**Cluster Computing manuscript No.**
(will be inserted by the editor)

# Towards a Framework for Monitoring Cloud Application Platforms as Sensor Networks

**Rustem Dautov · Iraklis Paraskakis · Mike Stannett**

Received: date / Accepted: date

**Abstract** With the continued growth in software environments on cloud application platforms, self-management at the PaaS level has become a pressing concern, and the runtime monitoring, analysis and detection of critical situations are all fundamental requirements if we are to achieve autonomic behaviour in complex PaaS environments. In this paper we focus on cloud application platforms offering their customers a range of generic built-in re-usable services. By identifying key characteristics of these complex dynamic systems, we compare cloud application platforms to distributed sensor networks, and investigate the viability of exploiting these similarities with a case study. We treat cloud data storage services as "virtual" sensors constantly emitting monitoring data, such as numbers of connections and storage space availability, which are then analysed by the central component of a monitoring framework so as to detect and react to SLA violations. We discuss the potential benefits, as well as some shortcomings, of adopting this approach.

**Keywords** Cloud Application Platforms · Service-Oriented Computing · Autonomic Computing · Monitoring · Sensor Network · Sensor Web

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7-PEOPLE-2010-ITN) under grant agreement n° 264840 (`http://www.relate-itn.eu/`).

R. Dautov · I. Paraskakis
South-East European Research Centre, CITY College, 24 Proxenou Koromila, Thessaloniki 54622, Greece
E-mail: rdautov@seerc.org, iparaskakis@seerc.org

M. Stannett
Department of Computer Science, University of Sheffield, Regent Court, 211 Portobello, Sheffield S1 4DP, United Kingdom
E-mail: m.stannett@sheffield.ac.uk

## 1 Introduction

In order to remain competitive in a rapidly changing market, it is essential that service providers monitor and respond to their existing and potential customers' needs. For companies operating traditional deployment models this means continually developing new systems and retiring old solutions in relatively short periods of time [28]. For cloud service providers, it means constantly monitoring the services that are available to customers, identifying bottlenecks and breakdowns, checking for Service Level Agreement (SLA) failures and mismatches, and providing seamless hand-overs from one service to another.

This is particularly important at the PaaS (Platform-as-a-Service) level, where a large number of generic services may be available, such as *data storage*, *queue messaging*, *searching*, *E-mail* and *SMS*, *logging* and *caching*, among others.[1] These can be re-used and integrated by users, generating complex interrelationships between services and user applications. Consequently, if a service becomes unexpectedly inaccessible, the platform provider needs to recognise this problem as quickly as possible, and to take steps to replace the 'broken' application with an equivalent working alternative. The definition of 'equivalent' in this context will depend, of course, on the SLAs associated with all users previously using the now-inaccessible service.

This requirement is an inevitable business consequence of the Service-Oriented Computing (SOC) paradigm, reflecting as it does the idea that services should be used as basic building blocks from which to assemble rapid and low-cost, yet secure and reliable, applications [24]. To be used in this way, services need to be largely autonomous, as well as discoverable, self-describing, reusable, and above all highly interoperable, while offering a wide range of capabilities in-

[1] Heroku, for example, provides up to 150 add-ons (`http://addons.heroku.com`).

cluding everything from the performance of simple calculations to the execution of complicated business processes in widely distributed environments [28].

The emergence of SOC opened new business opportunities for enterprises who migrated their IT systems from traditional monolithic approaches towards highly modular and re-usable service-based architectures, and allowed organisations to develop highly distributed software systems in a short period of time by dynamically assembling basic services supplied by multiple service providers and hosted on different hosting platforms [28]. To meet the demands of these service providers, it has been predicted that the resulting "service cloud" will come to comprise a federation of resources offered by multiple infrastructure providers [8], and cloud platforms will continue to play an increasingly important role in this context.

Today's cloud application platforms (e.g., Google AppEngine[2], Windows Azure[3] and Heroku[4]) already host numerous services, ranging from simple operations to complicated business logic, and ready to be integrated into an ever-expanding range of service compositions. However, the continuing paradigm shift towards cloud computing has introduced a new and pressing challenge: the complexity of next-generation service-based cloud environments is soon expected to outgrow our capacity to manage them in a manual manner [6, 10], and current cloud computing providers do not offer user-customised management and monitoring mechanisms as part of their infrastructure [28]. Since complexity is known to be inversely related to software reliability [19, 26], the challenge of service management and monitoring is especially pressing in the case of hybrid clouds, where different parts of an application system are deployed on private and public clouds, and multi-clouds, where an application system is distributed across several clouds at the same time. In this context, maintaining the ever-expanding software environment of a cloud application platform is a major challenge. Platform providers must be enabled to exercise control over all critical activities taking place on the platform, with the introduction of new services and applications, and the modification of existing ones, to maintain the platform's and deployed applications' stability and performance [19].

Meeting these challenges involves the rapid and scalable processing of large quantities of real-time data, generated by a widely distributed system of performance monitors. In this paper we argue that these monitors can be regarded as forming a *sensor network*, which in turn allows techniques developed by the sensor web community to be used to support the development of self-management mechanisms for cloud application platforms. Of particular benefit is the potential to monitor cloud application platforms (deployed applications, platform components, third-party services registered with the platform, etc.) and detect critical situations in a timely manner so as to provide a basis for run-time self-adaptation [10].[5] We focus in this paper on the PaaS level, although it should be noted that the techniques we propose could potentially be used, albeit with some modifications and various caveats, by SaaS (Software-as-a-Service) and IaaS (Infrastructure-as-a-Service) providers as well.

To demonstrate the viability of our approach, we present the results of a small case study, which focuses on the Heroku platform and the interpretation of its data storage services as sensors. Our approach uses a self-adaptation framework developed using 'Semantic Sensor Web' techniques, in which we represent heterogeneous values collected from services (e.g., number of client connections and occupied storage space) using RDF triples, and stream these data to an *autonomic manager* – the central component of the network, responsible for situation assessment, problem diagnosis and adaptation planning. To avoid overloading the autonomic manager, the data pass through a pre-processing (filtering) stage, so that only critical, adaptation-relevant values are allowed through. The case study can be regarded as a partial proof of concept. Nonetheless, while we identify several potential benefits associated with our approach – extensibility and scalability, opportunities to re-use existing solutions, the concept of routing nodes, and platform-independence – potential shortcomings can also be identified. These include both problems concerning our ability to instrument proprietary services, and the extent to which this may entail an intrusive approach to monitoring.

The rest of the paper is organised as follows. Section 2 is dedicated to the foundations of self-management and autonomic computing in cloud computing, in particular at the PaaS level. It also explains the essential role of monitoring and analysis components in self-managing systems. Section 3 further discusses monitoring in the context of cloud application platforms. Taking Heroku as an illustrative example, we analyse the challenges associated with management of such complex systems and draw parallels between cloud application platforms and sensor networks. In Section 4 we present the Heroku-based case study by applying our ideas to data storage services. Section 5 summarises the potential benefits and shortcomings of our approach, and suggests ways in which it can be expanded to include the IaaS and SaaS levels. Section 6 concludes the paper.

[2] `http://appengine.google.com/`
[3] `http://www.windowsazure.com/`
[4] `http://www.heroku.com/`

[5] It should perhaps be noted that we are not attempting here to compete with existing approaches for run-time monitoring in clouds, but rather to offer complementary concepts, which can be reused when developing cloud monitoring mechanisms.

## 2 Self-Management at the PaaS Level

With the continuing paradigm shift towards cloud computing, service-based cloud environments can be expected to become ever more complex, outgrowing our capacity to manage them in a responsive manual manner. Both academia and industry have consequently been putting considerable effort into finding potential solutions to this problem. These solutions, mainly based on the principles of Autonomic Computing [15], have as their focus the creation of self-adaptive cloud systems which are capable of *self-management* – the ability to manage themselves without human intervention.

To date, attempts to equip clouds with autonomic behaviour have mainly focussed on the IaaS (Infrastructure-as-a-Service) level, whether by developing efficient mechanisms for distributing the varying volumes and types of user requests across different computational instances (*load-balancing*), or by reserving and releasing computational resources on demand (*elasticity*) [3,23]. Both load balancing and elasticity are now seen as essential characteristics of cloud computing [22].

Building on techniques developed by the grid and HPC computing communities, these existing approaches usually rely on collecting such data as CPU load, memory utilisation and network bandwidth to execute adaptation actions. Depending on adaptation policies, such actions target various goals (e.g., increasing application performance or reducing cost and energy consumption) and typically include *replication* and *resizing* techniques [13]. The former refers to adding and removing computational instances to meet ever-changing resource requirements, whereas in the latter case resources are added or removed within a single computational instance. Load balancing techniques are often used in combination with replication to spread workload equably across available instances. Existing IaaS adaptation solutions can be further classified as *reactive* or *predictive* – the former are commonly employed by many IaaS cloud providers, while the latter use various heuristic analytical and machine learning techniques to predict workload and scale resources accordingly. For a survey of existing elasticity and load balancing techniques, the interested reader is referred to [13].

Unfortunately, the development of self-management capabilities at the Platform-as-a-Service (PaaS) level is far less developed. Even though various approaches have been described as targeting the PaaS level, these typically do not act at the PaaS-level directly, but rely on lower level, IaaS-related adaptation actions [13]. Neither researchers nor platform providers have offered solutions which would allow hosted applications to modify their internal structure and/or behaviour at run-time by adapting to changing context (e.g., by substituting one Web service for another). This task has instead been shifted to the Software-as-a-Service (SaaS) level – that is, it has been left to software developers, the target customers of the PaaS offerings, to implement self-adaptation logic within specific applications.

It is our belief that self-management at the PaaS level is equally important, and development of self-adaptation mechanisms at the this level is essential in order to prevent cloud platforms from dissolving into "tangled" and unreliable environments. Our research therefore targets the PaaS, and more specifically the Application-Platform-as-a-Service (aPaaS) level of cloud computing [20] – a cluster of cloud platforms, which extend the default functionality offered to customers, such as an operating system and execution environment, with a selection of generic re-usable services and tools to create applications and have them deployed and executed in a cloud environment.[6] We adopt the viewpoint of a supplier offering its customers a range of built-in or third-party services (whose behaviour we cannot easily change), and ask how best to monitor and manage their life-cycles.

### 2.1 Monitoring and Adaptation

The growing importance of distributed systems, including service-based applications and clouds, has also motivated the scientific community to investigate the adaptability and sustainability aspects of such systems. Early theoretical work, inspired by Paul Horn's Autonomic Computing 'manifesto' [15] and referring to the original 'self-*' characteristics of autonomic systems, served as groundwork for a wide range of prototypical implementations of self-managing mechanisms for various computer system structures, including service-based applications and clouds [7]. Existing self-adaptation mechanisms typically implement control feedback loops, such as MAPE-K[7] [18] or CADA[8] [12], where the monitoring activity acts as a trigger for adaptation: whenever monitored variables move beyond pre-specified bounds, this triggers adaptation (for example, re-balancing of workloads) in an attempt to restore normality.

In a broad sense, then, monitoring may be defined as the process of collecting and reporting relevant information about the execution and evolution of a computer system, and can be performed by any mechanism capable of checking whether the currently observed situation meets expectations [17].[9] These monitoring processes typically target the collection of data concerning a specific artefact, the *monitored subject* [5]. In the context of cloud application platforms,

[6] Throughout this paper, we use the terms cloud application platform (CAP), cloud platform, service-based cloud, servicebased cloud platform, PaaS, aPaaS, etc. interchangeably to refer to the same concept.

[7] MAPE-K: Monitor, Analyse, Plan, Execute, Knowledge.

[8] CADA: Collect, Analyse, Detect, Act.

[9] We focus in this paper on run-time monitoring. Related activities can also include such techniques as post-mortem log analysis, data mining, and online or offline testing – the interested reader is referred to [17].

the monitored subject can be a platform component, an application, a service composition, a single service, etc., and depending on the implementation of the monitoring mechanism, monitoring can be either intrusive or non-intrusive.[10] It is also important to consider who is responsible for driving the monitoring process: in *polling* mode, the monitor is responsible for querying the subject at regular intervals, while in *push* mode the subject is responsible for notifying the monitor whenever a significant event occurs [5].

In our view, implementing self-management at the PaaS level requires a flexible combination of approaches. On the one hand, we could make it a condition of deployment that applications must conform to a platform-specific API that includes the provision of 'hooks' to which monitors could be attached, thereby facilitating an intrusive approach. But since these hooks necessarily provide access to application performance data, this could in turn be seen as introducing a potential security risk; this may be unacceptable where particularly sensitive applications are concerned, in which case a non-intrusive approach would be required. Moreover, a 'broken' application cannot be relied upon to act according to its specification, and in particular, therefore, cannot be relied upon to push a report to the monitor, signaling its own failure. It is therefore essential that a PaaS-level monitor polls applications at regular intervals to determine whether they are still active. On the other hand, the SOC-inspired ability to combine applications in novel and unexpected ways makes it impossible to anticipate in advance all of the situations of which the monitor needs to be made aware – in such circumstances, it is also important that applications can push event data to the monitor, since the monitor may not itself issue the required queries.

## 3 Monitoring Services as Sensors

Let us consider a hypothetical scenario in which an online store is deployed on Heroku. This application has several *web dynos* (computational instances for processing incoming HTTP requests) and several *worker dynos* (computational instances for performing tasks on the server side), which can be scaled up and down. In order to fully utilise the advantages of the cloud platform, this e-store application may employ various add-on services – data storage, authentication, e-payment, search, notification, message queueing, etc. As this example suggests, the flexibility arising from the freedom to choose from a range of pre-existing services is appealing from the software developer's point of view – using just six services has already saved considerable financial, time and human resources expenditure.

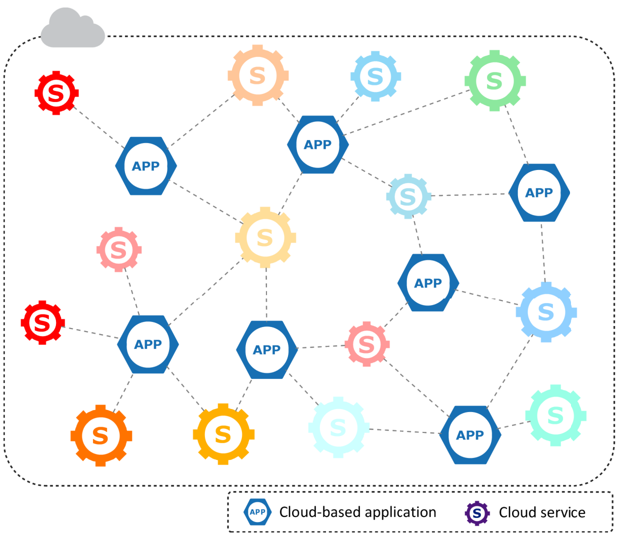

**Fig. 1** Cloud-based applications are coupled to the platform's services, with multi-tenancy allowing the same services to be used by more than one application. The resulting 'tangled' structure results in a highly complex maintenance problem.

From the platform provider's point of view, however, offering this level of flexibility comes at a price. With add-ons replicated across multiple computational instances (Figure 1), and coupled to more than one million deployed applications,[11,12] it becomes a challenging task to monitor the execution of the resulting PaaS environment so as to detect failures and suboptimal behaviours. Maintaining the whole system at an operational level – that is, satisfying SLAs between the provider and its consumers – is an inherently difficult problem.

The autonomic management framework we have been developing for service-based cloud environments [10] relies on monitoring and detection of critical situations, which then trigger appropriate adaptations. In other words, we treat cloud application platforms as devices equipped with *sensors* – whenever a significant condition is picked up by a 'sensor', a corresponding action is invoked by the 'device'. Adopting this approach leads immediately to the idea that cloud application platforms can be treated as *sensor networks*. Until recently sensor networks could be regarded as relatively scattered groups of sensor components, each based on its own proprietary standards and serving its own individual purposes. This situation changed in 2003 with the launch of the Sensor Web Enablement (SWE) initiative by the Open Geospatial Consortium[13] (OGC), whose members are "specifying interoperability interfaces and metadata en-

[10] Intrusive monitoring requires the monitored subject to be instrumented with probes to facilitate inspection of its characteristics. As with code instrumentation, it is essential that this is done with care, since the instrumentation can itself potentially affect the subject's performance, providing a flawed picture of its inherent capabilities.

[11] `http://gigaom.com/2012/05/04/heroku-boss-1-5m-apps-many-not-in-ruby/`

[12] `http://www.crunchbase.com/company/heroku`

[13] `http://www.opengeospatial.org/`

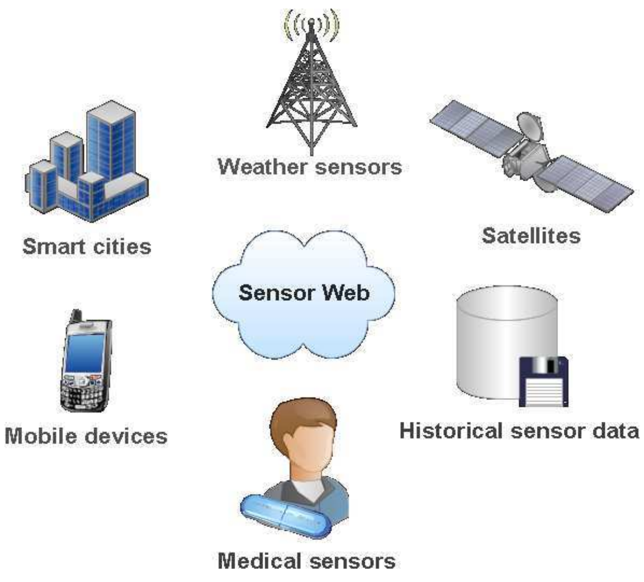

**Fig. 2** The concept of the Sensor Web.

codings that enable real time integration of heterogeneous sensor webs into the information infrastructure".[14] The Sensor Web can be seen as a collection of protocols and APIs, coupled to and providing access to an interconnected network of Web-accessible sensor networks and historical data repositories (Figure 2).

While sensor networks are more usually thought of as computer accessible networks of distributed devices using sensors to monitor continually varying conditions at different locations [4] (Figure 3), there are clear similarities with our own problem domain, which requires the monitoring of multiple distributed information sources on a cloud application platform in order to support self-management. Looking at cloud application platforms from an information management point of view, the commonalities can be summarised as follows:

– Volume: as in sensor webs, the amount of raw data generated by deployed applications, components of the platform, users, services, etc. is huge. Even if we neglect 'noise' (information flows that are not relevant for monitoring in the given context) the amount of information remaining is still considerable.
– Dynamism: in both sensor webs and cloud application platforms, various information sources are constantly generating data (which is then processed, stored, deleted, etc.) at an unpredictable rate. The various platform components evolve, with new services being added and old ones removed, making the whole system even more dynamic.

[14] `http://www.opengeospatial.org/projects/groups/sensorwebdwg`

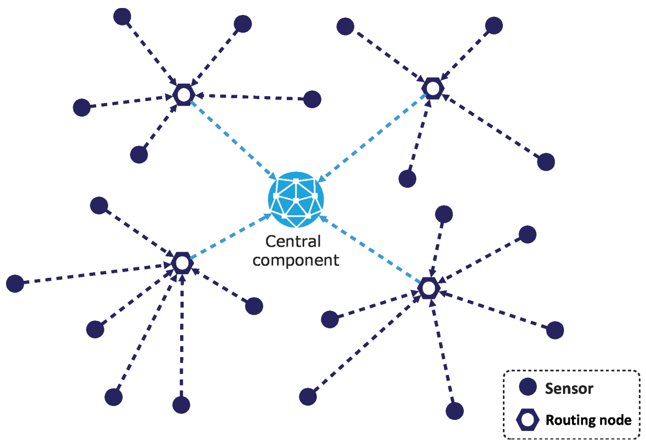

**Fig. 3** Schematic of a sensor network.

– Heterogeneity: just as networked sensors can be attached to a wide range of different devices, so platform data can originate from a wide range of distributed sources (applications, databases, user requests, services, etc.). This information is inherently heterogeneous, both in terms of data representation (different formats, encodings, etc.) and in terms of semantics. For example, two completely separate applications from different domains with different business logic may store logging data in XML. In this case, the data is homogeneous in its format – and potentially also in structure – but heterogeneous at the semantic level.
– Distribution: the information provided by both sensors and platform monitors may come from various logically and physically distributed sources. On the logical side, platform data may originate from databases, file systems, running applications, external Web services, and these may be physically deployed on distinct virtual machines, servers and even data centres.

These similarities allow us to draw parallels between cloud application platforms and problem domains for which solutions proposed by the Sensor Web research community, based on sensor technology, have already been shown to be effective. The parallels give us confidence that we can re-use the positive experience of the Sensor Web community in the context of dynamic monitoring and analysis of continuously flowing streams of data within a cloud application platform.

In summary, by extending the notion of sensors to include not just physical devices, but anything that calculates or estimates a data value – e.g., an application component, an SQL query, or a Web service – we can think of a particular service as a sensor and the whole platform as a network of such sensors. For example, we may be interested in monitoring response times from a Web service which is part of a more complex service composition. In this case, the service is the monitored subject, response time is the monitored as-

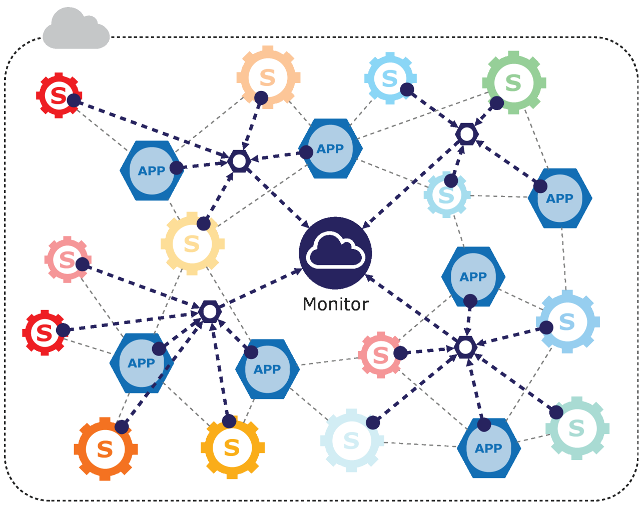

**Fig. 4** Schematic of a "sensor-enabled" cloud application platform.

pect, and the software functionality measuring and sending these values is the sensor. There may be cases when a monitored subject has more than one aspect to monitor – that is, a single Web service may be equipped with several sensors (though possibly implemented within a single piece of programming code) measuring not only response time, but also, for example, availability, the number of incoming requests or price.

Figure 4 illustrates the concept of a sensor-enabled cloud application platform. The monitored subjects – services, applications, platform components – are equipped with sensors – software components responsible for measuring some values – which are connected into a network. Following sensor network principles, values from a sub-network of monitored subjects may first go through a routing node – a software component responsible for transporting the values further to a central component and/or initial processing and aggregation of incoming information. Depending on the purposes of monitoring the central component may perform various functions, ranging from simply storing monitored values to sophisticated analysis of those values, diagnosing problems, or even executing appropriate adaptation actions through a feedback mechanism. For example, in [10] the autonomic manager is responsible for analysis of the incoming data, detection of critical situations and execution of relevant adaptation actions on the managed cloud application platform.

## 4 Proof of Concept

In this section we present the initial results of our experiments treating a cloud application platform as a distributed network of virtual sensors. As a test bed for our experiments we have chosen Heroku, which currently offers its customers up to 150 generic re-usable add-on services. A considerable number of these services are dedicated purely to data storage, including traditional relational databases (e.g., ClearDB MySQL[15] and Heroku Postgres[16]), NoSQL databases (e.g., MongoHQ[17] and Redis Cloud[18]), and caching services (e.g., MemCachier[19] and MemcachedCloud[20]), among others. Heroku's pricing model assumes that cloud customers, when subscribing to a particular service, are free to choose from a range of subscription plans. These plans determine the resources and support the customer receives for a given price, and essentially act as SLAs between the customer and platform provider. Typically, subscription plans range from very limited free accounts (for testing and trial purposes) to full-blown enterprise accounts (priced at up to $6000 per month). The two typical metrics determined in a subscription plan for data storage services are *number of simultaneous client connections* and *storage capacity* (defined either in terms of MB or number of rows). Heroku itself constantly monitors the resources used by its customers and compares this against the associated quotas; however, it does not notify the application providers in any way when a critical threshold value is approaching, and this can result in a situation where an application is unexpectedly restricted from connecting to a database (due to the connections limit) or inserting new values (due to the storage limit).

Accordingly, our goal in this case study was to fit data storage services with sensing capabilities, so that application providers can be notified in advance whenever a threshold is approaching and can take appropriate action – for example, by disconnecting idle client connections or by upgrading their subscription plan.

To this end, we have implemented a self-adaptation framework to support autonomicity of cloud application platforms. The framework is based on the MAPE-K reference model, and follows our interpretation of cloud application platforms as sensor networks. By annotating monitored values emitted by the sensors (i.e. services) with semantic descriptions (RDF triples), we are able to represent raw data in a uniform format and facilitate semantic interoperability between heterogeneous data sources. This approach also enables us to apply run-time querying (using a streaming query language) and formal reasoning to perform situation assessment, problem diagnosis and adaptation planning (these activities are performed by the 'autonomic manager'). Due to limitations of space we refer interested readers to [10] for implementation details.

Using our framework we manually annotated sensor data (in this case, the current pool of client connections, and util-

[15] https://www.cleardb.com/
[16] https://www.heroku.com/postgres
[17] http://www.mongohq.com/
[18] https://redislabs.com/redis-cloud
[19] https://www.memcachier.com/
[20] http://redislabs.com/memcached-cloud

isation of storage space) with semantic descriptions defined in an OWL ontology, and, using a publish-subscribe mechanism, streamed these values to the autonomic manager (the central component of the network) via a messaging queue. Before the values were delivered to the autonomic manager, they first passed through one or more routing nodes which performed initial filtering and aggregation; by registering standing queries against an RDF stream we selected only critical values (those violating the thresholds), which were then passed on to the autonomic manager. Thus, our approach off-loads some of the computational workload, which might otherwise become a bottleneck of the system. The autonomic manager picks up the critical values and, by reasoning over SWRL rules, determines the nature of the problem and suggests a possible adaptation strategy. In this case study, we defined the critical thresholds as 90% of the overall quotas, and restricted possible adaptation strategies to (a) disconnection of idle client connections and (b) deleting older database values. In practice, of course, the possible adaptation strategies should be somewhat more sophisticated and intelligent.

The results of the case study support the viability of our approach, albeit in a controlled environment, and suggest that further investigation is worthwhile – by treating services as sensors and connecting them into a network, we were able to monitor values coming from distributed sources and detect critical condition violations at run-time in a timely manner. Re-using existing techniques from the Semantic Sensor Web area allowed us to perform dynamic in-memory analysis of monitored data, and create an extensible architecture.

It should be noted that we relied here on the use of existing service metrics, which can be measured via standard mechanisms and do not require the insertion of additional probes into applications' source code. For example, PostgreSQL exposes its run-time usage statistics through the table `pg-stat-activity`, and Redis offers such information through a standard API command, `INFO`. However, various other metrics would require a more intrusive approach, and may require developers to instrument the source code of platforms, user applications or services with sensing functionality. We discuss this, and other potential drawbacks to our approach, in Section 5.3.

## 5 Discussion

There are a number of taxonomies and classifications discussing existing approaches to the monitoring and adaptation of service-based enviroments (we refer the interested reader to [17]). We highlight here two approaches that most closely relate to our own.

Ardissono et al. [2] presented a framework for fault tolerant Web service orchestration and introduced the idea of local and global diagnosers. A local diagnoser is a software component coupled with a Web service, which is responsible for diagnostic hypotheses explaining exceptional occurrences from a local point of view. Then local hypotheses are sent to a central (i.e., global) diagnoser, which is responsible for global reasoning about the whole composite service application. Accordingly, local diagnosers can be seen as sensors, and the global diagnoser as a central component of a sensor network.

A similar concept was introduced by Ameller et al. [1] as part of the SALMon project – a SOA system which uses a monitoring technique to provide run-time Quality of Service information that is needed to detect and eventually correct SLA violations. Monitored SOA systems are equipped with measuring instruments (which are essentially sensors), which are used to obtain basic metrics assocaited with the selected quality attributes. The metrics are then sent to the *Monitor*, *Analyser* and *Decision Maker* (software components associated with the given SOA system). In this sense, they play a role of routing nodes, as they only collect and analyse values from a particular SOA system. However, there is no central component in the SALMon architecture, and managed SOA systems are isolated from each other.

With regard to our own approach, the potential benefits and drawbacks of treating PaaS monitoring as a Web Sensor problem can be summarised as follows.

### 5.1 Potential Benefits

*Extensibility and Scalability.* Existing sensor networks typically comprise a vast number of sensing devices spread over a large area (e.g., traffic sensors distributed across a city-wide road network) and have the capacity to be easily extended (as modern cities continue to grow in size, more and more sensors are being deployed to support their associated traffic surveillance needs). Once the sensors are connected to the existing network, they are ready to report on the current situation. A similar extensible architecture may be introduced to the domain of monitoring cloud platforms, and it follows that the monitoring architecture should support such ad-hoc networks, so that when a new service is deployed on a cloud and integrated into an existing service-based environment, it is ready to be monitored. The same applies to the case when a service component is uninstalled – it should be seamlessly disconnected from the monitoring network.

*Existing solutions and best practices.* Treating a cloud application platform as a sensor network allows us to re-use existing solutions, developed and validated by the Sensor Web community, in the context of monitoring and analysis of streaming heterogeneous sensor data. A particularly promising direction to pursue is applying techniques from the Semantic Sensor Web area – a combination of the Semantic Web and the Sensor Web – so as to enable situation awareness by providing enhanced meaning for sensor observations [27]. This can be done by adding semantic annotations to existing standard Sensor Web languages (typically, by expressing sensor values in the form of RDF triples), thus providing more meaningful descriptions and enhanced access to sensor data. Moreover, this extra layer helps to bridge "the gap between the primarily syntactic XML-based metadata standards of the SWE and the RDF/OWL-based metadata standards of the Semantic Web" [27]. The advantage of this approach is that semantically-enriched sensor data helps to homogenise various data representation formats existing in the SWE and also facilitates more intelligent analysis of observed sensor values by applying formal reasoning. The research work presented in [10] describes our initial results applying techniques from the Semantic Sensor Web to create a self-adaptation framework for managing cloud application platforms. The main advantage of the described framework is the support for run-time analysis of the heterogeneous monitored values by means of reasoning over ontologies and rules.

*Routing nodes.* When implementing a monitoring framework for a cloud platform, intermediate routing or filtering nodes may be of great use. Their responsibility is

(i) to transfer the monitored values to the central component from a physical component (e.g., server, data centre), virtual component (e.g., application container, virtual machine), or logical component (e.g., application system, database) of the monitored platform; and
(ii) to perform initial processing of the incoming values – that is, by filtering and aggregating monitored values it is possible to offload some of the computational tasks from the central monitoring component (which otherwise may become a bottleneck of the whole system), and make the whole framework more scalable.

*Platform independence.* Our framework can act as a high-level conceptual model for creating monitoring frameworks, which is independent of the underlying technology and implementation. In other words, it does not constrain how communications between sensors and the monitoring components should be implemented – depending on a given cloud application platform it may be performed via publish-subscribe messaging queues, a message broker, via the SOAP protocol, etc. Our framework also does not constrain the underlying platform and programming languages – Java, .Net, Ruby, etc., are all equally acceptable.

## 5.2 Extending the approach to IaaS and SaaS levels

Throughout this paper we have specifically focused on monitoring at the aPaaS level. However, it can be potentially extended and applied to the IaaS and SaaS levels as well. That is, it is possible to monitor CPU/memory utilisation and network bandwidth usage by equipping the underlying infrastructure with probes and gauges and connecting them into a sensor network. Similarly, embedding application-specific sensors at the software level can help perform monitoring in terms of business goals.

There are, however, certain caveats to bear in mind. At the lowest (IaaS) level the set of monitoring metrics is rather limited, but quite defined and standardised – network bandwidth and latency, virtual machine CPU/memory utilisations, etc. [16]. Moving up the stack to the PaaS level, the set of possible parameters to be monitored increases, but is still limited to what a given cloud platform offers to its customers (for example, it may include OS-specific metrics like read/write frequencies, the number of running applications, thread counts, etc.), as well as metrics related to utilisation of platform-level services (size of messaging queues, execution times of worker processes, availability of data storage, etc.). At the uppermost SaaS level, the set of monitored metrics expands still further and is almost unlimited, as it covers all possible business-related parameters.

In this context, it would be interesting to explore how the role of routing nodes may change across the three cloud service models. At the IaaS level, we anticipate partitioning into sub-networks to be based on the physical location of monitored elements – a routing node may connect, for example, computational instances hosted on a single physical web server. At the platform level, partitioning may take place based on virtual distribution of sensors either horizontally (those collecting monitored values from multiple applications belonging to a specific user) or vertically (monitoring all instances of a specific service). At the SaaS level, we would expect it to be difficult to find common shared parameters on which we can perform partitioning. Nevertheless, identifying logical clusters – that is, applications with similar business logic – is still possible (for example, monitoring various Twitter-connected applications and identifying trending hash tags can help perform crowd sentiment analysis [11]).

## 5.3 Potential Drawbacks

*Portability.* A major challenge posed by the ideas presented in this paper is that implementing a monitoring framework based on this high-level abstraction is not straightforward. It is not possible to provide truly general guidelines as to how to implement the monitoring functionality, as implementation depends on the characteristics of the particular cloud application platform (its architecture, supported programming languages and frameworks, etc.). For example, our own work in this direction (described in [10]) showed clearly that a monitoring framework, implemented in Java

Spring and deployed on VMWare's Cloud Foundry[21], was portable to another cloud platform only with difficulty, due to its dependence on Cloud Foundry's built-in message queueing service RabbitMQ[22] as a means of transporting monitored values within the monitoring framework.

*Intrusive monitoring.* As we have seen, another shortcoming is the likely need to instrument monitored subjects with sensors, since the number of monitoring metrics exposed by the service providers and application developers and accessible through standard APIs is quite limited – that is, it may be necessary in many cases to follow an intrusive approach to monitoring. Unlike real sensor networks where the sensing functionality is implemented by default, we have to somehow "inject" probes into existing software assets. Even though there exist several open-source PaaS offerings on the market which can be enabled with sensors (e.g., AppScale[23], and the beta version of Cloud Foundry), the majority of cloud application platforms are proprietary, making third-party integration of new sensor capabilities problematic. Therefore, it may be necessary for systems adopting our framework to be implemented as 'trusted entities,' equipped with sufficient access rights to platform components and applications.

## 6 Summary and Concluding Remarks

We have outlined a novel framework for implementing monitoring – a key element for performing run-time self-adaptations. Our approach interprets service-based cloud application platforms as sensor networks, thereby allowing us to apply sensor web techniques to the problems of PaaS-level autonomic management. The main benefits of this approach are extensibility and scalability, the existence of efficient solutions, platform independence, genericity and complementarity to other approaches, the possibility of using routing nodes as filters to avoid bottlenecks; and the potential to address challenges of multi-clouds.

Maintaining application systems hosted on different clouds is a challenging task [4], and prompt run-time monitoring of the kind envisaged here may be a solution to this problem. There are novel ways of (partially) eliminating borders between clouds by establishing, for example, virtual networks between VMs hosted on different platforms [25]. Adopting virtual links and utilising the sensor network approach, it would be possible to regard separate clouds as part of a single network so as to perform monitoring. By thinking in terms of sub-networks and routing nodes, we can treat the whole multi-cloud system as a single sensor network and each of the participating clouds as sub-networks. In summary, we believe the approach we have outlined to be promising, but further investigation is required.

**Acknowledgements** The authors would like to thank Dimitrios Kourtesis for his valuable suggestions and comments on the contents of the paper.

[21] `http://www.cloudfoundry.com/`
[22] `http://www.rabbitmq.com/`
[23] `http://www.appscale.com/`

## References

1. Ameller, D., Franch, X.: Service Level Agreement Monitor (SALMon). In: Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008), pp. 224–227 (2008). DOI 10.1109/ICCBSS.2008.13
2. Ardissono, L., Furnari, R., Goy, A., Petrone, G., Segnan, M.: Fault Tolerant Web Service Orchestration by Means of Diagnosis. In: Gruhn, V., Oquendo, F. (eds.) Software Architecture, *Lecture Notes in Computer Science*, vol. 4344, pp. 2–16. Springer, Berlin Heidelberg (2006). DOI 10.1007/11966104_2
3. Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: Above the Clouds: A Berkeley View of Cloud Computing. Tech. Rep. UCB/EECS-2009-28, Electrical Engineering and Computer Sciences, University of California at Berkeley (2009)
4. Baryannis, G., Garefalakis, P., Kritikos, K., Magoutis, K., Papaioannou, A., Plexousakis, D., Zeginis, C.: Lifecycle management of service-based applications on multi-clouds: a research roadmap. In: Proceedings of the 2013 International Workshop on Multi-cloud Applications and Federated Clouds, New York, NY, USA, pp. 13–20 (2013)
5. Bratanis, K., Dranidis, D., Simons, A.: The Challenge of Engineering Multi-Layer Monitoring & Adaptation in Service-Based Applications. In: Proceedings of the 7th Annual South East European Doctoral Student Conference, pp. 497–503 (2012)
6. Brazier, F., Kephart, J., Van Dyke Parunak, H., Huhns, M.: Agents and Service-Oriented Computing for Autonomic Computing: A Research Agenda. IEEE Internet Computing 13(3), 82–87 (2009). DOI 10.1109/MIC.2009.51
7. Breskovic, I., Haas, C., Caton, S., Brandic, I.: Towards Self-Awareness in Cloud Markets: A Monitoring Methodology. In: Proceedings of IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC), IEEE, pp. 81–88 (2011). DOI 10.1109/DASC.2011.38
8. Clayman, S., Galis, A., Chapman, C., Toffetti, G., Rodero-Merino, L., Vaquero, L., Nagin, K., Rochwerger, B.: Monitoring service clouds in the future internet. In: Tselentis, G., Domingue, J., Galis, A., Gavras, A., Hausheer, D., Krco, S., Lotz, V., Zahariadis, T. (eds.) Towards the Future Internet: Emerging Trends from European Research, pp. 115–126 (2010). DOI 10.3233/978-1-60750-539-6-115
9. Cox, M., Ellsworth, D.: Application-Controlled Demand Paging for Out-of-Core Visualization. In: Proceedings of Visualization '97, pp. 235–244 (1997). DOI 10.1109/VISUAL.1997.663888
10. Dautov, R., Paraskakis, I., Kourtesis, D., Stannett, M.: Addressing Self-Management in Cloud Platforms: a Semantic Sensor Web Approach. In: Proceedings of the International Workshop on Hot Topics in Cloud Services (HotTopiCS'13), April 20-21, 2013, Prague, Czech Republic. ACM (2013)
11. Diakopoulos, N.A., Shamma, D.A.: Characterizing debate performance via aggregated Twitter sentiment. In: Proceedings of the 28th International Conference on Human Factors in Computing Systems (CHI 10), pp. 1195–1198 (2010). DOI 10.1145/1753326.1753504

12. Dobson, S., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., Zambonelli, F.: A survey of autonomic communications. ACM Transactions on Autonomous and Adaptive Systems (TAAS) 1(2), 223–259 (2006)
13. Galante, G., de Bona, L.C.E.: A survey on cloud computing elasticity. In: Proceedings of IEEE Fifth International Conference on Utility and Cloud Computing (UCC), IEEE, pp. 263–270 (2012).
14. A special report on managing information: Data, data everywhere. The Economist (25 February 2010)
15. Horn, P.: Autonomic Computing: IBM's Perspective on the State of Information Technology. Computing Systems 15, 1–40 (2001)
16. Iosup, A., Ostermann, S., Yigitbasi, M.N., Prodan, R., Fahringer, T., Epema, D.H.J.: Performance analysis of cloud computing services for many-tasks scientific computing. In: IEEE Transactions on Parallel and Distributed Systems 1(6), 931–945 (2011)
17. Kazhamiakin, R., Benbernou, S., Baresi, L., Plebani, P., Uhlig, M., Barais, O.: Adaptation of Service-Based Systems. In: Papazoglou, M., Pohl, K., Parkin, M., Metzger, A. (eds.) Service Research Challenges and Solutions for the Future Internet, *Lecture Notes in Computer Science*, vol. 6500, pp. 117–156. Springer, Berlin Heidelberg (2010). DOI 10.1007/978-3-642-17599-2_5
18. Kephart, J., Chess, D.: The vision of autonomic computing. Computer 36(1), 41–50 (2003)
19. Kourtesis, D.: Towards an ontology-driven governance framework for cloud application platforms. Tech. Rep. CS-11-11, Department of Computer Science, The University of Sheffield (2011)
20. Kourtesis, D., Bratanis, K., Bibikas, D., Paraskakis, I.: Software co-development in the era of cloud application platforms and ecosystems: the case of CAST. In: Camarinha-Matos, L.M., Xu, L., Afsarmanesh, H. (eds.) Collaborative Networks in the Internet of Services, *IFIP Advances in Information and Communication Technology*, vol. 380, pp. 196-204. Springer Berlin Heidelberg (2012). DOI 10.1007/978-3-642-32775-9_20
21. Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., Byers, A.H.: Big data: The next frontier for innovation, competition, and productivity. Tech. Rep., McKinsey Global Institute (2011)
22. Mell, P., Grance, T.: The NIST Definition of Cloud Computing. Tech. Rep. Special Publication 800–145, National Institute of Standards and Technology (2011)
23. Natis, Y., Knipp, E., Valdes, R., Cearley, D., Sholler, D.: Who's Who in Application Platforms for Cloud Computing: The Cloud Specialists. Tech. Rep., Gartner Research (2009)
24. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: a research roadmap. International Journal of Cooperative Information Systems 17(2), 223–255 (2008)
25. Rochwerger, B., Breitgand, D., Epstein, A., Hadas, D., Loy, I., Nagin, K., Tordsson, J., Ragusa, C., Villari, M., Clayman, S., Levy, E., Maraschini, A., Massonet, P., Muñoz, H., Tofetti, G.: Reservoir – When One Cloud Is Not Enough. Computer 44(3), 44–51 (2011). DOI 10.1109/MC.2011.64
26. Sha, L.: Using simplicity to control complexity. IEEE Software 18(4), 20–28 (2001). DOI 10.1109/MS.2001.936213
27. Sheth, A., Henson, C., Sahoo, S.: Semantic Sensor Web. IEEE Internet Computing 12(4), 78–83 (2008). DOI 10.1109/MIC.2008.87
28. Wei, Y., Blake, M.: Service-Oriented Computing and Cloud Computing: Challenges and Opportunities. IEEE Internet Computing 14(6), 72–75 (2010). DOI 10.1109/MIC.2010.147