2012 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing

SENSAPP as a Reference Platform to Support Cloud Experiments: From the Internet of Things to the Internet of Services

Sébastien Mosser, Franck Fleurey, Brice Morin, Franck Chauvel, Arnor Solberg
SINTEF ICT, Dept. of Networked Systems and Services
Oslo, Norway
first.last@sintef.no

Iokanaan Goutier
SINTEF ICT, Polytech'Lille
Lille, France
first.last@polytech-lille.net

***Abstract*—The Cloud-computing paradigm was considered as a revolution. Thanks to the abstraction of computing resources "in the clouds" this paradigm provides "anything" as a service, on a pay-as-you-go basis. Unfortunately, there is no reference software that one can use to properly compare a cloud approach against others. We propose the SENSAPP platform to tackle this challenge. SENSAPP is designed as a prototypical cloud application and is provided as an open-source service-based application used to store and exploit data collected by the Internet of Things. We propose SENSAPP as a reference implementation to compare different cloud approaches. In this paper we present initial experiments about scalability based on this platform.**

***Keywords*-Cloud-computing; Reference; Experiments; Internet of Things; Internet of Services;**

I. INTRODUCTION

Cloud–Computing [1] was considered as a *revolution*. Taking its root in distributed systems design, this paradigm advocates the share of distributed computing resources designated as "*the cloud*". The main advantage of using a cloud-based infrastructure is the associated scalability property (called *elasticity*). Since a cloud works on a *pay–as–you–go* basis, companies can rent computing resources in an elastic way. A typical example is to temporary increase the server–side capacity of an e–commerce website to avoid service breakdowns during a load peak (*e.g.*, Christmas period). However, there is still a huge gap between the commercial point of view and the technical reality that one has to face when exploiting "*the cloud*". As any emerging paradigm, and despite all its intrinsic advantages, Cloud–Computing still relies on fuzzy definitions[1] and lots of buzzwords (*e.g.*, the overused "*IaaS*", "*PaaS*" and "*SaaS*" acronyms that does not come with commonly agreed definitions).

To clarify the situation, it becomes necessary to define a fixed point in the cloud-computing technological space, that is, *something* to be shared among research groups and used as a comparison reference. From an industrial point of view, the Java *Pet Store* [2] was such a fixed point, specified as a reference application for the J2EE platform. From a research point of view, the component-based development community defined a "common modelling example" to support component system comparisons [3]. The same approach was followed by the *Aspect-Oriented Modelling* (AOM) research community. In 2009, Kienzle *et al.* proposed a common case study for AOM [4] based on a *Car Crash Crisis Management System* (CCCMS). The CCCMS was then exploited in several scientific work as a reference to *(i)* compare AOM approaches against others [5] and *(ii)* reify tool chains when these approaches worked a different level of abstractions [6]. According to Google Scholar (July 2012), up to 32 papers are referring to this case study.

The cloud computing research will benefit from making available similar reference software platforms and applications. We believe a reference application for Cloud-computing must match at least the following criteria:

- C_1: Reality-driven: the application cannot be just a toy example, it needs to be applied to real situations.
- C_2: Implementation: "Theory guides, Experiments decides" is a well known adage. Without a reference implementation, proper comparison of technical achievements will never be possible.
- C_3: Positioning with respect to Cloud-computing challenges: the cloud domain is still young and immature, filled with well-known challenges to be faced in the upcoming years. A reference application must describes how it can be used to support research conducted to tackle theses challenges. This includes key cloud properties such as elasticity, scalability, cost etc (this is further elaborated in section III)

Our contribution in this paper is to describe SENSAPP as a reference application for Cloud-computing. This platform leverages the *Internet of Services* (IoS) and the *Internet of Things* (IoT) to support the definition of innovative applications based on sensors data. Section II describes the SENSAPP platform (addressing criteria C_1 and C_2). We believe this is well suited for the purpose, since such applications typically requires what cloud claim to offer (*e.g.*, "big data", peak periods etc). The appropriateness of SENSAPP is thoroughly elaborated in Section III, positioning SENSAPP *w.r.t.* cloud-

[1]The Cloud–Standard initiative (http://cloud-standards.org/) lists dozens of overlapping standards related to Cloud–Computing. They focus on infrastructure modeling or business modeling.

978-0-7695-4934-7/12 $26.00 © 2012 IEEE
DOI 10.1109/SYNASC.2012.71

Fig. 1. Bike with sensors (*e.g.*, altitude, speed, location) and cameras

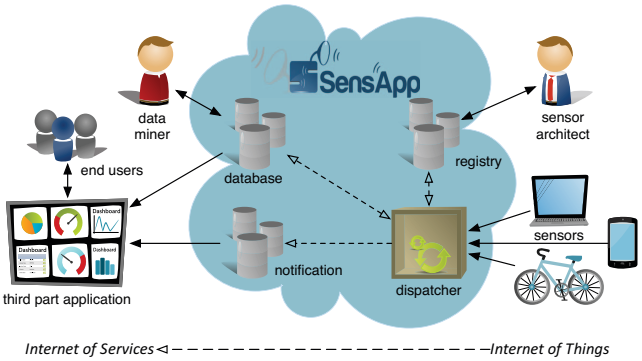


Fig. 2. Overview of the SensApp system

computing challenges (addressing C_3). Section IV describe an initial experiment made with this platform. Finally, section V conclude this paper by exposing immediate perspectives.

II. SensApp, a Cloud Platform for the IoT

This section describes the SensApp platform as a distributed set of services, and then present the reference implementation available as open-source software.

SensApp addresses the IoT domain, which relies on the following assumption: *things* can communicate about their state with other *things*, creating a network of interconnected *things*. Things rely on *sensors* to collect data. For example in an "intelligent building", temperature sensors collect room temperature, radiators (or A/C systems) are remotely activated to regulate such a temperature. In a daily context, smartphones can collect information from their hardware sensors (*e.g.*, user's location, battery level, wifi signal strength) and broadcast them to telephone companies for statistical purpose. Means of transport with accurate sensors (*e.g.*, speed, location) can log context information about a given travel. For example, the bike depicted in Figure 1 holds 12 sensors and 2 cameras[2]. The collected data are then used to collect information about the city surroundings.

A. Coarse-Grained Description

SensApp is designed to support application of the IoT. It provides elementary bricks a software developer can use to define cloud-based IoT application. Figure 2 describes the SensApp platform from a coarse-grained point of view.

We consider three different roles for the users. First, the "sensor architect" is in charge of the definition of the sensors that feed the system. Then, the "data miner" can work with the collected data to exploit them (*e.g.*, inferring new knowledge like air quality as a combination of carbon monoxide and ozone levels). Finally, "end users" (non technical) visualise the collected data through third-party applications that reshape the collected raw data into something they can understand (*e.g.*, air quality displayed according to a color code).

SensApp provides four elementary services to support the definition of IoT applications. The **`registry`** is fed by the sensor architect, and stores information (*e.g.*, description, creation date) about the sensors that are involved in the application. The **`database`** service implements a storage facility for sensor data, providing dedicated queries to expose the collected data to the data miner. The **`notification`** service is used by third party application to subscribe for notification when relevant data are pushed (in the previous example, the application will subscribe for new data collected by the *air quality* sensor, updating the displayed colors according to the latest quality level). Finally, the **`dispatcher`** received the data from the sensors, store the data in the relevant **`database`** according to the information stored in the **`registry`**, and finally triggers the **`notification`** mechanisms with the newly arrived data.

By default, SensApp is deployed as a monolith: all the services are deployed on the very same server. But the described architecture is intrinsically distributed: the services provided by SensApp interacts through well defined remote interfaces, thus it is possible to deploy the platform on multiple servers, as described in Figure 3. In this particular topology, the platform relies on three independent hosts to support its deployment: *(i) localhost*, *(ii)* a private database deployed in SINTEF' premises and *(iii)* a server used as a public demonstration. The configured platform relies on a local dispatcher to receive data, and store the collected data in the private database. The public server is used to support sensor registration and third-party notification.

B. Reference Implementation

We provide an open-source (LGPL) implementation of the SensApp platform, freely available[3]. The services are implemented as REST services [7] exposed on top of the HTTP protocol. Sensor data are represented with the SenML standard [8], used as an intermediary representation of sensor data in the platform. SensApp supports two databases: *(i)*

[2]http://www.youtube.com/watch?v=kia5Vkx59nY

[3]http://sensapp.modelbased.net

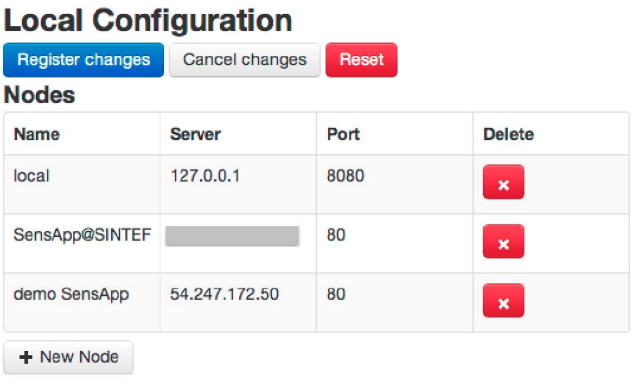


Fig. 3. Topology in SENSAPP administration panel

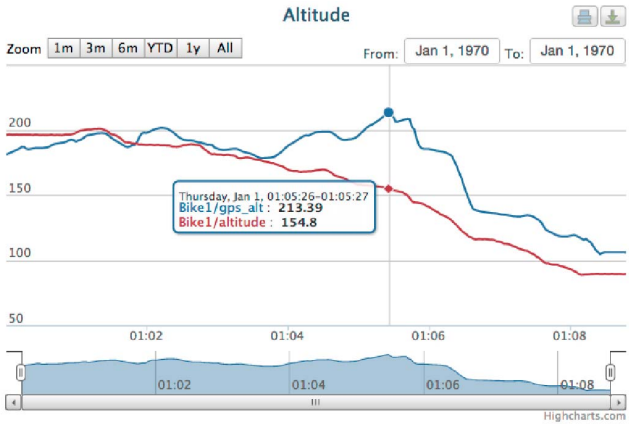


Fig. 4. Comparing sensors accuracy using SENSAPP dashboard

a MongoDB database used to store raw data and *(ii)* a Round-Robin database to store journaled data. SENSAPP also provides a web-based administration interface built on top of the offered REST interface. This interface supports the manipulation of sensors, and provide graphical widgets to visualise data collected in SENSAPP (see Figure 4). To support the access to resources with cross origin through plain HTTP, it implements the *Cross Origin Resource Sharing* standard [9].

This reference implementation is small enough to be easily understood, shared and deployed (mainly implemented in Scala, see TAB. I). But it is also expressive enough to support real-life experiments, and the platform is shared with other research groups (*e.g.*, SINTEF Communication Systems, the Rainbow project-team at the University of Nice Sophia Antipolis — I3S UMR CNRS UNS 7271). SENSAPP is daily used in SINTEF' premises (as a *living lab*) to collect work environment information in key offices. We also propose public data sets on the website.

TABLE I
SENSAPP REFERENCE IMPLEMENTATION SIZE (LOC)

Language	Files	Blank	Comment	Code
Scala	127	1130	4197	4742
XML	66	248	280	2867
Bourne Shell	9	51	33	160
Total	202	1429	4510	7769

Reproducibility using `git` and `cloc` commands:

```
$ git clone git://github.com/mosser/SensApp.git
$ cloc.pl SensApp
```

III. SENSAPP IN THE CLOUDS

The previous section demonstrated the intrinsic distribution of the SENSAPP platform. In this section, we position the platform *w.r.t.* the ten major "obstacles and opportunities" identified in the Cloud-computing seminal paper [1]. The key point is to illustrate how SENSAPP can be used to setup experiments and compare platforms *w.r.t.* these points.

A. Availability of Services

This obstacle is based on the recent outages encountered by major cloud vendors (up to several hours). As a cloud application is hosted in the cloud, it becomes dependent of the availability and the *Service Level Agreement* (SLA) of the underlying cloud vendor. A cloud outage transitively impacts the application, at different levels (*e.g.*, unavailability, data loss). To avoid such a situation, one can use a multi-tenant cloud through the federation of multiples clouds [10]. In this configuration, the federated cloud aggregates resources from different vendors, minimizing the risk of complete outage.

SENSAPP can be used to support research in this direction. Its very simple architectures does not require boilerplate configuration to deploy it. Thus, it is easy to deploy several instances of SENSAPP on multiple resources. One can easily simulate an outage by simply shutting down a subset of SENSAPP services, and then take adaptation decision (*e.g.*, starting new resources in the remaining providers) to ensure continuous service. Moreover, the notification mechanism embedded in SENSAPP makes easy the definition of controlled experiments to assess service availability: a sensor sends controlled data to the instance that encounter the simulated outage, and a checker subscribe to notification associated to this sensors. One can then compare sent and received data, as well as measure latency with or without outage.

B. Data Lock-in

Cloud vendors offer proprietary interfaces to their internal data storage facilities. Thus, the migration of an application from one cloud to another (*e.g.*, due to a change in the commercial offer) requires dedicated development. The obvious solution to this problem is the definition of a standard API to interact with cloud storage, but this solution is not accepted by major vendors (as data lock-in enforce customer fidelity).

SENSAPP is built on top of the SENML standard to model sensors data. This standard is extremely small (24 pages, implemented in less than 500 lines of code using the Scala language[4]). This uniform data representation is a good candidate for data migration techniques: the cost of writing a transformation from SENML to another data format is low, and SENSAPP provide public data sets containing 250,000 entries. The scalability of a migration technique τ can then be measured according to two dimensions: *(i)* the difficulty to write the adapter from SENML to the proprietary format using τ and *(ii)* the efficiency of τ while migrating the public data set.

C. Data Confidentiality and Auditability

Data access is one of the major flaws that prevents a large adoption of Cloud-computing. One the one hand, companies are reluctant to send critical data to external partners, for security and control reasons. Usually, companies tackle this challenge by using hybrid clouds: critical information are stored and processed in a private cloud (hosted by the company), and public clouds are used to host non-critical processes and data. On the other hand, some, such as governments might not be allowed to host sensitive data outside of some physical borders (*e.g.*, regions such as country borders, Europe). It is then critical for public actors to have a clear control on the data location, for auditability reasons.

SENSAPP collects data from sensors. Thus, the confidentiality of these data is transitively connected to the confidentiality associated to the physical sensor that collects the data. A luminance sensor collecting the amount of light measured in front of SINTEF' premises is clearly public, while the GPS location sensor of a smart-phone collects highly confidential information [11]. Classification techniques to model data privacy can then be applied on SENSAPP sensors to reify such knowledge. As the underlying database can be replicated and deployed on multiples host, it is possible to simulate hybrid clouds with one private database and one public database. The published available data sets contains both public (*e.g.*, GPS signal strength) and private (*e.g.*, location) information.

D. Data Transfer Bottleneck

Cloud applications are usually data-intensive. As a consequence, such applications require a lot of bandwidth to exchange data between processing nodes. Bandwidth that is usually charged by the cloud vendor as part of its business model. Tremendous savings can be collected thanks to a careful deployment of the data set in the clouds to avoid external communication when not necessary[5]. In addition to this challenge, the data locality factor is important for the customer of a cloud application. Considering that the application is hosted by the Amazon European data centre (located in Dublin, Ireland), if all clients are located in the United States, it makes sense to migrate it to one of the data centres available in this region (*e.g.*, the one located in Virginia).

SENSAPP is based on SENML for data representation, which provides a fixed representation for the exchanged data. Public data sets can therefore be used to feed cost models and then infer a deployment topology that minimises the data cost.

E. Performance Unpredictability

Cloud infrastructures rely on the concept of *Virtual Machine* (VM), started on demand according to user's needs. Contrarily to a "concrete" machine that is physically dependent of a given hardware, several VMs can be executed on the same "concrete" machine at the very same time. As a consequence, the performances of a VM can be degraded by another independent VM that over-consumes physical resources. In this shared model, reproducibility of experiments is not ensured, as it might be impacted by external and uncontrollable factors.

SENSAPP provides real-life data sets obtained from concrete situations. Thus, a scientific contribution in VM isolation can use the provided data sets to re-play again and again the same scenario. The main advantages are the following: *(i)* data sets are collected through real-life sensors and *(ii)* the collected data are explicitly timestamped by nature, allowing one to *exactly* replay a scenario by following the timestamps.

F. Scalable Storage

The immediate drawback of a data intensive application is that it consumes and/or produces tremendous amount of data. Handling such amounts of data becomes challenging, according to two orthogonal concerns. First, the storage capability must scale, *i.e.*, the underlying infrastructure must transparently absorb very large amount of data (measured in Tb). Secondly, as applications consumes these data, the query interface provided by the infrastructure must scale to extract relevant data subsets according to user's expectations.

SENSAPP is dedicated to a domain that both generates and consumes large volumes of data. Considering a sensor sending one value per second (this is typical for sensors that are deployed in buildings, as there are powered by the electrical network of the building instead of embedded battery), it produces 86400 data in a single day. A SENML message encoded in JSON (*i.e.*, plain text) weights in average 100 bytes. Thus, such a sensor produces almost 9 megabytes of data per day, and 3 gigabytes per year. Multiplied by the number of sensors involved in an application (our building monitoring system collects data from 16 sensors, which is a *small* system),

[4]https://github.com/mosser/SensApp/tree/master/net.modelbased.sensapp.library.senml

[5]The communication between instances from the same vendor but located in different world regions are usually considered as external communications

it immediately triggers storage issues. Moreover, third part applications rely on SENSAPP database to feed themselves. Thus, the response time of the underlying database is critical for SENSAPP. According to the simplicity of the platform, one can easily change the internal data representation (it only implies to re-implement the marhsalling function used by the service framework) to assess contribution in this domain. Database back-end can also be easily changed in the current implementation to assess scalability of querying systems.

G. Bugs in Large Distributed Systems

This empirical obstacles is based on the following facts: cloud-specific bugs appears in very large distributed systems, triggered by cloud side effects (*e.g.*, high latency, lost of connexion, VM migration). These bugs cannot be replicated outside of the cloud environment, as they are triggered by external factors.

SENSAPP provides an open source code, allowing one to apply offline static analysis on the source code, as well as runtime analysis and assessments using the reflexive capabilities of the Java VM. The services used to compose SENSAPP are by default hosted in Jetty servers, which are lightweight java-based web servers. Thus, it is possible to activate remote monitoring of the involved JVMs to apply bug discovering techniques dedicated to distributed systems [12]. SENSAPP is also easy to mock according to its reduced size, and is therefore a good candidate for fault injection techniques.

H. Scaling Quickly

Clouds are "elastic" by nature. Elasticity can be vertical (*e.g.*, increasing the number of core allocated to a VM) or horizontal (*e.g.*, increasing the number of VM involved in the application). A typical cloud scenario couples load balancing techniques, automated provisioning and adaptation policies to ensure the scalability of the deployed application. These mechanisms must react accordingly when a load peak is encountered, hiding this overload from the end user point of view.

SENSAPP can easily be deployed according to this kind of architecture: it is possible to hide the services behind load balancers, as they rely on plain HTTP requests (conforming to the uniform interface principle of the REST architectural style). It is then possible to validate the scalability according to two dimensions. First, one can assess the capability of the platform to absorb large data sets collected by real-life sensors. Secondly, as sensors can easily be simulated (being a sensor means to send SENML messages to a given URL), it is possible to stress the platform with billions of parallel "fake" sensors.

I. Reputation Fate Sharing

According to the cloud paradigm, a user books VMs hosted in the clouds. We consider here a "nasty" VM, used for spam or "denial of services" purpose. External systems might blacklist the IP address of such a VM. As a consequence, if a normal VM is eventually hosted using the same IP address, the application executed in this VM is also blacklisted!

SENSAPP provides notification mechanisms that can forward such information (*e.g.*, "unable to connect to ..."). The source code is simple enough to be modified in order to implement a reputation protocol and test its efficiency.

J. Software Licensing

The main idea of the cloud is to provide "anything" as a service, *e.g.*, infrastructure, platform, software. The notion of service is often connected to the notion of business. Thus, cloud business models implies to generate value based on the provided service. Infrastructure providers base their fees on CPU consumption and consumed bandwidth, where service providers usually charge account based licenses.

SENSAPP public data sets can be used to feed cost models and then apply multi-criteria analysis techniques to find a cloud provider that maximize user's expectations.

IV. INITIAL EXPERIMENTS: SCALABILITY

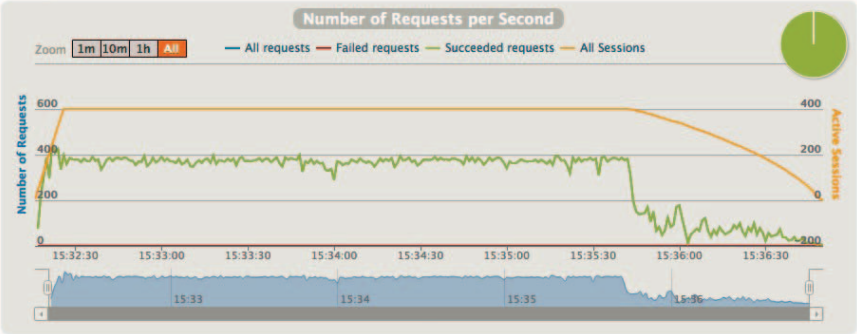
In this section, we describe initial experiments made with the SENSAPP platform to address the previously described challenges. The experiments were made using the Amazon EC2 cloud infrastructure (European data centre). The results presented here does not aim to compare cloud providers, as such a goal requires the set-up of an exhaustive protocol. Our goal here is to sketch how such a protocol can be defined, by focusing on the definition of artefacts to assess it. The underlying idea of this experiment is to investigate challenges F ("scalable storage") and G ("scaling quickly"). It is also related to challenges A ("availability of services") and D ("data transfer bottleneck").

The objective is to implement a set of *scenarios* that simulates the use of the SENSAPP platform. These scenario can be "normal" situations, *e.g.*, n sensors pushing a single data according to a given sampling frequency f. We can then work on n and f to assess instances of SENSAPP deployed in the clouds according to different topologies. We can also implement *stress scenario*, used to model peak load. For example, a stress scenario can model defective sensors that flood the system with irrelevant data[6]. It is then possible to implement a library of scenarios, available off-the-shelf (with reference value associated to each scenario). One can then confront a SENSAPP instance deployed according to its research contribution *w.r.t.* the reference value obtained in a similar context.

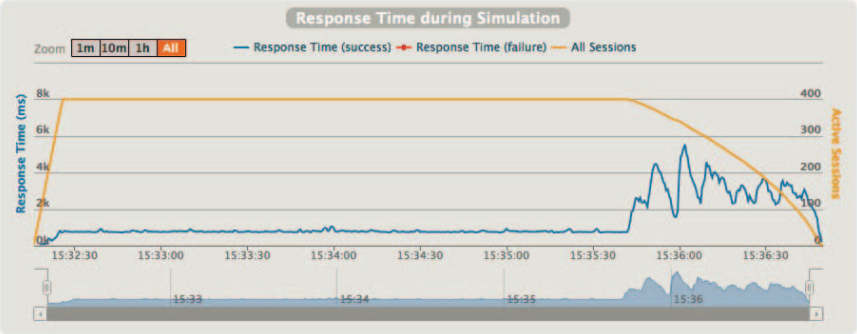
To model these scenarios, we used the Gatling Stress tool, provided as open source by Excylis[7]. It provides a domain-specific language (internal to Scala) to properly implement HTTP based scenario. Listing 1 describes a scenario implemented with the Gatling language. A scenario is a class that implements the `Simulation` class (l. 1). It defines an `apply` method that returns a list of scenario to be executed in this

[6]This situation actually happened in our living lab, where a router lost the physical connection to one of its associated sensors and enter in an infinite loop where it was sending all the collected data again and again to the SENSAPP instance.

[7]https://github.com/excilys/gatling



(a) Number of requests per second sent to the SENSAPP instance



(b) Response time measured on client side (800ms in average, including network latency)

Fig. 5. Stressing a SENSAPP instance deployed in Amazon EC2 cloud

```
1  class SensorPushSimulation extends Simulation {
2
3    val numberOfData: Int = 200
4    val maxDelayBetweenPush: Int = 400
5    val url = "http://..."
6
7    def apply = List(sensorPush.configure.users(10).ramp(10))
8
9    val headers = Map("Content-Type" -> "application/json",
       "Accept" -> "text/plain,application/json")
10
11   val sensorPush =
12     scenario("Sensor pushing Data")
13       .exec{
14         http("Alive?").get(url).check(status is 200)
15       }.pause(100, 200, MILLISECONDS)
16       .exec {
17         http("Sensor Creation").post(url)
18           .headers(headers).body(genSensor())
19       }.pause(100, 200, MILLISECONDS)
20       .loop{ chain
21         .exec {
22           http("Random data").put(url)
23             .headers(headers).body(genRandomData())
24         }.pause(100, maxDelayBetweenPush, MILLISECONDS)
25       }.times(numberOfData)
26       .exec {
27         http("Sensor Deletion").delete(url)
28       }
29 }
```

Listing 1. Example of Gatling Stress Scenario

simulation (l. 7, here configured to start 10 concurrent users in a 10 seconds ramp). A scenario is defined as a sequence of **exec** steps, executed for each user in parallel. In this scenario, we first check if the platform is responding (l. 13). Then, we register a new sensor in the platform (l. 16). The scenario send random data associated to this newly registered sensor by entering in a finite loop (l.20→25), and finally deletes this sensor (l. 27). One must notice that all the measurements are done on the client side and not on the server (it does not requires to instrument the running instance).

We represent in Figure 5 the graphs generated by the Gatling platform after the execution of a scenario. Among others, these two graphs describe *(i)* the number of requests sent per second to the SENSAPP instance (Figure 5(a)) and *(ii)* the average response time of the platform (Figure 5(b)). The x-axis represents the time, and the y-axis represents the measured dimensions: *(i)* number of request handled, or *(ii)* average response time of a request.

Through the capture of prototypical situations encountered in the running SENSAPP systems, it is possible to provide a set of stress simulation scenario, implemented as Gatling scenario. Relying on publicly available data sets, it becomes possible to provide quantifiable metrics to assess cloud research on a common platform.

V. CONCLUSIONS AND PERSPECTIVES

In this paper, we described SENSAPP, a platform that leverages IoT and IoS to support the definition of sensor based applications. This platform is intrinsically distributed, and we

emphasized how it can be used as a reference platform for cloud computing research assessment, based on the ten major obstacles identified for cloud-computing research. We also proposed a first experiment to assess scalability issues in a comparable way.

Immediate perspective of this work is the definition of a benchmark that extends the initial experiment sketched in this paper. We plan to provide off-the-shelf usage scenario, as well as reference time for different topologies provisioned in major clouds.

ACKNOWLEDGEMENT

This work is partially funded by the EU Commission through the REMICS project (FP7-ICT, Call 7, contract number 257793, http://www.remics.eu) and by SINTEF through the MODERATES and SISAS strategic projects. Interactions with the Amazon EC2 IaaS platform are supported through the MOD4CLOUD project (AWS in Education grant award, http://sm.ace-design.eu/projects/proposal/mod4cloud).

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009.

[2] N. Kassem and E. Team, *Designing Enterprise Applications: Java 2 Platform*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

[3] A. Rausch, R. Reussner, R. Mirandola, and F. Plasil, Eds., *The Common Component Modeling Example: Comparing Software Component Models [result from the Dagstuhl research seminar for CoCoME, August 1-3, 2007]*, ser. Lecture Notes in Computer Science, vol. 5153. Springer, 2008.

[4] J. Kienzle, N. Guelfi, and S. Mustafiz, "Crisis Management Systems: A Case Study for Aspect-Oriented Modeling," *T. Aspect-Oriented Software Development*, vol. 7, pp. 1–22, 2010.

[5] S. Katz, M. Mezini, and J. Kienzle, Eds., *Transactions on Aspect-Oriented Software Development VII - A Common Case Study for Aspect-Oriented Modeling*, ser. Lecture Notes in Computer Science, vol. 6210. Springer, 2010.

[6] M. Alférez, N. Amálio, S. Ciraci, F. Fleurey, J. Kienzle, J. Klein, M. E. Kramer, S. Mosser, G. Mussbacher, E. E. Roubtsova, and G. Zhang, "Aspect-Oriented Model Development at Different Levels of Abstraction," in *ECMFA*, ser. Lecture Notes in Computer Science, R. France, J. M. Küster, B. Bordbar, and R. Paige, Eds., vol. 6698. Springer, 2011, pp. 361–376.

[7] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, 2000, aAI9980887.

[8] C. Jennings, Z. Shelby, and J. Arkko, "Media Types for Sensor Markup Language (SENML), draft #8," IETF, Tech. Rep., Jan 2012. [Online]. Available: http://tools.ietf.org/html/draft-jennings-senml-08

[9] A. van Kesteren, "Cross-origin resource sharing," W3C, W3C Working Draft, Mar. 2009, http://www.w3.org/TR/2009/WD-cors-20090317/.

[10] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres, M. Ben-Yehuda, W. Emmerich, and F. Galán, "The RESERVOIR Model and Architecture for Open Federated Cloud Computing," *IBM J. Res. Dev.*, vol. 53, no. 4, pp. 535–545, Jul. 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1850659.1850663

[11] S. Gambs, M.-O. Killijian, and M. N. del Prado Cortez, "Show Me How You Move and I Will Tell You Who You Are," *Transactions on Data Privacy*, vol. 4, no. 2, pp. 103–126, 2011.

[12] P. Reynolds, C. E. Killian, J. L. Wiener, J. C. Mogul, M. A. Shah, and A. Vahdat, "Pip: Detecting the Unexpected in Distributed Systems," in *NSDI*. USENIX, 2006.