Scalable and Configurable Monitoring System for Cloud Environments

A. Di Stefano, G. Morana, D. Zito
Dept. of Electrical, Electronic and Information Engineering
University of Catania
Email: giovanni.morana@dieei.unict.it

***Abstract*—The scale of the current cloud infrastructures and the heterogeneity of applications running on them pose significant challenges about resources management on clouds.**
This paper proposes a monitoring system that allows user to collect and aggregate data in a form flexible and adequate to the integration with the management policy that he adopt. This monitoring system is supplied by a a configurable and highly scalable version of publisher-subscriber pattern.
It gives user the capability of selecting data he should obtain from the provider with the desired grain of triggering and is suitable to support distributed and cooperative management systems.

I. Introduction

Today cloud Systems [1], [2] provide on-demand access to a shared pools of configurable computing resources that can be leased or acquired by means of a large set of ad-hoc services that face all the issues related with the exploitation of these resources, e.g. automatic scaling up/down, load balancing, faults management.
In particular Infrastructure as a Services (IaaS) systems allow users to build a virtual environment on top of the resources they rent. Such virtual environments are highly configurable: users can add, move, remove one or more cloud resources, and also modify the mapping of the application components on virtual machines [3] in order to better characterise performance of applications.
In such scenario, both the setting-up of the hosting virtual machine and the optimisation of the system performance imply a a detailed knowledge [4] about the state of both applications and resources.
For an effective management of the resources of the virtual environment got by the cloud provider, the user should be able to monitor and integrate, in a flexible manner and with an adequate grain, information relevant with the high level application and with the underlying real performance of the owned resources.
Unfortunately, triggering the behaviour of the application and at the same time evaluating the real performance of the resources is difficult since they regard two distinct point of view, that typically are hidden each other and are under the control of different subjects: user and provider.
The user can collect, indirectly, a lot of data by adopting suitable query strategies to get information on virtual hardware resources (e.g. CPU, memory or bandwidth from OS tools) and on cloud's software services (e.g., load alancing, databases, queue systems [5], [6]). However, these are desegregated data, heterogeneous for source, type, creation and delivery time and they make very difficult, for the users, identify common elements that allow to create valuable knowledge exploitable for optimising the performance of their applications.
To enforce the management of the application, the cloud provider should offer to users a service for an effective data collection and aggregation. This service should provide a common logical schema to associate a semantic characterisation to the monitored data. This simplifies the user-side process of data gathering and handling, enabling a suitable utilisation in the management policy. This paper proposes a scalable and configurable monitoring system that adopts, as local schema, a *common tagging model* allowing a logical selection and aggregation of monitored data.
The paper focus on three main tags: ”topic”, which provides information about the management policy contexts (e.g. performance, fault, accounting [7], [8]), ”context”, which identifies geographical distribution of the physical resources (e.g. from specific geographical regions or form different clusters of the same region), and ”content”, which provides several aspects of the same aggregated data. The choice of this kind of tags is the more natural but does not limit the approach that can consider any other possible characterisations.
The system here proposed, called Cloud Monitoring Architecture - CMA, is based on the publisher-subscribed pattern, as many other solutions and best practices proposed in literature for the designing of monitoring systems. Indeed, this messages-based, architectural patter has demonstrated to be valid to build effective monitoring systems, able to guarantee a high level of configurability and scalability, two key issues for facing the dynamics of clouds.
The publisher-subscriber pattern here proposed is designed for creating a distributed and collaborative management systems, a kind of management very useful in clouds where the entity controlling hardware resources is different from the entities controlling services running in these resources.
The paper describes the structure and the functioning of the proposed version of publisher-subscriber pattern, the importance of *context*, *topic* and *content* in filtering and organising the monitored information and how to use the pattern for creating a scalable, flexible and customisable monitoring system, supporting distributed and collaborative management.
The capability to create specific user-dependent sensors (Publishers) and to exploit sets of highly configurable Subscribers, collecting data basing on topics, context and content, enables users to define their own monitoring and analysis policies.
The service proposed represents the basic building block for introducing the concept of *management as a service*, which give to a user the ability to directly choose (or influence) the

978-0-7695-5002-2/13 $26.00 © 2013 IEEE
DOI 10.1109/WETICE.2013.57

management policies regulating the behaviour of its applications hosted on cloud resources.
The rest of paper is organised as follows. Section 2 provides a view of the reference scenario. Section 3 and 4 illustrate, in details, the system architecture. Section 6 shows the results of led simulations. Section 7 presents some related works. Section 8 concludes the work.

II. Reference Scenario

A. Resources

In this paper, the term ”cloud” identifies a set of nodes (Cloud Nodes, CNs) which are connected via shared communication channels.
Each CN is a physical entity that groups a number n of cores, sharing main memory, disk storage and a given quantity of bandwidth for network communications. Each CN hosts a set of Virtual Machines (VMs), which represent logical containers providing ”well defined” environments where applications and/or services are executed.
Each VM consists of a specific set of software resources (e.g. OS, scientific applications or libraries) running over virtual hardware resources. All the VMs hosted on a CN are handled by the local Virtual Machine Monitor (VMM): it takes care to start and to stop VMs, or to migrate them onto different CNs.
Example of this kind of (IaaS) cloud environments are Amazon EC2 [5] and RackSpace [6]: these public providers offer to users the ability to configure owned VMs (based on their hardware and software requirements) in order to build their private, virtual execution environments.

B. Roles and Targets

In general, it is possible to distinguish three main roles in the above described cloud environments: *Cloud Provider*. *Cloud Consumer* and *Services User*.

- *Cloud Provider*. The owner of hardware resources and software services composing the cloud infrastructure. It offers to Cloud Consumers the ability to exploit its resources by means of one or more virtualization layers.
- *Cloud Consumer*. The entity who exploits the virtualised resources supplied by the Cloud Provider either to run own applications or to provide a service to external end-users (following a third parties schema)
- *Service User*. The end-user, i.e. the entity that pays to use cloud services.

In the following, the term user will be used indifferently for both Cloud Consumer and Service User, a simplification adopted by the majority of research papers on the subject.

C. Monitoring: characterisation

In the reference scenario here considered, there are two different classes of variables in two different classes:

- *Infrastructure-dependent variables*, holding information about both the physical resources composing CNs (e.g. CPU number, type and status, physical memory, network consumption) and the VMM state.
This class of variables is managed exclusively by the Cloud Provider.
- *User-dependent variables*, related with the execution of the applications running on CNs (e.g number of running tasks, cpus and memory usage, response time).
In turn, these variables can be subdivided in: i) *directly observable* variables (i.e., variables monitored by specific sensors provided by the user) and ii) *indirectly observable* variables (i.e. variables inferred by multiple data coming from the VMM).

The greater the number of monitored variables, the greater the analysing and processing costs of them (in terms of computation power and time, storage space or bandwidth) and, as a consequence, the greater the cost of the management.

D. Problem statement

As said in Section II-A , a cloud is a large set of resources organised in clusters (CNs) geographically distributed. For effective managing them, each Cloud Provider has to be able to monitor and to control all these resources.
The management process is a complex task because it could take into account a large amount of different aspects, especially if multiple, unrelated goals have to be handled.
For example, in order to provide a QoS-guaranteed services, a management systems has to monitor hardware and software resources to assure the dealt performance (performance management), to prevent bad functioning (fault management) and to register data about their usage (accounting management).
Each one of these sub-kind of management aspects relies on several attributes coming from hardware and software resources: among these attributes, some are in common to all the resources whereas others are specific to given types of them.
In general, each resource is featured by a set of attributes that characterises its static and dynamic behaviour.
The monitoring process, which is at the base of management system, foresees that when a new data is available, the resource *publishes* it raising an event (event-based notification).
The task of the monitoring system consists in handling all these events, collecting the related data in a scalable, flexible, and effective way.
The solution here proposed envisages the adoption of a monitoring systems based on an optimised version of the publishers-subscribers pattern that combines i) the use of different topics, ii) the ability of adapting the content of data to the subscriber interests and, very important, iii) to organise dispatchers in different *monitored regions* mapping the underlying organisation in clusters.

III. Cloud Monitoring Architecture: Structure

The structure of the Cloud Monitoring Architecture (CMA), shown in the fig. 1, is based on a scalable version of the publisher-subscriber (P/B) pattern. It consists of six type of components: three of these, *Dispatchers*, *Subscribers*

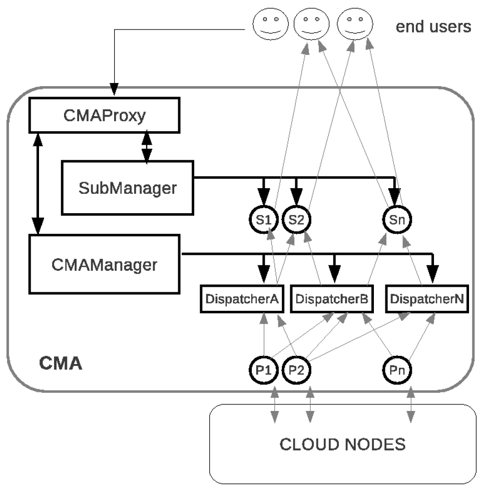

Fig. 1. Structure of CMA

and *Sensors* (i.e., Publishers) belong to the simple version of the P/S pattern whereas *CMAProxy*, *CMAManager* and *SubscribersManager* are the additional components that improve the pattern performance.

1) *CMAProxy* is a *event-driven proxy* responsible for accepting the requests coming from users (or software agents working on behalf of them), parsing, interpreting and forwarding them to the SubscribersManager.

2) *SubscribersManager* is the *subscribers coordinator* (SubManager in the fig. 1). It handles all the operations related to the creation of new subscribers, to the association between subscribers and dispatchers and to the coordination of the active subscribers.

3) *CMAManager* is the *dispatchers coordinator*. It coordinates all the activities related to dispatchers, including their configuration, deployment and maintenance. It also provide a discovery services for the information collected by the topics. It handles: i) the Semantic Schema for each type of monitoring variable, ii) the list of active dispatchers and, for each dispatcher, iii) the list of available monitored variables. It also monitors the state of active dispatchers (looking for failures).

4) *Dispatchers* are the *topics manager*. They are coordinated by the *CMAManager*. They handle all the lifecycle of the topics.

5) *Subscribers* are the *interface* to get data/variables. Selecting a given set of subscribers, each user defines the specific set of variables that wants to monitor.

6) *Sensors* are the lowest components of the monitoring system, are platform or application dependent and represent the data sources for the proposed solutions. They monitor the status of the software and hardware variables of the cloud and act as publishers (P_1, P_2, and P_n in 1).

IV. Cloud Monitoring Architecture: Components and their functioning

A. Sensors

Sensors collects information about resources belonging to cloud.
Each CN is characterised by a specific set of sensors $cn_i = \{s_1, s_2, ..., s_k\}$. Sensors are distinguished in Infrastructure and User sensors:

- *Infrastructure sensors* perform a continuos monitoring of the state of resources in each CN. They are provided by the Cloud Provider.
- *User sensors* perform i) an active monitoring of the *directly observable* variable (i.e., using readily available measurements, typically from virtualised operating system facilities and ii) a passive monitoring of the *indirectly observable* variables (i.e., estimating measurements through the tools provided by the VMM).

Sensors can capture the dynamic information of each CN using either Polling approach or Pushing (event driven/ interrupts) approach, according to what established from the sensor's owner.

1) Management of the monitored information: As said before, monitored variables are semantically characterised by means of three factors: *Topic, Context, Content*

- *Topic* identifies the kind of management in which these information can be involved (e.g., performance, fault or accounting management).
- *Context* focuses on the physical location of the CN within the cloud (e.g. acquiring data produced by resources belonging to a specific cluster).
- *Content* is the set of information related to an variable. It is strictly related with the number of attributes triggered by the specific event.

For each sensor (i.e., publisher), there is a XML schema containing all the parameters monitored and the related values of Context and Topic.

2) Publishing process: Each sensor publishes its data in four phases:

1) data is organised in subsets of data according to their 3-upla *Topic / Context / Content*;
    a) the Context is assigned basing on the location of the resource;
    b) basing on the Content, CMA extracts every attribute related with the observed data
    c) one or more Topics are assigned to each monitored data;
2) Select all the topics interested from each subset;
3) Update these topics with the information collected for that particular subset;

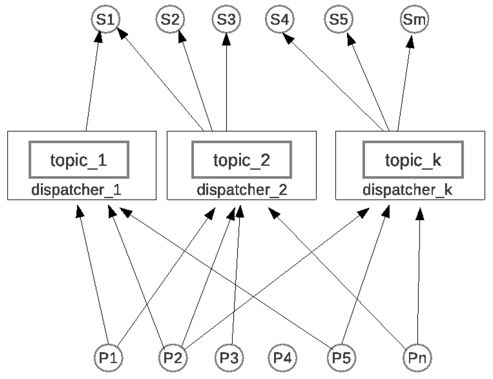

Fig. 2. Overlay of Dispatchers

4) Reiterate the last two passes, until any subset has been processed

B. Dispatchers, Topics and Context

Topics and Context are the main components of the architecture here described.
As described in the previous subsection IV-A, each CN is characterised from a specific set of sensors.
Topics aggregates the information related to sensors, according to specific policies.
Authors distinguish two main type of topics: *infrastructure* or *user-driven*, depending on who is the owner of the topic: the cloud provider in the first case, the user in the second case.
Dispatcher handles data coming from a given context: for each context, dispatchers are identified by the topics that hold. This means that every dispatcher is in charge of one or more topics. Basing on the value of 3-upla Topic/Context/Content, each sensor can publish its information on one or more topics.
Thus, differently from the classical Publishers/Subscribers schema, here each topic can be connected to one or more publishers. On the same time, topics can be accessed from one or more subscribers.
Therefore, by means of each topic a dispatcher groups: i) a set of publishers (those that publish information on it); and ii) a set of subscribers (those interested in the update of the information collected by the topic).
This implies that the set of dispatchers creates an overlay network that links together subscribers with publishers (see 2).
Such virtual topology is given by a graph, which is assumed to be acyclic and connected.
Note that Publishers/Subscribers have exactly one link and act as senders or receivers of events, while the Dispatchers are participants with more than one link and act as filter for events that transit through them.

C. CMAManager

CMAManager is the coordinator of dispatchers.
It is responsible both for the creation and the starting of each dispatcher in the monitoring system. Once a new dispatcher is started, *CMAManager*: i) configures it according to the information stored in a configuration file, ii) assigns it an URI (IP address) for being addressed by publishers and iii) adds an entry in its internal repository of active dispatchers.

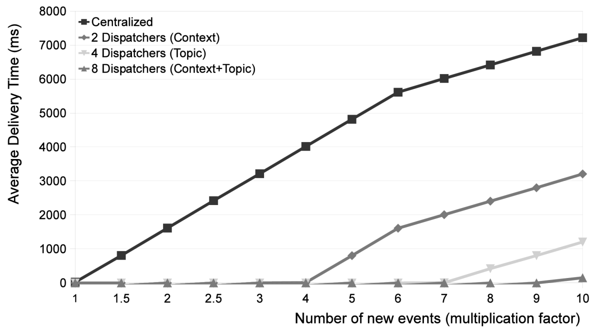

Fig. 3. Average Delivery Time

The dispatcher becomes visible both to publishers and to subscribers.
CMAManager can also modify active dispatchers i) adding or removing Topics, ii) changing the semantic meaning of a given Topic and iii) distributing the management of a given Topic in two or more different dispatchers.
The internal repository of the *CMAManager* consist of an XML-based database storing the information related with every dispatcher in the system.

D. Subscribers

Subscribers represent the main components of the CMA because are the access points to the monitoring systems here described. Each user can set one or more subscribers to be notified about the lifecycle of the variables that he wants to monitor.

E. SubscribersManager

SubscribersManager is the subscribers' coordinator.
It drives the user in the choice of the subscribers by means of which monitoring the cloud.
If a user want to monitor a specific set of cloud nodes and or a specific set of applications, firstly he has to contact the *SubscribersManager* in order to know the list of available monitoring variables for that cloud; afterwards he has to send a subscription request to all the dispatchers managing the monitoring variables he is interested. These requests are collected from the SubscribersManager and forwarded to the desired dispatchers. Each dispatcher, on the basis of the policy it implements, can accept or refuse the user requests. The *SubscribersManager* collects all the responses and starts a subscription for each dispatcher that has sent a subscription confirm. At the same time it notifies the user with the rejects of all the dispatchers that have refused the subscription.

F. CMAProxy

It is a simple *event-driven proxy* responsible for accepting the requests coming from the users. It parses them, checks for their correctness and if it is possible, forwards them to the *SubscribersManager*.

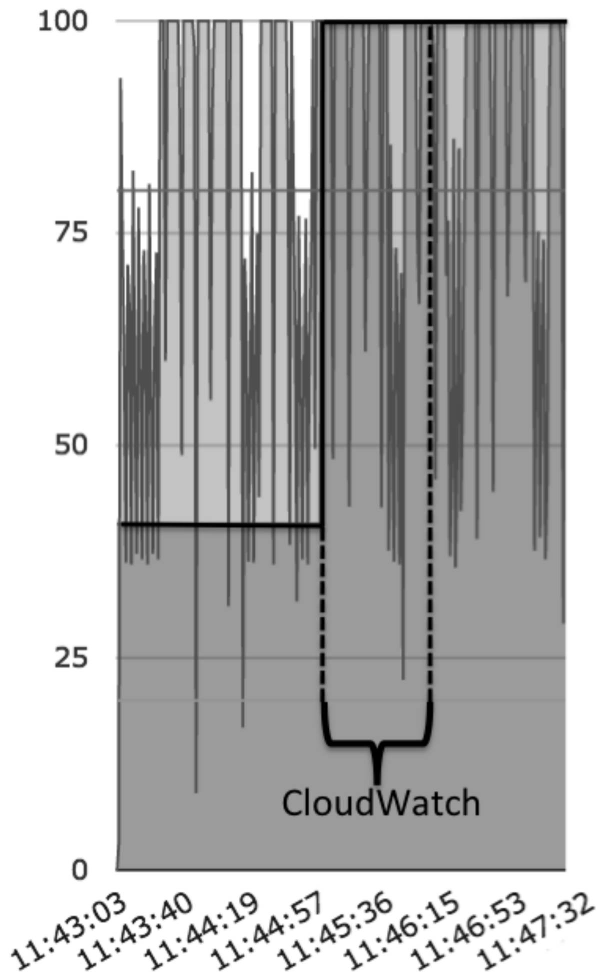

Fig. 4. CMA vs. CloudWatch (CPU usage in Amazon VM)

V. Performance evaluation

This section illustrates and discusses the results of simulations done for demonstrating the advantages obtained in using the proposed cloud monitoring system. In particular, these simulations aim to demonstrate the ability of the proposed solution to be responsive when the number of generated events (i.e. new data) increases (*scalability*).

The index used for the evaluation and comparison of the system is the *average delivery time* (aDT), i.e. a measure (in 10^{-3} seconds) of the time elapsed from the generation of an event in the publisher to the reception of the same event in the subscriber(s): the shorter this time range, the better the performance of the monitoring system.

Simulations take into account an increasing number of publishers generating events and four different number of dispatchers redirecting events (1 - centralised approach, 2, 4 and 8 dispatcher): the number of subscribers receiving events is maintained fixed.

Each publisher generates its events with a given frequency (ranging from 0.3 to 3 event/s) and each event is characterised by different values of the 3-uple $<$ *topic, context, content* $>$.

The Fig. 3 summarises the obtained results.

This figure shows the aDT's value for a monitoring system having a single dispatcher (*centralised* solution), for a system with two dispatchers (referring to 2 different contexts, CN_A and CN_B), for a systems with four dispatchers (one for each considered topic - e.g., Faults, Performance, Accounting and Security) and, lastly, for a system with eight different dispatchers, each one handling a specific topic within a specific context.

Starting from an initial steady-state condition, i.e. when even a single dispatcher is able to handle all the generated events without introducing an appreciable delay, the Fig. 3 shows the increment of the aDT's value when the total number of generated events increases (from 1.5 to 10 times in respect with the initial value, i.e. 100 event/s). As expected, the greater the number of dispatchers in the system, the lower the value of the aDT in delivering events.

However, the ratio among delays does not follow a linear trend. In fact, even if in the centralised solution the aDT starts to grow, immediately and linearly, with the number of new events , all the other solutions present an *activation* threshold (different for each considered solution).

Until this threshold is not reached, the values of aDT for systems having multiple dispatchers is not appreciable. When the threshold is reached, the aDT starts to grow linearly and assumes values proportional to the ones assumed by the centralised solution. This threshold (equals to 4 - multiplication factor - for 2 Dispatcher, 6 for 4 dispatchers and 9 for 8 dispatchers) gives a measure of the scalability of the proposed solution in respect to the centralised solution: the greater the number of used dispatcher, the greater this threshold, the greater the system scalability, i.e. the ability to maintain almost constant the aDT when the number of new events grows.

An important consequence of a high-responsive monitoring system is the ability to follow out variables that change their values very quickly. The fig . 4 shows the values of CPU usage measured in an Amazon VM [5] using the proposed approach (in light-grey) compared with the ones measured using the standard monitoring system provided by Amazon (CloudWatch, in black/dark-grey).

VI. Related Work

Monitoring is a critical topic in the IT industry: there are a lot of tools and solutions proposed for its management.

Within the PaaS context, Google App Engine [17] provides an App Engine System Status Dashboard to show to the application developers the status of the applications on the cloud.

Within the IaaS context, Amazon provides a service called CloudWatch [19] for the high level resource monitoring of Amazon services like EC2, Elastic Load Balancing, and Amazon's Relational Database Service. CloudWatch collects the information related with different configurable measurement types form its target and stores them implicitly for a period of two weeks.

There also third-party tools that are developed to keep watch over the clouds, such as CloudStatus developed by Hyperic Inc. [18], which can both monitoring Amazon service and Google App Engine.

All these enterprise solutions present two big differences with the one proposed in this paper.

First, solutions as App Engine System Status Dashboard and CloudWatch collect and organise only high level information (those associated with the virtual machine and only in some case with the applications themselves), while the CMA provide a complete view of all the cloud resources involved in the execution of applications, both Infrastructure and User dependent. Second, the time grain for the monitoring among this reviewed solutions and CMA is different. CMA allows the user to establish (by means of ad hoc sensors) the temporal grain use for its monitoring, while e.g. CloudWatch imposes a wide time-grain (minute(s)) that makes impossible the management of variables that change their values very quickly (see Fig. 4). CMA adopts both *polling* and continuos monitoring approaches.
Polling protocols have been extensively studied in the literature [9], [10], [11], [12].
Note that continuous monitoring can be approximated through periodic polling (sometimes called: *N-time queries* (e.g., [11], [13]), at the risk of missing changes on the aggregation during the polling intervals. Protocols for continuous monitoring of aggregates have also been proposed in the literature [14], [15], [16]. The common property of these protocols is that they use incremental aggregation techniques.

VII. Conclusion

In this paper the authors presented a cloud monitoring system (CMA) based on a configurable and highly scalable version of the publisher-subscriber pattern.
The main goal of this system is to provide to users the ability to aggregate data on cloud (coming from multiple and heterogeneous sources) basing on topics, context and content, allows them to define their own monitoring and analysis policies for controlling the behaviour of their own applications running on cloud.
In particular, in this paper the authors described the structure and the functioning of the proposed monitoring systems, explained the importance of using tags (topic, context and content) in filtering and aggregating data and shown the advantages in using the proposed publish-subscriber based solution.
Future investigations will evaluate the advantages on CMA in the context of DIME Networks [4], [20].

References

[1] William Voorsluys, James Broberg, and Rajkumar Buyya, Introduction to Cloud Computing, Cloud Computing: Principles and Paradigms, 1-41pp, R. Buyya, J. Broberg, A. Goscinski (eds), ISBN-13: 978-0470887998, Wiley Press, New York, USA, February 2011.

[2] Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, and Albert Zomaya, A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems, Advances in Computers, Volume 82, 47-111pp, M. Zelkowitz (editor), ISBN: 978-0-12-385512-1, Elsevier, Amsterdam, The Netherlands, March 2011.

[3] Antonella Di Stefano, Giovanni Morana, Daniele Zito: Improving the Allocation of Communication-Intensive Applications in Clouds Using Time-Related Information. ISPDC 2012: 71-78

[4] Mikkilineni, R.; Morana, G.; , "Injecting the Architectural Resiliency into Distributed Autonomic Systems Using DIME Network Architecture," Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on , vol., no., pp.867-872, 4-6 July 2012 doi: 10.1109/CISIS.2012.170

[5] Amazon EC2 website. http://aws.amazon.com

[6] RackSpace website. http://www.rackspace.com

[7] Morana, G.; Mikkilineni, R.; Scaling and Self-repair of Linux Based Services Using a Novel Distributed Computing Model Exploiting Parallelism. 20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), pp.98-103, 27-29 June 2011 doi: 10.1109/WETICE.2011.18

[8] Eberbach E., Mikkilineni R., Morana G., Computing Models for Distributed Autonomic Clouds and Grids in the Context of the DIME Network Architecture, Proc. of 21st IEEE Intern. Conf. on Collaboration Technologies and Infrastructures WETICE 2012, Toulouse, France, June 25-27, 2012, 125-130, DOI: 10.1109/WETICE.2012.10.

[9] C. Avin and C. Brito. Efficient and robust query processing in dynamic environments using random walk techniques. In International Symposium on Information Processing in Sensor Networks, pages 277-286, New York, NY, USA. ACM Press. 2004.

[10] M.Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. ACM Transactions on Computer Systems, 23(3):219-252, 2005.

[11] A.Kertsz and P. Kacsuk. A taxonomy of grid resource brokers. In Pter Kacsuk, Thomas Fahringer, and Zsolt Nmeth, editors, Distributed and Parallel Systems, pages 201- 210. Springer US, February 2007.

[12] A. G. Prieto. Adaptive Real-time Monitoring for Large-scale Networked Systems. PhD thesis, KTH Royal Institute of Technology, October 2008.

[13] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny Aggretation service for ad-hoc sensor networks. In Symposium on Operating Systems Design and Implementation, pages 131-146, 2002

[14] N. Ahmed, D.Hadaller, and Srinivasan Keshav. Incremental maintenance of global aggregates. Technical report, School of Computer Science, University of Waterloo, CS-2006-19. 2006.

[15] A. G. Prieto and R. Stadler. A-GAP: An adaptive protocol for continuous network monitoring with accuracy objectives. IEEE Transactions on Network and Service Management, 4(1):2-12, June 2007.

[16] A. Silberstein, R. Braynard, and J. Yang. Constraint chaining: on energy-efficient continuous monitoring in sensor networks. In ACM SIGMOD International Conference on Management of Data, pages 157-168, New York, NY, USA. ACM Press. 2006

[17] Google Application Engine. http://code.google.com/appengine/

[18] Hyperic official website. www.hyperic.com/

[19] Amazon CloudWatch website. http://aws.amazon.com/cloudwatch/

[20] Mikkilineni R., Designing a New Class of Distributed Systems, ISBN 978-1-4614-1923-5, Springer, 2011