CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2015; **00**:1–20
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

A Service Oriented Cloud-based Architecture for Mobile Geolocated Emergency Services

C. Rossi*, M. H. Heyi, F. Scullino

Istituto Superiore Mario Boella (ISMB), Torino, Italy

SUMMARY

Despite the growing development of space-based systems aimed at monitoring and studying natural hazards, they continue to harm humankind worldwide, causing enormous human and economic losses. In a view of improving the effectiveness and the timeliness of existing emergency systems, we propose a service oriented cloud-based software architecture for mobile sensing applications. Exploiting existing Global Navigation Satellite Systems (GNSS) and public Cloud Computing services, we implement a set of mobile geolocated services enabling mobile devices to send real-time in-field observations. Such observations could be used by authorities and first responders for both Early Warning and Emergency Response services complementing in real-time the situational assessment provided by existing means, e.g., remote sensed information and in-situ sensors. We propose a Service Oriented Architecture that can be used as a reference for implementing the back-end of mobile applications requiring to send crowdsourced geolocated reports. We fully implement a real application and we evaluate its performances with Microsoft Azure varying the user load and the main deployment parameters. Our results can be taken as reference to assess the capability of future applications with similar requirements. Copyright © 2015 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: Cloud; mobile sensing; GNSS; performance; evaluation

1. INTRODUCTION

According to the Emergency Events Database (EM-DAT) [1], disasters caused by natural hazards summed up around 2.6 million causalities and up to 2.424 trillion of economic losses between 1980 and 2015. Among all natural hazards, flood is the biggest one with respect to recorded events and affected people, and its future impacts will be exacerbated by climate change effects. Between 1980 and 2015 more than 3.880 flood events occurred, causing around 232.800 fatalities and affecting more than 3 billion people. The associated total economic losses for the same period are estimated around 603 billion. According to Ciscar et al. [2], the increase in direct damages from future river flooding in 60 years will double compared to the annual average damages in the period 19611990. The Federal Emergency Management Agency (FEMA) of the United States stated that for every

*Correspondence to: 61, Via Pier Carlo Boggio, 10138 Torino, Italy

1 invested in disaster prevention, 4 to 7 are saved in disaster response. Thus there is an increasing need to improve existing systems by investing in novel solutions for disaster risk reduction, early warning and emergency response. At the same time, an effective prevention and management of disasters through an appropriate involvement of citizen communities and volunteers is considered a key driver for impact reduction of natural hazards.

Nowadays, emergency management systems depend on the collection and on the dissemination of Earth Observations (EO): the gathering of information about planet Earths physical, chemical and biological systems via remote sensing technologies supplemented by earth surveying techniques. The European Copernicus Programme for Earth Observation, Copernicus EMS [4] (formerly known as GIO-EMS, GMES Initial Operations Emergency Management Service) provides geographical mapping services of natural disasters, with accurate geospatial information derived from satellite observations and complemented with in situ or open data sources, whenever available. The Copernicus program also provides a 25 m Digital Elevation Models (DEM) [5] covering all Europe. Copernicus EMS represents a very powerful technology for land monitoring and it can provide several type of maps. Such maps are classified in reference, delineation and grading maps, which are generated prior, during, and after the event, respectively. However, due to both the administrative operations required by the activation procedures, and the time required by the satellites to acquire, process and map the data, some days can pass between the occurrence of an event and its first mapping. This delay is not ideal during rapidly occurring events such as flash floods, posing a significant limit in providing timely and updated maps.

In order to overcome the aforementioned limitations we propose to complement existing information with mobile sensing, i.e., the collection of geolocalized in-field reports that can be sent by both citizens and professionals (volunteers, civil protection, agents, other first responders) through their mobile devices (smartphones, tablets). For certain type of hazards, like floods and fires, such geolocated reports could be used together with Copernicus maps to compute near real-time extent map, thus improving their update frequency and accuracy.

We build on top of existing European solutions, using cloud computing architectures [6] and the mobile sensing paradigm to let users send in-field reports through their mobile devices. In order to increase the accuracy of such reports, we propose to validate and augment the position of mobile devices through a cloud-based service that uses the Data Access Service (EDAS) of the European Geostationary Navigation Overlay Service (EGNOS) [7]. EDAS delivers EGNOS differential corrections for GPS through a specific API, allowing their use for computing the position integrity [9] in the form of the so-called protection levels and for increasing the position accuracy.

The contribution of this paper are (i) the design of a Service Oriented Architecture (SOA) aimed at supporting the realization of mobile applications for emergency services requiring geolocation and in-field reports; (ii) the implementation of the proposed SOA and its deployment in a Could-Computing platform, i.e., Microsoft Azure; (iii) the implementation of a sample cross-platform mobile application for flood monitoring and reporting, (iv) the performance evaluation of both the mobile application and the Cloud web services required to collect geolocalized in-field reports.

Our results demonstrate that the proposed architecture is able to handle big concurrent user-generated content, enabling the crowdsourcing approach under critical and highly dynamic emergency scenarios such as natural hazards.

2. SPACE ASSETS

In the following subsections, we give a brief overview of both Copernicus EMS and of EGNOS/EDAS in a view of allowing the reader to understand the importance of these providers in the proposed architecture.

2.1. Copernicus EMS

Copernicus EMS mapping products provide accurate geospatial information derived from satellite sensing to all actors involved in the management of natural disasters, man-made emergency situations, and humanitarian crises. As soon as an emergency event is activated, Copernicus EMS triggers the production of maps to support the emergency management. The Copernicus EMS service can be used by authorized, associated and general public users. Authorized and associated users can trigger the mapping activation, whereas public users are not authorized to trigger the service, but they can be informed of an activation request through the web portal. Overall, the list of activations is sorted according to the disaster type and structured with an activation code, which is complemented with metadatas including the region, the country, and the activation date for every emergency event. Each activation is advertised through the so-called GeoRSS, which is a RSS feed that reports all activation details including the links to get the final map products and the associated raw data, which are distributed in the form of shapefiles. Any external emergency management system can thus obtain the Copernicus EMS data by subscribing to and processing the GeoRSS feed.

2.2. EGNOS/EDAS

EGNOS is the first pan-European satellite navigation system and it implements a Satellite Based Augmentation System (SBAS), improving the supported satellite navigation system t make it suitable for safety critical applications. EGNOS provides corrections and integrity for thr GPS signal over a broad area centered in Europe. It is composed by two segments: the space segment and the ground segment. The space segment comprises three geostationary (GEO) satellites, which broadcast corrections and integration information for GPS satellites in the L1 frequency. The ground segment comprises a network of different stations and facilities. They collect measurements from GPS satellites and process them in order to compute a sets of corrections, which are transmitted to EGNOS satellites, which in turn broadcast them to users through the so-called Signal-In-Space (SIS). These corrections apply to clock errors for each GPS satellite in view, ephemeris errors, and ionospheric errors. The EGNOS system can also warn users in case of anomalies in the GPS data, which are detected in a very short timeframe.

However, due to obstructions caused by tall buildings, the user may not be able to receive the SIS, which happens very frequently in dense urban scenario. EDAS [7] provides an alternative way to deliver EGNOS data by implementing a service reachable by any ground-based network, including the cellular network. The high-level architecture of EGNOS/EDAS is shown in Figure 1, which sketches the different delivery scheme of the system.

In order to show the output of the EGNOS/EDAS service, we report in Figure 2 (left) the Horizontal Protection Level (HPL) computed through EDAS, and the Horizontal Alert Limit (HAL).

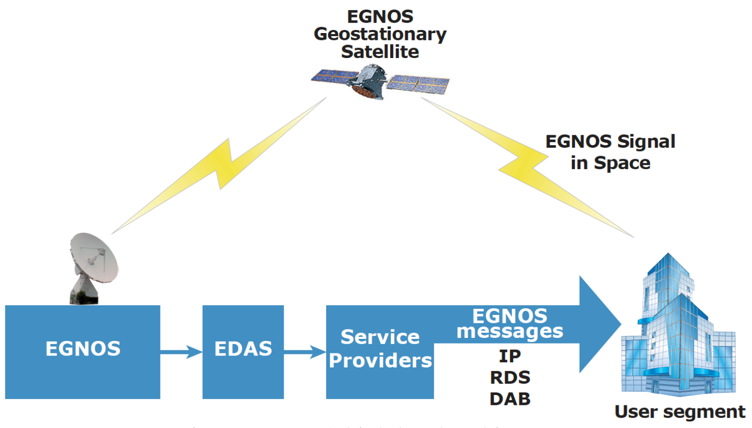
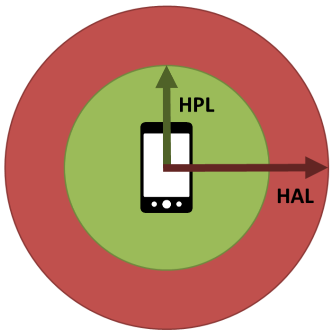


Figure 1. EDAS high level architecture.



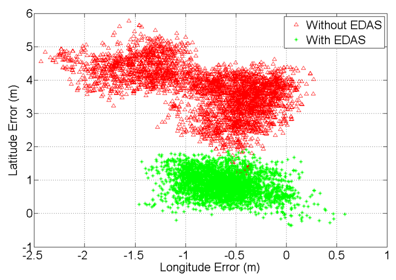


Figure 2. Horizontal Protection Level representation (left), and EGNOS/EDAS accuracy evaluation at a single known location (right)

The HPL provides position integrity, assuring that the true position falls within the area defined by the HPL radius. The HAL is the maximum limit for the HPL, after which the position obtained cannot be considered valid for the application. The HAL was initially conceived for plane navigation applications. Figure 2 (right) shows the position errors with and without EDAS, which we obtain collecting one sample every second for several hours at a single known position. On average, the EDAS augmentation algorithm reduces by 0.18 m and 2.78 m the longitude and the latitude error, respectively. The HAL is not standardized and its value depends on the application. For emergency services we suggest that the HAL should be $\leq 10m$.

3. SYSTEM ARCHITECTURE AND DATA SPECIFICATION

In this section we describe the proposed SOA, which is designed to allow the implementation of geolocated mobile sensing applications targeting emergency management services. We motivate the architectural choices, describe the functionality of each system component, and detail the common data structures.

3.1. System Architecture

An emergency service can be subject to usage peaks at certain times, i.e., during the emergency event, while being completely inactive otherwise. To cope with usage bursts, we propose to

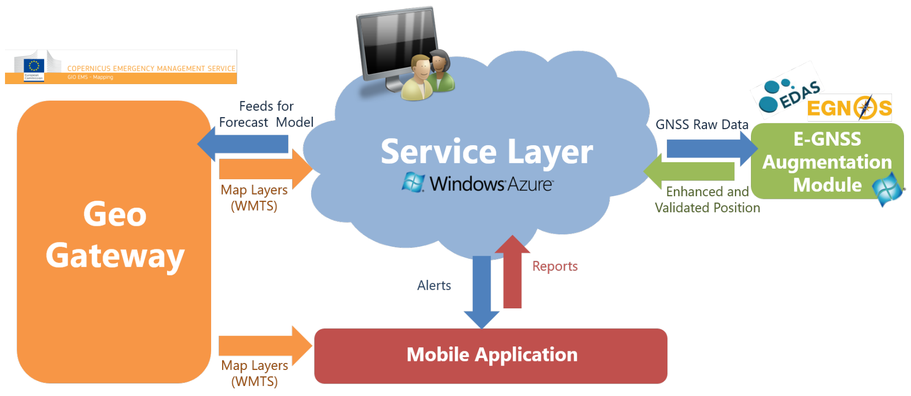


Figure 3. Proposed Service Oriented Architecture (SOA).

implement a cloud-based Service Oriented Architecture (SOA). We break down the SOA in different sub-systems in a view of allowing each one of them to be deployed independently. The main components of the SOA are:

A Service Layer (SL) that acts as the centralized service provider implementing (i) web services to receive/provide geolocalized flood reports from/to mobile devices; (ii) a service that provides geolocated reports of a given area to external services, (iii) a web site that acts as back-end and Decision Support System (DSS) for CPs and Disaster Management Centers (DMCs). The DSS allows the authorities to geographically and interactively visualize all data handled by the SL, see historical data, validate or reject user reports, check user behaviors, control user accounts, see the position of in-field agents, and send massive push notification to mobile devices;

An Augmentation Module (AM) that, leveraging on the EDAS service, receives GNSS raw data from mobile devices and computes an augmented and validated GNSS position, thus providing increased accuracy and integrity to the geolocation of in-field devices;

A multi-platform Mobile Application (MA) that (i) sends real-time geolocalized reports, (ii) receives base maps (street, terrain, satellite) together with additional map layers (extent, nowcast, forecast), and nearby user reports, (iii) receives alerts sent by the authorities via the SL back-end;

A GEO Gateway that implements the interfaces with existing emergency systems like Copernicus EMS and EFAS [10] in order to collect useful data and map layers. The GEO Gateway can also host algorithms and procedures to collect or create useful map layers, which can be disseminated to other system components. Hence, the GEO Gateway also provides the required interfaces to deliver such layers to other components, e.g., mobile devices;

Figure 3 shows the architectural scheme of the SOA, highlighting the four sub-systems and their interactions.

3.2. Data Specification

The main data structure of the proposed architecture is the in-field report. In order to suite different type of natural hazard and to cover both the monitoring and the emergency response phase, we propose a common data structure, which is reported and described in Table I. The risk level is not internationally standardized, hence its definition is left to the application. However, it must cover different risk situations (low, medium, high) including a specific value to declare an ongoing event. A state is associated to each report, which can be submitted, flagged, validated, or rejected. We

Table I. In-field report data structure

Name	**Description**	**Type**
UserID	the user's ID	String
Timestamp	the time at which the report has been submitted	Datetime in ISO 8601
Long	Longitude as given by the GNSS chipset of the device	WGS 84
Lat	Latitudes given by the GNSS chipset of the device	WGS 84
Level	perceived risk level, including ongoing event	String
Status	status of the report (submitted, flagged, validated, rejected)	String
Picture	picture taken by the device at report creation	base64
Ext	extended structure	Object

allow the mobile devices to flag as inappropriate a report submitted by another user, thus allowing a peer-to-peer feedback mechanism. Flagged report are clearly marked and grouped in a separate view in the DSS in order to allow the authorities to easily validate or reject them. The specification of an automatic validation technique for crowdsourced data is outside the scope of this paper and it is left as future work. When the report is sent it goes in the submitted state, from which it can go to the flagged state if reported as inappropriate by another user, or rejected/validated by authorities through the DSS. To manage validation and trust issues, we propose two level of users, namely citizens and professionals, where professional accounts can validate and reject reports made by citizens. Note that everybody can flag a submitted report as inappropriate, provided that it is not validated. We let the back-office interface implemented in the Service Layer to support data structure extensions, which can be implemented using the well-known Entity Attribute Value scheme. Thus, the report and the user profile data structures can be extended according to the application needs.

4. SYSTEM IMPLEMENTATION

In this section we detail a real implementation of the proposed SOA architecture, and we briefly describe the implementation of a sample application, named FLOODIS, which is aimed at improving flood management. The implementation of the GEO Server has been described in [12] and it not detailed further.

4.1. *Service Layer*

In order to implement the Service Layer (SL), we use the 3-tier programming pattern. This pattern divides the program into three logically and architecturally different layers: the Data Access (DA), the Business Logic (BL) and the Presentation (P) layer. The DA deals with the interaction with the database, the BL provides common algorithms and processing, while P implements the User Interface (UI) together with the client-side scripts and validation rules. In Figure 4 (left) we report the 3-tier architecture we implement, together with the main technologies we use. We adopt NHibernate as Object Relation Mapper (ORM), and SQL Azure as database. We deploy the service

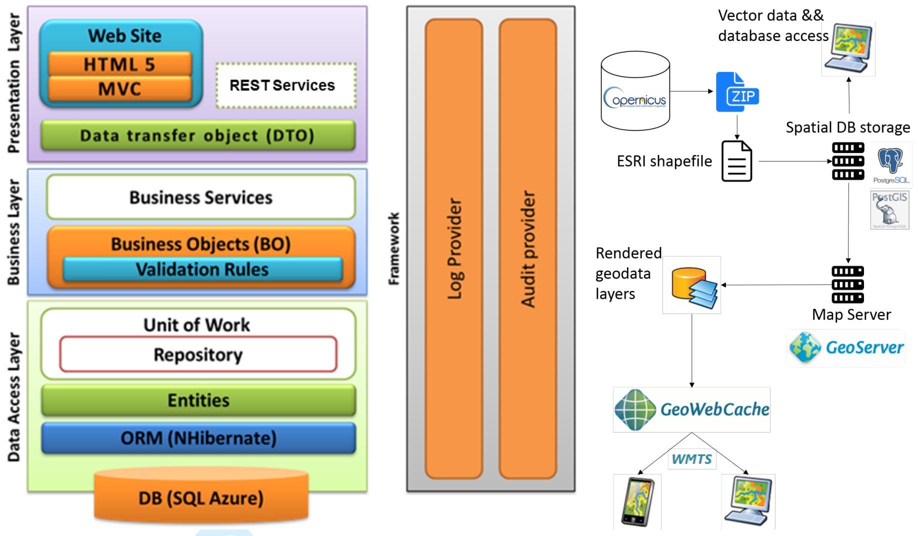


Figure 4. Service Layer (left) and GEO Gateway (right) architecture.

layer using the Platform as a Service (PaaS) approach, which enable auto scaling according to the real-time usage. We use one Microsoft Azure Cloud service for the SL, one Azure SQL database for user reports textual information and positions, and one Azure Blob service for photos and logs. We implement web services for collecting and querying crowdsourced data using the REST architecture, the JSON data format, and the .NET WEB API 2.0 framework. We use OAuth2.0 for authentication and login from mobile devices, and we implement the registration procedure in the web site for all users. To realize the registration, the login, and the user authorization, we rely on the .NET Identity framework 2.1.0. The web site DSS hosted by the SL is implemented using HTML5/CSS3, the Model View Controller (MVC) pattern, and the .NET framework 4.5.1.

4.2. Augmentation Model

Since the majority of mass-market mobile devices are unable to provide GNSS raw data, which are required to perform the augmentation, we implement the Augmentation Module (AM) as a stand-alone service. We apply the same 3-tier patter also for the AM, changing the data access layer to work with an Azure Table [18], which is a NOSQL database. Since the position raw data can be easily mapped into a key-value structure, we select a NOSQL database so as to exploit the greater speed of such solutions. We implement the EGNOS/EDAS correction algorithm as described in our preliminary works [20] in the business layer. The augmentation algorithm allows to improve the position accuracy, achieving the so-called position augmentation, to compute the position integrity, obtaining the HPL, or both.

The system architecture of the AM is composed by three main components:

- The EGNOS Data Access Service (EDAS), which provides EGNOS corrections through an Internet service. We store historical correction in order to support also applications requiring post-processing;
- The Cloud Application (CA), which takes the GNSS raw data from the User Device (UD) and the correction parameters from EDAS to perform the augmentation algorithm, thus obtaining

as output the augmented Position, Velocity and Time (PVT) vector plus the HPL, which are communicated back to the user device.

- The User Device (UD) featuring a GNSS receiver capable of providing raw GNSS data to the CA.

This solution provides two services to users:

- The dissemination of EGNOS/EDAS information, through a web service that can be queried for current or past corrections. Indeed, the AM constantly parse and store the binary streams coming from the EDAS client;
- A cloud-based augmentation algorithm providing PVT and HPL computation. The service can be queried also for augmenting recorder traces, i.e., raw data referring to the past.

4.2.1. Dissemination of EGNOS/EDAS information The EDAS service has been internationally standardized [7], and it can be provided according to different service levels. The AM uses the SL2, which is delivered with the RTCM 3.1 format [8]. Every EDAS message contains a header and a payload, where the latter can contain multiple RTCM messages. Thus, the AM performs the following operations in order to extract the required data:

- It determines the message types and act accordingly;
- If the message type corresponds to 4085 subtype 2, it identifies and get all EGNOS corrections the message contains. EGNOS corrections are 250 bit long and have a 6 bit identifier, which allows 64 EGNOS messages;
- If the message type corresponds to 4085 subtype 5, it identifies and gets the Klobuchar ionospheric corrections;
- If the message type is 1019, it parses and gets the GPS ephemerides;
- It stores all parsed data into the Azure Table, using one table for each data type.

The data type stored are: Fast Correction, Fast Correction Degradation Factor, Long Term Correction, Ionospheric Delay, Degradation Factor for Ionospheric Delay and Long Term correction, IGP mask, Integrity Information, PRN mask, Service Message, GPS ephemerides, Klobuchar ionospheric corrections. The details about the aforementioned data types are omitted for brevity and they can be found in [8]. The added value of having a centralized access to EDAS data though a cloud service is to:

- Avoid to install the EGNOS client locally, i.e., on each user device or embedded system;
- Avoid to register the IP address of each single embedded device, which should be known to EDAS in order to grant the access to its service;
- Simplify the access to EDAS data by providing a web service interface, which can be reachable by all mobile devices;
- Reduce the computational effort to get the EDAS data while providing scalability.

4.2.2. Cloud-based augmentation service The cloud-based augmentation service computes the improved PVT vector and the HPL. The service gets the GNSS raw data and the time, expressed using the Time of Week format (TOW), from the user request, retrieves the corresponding EDAS data form the Azure Table, applies the corrections, and computes the PVT using the Least Mean

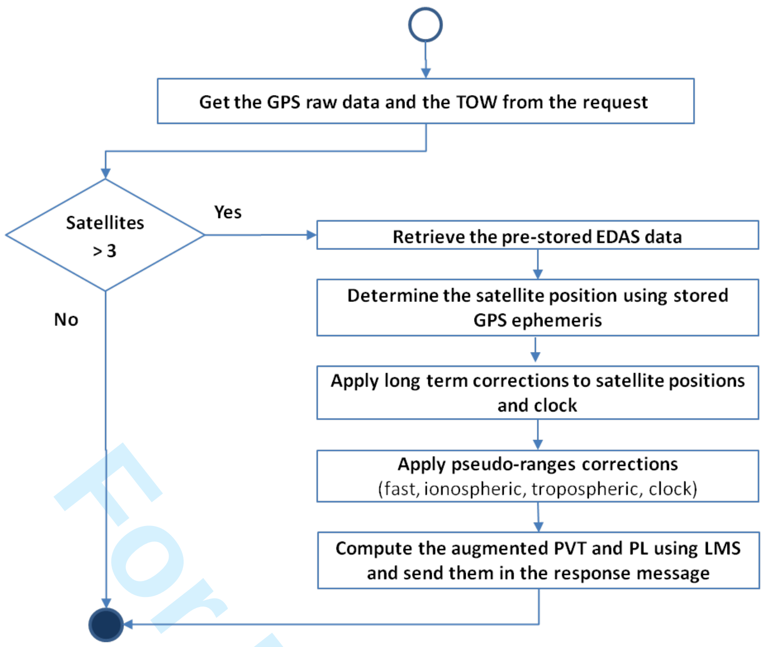

Get the GPS raw data and the TOW from the request
Satellites > 3
Yes
No
Retrieve the pre-stored EDAS data
Determine the satellite position using stored GPS ephemeris
Apply long term corrections to satellite positions and clock
Apply pseudo-ranges corrections (fast, ionospheric, tropospheric, clock)
Compute the augmented PVT and PL using LMS and send them in the response message

Figure 5. Flow chart describing the steps performed by the RESTful web services that provides augmentation and integrity.

Square (LMS) estimator, and the HPL. The AM corrects the pseudoranges for each GPS satellite included in the raw data, excluding the pseudoranges for satellites declared as Do Not Use and Not Monitored by EGNOS. The most important corrections applied by the AM to obtain the PVT are:

- Satellite positions, used to compute the range from the satellites to the GNSS receiver;
- Fast corrections, used to compensate short-term disturbances in the GPS signals, generally due to satellite clocks errors;
- Long-term corrections, used to compensate for the longer-term drift in satellite clocks and the errors in the broadcast satellite orbits;
- Ionospheric corrections, defined as a grid of vertical delays (GIVD) from which a user receiver can determine a slant correction to be applied on each range measurement to compensate for the delay experienced by the signal as it passes through the ionosphere.

An authenticated UD who wants to perform a position augmentation and validation needs to call the RESTful web services exposed by the AM, passing as parameters the TOW, and the raw data coming from the GNSS receiver. After validating the request, the AM retrieves the required data from the cloud storage and performs both the augmentation and the validation algorithms, returning the improved PVT and the HPL to the UD. The steps performed by the AM at each request are show in Figure 5, while the overall process of acquiring the data and providing the service is sketched in Figure 6.

4.3. FLOODIS

FLOODIS is a novel service that provides a faster, flexible and scalable flood emergency system. It leverage on the proposed SOA and on a specific mobile sensing application to let users report on-ground flood status through their mobile devices. A user Report contains a short description, one

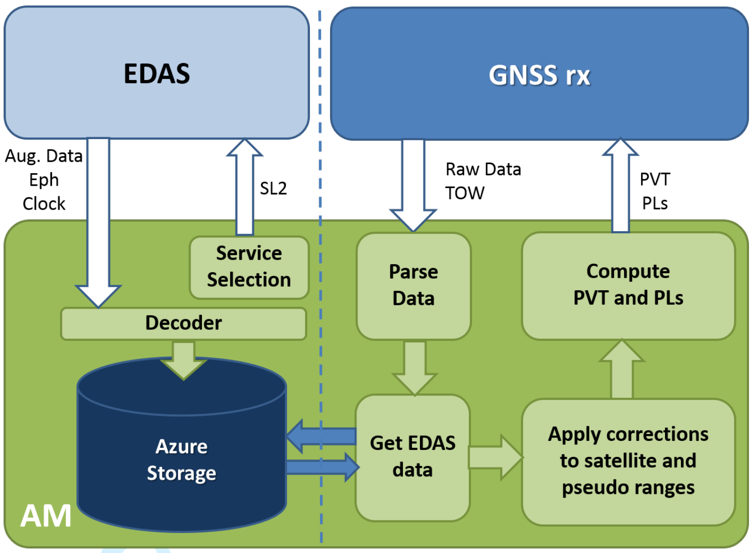


Figure 6. Mid-level block diagram describing the augmentation algorithm structure and data flow.

photo, and an indication of the water level, which can be binary (to indicate the presence of water) or an actual water level estimation for normal and professional accounts, respectively. It integrates these geolocated Reports with EOs in order to provide a Decision Support System (DSS) for public administrations, targeting early flood notification for citizens as well as for emergency teams. FLOODIS improves flood extent map updates by creating flood nowcast maps taking into account crowdsourced data, and computing flood forecasts. The flood nowcast is constructed starting from user Reports, which are collected form citizen and professional users operating in the field, while the flood forecast is computed considering post-processed river flow data from the EFAS [10] Sensor Observation Service (SOS), the EU-DEM [5], the CORINE Land Cover [14], and Open Street Map (OSM) river information. The latter is user to extract the area of interest, which is the area affected by the flood. The flood is modeled using the LISFLOOD-FP [15, 16]. A full specification of FLOODIS is given in [11], while the flood nowcast and forecast algorithms are detailed in [12].

5. PERFORMANCE EVALUATION

In this section we evaluate the performance of our implementation using a real mobile application and a Microsoft Azure cloud deployment of both the AM and of the FLOODIS SL. First, we evaluate the AM with a single mobile application, and then we perform the evaluation of its Cloud service at scale using the Visual Studio Online stress test tool, which creates multiple client using visualized machines. Then, we evaluate the FLOODIS SL using the same methodology. In order to select the range of concurrent users, we take as baseline the daily amount of Tweets generated during recent emergency events occurred in Italy, which spanned from 100 k (2013, flood in Sardinia) up to 260 k (2012, earthquake in the Emilia region), as measured in [24]. Considering the upper bound, which is referred to a very densely populated area, the Tweet rate per second is around 4. Assuming a 3 s service time [†] 12 parallel users would be required to achieved such rate, on average.

[†]This limit was declared as acceptable by the civil protections involved in the FLOODIS project.

Table II. Microsoft Azure Instance types used.

Type	Cores	RAM
P1/Small	1	1.75GB
P2/Medium	2	3.5GB
P3/Large	4	7GB

Since emergency services using crowdsourced data are still at early stages, we challenge on this number by considering up to 200 parallel users in our performance evaluation at scale. For the Cloud deployment we selected the West Europe datacenter, keeping it equal for the agents (clients) used to perform the stress tests. In this way we minimize the contribution of the networking delay, which could introduce high variability. The assessment of performance differences between different data centers is outside the scope of this paper and it is left as future work.

We start by validating the implementation of the AM in a real urban setting with a single client, evaluating its performance in terms of position accuracy and HPL. We deploy the AM in Microsoft Azure, usign the webapp PaaS service with a single *P1* instance. The specification of the instance types that we use is reported in Table II as reference. We drive for 3.5 km at walking speed along an heterogeneous path, which passes along high buildings, below underpasses, and wide roads. We take position samples every 3 sec from the receiver and, at the same time, we perform the position augmentation and validation by calling the AM web-service. For this test we use a mobile phone, namely the Nokia 735, equipped with Windows Phone 8.1, with an external GNSS receiver, namely the SIRF GlobalSat BT-821. The full path is shown in Figure 7 (left), together with the obtained augmented positions and the corresponding circles resulting from the computed HPL. In the bottom (southern) part of the path, which is highlighted in Figure 7 (right), the AM performs very well and consistently due to the absence of tall buildings and obstructions. Conversely, in the top (northern) part of the path, we observe some deviations, which becomes very huge at certain locations, especially in correspondence with underpasses and higher building density. In Figure 8 (left) we plot the empirical cumulative distribution function (ecdf) of the HPLs obtained by removing the top 95th percentile from the entire HPL dataset, reporting a median value of 7.1 m, and with the 90% of the samples well below 8 m. This result must be evaluated together with the maximum threshold allowed by the application, which we set at 10 m because most DEMs - which we require to compute the flood nowcast - have a grid greater than that. Using the same dataset, we also plot in Figure 8 (right) the ecdf of the difference between the latitude and the longitude of the position computed by the SIRF receiver, with the corresponding quantities computed by our AM. We can assume such differences to be the actual improvement, noting a median value of 0.49 m and 2.24 m for the latitude and longitude, respectively. During such test, we also measure all delay components of our systems, namely the 3G network delay, i.e., the time required to send the REST request plus the time to receive the response, the time required to get the EDAS corrections from the Azure Table database, and the time taken by the augmentation algorithm. We report in Table III the average and standard deviation of each delay contribution as well as the total delay achieved. We always obtain an average total delay below the 1 s threshold, thus achieving the standard target performance.

We now evaluate the performance of the AM at scale varying the number of concurring users, the instance number and type. For all tests we set the request timeout time to 10s, and we let each user

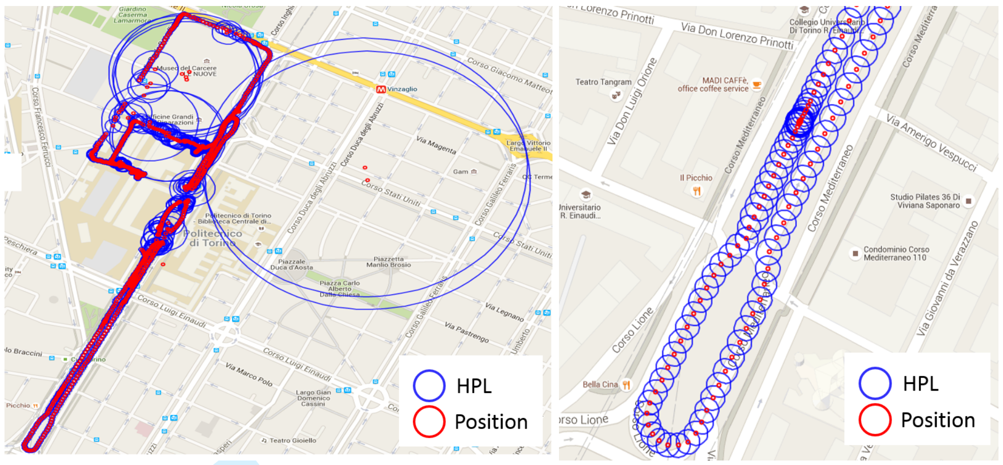


Figure 7. Map of the urban path on whcih the AM is evaluated (left) and a zoom over the southern area (right), showing the base street map, the augmented position, and the HPL.

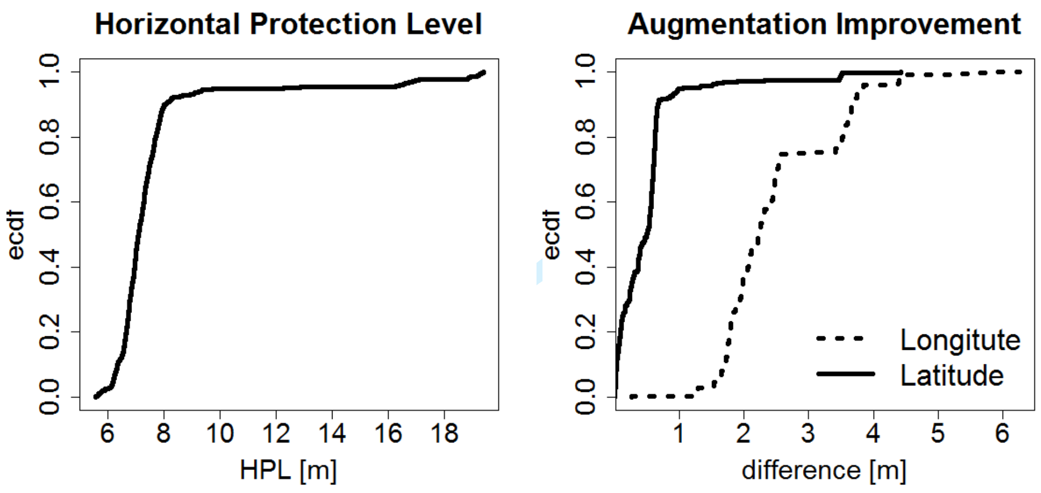


Figure 8. Edcf of the HPL (left), and ecdf of the latitude and the longitude differences (right) obtained using the AM on an urban path.

Table III. Median delay contribution of single device tests

System	Delay contributions (μ,σ) [ms]			
	Network (3G)	DB extraction	AM algorithm	TOTAL
Cloud	(360, 35)	(110,5)	(511,20)	(981,50)

repetitively call the augmentation service for 30 min, averaging the metrics with sampling intervals of 15 s.

We start by assessing the impact of users on a deployment of 20 instances[‡] of type *P1*. As shown in the boxplot reported in Figure 9 (left), the averaged response time grows with the number of parallel users, scoring 0.341 s, 0.486 s, 0.519 s, 0.7692 s with 50, 100, 150, 200 users, respectively. From the boxplot reported in Figure 9 (right) we observe the number of requests per second to

[‡]for a webapp 20 instances is the maximum allowed with a standard Azure subscription.

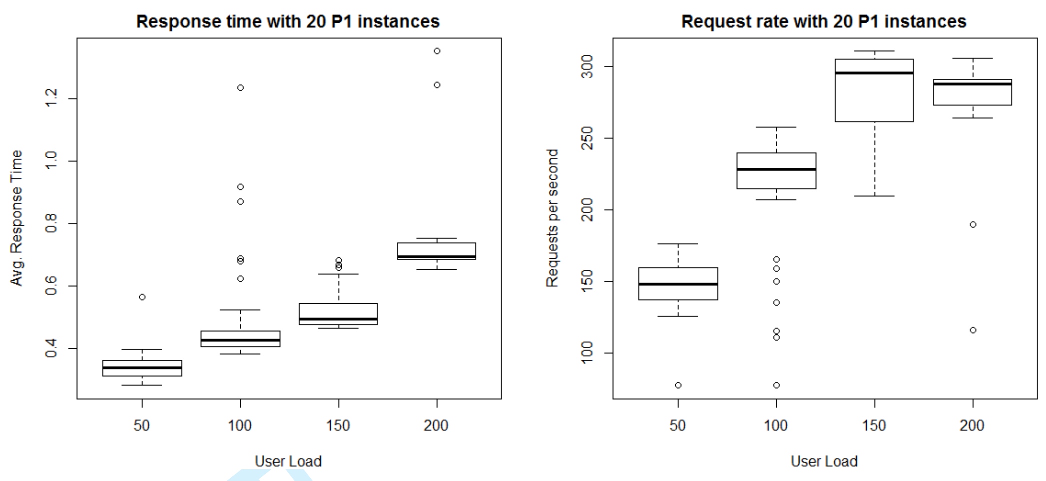


Figure 9. AM response time (left), and requests per second (right) in function of the user load with 20 instances of type *P1*.

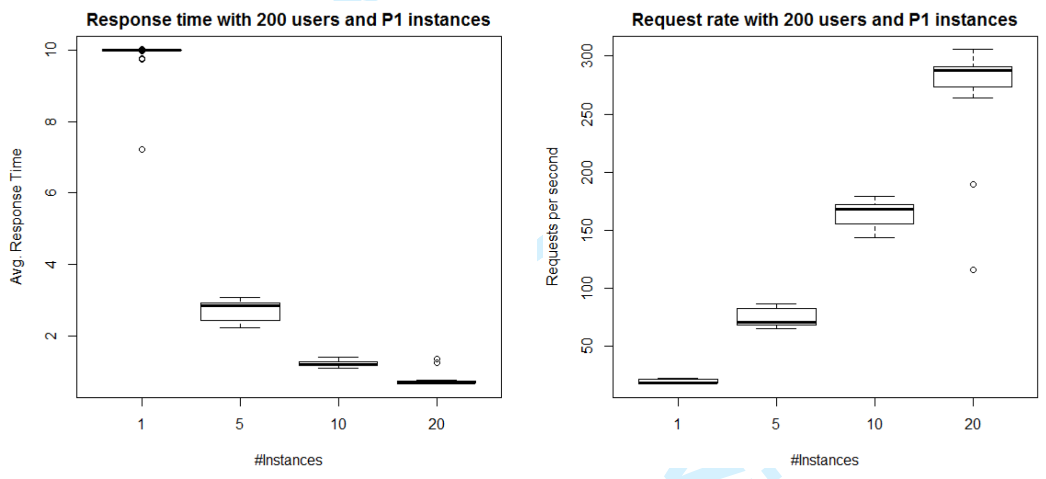


Figure 10. AM response time (left), and requests per second (right) in function of the number of *P1* instances with a fixed load of 200 parallel users.

increase with users up to 282.4 req/s with 150 users, and to fall down to 171.1 req/sec with 200 users. This behavior indicates that a saturation condition is achieved with this setup. A further increase of the user load would not result in an increased response rate.

Then, we keep the user load fixed at 200 while changing the number of instances from 1 to 20. As shown in the boxplot reported in Figure 10 (left), all requests experience a timeout with a single instance, while the mean response time decreases to 2.72 s, 1.217 s, 0.77 s, with 5, 10, 20 instances, respectively. This trend is reflected in the boxplot of Figure 10 (right), which shows the request rate. Indeed, as the number of instances increases from 1 to 20, the average request rate increases from 19.88 req/s up to 270 req/s.

Next, we keep fixed both the user load and the number of instances to 200 and 20, respectively, varying the instance type from *P1* to *P3*. As expected, as the instance cores and RAM grows, the AM is able to respond faster, and also in this case the request rate increases with the instance size, as shown in the boxplots of Figure 11. The average response time is 0.77 s, 0.52 s, and 0.36 s with

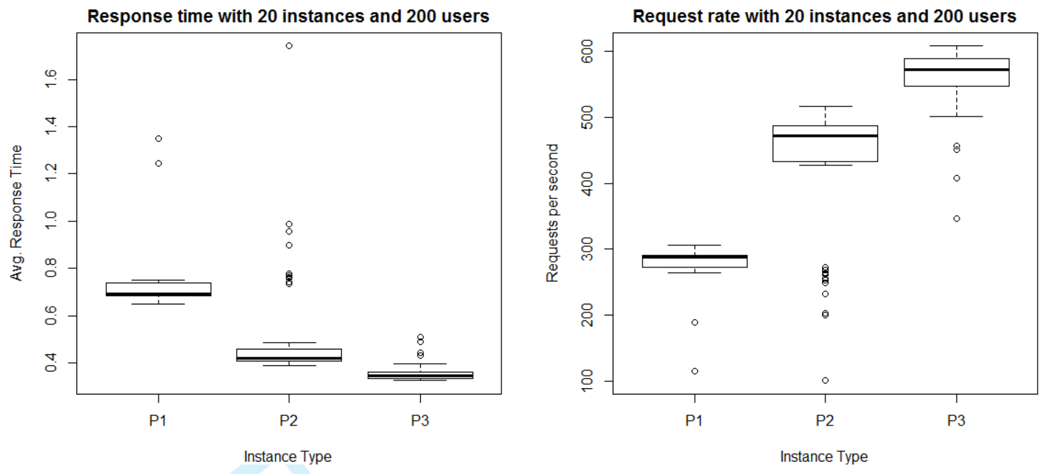


Figure 11. AM response time (left), and requests per second (right) in function of the instance type (*P1*, *P2*, *P3*) with a fixed number of 20 instances with a fixed load of 200 parallel users.

type P1, P2 and P3, respectively, while the request rate is 270.2 req/s, 423.4 req/s, and 555 req/s, respectively.

We now evaluate the FLOODIS SL implementation, which we deploy it with a Microsoft Azure Cloud service. Since FLOODIS uses a SQL database, we keep its capacity fixed to 500 Data Transfer Unit (DTU), which is the median value for Azure premium plans as of 02/2016 [23]. We chose this value high enough to avoid placing the bottleneck at the database. The evaluation of the performances of the FLOODIS SL in function of the DTU and the DB load is outside the scope of this paper and it is left as future work. We test the POST Report service, which have a HTTP request payload of 310 B and 429 KB without and with image, respectively. The HTTP response payload is negligible as it contains only the HTTP response code.

First, we provide a baseline performance computed with a single client, and one *Small* instance selecting as metrics the different delay contributions. The client delay, defined as the delay between the HTTP request and the response measured at the device (client). The network delay, defined as the sum of the network delay of the HTTP request plus the network delay of the HTTP response; and the server delay, which is the delay between the reception of the HTTP request and the dispatch of the HTTP response at the Cloud application. First, we measure the aforementioned delay contributions with one client and without attaching an image to the Report, but varying the client type and the network conditions. Figure 12 (left) shows the results obtained with the mobile application deployed as web application, and run with a Nexus 7 device and a 3G connection throttled at 750 kbps using the Chrome device emulator. With this setup, we obtain a median delay of 267 ms, 255 ms, and 11 ms for client, network, and server, respectively. Clearly, in this case the delay is dominated by the network. We perform the same test using a real Android Nexus 7 device connected with 3G, obtaining slightly better results, namely a median delay of 199 ms, 188 ms, and 11 ms for client, network, and server, respectively; as reported in Figure 12 (right). This improvement is motivated by the greater speed of the 3G connection. We enable the 4G connection on the Nexus device and we perform the POST Report with and without image, obtaining the results shown in Figure 13. Despite the greater speed of the 4G connection, without picture the median delay contributions are very similar to the 3G case, specifically 197 ms, 184 ms, and 11 ms, for client, network, and server,

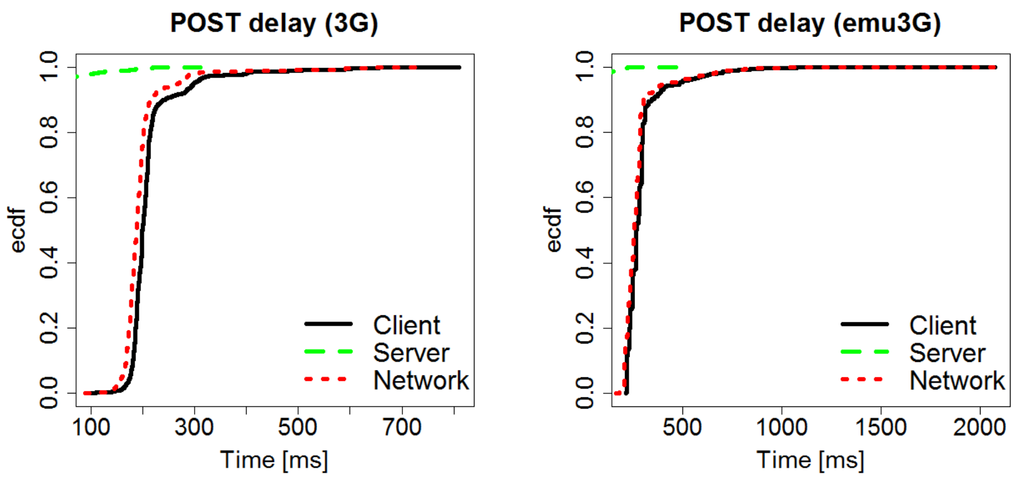


Figure 12. ecdf for delay contributions for the POST Report service without image, used by the Chrome device emulator with 750kbps 3G (left) and with Android Nexus 7 device with a real 3G connection (right).

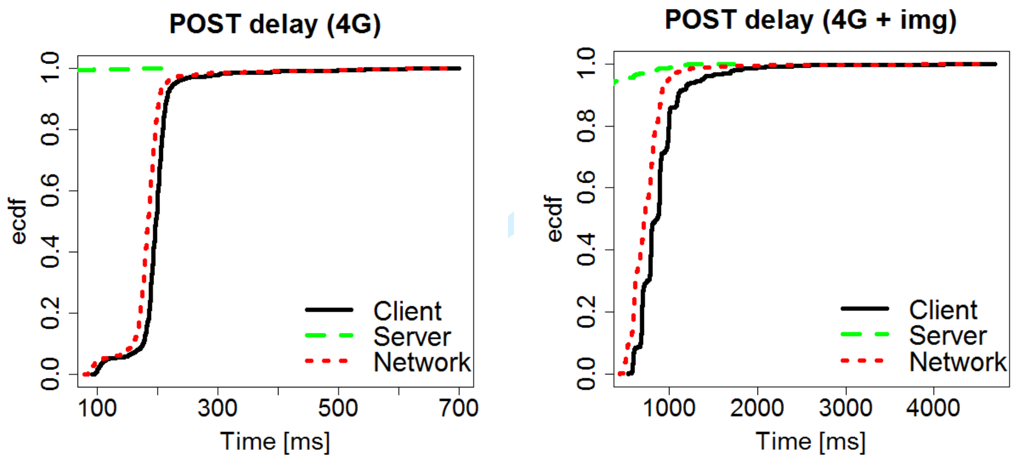


Figure 13. ecdf for delay contributions for the POST Report service without (left) and with (right) attached image, called by and Android Nexus 7 device connected with 4G.

respectively. This is due to the little payload (310 B), that, given the channel allocation scheme of the 4G network, cannot benefit for the greater channel capacity . We include in each report an image of size 429 KB, and we observe that the median delay increases of almost one order of magnitude, specifically at 869 ms, 721 ms, 127 ms, for client, network, and server, respectively. The aforementioned tests, that we summarize in Table IV, should be considered as a best case because they consider only one device interacting with the Cloud.

In order to understand the performance of our FLOODIS SL Cloud deployment at scale, we use again the Azure Stress Test Tool provided by Visual Studio Online. First, we enable the Azure autoscale function from one up to a maximum of five *Small* instances. We simulate 100 parallel users repetitively submitting a report with image for 1 hour, i.e., we let each user continuously post a report with image one at a time. We plot in Figure 14 and Figure 15 the average pages (requests) per second and the average service time, respectively. It is clearly visible that the Azure Cloud Service performs two scaling operations around minute 25 and minute 40, increasing the number of

Table IV. Median delay contribution of single device tests.

Connection Type	Median Delay [ms]		
	Client	Network	Server
3G @ 750kbps	267	255	11
3G	199	188	11
4G	197	184	11
4G with image	869	721	127

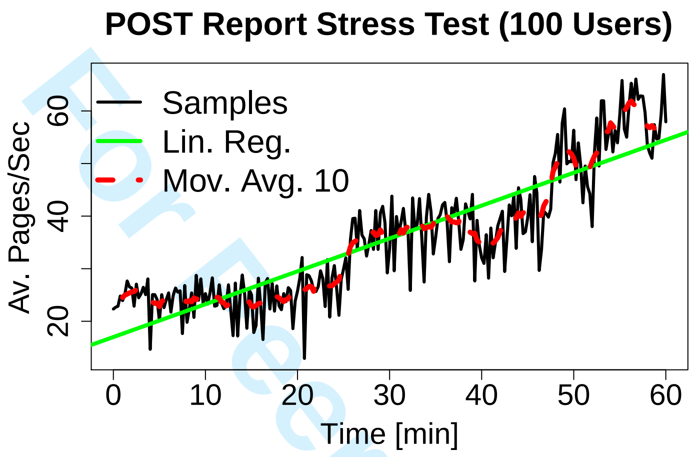


Figure 14. Average Pages per Second achieved with a stress test of the POST Report (with image) service with 100 parallel users. The Azure Cloud Service is deployed with 1 up to 5 *Small* instances, with autoscaling in steps of 2, and with Azure SQL database.

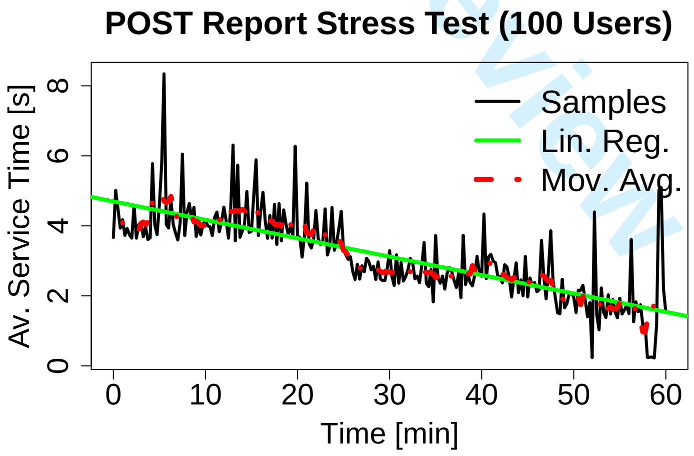


Figure 15. Average Service Time achieved with a stress test of the POST Report (with image) service with 100 parallel users. The Azure Cloud Service is deployed with 1 up to 5 *Small* instances, with autoscaling in steps of 2, and with Azure SQL database.

instances to 3 and 5, respectively. We note that the scaling time between two configurations is around 20 min, despite we set Azure to perform scaling operations at 5 min distance, and that performances are less consistent with the number of active instances.

Similarly to the AM, we further evaluate the FLOODIS SL performances varying the number of users, instances, and instance type. As before, the duration of every test is set to 30 min, and the metrics are sampled every 15 s. We test the post report service with image, because it is the most important use case in crowdsourced applications for emergency services. We start by varying the number of users, while keeping fixed the number of instances, which we set equal to 20, and the instance type, which we set to *Small*. The boxplot reported in Figure 16 (left), shows that the averaged response time grows with the number of parallel users, scoring 0.575 s, 0.7445 s, 1.128 s, 1.650 s with 50, 100, 150, 200 users, respectively. From the boxplot reported in Figure 16 (right) we observe that the number of requests per second to increase with users up to 134.5 with 150 users, and to fall down to 121.1 with 200 users. Similarly to the AM, this behavior suggests that a saturation condition is achieved with this setup. Since both the payload size of the request and the DB are different, we argue that this saturation is due to the number of cores (CPUs) available.

Then, we keep the user load fixed at 200 while changing the number of instances from 1 to 20. As shown in the boxplot reported in Figure 17 (left), all requests experience a timeout with a single instance, while the mean response time decreases to 4.46 s, 2.73 s, and 1.65 s, with 5, 10, and 20 instances, respectively. This trend is reflected in the boxplot of Figure 17 (right), which shows the request rate. Indeed, as the number of instances increases from 1 to 20, the request rate increases from 19.97 req/s up to 121.5 req/s.

Next, we keep fixed both the user load and the number of instances to 200 and 20, respectively, varying the instance type from *Small* to *Large*. As expected, as the instance cores and RAM grows, the FLOODIS SL is able to respond faster, and also in this case the request rate increases with the power of the instance, as shown in the boxplots of Figure 18. The average response time is 1.65 s, 1.15 s, and 0.99 s with type *Small*, *medium* and *Large*, respectively, while the request rate is 121.5 req/s, 165.4 req/s, and 194.1 req/s, respectively.

We note that the AM performs better than the FLOODIS SL for two reasons. First, the payload of the report contains an image, which needs to be uploaded and saved in an Azure Blob storage. Second, the AM uses a NOSQL database (Azure Table), while the FLOODIS SL uses a relational one (Azure SQL). The aforementioned tests provide a first assessment both for the AM and for the FLOODIS SL as separated services when deployed using Microsoft Azure. We omit for brevity the results of the integrated service, i.e., a post report requiring a position augmentation, because the average response time can be approximated with the sum of the two services (SL plus AM). Note that both the SL and the AM must be deployed with the same settings in terms of number and type of instance. We stress that in our evaluation we have used the greatest deployment possible for a normal Azure account, i.e., up to instance P3/Large type and up to 20 instances.

The suitability of the evaluated deployment ultimately depends on the application requirements in terms of response time, throughput (request/second), and user load.

6. CONCLUSIONS

We designed a general architecture for emergency services requiring to gather in real-time crowdsourced information with accurate and validated positioning. We implement a Could based Augmentation Module (AM) for improved positioning and a sample application named FLOODIS:

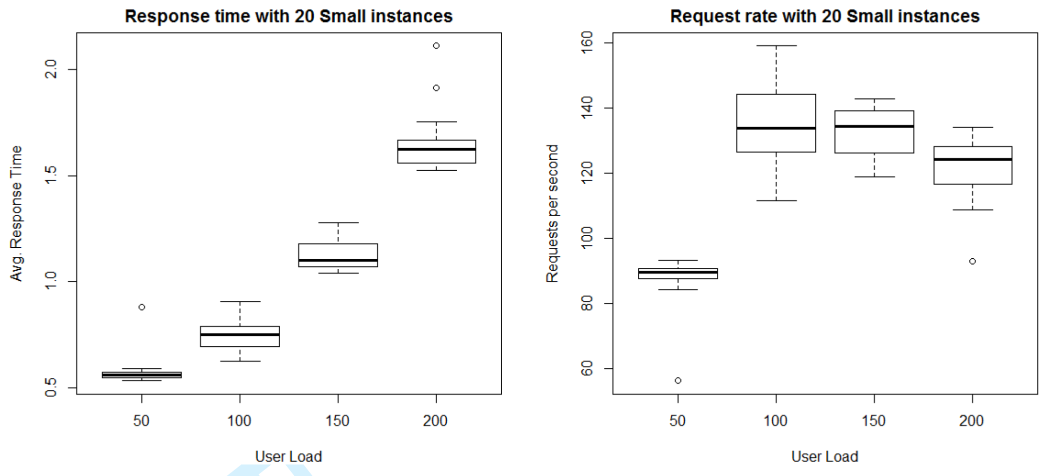


Figure 16. FLOODIS SL response time (left), and requests per second (right) in function of the user load with 20 instances of type *Small*.

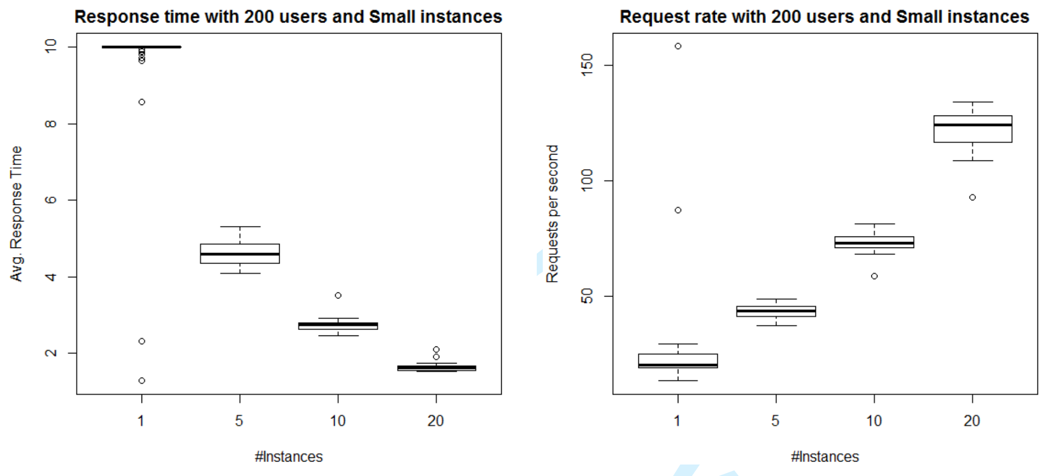


Figure 17. FLOODIS SL response time (left), and requests per second (right) in function of the number of *Small* instances with a fixed load of 200 parallel users.

a novel flood emergency service exploiting mobile sensing and cloud architecture to improve management and operations in case of floods. We evaluate the performance of both the AM and FLOODIS in real conditions using Microsoft Azure Cloud services varying the user load, the instance number and type. The results shows the achievable performances of the proposed architectures, which can be used as reference for future crowdsourced applications targeting a Microsoft Azure deployment, especially for those targeted at mobile emergency services requiring position augmentation.

ACKNOWLEDGMENTS

This work was supported by the European Union through the FLOODIS project (FP7-SPACE-2013-1). We thank Vito Macchia for the development of the FLOODIS mobile application and Alfredo FAvenza for the early code implementing the the Augmentation Module.

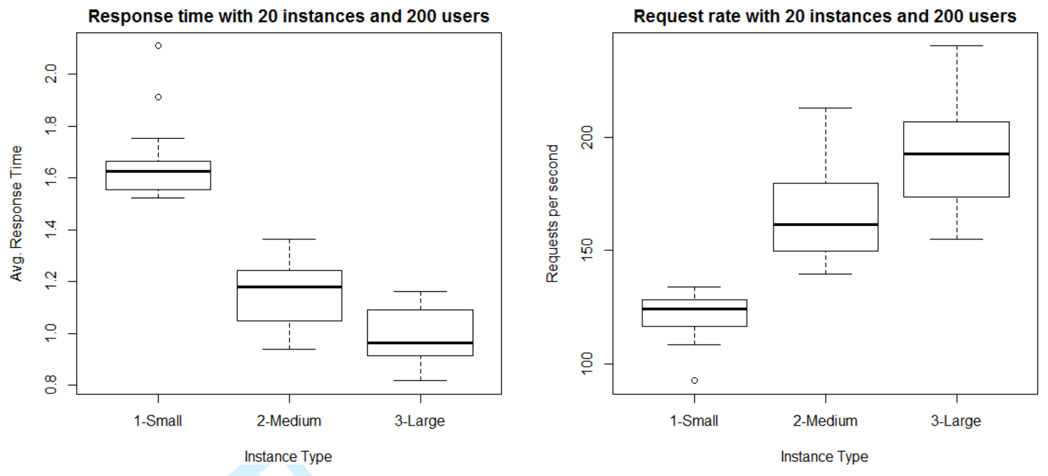


Figure 18. FLOODIS SL response time (left), and requests per second (right) in function of the instance type (*Small*, *Medium*, *Large*) with a fixed number of 20 instances with a fixed load of 200 parallel users.

REFERENCES

1. D. Guha-Sapir, R. Below, Ph. Hoyois, “EM-DAT: The CRED/OFDA International Disaster Database,” www.emdat.be, Universit Catholique de Louvain.
2. J.-C. Ciscar, A. Iglesias, L. Feyen, L. Szab, D. Van Regemorter, B. Amelung, R. Nicholls, P. Watkiss, O. B. Christensen, R. Dankers, L. Garrote, C. M. Goodess, A. Hunt, A. Moreno, J. Richards and A. Soria, “Physical and economic consequences of climate change in Europe.”, http://www.pnas.org/content/108/7/2678, 2010.
3. B. Jongman, S. Hochrainer-Stigler, L. Feyen, J. C. J. H. Aerts, R. Mechler, W. J. W. Botzen, L. M. Bouwer, G. Pflug, R. Rojas, P. J. Ward, “Increasing stress on disaster-risk finance due to large floods,”, *Nature journal on Climate Change*, no. 4, pg. 264268, 2014.
4. Copernicus EMS, http://emergency.copernicus.eu/, accessed the 27/04/2015.
5. The Digital Elevation Model over Europe from the GMES RDA project (EU-DEM), http://www.eea.europa.eu/data-and-maps/data/eu-dem, accessed the 27/04/2015.
6. NIST, “The NIST Definition of Cloud Computing,” September 2011. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, accessed the 17/04/2015.
7. EC, “Data Access Service (EDAS) Service Definition Document (v2.0),” http://ec.europa.eu/DocsRoom/documents/3985/attachments/1/translations/en/renditions/pdf, accessed the 06/10/2015.
8. RTCM standard 10403.1, website http://www.rtcm.org, accessed the 14/10/2015.
9. Position Integrity, http://www.navipedia.net/index.php/Integrity, accessed the 27/04/2015.
10. EFAS, https://www.efas.eu/, accessed the 27/04/2015.
11. C. Rossi, W. Stemberger, C. Bielski, G. Zeug, N. Costa, D. Poletto, E. Spaltro, and F. Dominici, “Coupling Crowdsourcing, Earth Observations, and E-GNSS in a Novel Flood Emergency Service in the Cloud”, IEEE IGARSS, year 2015.
12. C. Rossi, A. Favenza, F. Scullino, V. Macchia, G.L. Spoto, F. Dominici, “Evaluating FLOODIS: Mobile Sensing for a Flood Emergency Service in the Cloud”, IEEE CLOUDTECH, year 2015.
13. Microsoft Azure, “Microsoft Azure Cloud Computing Platform,” http://www.microsoftazure.com, accessed the 27/04/2015.
14. Corine Land Cover Classes, http://sia.eionet.europa.eu/CLC2000/classes, accessed the 27/04/2015.
15. J. M. Van Der Knijffa, J. Younisa, A. P. J. De Rooa “LISFLOOD: a GISbased distributed model for river basin scale water balance and flood simulation,” *International Journal of Geographical Information Science*, vol. 24, issue 2, 2010.
16. P.D. Bates, A.P.J. De Roo, “A simple raster-based model for floodplain inundation,” *Journal of Hydrology*, no. 236, pg.54-77, year 2000.
17. G. Di Baldassarre, “Floods in a Changing Climate: Inundation Modelling,” Cambridge University Press, 2002.

18. B. Calder, J. Wang, A. Ogus, et. al., "Windows azure storage: a highly available cloud storage service with strong consistency," *ACM SOSP*, pages 143-157, year 2011.
19. RTCA, "Minimum Operational Performance Standards for Global Positioning System/Satellite-Based Augmentation System Airborne Equipment," http://www.rtca.org/store_product.asp?prodid=1101, accessed the 13/10/2015.
20. A. Favenza, C. Rossi, M. Pasin, F.Dominici, "A Cloud-based Approach to GNSS Augmentation for Navigation Services," *ACM/IEEE UCC*, year 2014.
21. GPS Interface Specification IS-GPS-200 document, http://www.gps.gov/technical/icwg/, accessed the 13/10/2015.
22. B. W. Parkinson, J. J. Spilker Jr., "Global Positioning System: Theory and Applications, vol. 1," Washington, DC: American Institute of Aeronautics and Astronautics, Inc., 1996.
23. Azure SQL DTU, https://azure.microsoft.com/en-us/pricing/details/sql-database/, accessed the 15/02/2016.
24. Risknet project financed by ALCOTRA. http://www.risknet-alcotra.org/, accessed the 15/02/2016.