quinn, and Bayen, Alexandre M. "Evaluation of traffic data obtained via GPS-enabled mobile phones: The Mobile Century field experiment". In: *Transportation Research Part C: Emerging Technologies* 18.4 (2010), pp. 568–583.

[66] Huang, Kuan Lun, Kanhere, Salil S, and Hu, Wen. "A privacy-preserving reputation system for participatory sensing". In: *Local Computer Networks (LCN), 2012 IEEE 37th Conference on*. IEEE. 2012, pp. 10–18.

[67] Huang, Kuan Lun, Kanhere, Salil S, and Hu, Wen. "Are you contributing trustworthy data?: the case for a reputation system in participatory sensing". In: *Proceedings of the 13th ACM international conference on Modeling, analysis, and simulation of wireless and mobile systems*. ACM. 2010, pp. 14–22.

[68] Hull, Bret, Bychkovsky, Vladimir, Zhang, Yang, Chen, Kevin, Goraczko, Michel, Miu, Allen, Shih, Eugene, Balakrishnan, Hari, and Madden, Samuel. "CarTel: a distributed mobile sensor computing system". In: *Proceedings of the 4th international conference on Embedded networked sensor systems*. ACM. 2006, pp. 125–138.

[69] Information Society, Directorate-General for the and Commission), Media (European. *Advancing and Applying Living Lab Methodologies*. Tech. rep. July 2010.

[70] Irwin, Alan. *Citizen science: A study of people, expertise and sustainable development*. Routledge, 2002.

[71] Isenburg, Martin, Lindstrom, Peter, and Snoeyink, Jack. "Lossless compression of floating-point geometry". In: *Computer-Aided Design and Applications* 1.1-4 (2004), pp. 495–501.

[72] *jana*. `www.jana.com`.

[73] Jayaraman, Prem Prakash, Perera, Charith, Georgakopoulos, Dimitrios, and Zaslavsky, Arkady. "Efficient opportunistic sensing using mobile collaborative platform mosden". In: *Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013 9th International Conference Conference on*. IEEE. 2013, pp. 77–86.

[74] Jeung, Hoyoung, Sarni, Sofiane, Paparrizos, Ioannis, Sathe, Saket, Aberer, Karl, Dawes, Nicholas, Papaioannou, Thanasis G, and Lehning, Michael. "Effective metadata management in federated sensor networks". In: *Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC), 2010 IEEE International Conference on*. IEEE. 2010, pp. 107–114.

[75] Jing, Yao, Guo, Bin, Wang, Zhu, Li, Victor OK, Jacqueline, CK, and Yu, Zhiwen. "CrowdTracker: Optimized Urban Moving Object Tracking Using Mobile Crowd Sensing". In: *IEEE Internet of Things Journal* (2017).

[76] Kalim, F., Jeong, J. P., and Ilyas, M. U. "CRATER: A Crowd Sensing Application to Estimate Road Conditions". In: *IEEE Access* 4 (2016), pp. 8317–8326.

[77] Kanakatte, Aparna, Subramanya, Rakshith, Delampady, Ashik, Nayak, Rajarama, Purushothaman, Balamuralidhar, and Gubbi, Jayavardhana. "Cloud solution for histopathological image analysis using region of interest based compression". In: *Engineering in Medicine and Biology Society (EMBC), 2017 39th Annual International Conference of the IEEE*. IEEE. 2017, pp. 1202–1205.

[78] Kantarci, Burak and Mouftah, Hussein T. "Trustworthy sensing for public safety in cloud-centric internet of things". In: *IEEE Internet of Things Journal* 1.4 (2014), pp. 360–368.

[79] Kargupta, Hillol, Sarkar, Kakali, and Gilligan, Michael. "MineFleet®: an overview of a widely adopted distributed vehicle performance data mining system". In: *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2010, pp. 37–46.

[80] Kazemi, Leyla and Shahabi, Cyrus. "Geocrowd: enabling query answering with spatial crowdsourcing". In: *Proceedings of the 20th international conference on advances in geographic information systems*. ACM. 2012, pp. 189–198.

[81] Kazemi, Leyla, Shahabi, Cyrus, and Chen, Lei. "Geotrucrowd: trustworthy query answering with spatial crowdsourcing". In: *Proceedings of the 21st acm sigspatial international conference on advances in geographic information systems*. ACM. 2013, pp. 314–323.

[82] Khan, Zaheer, Anjum, Ashiq, and Kiani, Saad Liaquat. "Cloud based big data analytics for smart future cities". In: *Proceedings of the 2013 IEEE/ACM 6th international conference on utility and cloud computing*. IEEE Computer Society. 2013, pp. 381–386.

[83] Kincaid, J. *Googles open spot makes parking a breeze, assuming everyone turns into a good samaritan*. 2010.

[84] Koukoumidis, Emmanouil, Peh, Li-Shiuan, and Martonosi, Margaret Rose. "Signalguru: leveraging mobile phones for collaborative traffic signal schedule advisory". In: *Proceedings of the 9th international conference on Mobile systems, applications, and services*. ACM. 2011, pp. 127–140.

[85] Kulkarni, Anand, Can, Matthew, and Hartmann, Björn. "Collaboratively crowdsourcing workflows with turkomatic". In: *Proceedings of the acm 2012*

conference on computer supported cooperative work. ACM. 2012, pp. 1003–1012.

[86] La, Hyun Jung and Kim, Soo Dong. “A conceptual framework for provisioning context-aware mobile cloud services”. In: *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE. 2010, pp. 466–473.

[87] Lamy-Perbal, Sylvie, Boukallel, Mehdi, and Castaneda, Nadir. “An improved pedestrian inertial navigation system for indoor environments”. In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE. 2011, pp. 2505–2510.

[88] Lane, Nicholas D, Miluzzo, Emiliano, Lu, Hong, Peebles, Daniel, Choudhury, Tanzeem, and Campbell, Andrew T. “A survey of mobile phone sensing”. In: *IEEE Communications magazine* 48.9 (2010).

[89] Laoudias, Christos, Constantinou, George, Constantinides, Marios, Nicolaou, Silouanos, Zeinalipour-Yazti, Demetrios, and Panayiotou, Christos G. “The airplace indoor positioning platform for android smartphones”. In: *Mobile Data Management (MDM), 2012 IEEE 13th International Conference on*. IEEE. 2012, pp. 312–315.

[90] Larkou, Georgios, Metochi, Julia, Chatzimilioudis, Georgios, and Zeinalipour-Yazti, Demetrios. “CLODA: A crowdsourced linked open data architecture”. In: *Mobile Data Management (MDM), 2013 IEEE 14th International Conference on*. Vol. 2. IEEE. 2013, pp. 104–109.

[91] Lasnia, Damian, Broering, Arne, Broering, A, Jirka, Simon, and Remke, Albert. “Crowdsourcing sensor tasks to a Socio-Geographic network”. In: *AGILE 2010: proceedings of the 13th AGILE International Conference on Geographic Information Science: geospatial thinking, Guimaraes, Portugal*. AGILE. 2010.

[92] Ledlie, Jonathan, Odero, Billy, Minkov, Einat, Kiss, Imre, and Polifroni, Joseph. “Crowd translator: on building localized speech recognizers through micropayments”. In: *ACM SIGOPS Operating Systems Review* 43.4 (2010), pp. 84–89.

[93] Li, Hongzhi and Hua, Xian-Sheng. “Melog: mobile experience sharing through automatic multimedia blogging”. In: *Proceedings of the 2010 ACM multimedia workshop on Mobile cloud media computing*. ACM. 2010, pp. 19–24.

[94] Liu, Chi Harold, Zhang, Bo, Su, Xin, Ma, Jian, Wang, Wendong, and Leung, Kin K. “Energy-aware participant selection for smartphone-enabled mobile crowd sensing”. In: *IEEE Systems Journal* 11.3 (2017), pp. 1435–1446.

[95] Liu, Qun, Kumar, Suman, and Mago, Vijay. “SafeRNet: Safe transportation routing in the era of Internet of vehicles and mobile crowd sensing”. In: *Consumer Communications & Networking Conference (CCNC), 2017 14th IEEE Annual*. IEEE. 2017, pp. 299–304.

[96] Liu, Songbin, Huang, Xiaomeng, Ni, Yufang, Fu, Haohuan, and Yang, Guangwen. “A versatile compression method for floating-point data stream”. In: *Net-*

working and Distributed Computing (ICNDC), 2013 Fourth International Conference on. IEEE. 2013, pp. 141–145.

[97] Liu, Xuan, Lu, Meiyu, Ooi, Beng Chin, Shen, Yanyan, Wu, Sai, and Zhang, Meihui. “Cdas: a crowdsourcing data analytics system”. In: *Proceedings of the VLDB Endowment* 5.10 (2012), pp. 1040–1051.

[98] Liu, Yefeng, Alexandrova, Todorka, and Nakajima, Tatsuo. “Using stranger as sensors: temporal and geo-sensitive question answering via social media”. In: *Proceedings of the 22nd international conference on World Wide Web*. ACM. 2013, pp. 803–814.

[99] Lohrmann, Björn and Kao, Odej. “Processing smart meter data streams in the cloud”. In: *Innovative Smart Grid Technologies (ISGT Europe), 2011 2nd IEEE PES International Conference and Exhibition on*. IEEE. 2011, pp. 1–8.

[100] Machado, Rodrigo Prestes, Conforto, Débora, and Santarosa, Lucila. “Sound chat: Implementation of sound awareness elements for visually impaired users in web-based cooperative systems”. In: *Computers in Education (SIIE), 2017 International Symposium on*. IEEE. 2017, pp. 1–6.

[101] Manzoor, Atif, Patsakis, Constantinos, McCarthy, Jessica, Mullarkey, Gabriel, Clarke, Siobhán, Cahill, Vinny, and Bouroche, Mélanie. “Data sensing and dissemination framework for smart cities”. In: *MOBILe Wireless MiddleWARE, Operating Systems and Applications (Mobilware), 2013 International Conference on*. IEEE. 2013, pp. 156–165.

[102] Mathur, Suhas, Jin, Tong, Kasturirangan, Nikhil, Chandrasekaran, Janani, Xue, Wenzhi, Gruteser, Marco, and Trappe, Wade. “Parknet: drive-by sensing of road-side parking statistics”. In: *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM. 2010, pp. 123–136.

[103] Matyas, Sebastian, Matyas, Christian, Schlieder, Christoph, Kiefer, Peter, Mitarai, Hiroko, and Kamata, Maiko. “Designing location-based mobile games with a purpose: collecting geospatial data with CityExplorer”. In: *Proceedings of the 2008 international conference on advances in computer entertainment technology*. ACM. 2008, pp. 244–247.

[104] McEntire, David A. “Anticipating Human Behavior in Disasters: Myths, Exaggerations and Realities”. In: *Disaster response and recovery* (2006), pp. 62–85.

[105] Medvedev, Alexey, Zaslavsky, Arkady, Grudinin, Vladimir, and Khoruzhnikov, Sergey. “Citywatcher: annotating and searching video data streams for smart cities applications”. In: *International Conference on Next Generation Wired/Wireless Networking*. Springer. 2014, pp. 144–155.

[106] Mell, Peter, Grance, Tim, et al. “The NIST definition of cloud computing”. In: (2011).

[107] *Metrosense*. `http://www.cs.dartmouth.edu/~sensorlab/metrosense/`.

[108] Millard, Peter, Saint-Andre, Peter, and Meijer, Ralph. “XEP-0060: publish-subscribe”. In: *XMPP Standards Foundation* 1 (2010), p. 13.

[109] Miluzzo, Emiliano, Lane, Nicholas D, Eisenman, Shane B, and Campbell, Andrew T. “CenceMe–injecting sensing presence into social networking applications”. In: *European Conference on Smart Sensing and Context*. Springer. 2007, pp. 1–28.

[110] Min, Hong and Scheuermann, Peter. “A Hierarchical back-end architecture for smartphone sensing”. In: *Proceedings of the 2012 ACM Research in Applied Computation Symposium*. ACM. 2012, pp. 434–439.

[111] Minker, Jack. “On indefinite databases and the closed world assumption”. In: *International Conference on Automated Deduction*. Springer. 1982, pp. 292–308.

[112] *Mobile Advertising*. `http://www.mobads.com`.

[113] *Mobile cloud computing forum*. `http://www.mobilecloudcomputingforum.com/`.

[114] Mohan, Prashanth, Padmanabhan, Venkata N, and Ramjee, Ramachandran. “Nericell: rich monitoring of road and traffic conditions using mobile smartphones”. In: *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM. 2008, pp. 323–336.

[115] Mühl, Gero, Fiege, Ludger, and Pietzuch, Peter. *Distributed event-based systems*. Springer Science & Business Media, 2006.

[116] Nicolae, Bogdan, Antoniu, Gabriel, Bougé, Luc, Moise, Diana, and Carpen-Amarie, Alexandra. “BlobSeer: Next-generation data management for large scale infrastructures”. In: *Journal of Parallel and distributed computing* 71.2 (2011), pp. 169–184.

[117] Nivedha, B, Priyadharshini, M, Thendral, E, and Deenadayalan, T. “Lossless Image Compression in Cloud Computing”. In: *Technical Advancements in Computers and Communications (ICTACC), 2017 International Conference on*. IEEE. 2017, pp. 112–115.

[118] *O2 Wi-Fi Network*. `https://www.o2.co.uk/connectivity/free-wifi`.

[119] *OpenIoT Architecture*. `https://github.com/OpenIotOrg/openiot/wiki/OpenIoT-Architecture`. Accessed: 12-4-2014.

[120] Ouyang, Robin Wentao, Srivastava, Animesh, Prabahar, Prithvi, Roy Choudhury, Romit, Addicott, Merideth, and McClernon, F Joseph. “If you see something, swipe towards it: crowdsourced event localization using smartphones”. In: *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*. ACM. 2013, pp. 23–32.

[121] Pallis, George. “Cloud computing: the new frontier of internet computing”. In: *IEEE internet computing* 14.5 (2010), pp. 70–73.

[122] Papakos, Panagiotis, Capra, Licia, and Rosenblum, David S. “Volare: context-aware adaptive cloud service discovery for mobile systems”. In: *Proceedings of the 9th International Workshop on Adaptive and Reflective Middleware*. ACM. 2010, pp. 32–38.

[123] Park, Jong Won, Yun, Chang Ho, Rho, Seong Woo, Lee, Yong Woo, and Jung, Hae Sun. “Mobile cloud web-service for U-City”. In: *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*. IEEE. 2011, pp. 1061–1065.

[124] Pascoe, Jason. “Adding generic contextual capabilities to wearable computers”. In: *Wearable Computers, 1998. Digest of Papers. Second International Symposium on*. IEEE. 1998, pp. 92–99.

[125] Paz, Leandro Ferreira, Maran, Vinicius, Machado, Alencar, and Augustin, Iara. “MECA: Mobile System Support for Brazilian Community Health Agents Program Based on Context-Awareness”. In: *IEEE Latin America Transactions* 15.8 (2017), pp. 1547–1555.

[126] Pelusi, Luciana, Passarella, Andrea, and Conti, Marco. “Opportunistic networking: data forwarding in disconnected mobile ad hoc networks”. In: *IEEE communications Magazine* 44.11 (2006).

[127] Peng, Dan, Wu, Fan, and Chen, Guihai. “Pay as how well you do: A quality based incentive mechanism for crowdsensing”. In: *Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing*. ACM. 2015, pp. 177–186.

[128] Podnar Zarko, Ivana, Antonic, Aleksandar, and Pripužic, Krešimir. “Publish/subscribe middleware for energy-efficient mobile crowdsensing”. In: *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*. ACM. 2013, pp. 1099–1110.

[129] Pryss, Rüdiger, Reichert, Manfred, Herrmann, Jochen, Langguth, Berthold, and Schlee, Winfried. “Mobile Crowd Sensing in Clinical and Psychological Trials–A Case Study”. In: *Computer-Based Medical Systems (CBMS), 2015 IEEE 28th International Symposium on*. IEEE. 2015, pp. 23–24.

[130] Pryss, Rüdiger, Reichert, Manfred, Langguth, Berthold, and Schlee, Winfried. “Mobile crowd sensing services for tinnitus assessment, therapy, and research”. In: *Mobile Services (MS), 2015 IEEE International Conference on*. IEEE. 2015, pp. 352–359.

[131] Puliafito, Antonio. “Sensorcloud: An integrated system for advanced multirisk management”. In: *Network Cloud Computing and Applications (NCCA), 2014 IEEE 3rd Symposium on*. IEEE. 2014, pp. 1–8.

[132] Qin, Zhaokun and Zhu, Yanmin. “NoiseSense: A Crowd Sensing System for Urban Noise Mapping Service”. In: *Parallel and Distributed Systems (ICPADS), 2016 IEEE 22nd International Conference on*. IEEE. 2016, pp. 80–87.

[133] Ra, Moo-Ryong, Liu, Bin, La Porta, Tom F, and Govindan, Ramesh. “Medusa: A programming framework for crowd-sensing applications”. In: *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM. 2012, pp. 337–350.

[134] Raento, Mika, Oulasvirta, Antti, Petit, Renaud, and Toivonen, Hannu. "ContextPhone: A prototyping platform for context-aware mobile applications". In: *IEEE pervasive computing* 4.2 (2005), pp. 51–59.

[135] Ramaiah, K Dasaradha and Venugopal, T. "A novel approach to detect most effective compression technique based on compression ratio and time complexity with high image data load for cloud migration". In: *Colossal Data Analysis and Networking (CDAN), Symposium on*. IEEE. 2016, pp. 1–5.

[136] Rana, Rajib Kumar, Chou, Chun Tung, Kanhere, Salil S, Bulusu, Nirupama, and Hu, Wen. "Ear-phone: an end-to-end participatory urban noise mapping system". In: *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*. ACM. 2010, pp. 105–116.

[137] Rasmussen, Carl Edward. "Gaussian processes in machine learning". In: *Advanced lectures on machine learning*. Springer, 2004, pp. 63–71.

[138] Ratanaworabhan, Paruj, Ke, Jian, and Burtscher, Martin. "Fast lossless compression of scientific floating-point data". In: *Data Compression Conference, 2006. DCC 2006. Proceedings*. IEEE. 2006, pp. 133–142.

[139] Reddy, Emmenual, Kumar, Sarnil, Rollings, Nicholas, and Chandra, Rohitash. "Mobile application for dengue fever monitoring and tracking via GPS: case study for fiji". In: *arXiv preprint arXiv:1503.00814* (2015).

[140] Reddy, Sasank, Shilton, Katie, Denisov, Gleb, Cenizal, Christian, Estrin, Deborah, and Srivastava, Mani. "Biketastic: sensing and mapping for better biking". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2010, pp. 1817–1820.

[141] Reddy, Sasank, Parker, Andrew, Hyman, Josh, Burke, Jeff, Estrin, Deborah, and Hansen, Mark. "Image browsing, processing, and clustering for participatory sensing: lessons from a DietSense prototype". In: *Proceedings of the 4th workshop on Embedded networked sensors*. ACM. 2007, pp. 13–17.

[142] Reinsch, Tobias, Wang, Yue, Knechtel, Martin, Ameling, Michael, and Herzig, Philipp. "CINA-A crowdsourced indoor navigation assistant". In: *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*. IEEE Computer Society. 2013, pp. 500–505.

[143] Reyna, Eva Arias-de, Dardari, Davide, Closas, Pau, and Djurić, Petar M. "Enhanced indoor localization through crowd sensing". In: *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*. IEEE. 2017, pp. 2487–2491.

[144] Rodrigues, Joao GP, Aguiar, Ana, Vieira, Fausto, Barros, Joao, and Cunha, Joao P Silva. "A mobile sensing architecture for massive urban scanning". In: *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*. IEEE. 2011, pp. 1132–1137.

[145] Ruiz-Alvarez, Arkaitz and Humphrey, Marty. "An automated approach to cloud storage service selection". In: *Proceedings of the 2nd international workshop on Scientific cloud computing*. ACM. 2011, pp. 39–48.

[146] Saint-Andre, Peter. “Extensible messaging and presence protocol (XMPP): Core”. In: (2011).

[147] Salomon, David and Motta, Giovanni. *Handbook of data compression*. Springer Science & Business Media, 2010.

[148] Samaraweera, Isuru and Corera, Sheran. “Sahana victim registries: Effectively track disaster victims”. In: *Proceeding of ISCRAM* (2007).

[149] Selviandro, Nungki, Sabariah, Mira Kania, and Saputra, Surya. “Context awareness system on ubiquitous learning with case based reasoning and nearest neighbor algorithm”. In: *Information and Communication Technology (ICoICT), 2016 4th International Conference on*. IEEE. 2016, pp. 1–6.

[150] *Sensor Web Enablement*. `http://www.opengeospatial.org/standards`. Accessed: 12-9-2014. 2011.

[151] Silva, Welington M da, Alvaro, Alexandre, Tomas, Gustavo HRP, Afonso, Ricardo A, Dias, Kelvin L, and Garcia, Vinicius C. “Smart cities software architectures: a survey”. In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM. 2013, pp. 1722–1727.

[152] Simonite, T. *Navigation im Schwarm*. `http://www.heise.de/tr/artikel/Navigation-im-Schwarm-1183752.html`. 2011.

[153] Sinaeepourfard, Amir, Garcia, Jordi, Masip-Bruin, Xavier, and Marin-Tordera, Eva. “A Novel Architecture for Efficient Fog to Cloud Data Management in Smart Cities”. In: *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE. 2017, pp. 2622–2623.

[154] Singer, Jane B, Domingo, David, Heinonen, Ari, Hermida, Alfred, Paulussen, Steve, Quandt, Thorsten, Reich, Zvi, and Vujnovic, Marina. *Participatory journalism: Guarding open gates at online newspapers*. John Wiley & Sons, 2011.

[155] *Smart City: San Francisco*. `http://smartcitysf.com`.

[156] *Smart City Wien*. `http://urbantransform.eu/2014/09/10/vienna-2050-ensuring-quality-of-life-through-innovation-adopting-the-smart-city-wien-framework/`.

[157] *Smart Seoul 2015*. `http://english.seoul.go.kr/wp-content/uploads/2014/02/SMART_SEOUL_2015_41.pdf`.

[158] *Smart Town*. `http://fujisawasst.com/EN/project/`.

[159] Srinivasan, Vijay, Moghaddam, Saeed, Mukherji, Abhishek, Rachuri, Kiran K, Xu, Chenren, and Tapia, Emmanuel Munguia. “Mobileminer: Mining your frequent patterns on your phone”. In: *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM. 2014, pp. 389–400.

[160] *StartUpAmsterdam*. `https://www.iamsterdam.com/en/business/startupamsterdam`.

[161] *StartUpDelta*. `https://www.startupdelta.org`.

[162] Stevens, Matthias and D?Hondt, Ellie. “Crowdsourcing of pollution data using smartphones”. In: *Workshop on Ubiquitous Crowdsourcing*. 2010.

[163] *Strategy Analytics*. `http://www.strategyanalytics.com`. Accessed: 20-9-2014.

[164] *Swarm App*. `https://www.swarmapp.com`.

[165] Szabó, Róbert, Farkas, Károly, Ispány, Márton, Benczúr, András A, Bátfai, Norbert, Jeszenszky, Péter, Laki, Sándor, Vágner, Anikó, Kollár, Lajos, Sidló, Cs, et al. "Framework for smart city applications based on participatory sensing". In: *Cognitive Infocommunications (CogInfoCom), 2013 IEEE 4th International Conference on*. IEEE. 2013, pp. 295–300.

[166] Talasila, Manoop, Curtmola, Reza, and Borcea, Cristian. "Mobile crowd sensing". In: *Google Scholar* (2015).

[167] Talebifard, Peyman and Leung, Victor CM. "Towards a content-centric approach to crowd-sensing in vehicular clouds". In: *Journal of Systems Architecture* 59.10 (2013), pp. 976–984.

[168] Thiagarajan, Arvind, Ravindranath, Lenin, LaCurts, Katrina, Madden, Samuel, Balakrishnan, Hari, Toledo, Sivan, and Eriksson, Jakob. "VTrack: accurate, energy-aware road traffic delay estimation using mobile phones". In: *Proceedings of the 7th ACM conference on embedded networked sensor systems*. ACM. 2009, pp. 85–98.

[169] Townsend, Kevin R and Zambreno, Joseph. "A multi-phase approach to floating-point compression". In: *Electro/Information Technology (EIT), 2015 IEEE International Conference on*. IEEE. 2015, pp. 251–256.

[170] Toyama, Kentaro, Logan, Ron, and Roseway, Asta. "Geographic location tags on digital images". In: *Proceedings of the eleventh ACM international conference on Multimedia*. ACM. 2003, pp. 156–166.

[171] Trehard, Guillaume, Lamy-Perbal, Sylvie, and Boukallel, Mehdi. "Indoor infrastructure-less solution based on sensor augmented smartphone for pedestrian localisation". In: *Ubiquitous Positioning, Indoor Navigation, and Location Based Service (UPINLBS), 2012*. IEEE. 2012, pp. 1–7.

[172] *Usahidi*. `http://haiti.ushahidi.com/`. 2007.

[173] Usevitch, Bryan E. "JPEG2000 extensions for bit plane coding of floating point data". In: *Data Compression Conference, 2003. Proceedings. DCC 2003*. IEEE. 2003, p. 451.

[174] Wang, Leye, Zhang, Daqing, Yan, Zhixian, Xiong, Haoyi, and Xie, Bing. "effSense: A novel mobile crowd-sensing framework for energy-efficient and cost-effective data uploading". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45.12 (2015), pp. 1549–1563.

[175] Wang, Qianru, Guo, Bin, Wang, Leye, Xin, Tong, Du, He, Chen, Huihui, and Yu, Zhiwen. "CrowdWatch: Dynamic Sidewalk Obstacle Detection Using Mobile Crowd Sensing". In: *IEEE Internet of Things Journal* 4.6 (2017), pp. 2159–2171.

[176] Wang, Xinlei, Govindan, Kannan, and Mohapatra, Prasant. "Collusion-resilient quality of information evaluation based on information provenance". In: *Sen-*

sor, Mesh and Ad Hoc Communications and Networks (SECON), 2011 8th Annual IEEE Communications Society Conference on. IEEE. 2011, pp. 395–403.

[177] Wang, Xinlei Oscar, Cheng, Wei, Mohapatra, Prasant, and Abdelzaher, Tarek. "Artsense: Anonymous reputation and trust in participatory sensing". In: *INFOCOM, 2013 Proceedings IEEE*. IEEE. 2013, pp. 2517–2525.

[178] Wang, Yan, Yang, Jie, Liu, Hongbo, Chen, Yingying, Gruteser, Marco, and Martin, Richard P. "Sensing vehicle dynamics for determining driver phone use". In: *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*. ACM. 2013, pp. 41–54.

[179] Wang, Zi, Guo, Bin, Yu, Zhiwen, Wu, Wenle, Zhang, Jiafan, Wang, Zhu, and Chen, Huihui. "PublicSense: A Crowd Sensing Platform for Public Facility Management in Smart Cities". In: *Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), 2016 Intl IEEE Conferences*. IEEE. 2016, pp. 114–120.

[180] *Weather*. `https://data.gov.ie/data`.

[181] *Weather Buoy Network*. `https://data.gov.ie/dataset/weather-buoy-network`.

[182] White, Jules, Thompson, Chris, Turner, Hamilton, Dougherty, Brian, and Schmidt, Douglas C. "Wreckwatch: Automatic traffic accident detection and notification with smartphones". In: *Mobile Networks and Applications* 16.3 (2011), p. 285.

[183] Wu, Fang-Jing, Luo, Tie, and Liang, Jason Cheah Jia. "A crowdsourced WiFi sensing system with an endorsement network in smart cities". In: *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2015 IEEE Tenth International Conference on*. IEEE. 2015, pp. 1–2.

[184] Wu, Yao, Wu, Yuncheng, Peng, Hui, Chen, Hong, and Li, Cuiping. "MagiCrowd: A crowd based incentive for location-aware crowd sensing". In: *Wireless Communications and Networking Conference (WCNC), 2016 IEEE*. IEEE. 2016, pp. 1–6.

[185] Wu, Yao, Wu, Yuncheng, Zeng, Juru, Chen, Hong, and Li, Cuiping. "PIE: A personalized incentive for location-aware mobile crowd sensing". In: *Computers and Communications (ISCC), 2017 IEEE Symposium on*. IEEE. 2017, pp. 981–986.

[186] Xie, Fei, Yan, Jun, and Shen, Jun. "Towards Cost Reduction in Cloud-Based Workflow Management through Data Replication". In: *Advanced Cloud and Big Data (CBD), 2017 Fifth International Conference on*. IEEE. 2017, pp. 94–99.

[187] Xu, Jia, Rao, Zhengqiang, Xu, Lijie, Yang, Dejun, and Li, Tao. "Mobile Crowd Sensing via Online Communities: Incentive Mechanisms for Multiple Coop-

erative Tasks”. In: *2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE. 2017, pp. 171–179.

[188] Yan, Tingxin, Kumar, Vikas, and Ganesan, Deepak. “Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones”. In: *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM. 2010, pp. 77–90.

[189] Yan, Zhixian, Eberle, Julien, and Aberer, Karl. “Optimos: Optimal sensing for mobile sensors”. In: *Mobile Data Management (MDM), 2012 IEEE 13th International Conference on*. IEEE. 2012, pp. 105–114.

[190] Yang, Chi and Chen, Jinjun. “A scalable data chunk similarity based compression approach for efficient big sensing data processing on cloud”. In: *IEEE Transactions on Knowledge and Data Engineering* 29.6 (2017), pp. 1144–1157.

[191] Yang, Dejun, Xue, Guoliang, Fang, Xi, and Tang, Jian. “Crowdsourcing to smartphones: Incentive mechanism design for mobile phone sensing”. In: *Proceedings of the 18th annual international conference on Mobile computing and networking*. ACM. 2012, pp. 173–184.

[192] Ye, Fan, Ganti, Raghu, Dimaghani, Raheleh, Grueneberg, Keith, and Calo, Seraphin. “Meca: mobile edge capture and analysis middleware for social sensing applications”. In: *Proceedings of the 21st International Conference on World Wide Web*. ACM. 2012, pp. 699–702.

[193] Ye, Zhi, Chen, Xin, and Li, Zhu. “Video based mobile location search with large set of SIFT points in cloud”. In: *Proceedings of the 2010 ACM multimedia workshop on Mobile cloud media computing*. ACM. 2010, pp. 25–30.

[194] Yeh, PS, Serafino, W, Miles, L, Kobler, B, and Menasce, D. “Implementation of CCSDS lossless data compression in HDF. Proc. Of NASA ESTO Conf. 2002, Pasadena, CA, June 11-13”. In: (2002).

[195] Zappatore, Marco, Longo, Antonella, and Bochicchio, Mario A. “Using mobile crowd sensing for noise monitoring in smart cities”. In: *Computer and Energy Science (SpliTech), International Multidisciplinary Conference on*. IEEE. 2016, pp. 1–6.

[196] Zhang, Hengyang, Huang, Tao, Liu, Yunjie, Zhu, Shixiang, Gui, Guan, and Chi, Yuanying. “Senz: A Context Awareness Middleware System Used in Mobile Devices”. In: *Vehicular Technology Conference (VTC Spring), 2017 IEEE 85th*. IEEE. 2017, pp. 1–7.

[197] Zhu, Chunsheng, Sheng, Zhengguo, Leung, Victor CM, Shu, Lei, and Yang, Laurence T. “Toward offering more useful data reliably to mobile cloud from wireless sensor network”. In: *IEEE Transactions on Emerging Topics in Computing* 3.1 (2015), pp. 84–94.

[198] Zhu, Xuan, An, Jian, Yang, Maishun, Xiang, Lele, Yang, Qiangwei, and Gui, Xiaolin. “A Fair Incentive Mechanism for Crowdsourcing in Crowd Sensing”. In: *IEEE Internet of Things Journal* 3.6 (2016), pp. 1364–1372.

[199] Ziftci, Celal, Nikzad, Nima, Verma, Nakul, Zappi, Piero, Bales, Elizabeth, Krueger, Ingolf, and Griswold, William. “Citisense: mobile air quality sensing for individuals and communities”. In: *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*. ACM. 2012, pp. 23–24.

[200] *zlib*. `http://www.zlib.net`.

Appendix A

“SenseAll” Mobile App

This appendix presents the “SenseAll” Android mobile app. This app was developed and installed in users’ mobile devices in order to conduct the sensing activity used throughout this thesis. Therefore, this appendix will show how users can sense using this mobile app.

The different screens in the mobile app are listed with their functionalities in this appendix. These screens are:

1. Login and registration
2. Dashboard
3. Sensing
4. Logout

A.1 Login and Registration Screen

In order for users to use the app, they need to register in the cloud, where their details will be validated. On the first screen (the Login screen), the user will press the “register” button to navigate to the registration screen, where the user will insert the required details. These details are:

1. Username
2. Password
3. Password validation

After filling in the required details, the user will press the “register” button in order to send the details to the cloud and complete the registration process, but first the app will check if all the details are in the desired form. The desired forms are in terms of username and password length and Internet connection. If the app verifies all the details, it will send these details to the cloud encapsulated in an HTTP request. Otherwise, an error message will appear.

When the cloud receives the details, it will insert the user into the database and assign a random ID if this user does not exist. This user ID is the ID used when the user starts sensing. However, if the user is already registered in the cloud, the cloud will send an error message to the app.

If the cloud verifies the user details, the user can go back to the Login screen and insert the details to grant access to the app.

A.2 Dashboard Screen

The Dashboard screen contains several icons (i.e. buttons), each of which represents a smart city domain, such as weather, water, traffic, etc. As an example, one icon is called “Weather”, which will transfer the user to the sensing screen where the user can sense data that are related to the weather context. Then, when these data are sent to the cloud, they are inserted into the weather database. The Dashboard screen is shown below.

In this thesis, the only icon used is a “Roads” icon, with which users can sense data related to road situations such as potholes. If the user presses this icon, the Sensing screen for that icon will appear. The characteristics of the Sensing screen that is related to the “Roads” icon are shown in section A.3.

A.3 Sensing Screen

The Sensing screen is the screen on which users can exploit the device sensors and start to sense. The data produced by the app for all icons contains time, GPS coordinates (latitude and longitude), three-dimensional accelerometer readings, the status of the user (e.g., at a standstill, walking, etc.), photos and voice notes. However, “Roads” is the only icon used throughout this thesis. With this icon, users can send GPS coordinates (latitude and longitude), three-dimensional accelerometer readings, the status of the user (e.g., at a standstill,

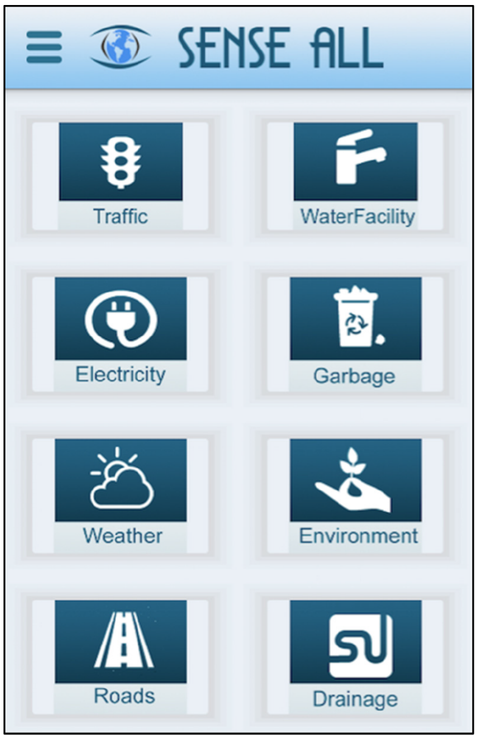


Figure A.1: Dashboard screen

walking, etc.) and photos. The status "Driving" is the default, in order to ensure user safety and reduce the usage of mobile devices while driving. Thus, in order to capture the accelerometer entries, the user will have to either shake the mobile device or simply place it on a surface in a car or on a bike, for example. The user status and photo entries are done by the user. However, a photograph is an optional entry and has a separate activity that is obtained when the user clicks "Photo?" button. GPS coordinates are attached once the user clicks "Send" button.

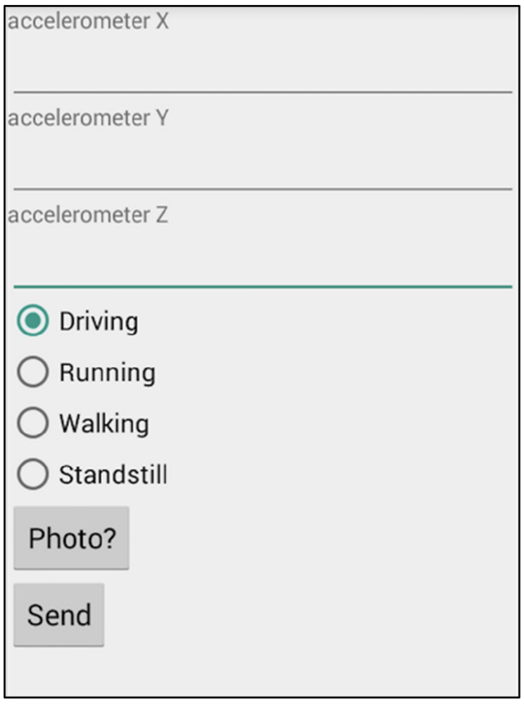


Figure A.2: Sensing screen of “Roads” icon

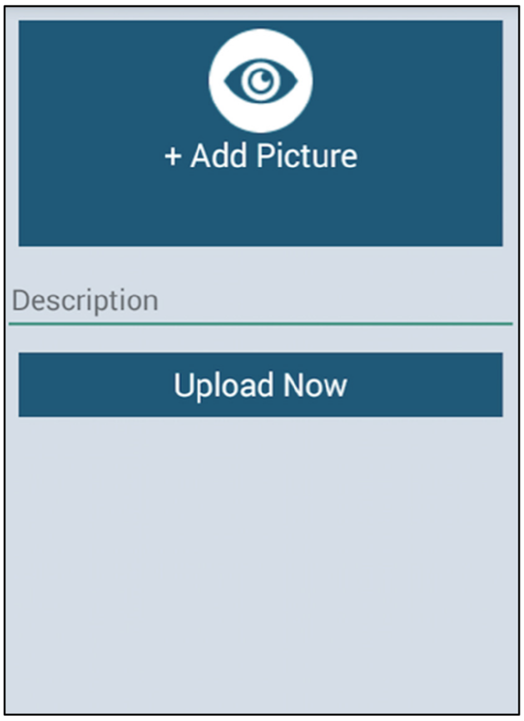


Figure A.3: Adding a photo

A.4 Logout Screen

The user can sign out from the "SenseAll" app by clicking the "Logout" button, as shown below. By clicking this button, a message will be sent to the cloud indicating that this user is no longer active and will not be part of the sensing activity. Then, the app will navigate the user to the Login screen.

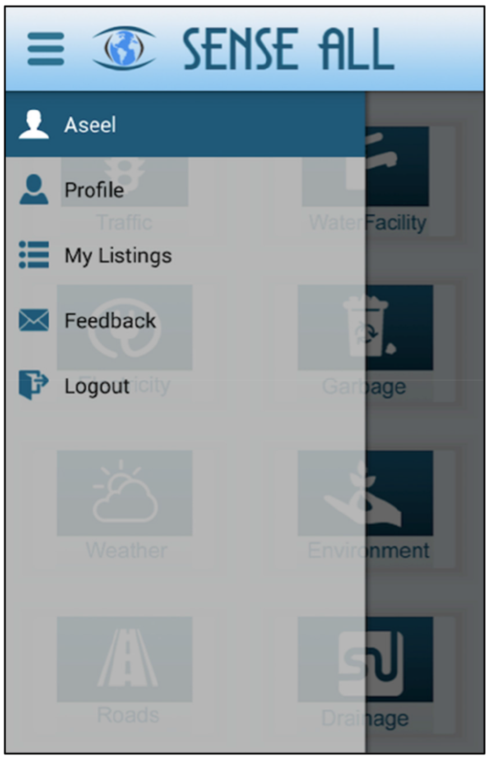


Figure A.4: Logout button

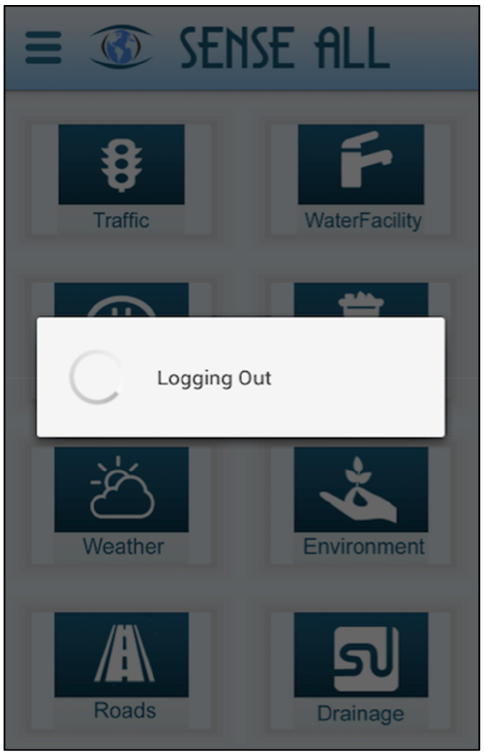


Figure A.5: Logging out

Appendix B

Trust and Reputation Scores

Day 1:

Users	**Users Status**	**Sensing Style**	**Loyalty**	**Similarity**	**Trust value**	**Trusted?**	**Final reputation**
User1	0.2	0.15	0.0	0.0	1.4	YES	1.4
User2	0.15	0.15	0.0	0.0	1.3	YES	$1.3 + 1.4 = 2.7$
	0.15	0.15	0.05	0.0	1.4	YES	
User3	0.15	0.15	0.0	0.0	1.3	YES	1.3
User4	0.1	0.15	0.0	0.0	1.3	YES	1.3
User5	0.05	0.05	0.0	0.0	1.1	NO	-3.0
User6	0.15	0.15	0.0	0.0	1.3	YES	$1.3 + 1.5 = 2.8$
	0.2	0.15	0.05	0.0	1.5	YES	
User7	0.05	0.15	0.0	0.0	1.2	NO	-3.3
User8	0.15	0.15	0.0	0.0	1.3	YES	1.3
User9	N/A	N/A	N/A	N/A	N/A	N/A	N/A
User10	0.15	0.15	0.0	0.0	1.3	YES	1.3
User11	0.2	0.15	0.0	0.0	1.4	YES	$1.4 + 1.8 = 3.2$
	0.2	0.35	0.05	0.0	1.8	YES	
User12	0.2	0.15	0.0	0.0	1.4	YES	1.4 + 1.5 + 1.6 = 4.5
	0.2	0.15	0.05	0.0	1.5	YES	
	0.2	0.15	0.1	0.0	1.6	YES	
User13	0.15	0.15	0.0	0.0	1.3	YES	1.3
User14	0.05	0.15	0.0	0.0	1.2	NO	-3.3

Day 2:

Users	Users Status	Sensing Style	Loyalty	Similarity	Trust value	Trusted?	Final reputation
User1	0.15	0.15	0.0	0.0	1.3	YES	$1.4 + 1.3 = 2.7$
User2	0.15	0.15	0.0	0.1	1.5	YES	$2.7 + 1.5 = 4.2$
User3	0.15	0.15	0.0	0.0	1.3	YES	$1.3 + 1.3 = 2.6$
User4	0.1	0.15	0.0	0.0	1.3	YES	$1.3 + 1.3 = 2.6$
User5	0.2	0.15	0.0	0.0	1.4	YES	$-3.0 + 1.4 +1.5 = -0.1$
User6	0.2	0.15	0.05	0.0	1.5	YES	$2.8 + 1.4+ 1.4 = 5.6$
	0.2	0.15	0.0	0.0	1.4	YES	
	0.15	0.15	0.05	0.0	1.4	YES	
User7	0.05	0.15	0.0	0.0	1.2	NO	$-3.3 - 3.3 = -6.6$
User8	0.2	0.15	0.0	0.1	1.6	YES	$1.3 + 1.6 = 2.9$
User9	0.15	0.15	0.0	0.0	1.3	YES	1.3
User10	0.2	0.15	0.0	0.0	1.4	YES	$1.3 + 1.4 = 2.7$
User11	0.15	0.35	0.0	0.0	1.6	YES	$3.2 + 1.6+ 1.5 = 6.3$
	0.2	0.15	0.05	0.0	1.5	YES	
User12	0.15	0.15	0.0	0.0	1.3	YES	$4.5 + 1.3+ 1.5 = 7.3$
	0.2	0.15	0.05	0.0	1.5	YES	
User13	0.05	0.15	0.0	0.0	1.2	NO	$1.3 - 3.3 = -2.0$
User14	0.15	0.15	0.0	0.0	1.3	YES	$-3.3 + 1.3 = -2.0$

Day 3:

Users	Users Status	Sensing Style	Loyalty	Similarity	Trust value	Trusted?	Final reputation
User1	0.2	0.15	0.0	0.0	1.4	YES	$2.7 + 1.4 = 4.1$
User2	0.2	0.15	0.0	0.1 (8)	1.6	YES	$4.2 + 1.6 = 5.8$
User3	0.15	0.15	0.0	0.0	1.3	YES	$2.6 + 1.3 = 3.9$
User4	0.05	0.15	0.0	0.0	1.2	NO	$2.6 - 3.3 = -0.7$
User5	0.15	0.15	0.0	0.0	1.3	YES	$-0.1 + 1.3 = 1.2$
User6	0.15	0.15	0.0	0.1 (12)	1.5	YES	$5.6 + 1.5 = 7.1$
User7	0.15	0.15	0.0	0.0	1.3	YES	$-6.6 + 1.3 = -5.3$
User8	0.2	0.15	0.0	0.1	1.6	YES	$2.9 + 1.6 = 4.5$
User9	0.1	0.15	0.0	0.0	1.3	YES	$1.3 + 1.3 = 2.6$
User10	0.15	0.15	0.0	0.0	1.3	YES	$2.7 + 1.3 = 4.0$
User11	0.2	0.15	0.0	0.0	1.4	YES	$6.3 + 1.4 + 1.5 = 9.2$
	0.2	0.15	0.05	0.0	1.5	YES	
User12	0.2	0.15	0.0	0.1	1.6	YES	$7.3 + 1.6 = 8.9$
User13	0.15	0.15	0.0	0.0	1.3	YES	$-2.0 + 1.3 = -0.7$
User14	0.2	0.15	0.0	0.0	1.4	YES	$-2.0 + 1.4 = -0.6$

Day 4:

Users	Users Status	Sensing Style	Loyalty	Similarity	Trust value	Trusted?	Final reputation
User1	0.1	0.15	0.0	0.0	1.3	YES	$4.1 + 1.3 = 5.4$
User2	0.15	0.15	0.0	0.0	1.3	YES	$5.8 + 1.3 + 1.6 = 8.7$
User3	0.15	0.15	0.05	0.1	1.6	YES	3.9
	N/A	N/A	N/A	N/A	N/A	N/A	
User4	0.15	0.15	0.0	0.0	1.3	YES	$-0.7 + 1.3 = 0.6$
User5	0.2	0.15	0.0	0.0	1.4	YES	$1.2 + 1.4 = 2.6$
User6	0.15	0.15	0.0	0.0	1.3	YES	$7.1 + 1.3 + 1.4 = 9.8$
	0.15	0.15	0.05	0.0	1.4	YES	
User7	0.2	0.15	0.0	0.0	1.4	YES	$-5.3 + 1.4 = -3.9$
User8	0.2	0.15	0.0	0.1	1.6	YES	$4.5 + 1.6 = 6.1$
User9	0.1	0.15	0.0	0.0	1.3	YES	$2.6 + 1.3 = 3.9$
User10	0.2	0.15	0.0	0.0	1.4	YES	$4.0 + 1.4 = 5.4$
User11	0.15	0.15	0.0	0.0	1.3	YES	$9.2 + 1.3 = 10.5$
User12	0.15	0.15	0.0	0.0	1.3	YES	$8.9 + 1.3 + 1.5 = 11.7$
	0.2	0.15	0.05	0.0	1.5	YES	
User13	0.2	0.15	0.0	0.0	1.4	YES	$-0.7 + 1.4 = 0.7$
User14	0.15	0.15	0.0	0.0	1.3	YES	$-0.6 + 1.3 = 0.7$

Day 5:

Users	Users Status	Sensing Style	Loyalty	Similarity	Trust value	Trusted?	Final reputation
User1	0.15	0.15	0.0	0.0	1.3	YES	5.4 + 1.3 = 6.7
User2	0.2	0.15	0.0	0.1 (8)	1.6	YES	$8.7 + 1.6 = 10.3$
User3	0.2	0.15	0.0	0.1 (8)	1.6	YES	$3.9 + 1.6 = 5.5$
User4	0.2	0.15	0.0	0.0	1.4	YES	$0.6 + 1.4 = 2.0$
User5	0.05	0.15	0.0	0.1 (11)	1.3	YES	2.6 + 1.3 = 3.9
User6	0.15	0.15	0.0	0.0	1.3	YES	$9.8 + 1.3 + 1.4 = 12.5$
	0.15	0.15	0.05	0.0	1.4	YES	
User7	0.15	0.15	0.0	0.0	1.3	YES	$-3.9 + 1.3 = -2.6$
User8	0.2	0.15	0.0	0.1	1.6	YES	$6.1 + 1.6 = 7.7$
User9	N/A	N/A	N/A	N/A	N/A	N/A	3.9
User10	0.15	0.15	0.0	0.0	1.3	YES	$5.4 + 1.3 = 6.7$
User11	0.15	0.15	0.0	0.0	1.3	YES	$10.5 + 1.3 + 1.4 = 13.2$
	0.05	0.15	0.05	0.1	1.4	YES	
User12	0.2	0.15	0.0	0.0	1.4	YES	$11.7 + 1.4 + 1.4 = 14.5$
	0.15	0.15	0.05	0.0	1.4	YES	
User13	0.1	0.05	0.0	0.0	1.2	NO	$0.7 - 3.3 = -2.6$
User14	0.15	0.15	0.0	0.0	1.3	YES	$0.7 + 1.3 = 2.0$

Day 6:

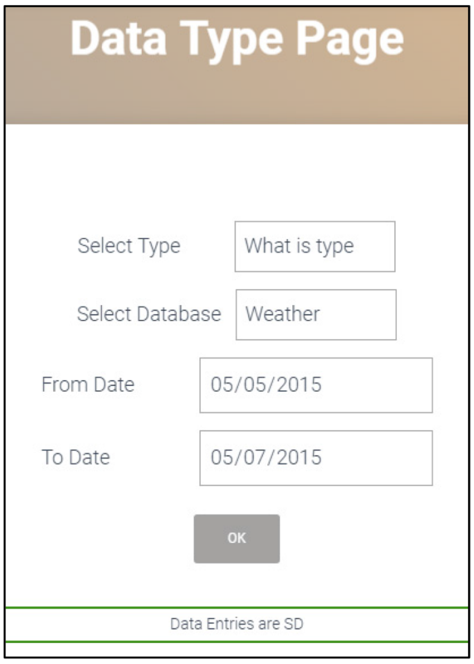
Users	Users Status	Sensing Style	Loyalty	Similarity	Trust value	Trusted?	Final reputation
User1	0.15	0.15	0.0	0.0	1.3	YES	$6.7 + 1.3 = 8.0$
User2	0.2	0.15	0.0	0.0	1.4	YES	$10.3 + 1.4 = 11.7$
User3	0.2	0.15	0.0	0.1 (8)	1.6	YES	$5.5 + 1.6 +$ $1.4 = 8.5$
	0.15	0.15	0.05	0.0	1.4	YES	
User4	0.2	0.15	0.0	0.0	1.4	YES	$2.0 + 1.4 = 3.4$
User5	0.05	0.15	0.0	0.0	1.2	NO	$3.9 - 3.3 = 0.6$
User6	0.05	0.05	0.0	0.0	1.1	NO	$12.5 - 3.0 = 9.5$
User7	0.2	0.15	0.0	0.0	1.4	YES	$-2.6 + 1.4 = -1.2$
User8	0.15	0.15	0.0	0.1	1.5	YES	$7.7 + 1.5 = 9.2$
User9	0.1	0.05	0.0	0.0	1.2	NO	$3.9 - 3.3 = 0.6$
User10	0.15	0.15	0.0	0.1	1.5	YES	$6.7 + 1.5 = 8.2$
User11	0.15	0.35	0.0	0.0	1.6	YES	$13.2 + 1.6 = 14.8$
User12	0.15	0.15	0.0	0.0	1.3	YES	$14.5 + 1.3 = 15.8$
User13	0.15	0.35	0.0	0.0	1.6	YES	$-2.6 + 1.6 = -1.0$
User14	0.2	0.15	0.0	0.0	1.4	YES	$2.0 + 1.4 = 3.4$

Day 7:

Users	Users Status	Sensing Style	Loyalty	Similarity	Trust value	Trusted?	Final reputation
User1	0.15	0.15	0.0	0.0	1.3	YES	$8.0 + 1.3 = 9.3$
User2	0.2	0.15	0.0	0.1	1.6	YES	$11.7 + 1.6 = 13.3$
User3	0.1	0.15	0.0	0.0	1.3	YES	$8.5 + 1.3 = 9.8$
User4	0.15	0.15	0.0	0.0	1.3	YES	$3.4 + 1.3 = 4.7$
User5	0.2	0.15	0.0	0.0	1.4	YES	$0.6 + 1.4 = 2.0$
User6	0.05	0.15	0.0	0.0	1.2	NO	$9.5 - 3.3 = 6.2$
User7	0.15	0.15	0.0	0.0	1.3	YES	$-1.2 + 1.3 = 0.1$
User8	0.2	0.15	0.0	0.1	1.6	YES	$9.2 + 1.6 = 10.8$
User9	0.1	0.15	0.0	0.0	1.3	YES	$0.6 + 1.3 = 1.9$
User10	0.2	0.15	0.0	0.0	1.4	YES	$8.2 + 1.4 = 9.6$
User11	0.15	0.35	0.0	0.0	1.6	YES	$14.8 + 1.6 = 16.4$
User12	0.2	0.15	0.0	0.0	1.4	YES	$15.8 + 1.4 = 17.2$
User13	0.15	0.15	0.0	0.0	1.3	YES	$-1.0 + 1.3 = 0.3$
User14	0.2	0.15	0.0	0.0	1.4	YES	$3.4 + 1.4 = 4.8$

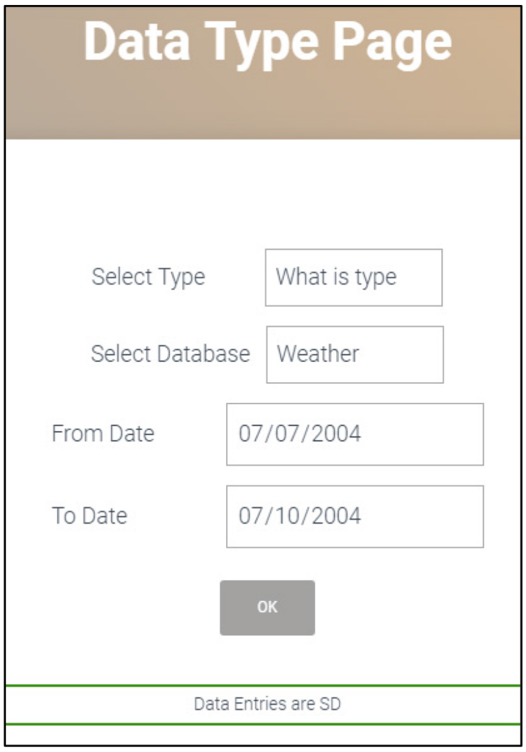
Appendix C

Consumers Interface Screenshots


Data Type Page

Select Type	What is type
Select Database	Weather
From Date	05/05/2015
To Date	05/07/2015

Figure C.1: Screenshot of the Data Type page (first request)

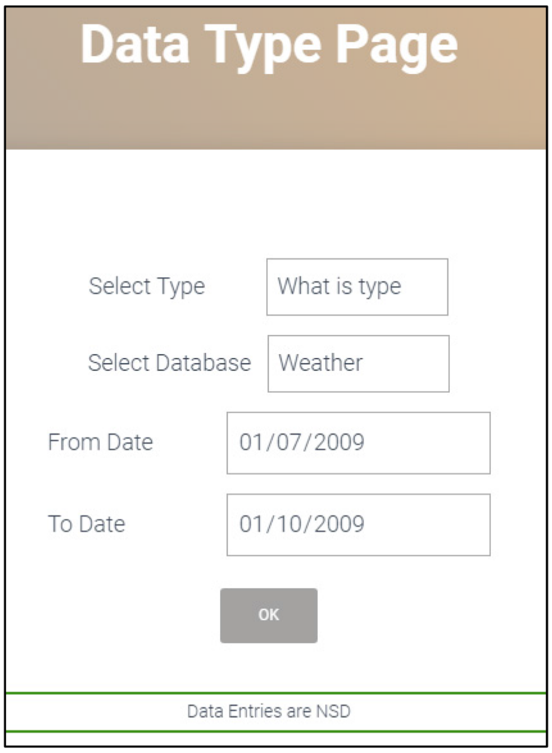


Data Type Page

Select Type	What is type
Select Database	Weather
From Date	07/07/2004
To Date	07/10/2004

Data Entries are SD

Figure C.2: Screenshot of the Data Type page (second request)

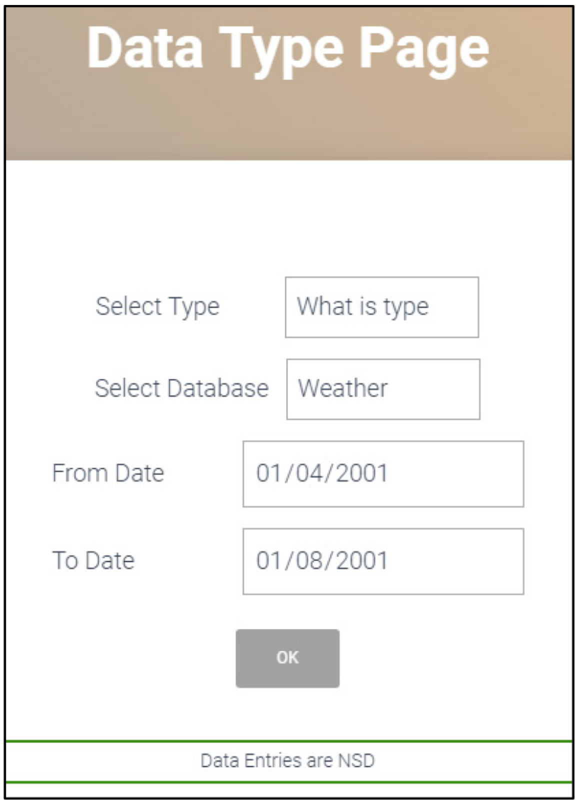


Data Type Page

Select Type	What is type
Select Database	Weather
From Date	01/07/2009
To Date	01/10/2009

Data Entries are NSD

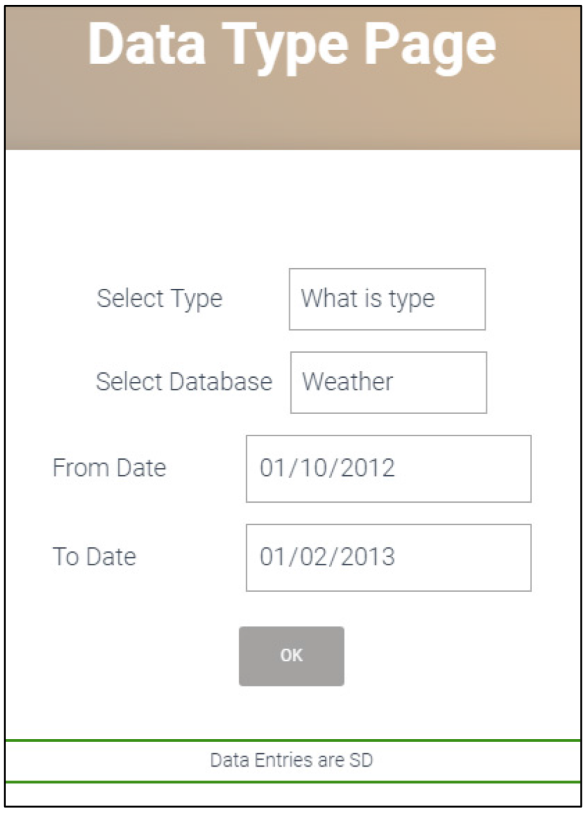
Figure C.3: Screenshot of the Data Type page (third request)


Data Type Page

Select Type	What is type
Select Database	Weather
From Date	01/04/2001
To Date	01/08/2001

Data Entries are NSD

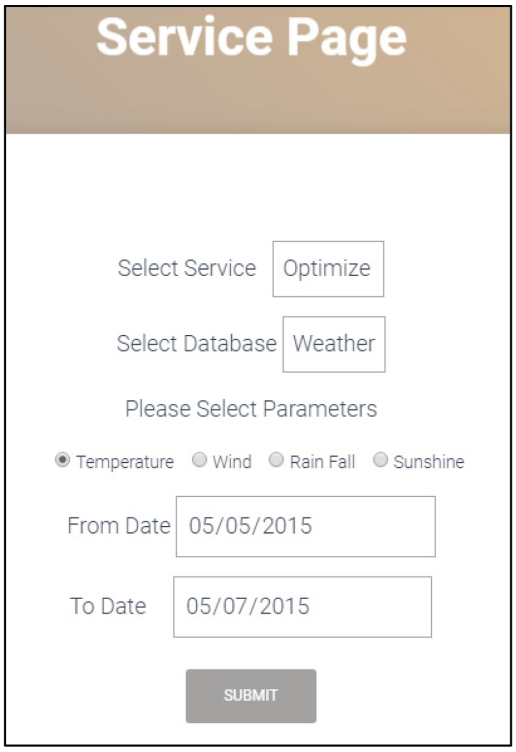
Figure C.4: Screenshot of the Data Type page (fourth request)


Data Type Page

Select Type	What is type
Select Database	Weather
From Date	01/10/2012
To Date	01/02/2013

Data Entries are SD

Figure C.5: Screenshot of the Data Type page (fifth request)



Service Page

Select Service: Optimize

Select Database: Weather

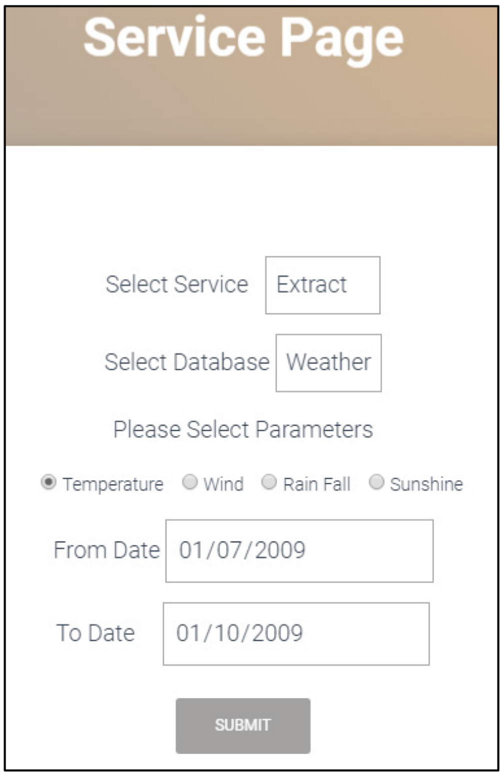
Please Select Parameters

[✓ Temperature] Wind Rain Fall Sunshine

From Date: 05/05/2015

To Date: 05/07/2015

Figure C.6: Screenshot of the Service page using “Optimize” service



Service Page

Select Service: Extract

Select Database: Weather

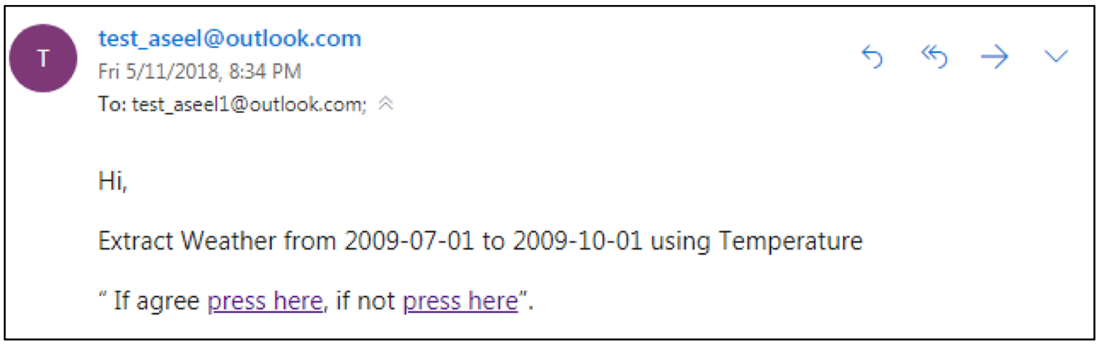
Please Select Parameters

[✓ Temperature] Wind Rain Fall Sunshine

From Date: 01/07/2009

To Date: 01/10/2009

Figure C.7: Screenshot of the Service page using “Extract” service



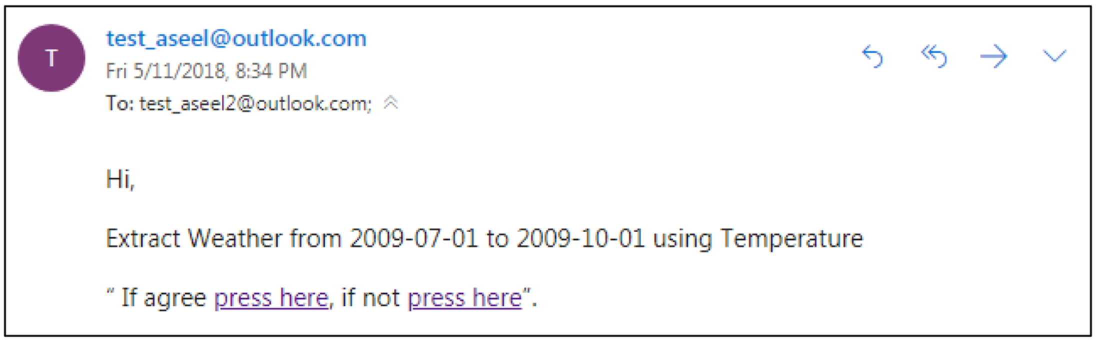
test_aseel@outlook.com
Fri 5/11/2018, 8:34 PM
To: test_aseel1@outlook.com;

Hi,

Extract Weather from 2009-07-01 to 2009-10-01 using Temperature

" If agree press here, if not press here".

Figure C.8: Screenshot of the notification email received by the second consumer "test_aseel1@outlook.com"



test_aseel@outlook.com
Fri 5/11/2018, 8:34 PM
To: test_aseel2@outlook.com;

Hi,

Extract Weather from 2009-07-01 to 2009-10-01 using Temperature

" If agree press here, if not press here".

Figure C.9: Screenshot of the notification email received by the third consumer "test_aseel2@outlook.com"

Appendix D

Symbols and Definitions

Symbol	**Definition**
S	User status (i.e. not moving, walking, or moving fast).
SS	Sensing Style (i.e. location, voice, photo, accelerometer).
N_u	Loyalty factor for user “u”.
S’	Similarity function, returns “0” for no similarity and “0.1” for similarity.
Sim	Similarity factor in the trust service.
T_u	Trust value for user “u”.
Rep$_u$	Reputation value for user.
sim$_u$	Similarity weight for user ”u” in the scheduler service
$d_{i,y}$	data entries that start from entry “i” to entry “y”
T	Time
AR	Access Rate
SD	Immutable part
NSD	Mutable part