Self-Management for Cloud Computing

Mariachiara Puviani
University of Modena and Reggio Emilia
Modena, Italy

Regina Frei
Imperial College London, London, UK
and University of Durham, Durham, UK

Abstract—**Cloud computing is an emergent technology in the process of becoming ubiquitous. This requires strategies to deal with challenging situations. While a P2P structure is suitable under usual circumstances, other structures may be required in case of strong perturbations and disruptions. This paper describes a partial blackout scenario: the Cloud uses self-management properties to switch from a peer-to-peer structure to a temporary centralised structure, and then returns to normal. The system remains adaptive at all times, while maintaining performance under aggravated conditions. To achieve such self-management, a specific design pattern is suggested. Properties for self-adaptive and self-managing system are described, and implementation perspectives are discussed.**

Keywords*—*Cloud computing; self-management; design patterns

I. Introduction

Cloud computing is a powerful technology coming with a set of challenges across many application scenarios. Social media and virtual networks are growing at an increased pace. People want to be connected at all times in every part of the world: they need to work and maintain contact with family and friends. Users may log into their accounts on remote servers, for instance to send job applications, manage their Linked-in profiles, use file sharing sites like Dropbox or Google Drive or have video-conferences through Skype. Examples are plentiful.

On the one hand, cloud computing has great potential to deal with huge volumes of computing requests, but on the other hand, it also comes with a lot of challenges like load balancing, changing nodes, varying application requirements, a need for fault tolerance and others. Moreover, a cloud computing system must be self-managed and self-adaptive, as it is impossible to manage by humans due to its sheer size and due to an extremely changing environment of a cloud.

Guidelines that permit a system to cope with these challenges and autonomously manage and adapt itself to new situations – including changing its configuration if necessary – can be very useful to developers of cloud computing systems. This paper provides such guidelines described as patterns, illustrates their use in a blackout scenario, and discusses advantages as well as challenges of such an approach.

Organisation of this paper: Section II reviews related work. Section III briefly discusses utility and challenges of cloud computing. Section IV explains how cloud computing may benefit from the implementation of self-* properties. Section V details patterns for self-* properties in cloud computing. Section VI discusses perspectives of an implementation scenario, and Section VII concludes the paper.

II. Related work

Work on **self-* properties** and self-management in particular is abundant in literature. For example, Horn [15] presents self-* properties as autonomic system / application characteristics. Also Parashar and Hariri [16] describe self-* properties for autonomic systems in depth, especially focusing on the self-management perspective addressed in four primary system / application aspects: configuration, optimisation, protection, and healing. Brun [17] presents an evaluation methodology on self-adaptation and self-management considering them the future of distributed systems.

Furthermore, Kramer and Magee [18] present self-management as one of the most powerful means to create systems that are scalable, support dynamic composition and rigorous analysis, and are flexible and robust in the presence of change. These are the challenges for cloud computing systems. Kramer and Magee propose an architectural reference model for identifying how self-management is possible.

Herrmann et al. [19] present self-management as a possible solution for IT infrastructure development (which usually are also the problems in a cloud computing system): it is not realistic for human operators to maintain control over a system that consists of thousands of nodes and large amounts of data. Also the challenges encountered with a self-managing approach are similar.

More closely related to our work, the use of feedback loops to create self-managing system is discussed in Patikirikorala et al. [20]. They propose a technique to design self-managing control systems based on multiple models. This provides better performances under changing operating conditions.

However, none of these works focus on the relationship between self-* properties (especially self-management) and Cloud computing, as this paper does.

Cloud computing is a young technology with many issues to address. Vaquero at al. [21] provide a definition of cloud computing along with a survey and a comparison with grid computing, to make users understand what these new systems really are. Also Zhang et al. provide a survey [22] on cloud computing. According to the authors, cloud computing systems are self-organising, as resources can be dynamically allocated and service providers manage their own resource consumption.

In our paper, we suggest a solution for cloud self-management at system level, whereas Brandic [23] presents an approach focusing on individual nodes. The objective is for the each node to assure that Service Level Agreements (SLAs) are not violated. Essentially, each node interacts with a MAPE loop [24] through interfaces for job management, negotiation and self-management.

Many other works are related to the **use of patterns** to create a self-adaptive and self-managing systems. Weyns et al. [25] introduce the concept of patterns for self-adaptive systems based on control loops. They describe how control loops are used to enforce adaptivity in a system and present a set of patterns. Furthermore, Puviani et al. [13] provide an example of a self-managing configuration of a robotic system.

In this paper, we suggest the use of adaptation patterns in the context of Cloud computing.

III. Cloud Computing

According to the (US) National Institute of Standards and Technology (NIST), Could Computing [1] can be defined as follows:

"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models."

Cloud computing provides users with resources such as virtual machines, storage space, processing power, or applications through the Internet [2]. Clouds are able to group together a large number of computer servers and other services and applications. Users can benefit from a wide range of resources without the necessity to install or manage them, which would often be complicated, costly and time-consuming. Users also have the possibility to add their own resources to the cloud or remove them at their choice.

Furthermore, an increasing number of companies provide their employees with a *private cloud* offering computing power, storage space and dedicated applications, coming with the possibility to work remotely. Depending on which resources and services are provided, three different layers of cloud solutions can be distinguished [3], viewed as *services* offered to the users.

- **Infrastructure as a Service:** Providing resources such as virtual machines, virtual network switches, and data storage.
- **Platform as a Service:** Providing development and execution platforms for cloud applications.
- **Software as a Service:** Providing applications, for example a complete e-mail, calendar or document collaboration solution.

From the perspective of the user, the most important property of cloud systems is that they are "always on" [4]. Hence, the user needs to be able to trust that the resources and applications in the cloud will never fail. This is the reason why self-management and self-adaptation to environmental changes are crucial for cloud computing: they allow the cloud to be running continuously, despite of problems that may occur.

To fully realise the potential of cloud computing, cloud services need means for being flexible in their service delivery to meet varying users requirements, while maintaining the isolation of the users from the underlying infrastructure [5]. Along with flexibility, cloud computing makes it possible for users to no longer have to worry about computing issues like server updating and others.

Users may join and leave the cloud at any time; authorisation to join is required for private clouds. Users joining the cloud with their devices provide the cloud community with further resources, and thus the cloud grows. However, this possibility to join or leave at will, requires the cloud parts to collaborate in a peer-to-peer structure.

The actual cloud is a collection of notebooks, desktop computers, servers, or virtual machines running the *Cloud Platform*. Each (virtual) machine is running an *instance* of the Cloud Platform, and each instance is considered as a service component of the cloud. Such service components are also called *nodes*. Multiple platforms and nodes communicate over the Internet forming a cloud system.

IV. Self-* in cloud computing

A cloud system is a distributed system that works without a central coordination manager, usually in a P2P fashion. Every service component (i.e. node) is autonomous and autonomic; each node can join and leave the cloud at will. To assure its adaptivity, the cloud must address some issues [6]:

- **Fail-safe operation:** It needs to continue working also if one or several nodes fail.
- **Load balancing / throughput:** Parallel executions have to take place only if the load exceeds a specified threshold.
- **Security:** All stored and shared data and all applications need to be secure.
- **Interoperability:** Applications must be portable between clouds or multiple infrastructures.
- **Energy conservation:** The system must be able to shut down virtual machines or de-configure virtual networks if they are no longer required.

To create such a cloud, a system designer may use the catalogue of adaptation patterns [7]. This catalogue assists in creating self-adaptive systems based on requirements, context and expected behaviour.

Patterns are classified according to the relation between service components (SC) and autonomic managers (AM). These AM build control loops into the systems, as well as the communication between SC (and also AM) to enact adaptation. The control loops (or feedback loops) follow the MAPE-K approach [8].

To address the challenges of a cloud system, the adaptation pattern based on negotiation described in Section V-A is the most suitable to be used. This pattern enables each SC (i.e. each node) to have an internal AM. It also enables each component to continually monitor itself and its environment and hence to adapt to node failures, to nodes appearing and

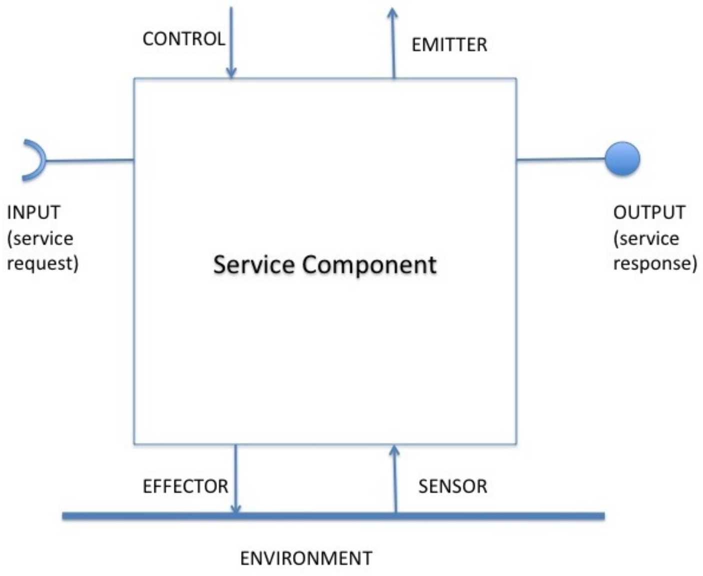

Fig. 1. Interface of a Service Component

disappearing, to the occurrence of overload and other issues. Alternatively, the AM can be external to the SC, as suggested in the *P2P Autonomic Managers Service Components Ensemble Pattern* [7]).

Systems based on these patterns make the cloud self-adaptive and additionally give it the following self-* characteristics:

- **Self-healing:** If a node fails or is shut down, the executing applications must be made available / resumed elsewhere.
- **Self-management:** Each node stays conscious of newly added nodes or disappearing ones and manages itself to make use of newly available resources or disappearing resources.
- **Self-optimisation:** If a node reaches the limits of its capacity, it must be able to transfer some of the load to another node. The same applies to overloaded links in the network.

V. Patterns for self-* properties

The dynamicity of a cloud infrastructure provides a lot of new possibilities, but also increases the system management complexity. Hence applying "traditional" patterns for self-management and self-adaptation is not always sufficient.

In particular, in situations of massive changes in the cloud structure (e.g. number of nodes, number of services requests) there is a need for new self-management strategies. These strategies must take into account the possibility of reconfiguration of the whole system or part of it when needed.

The following two subsections present the most important aspects of such patterns, regarding their role in developing a cloud system. The first one is a pattern for negotiating collaborations between peers under regular conditions, and the second one is a pattern for forming a temporary centralised structure among nodes that are normally equal peers. For more detailed information, please refer to [7].

To understand how components behave in a pattern, it is necessary to understand their interfaces, depicted in Figure 1.

The component receives service requests as INPUT and replies with an OUTPUT. SENSORS provide the component with information from outside the system (e.g. others components and/or the environment). EFFECTORS allow the component to manage the external system (e.g. to act on the environment). EMITTERS provide information about the internal state of the component. Through this interface, the component can also share other information with its peers; for instance sensory information about the environment or other components. Finally, CONTROLLERS make it possible to an external adaptation manager to change and adapt the component's internal state.

A. P2P Negotiation Service Components Ensemble Pattern (P2PN)

The structure of the P2PN pattern is illustrated in Figure 2. The pattern is designed around an implicit autonomic feedback loop for each component. The components must be proactive (impossible for reactive components). Each component is able to monitor and adapt to the environment around it (including the other nodes). The components can communicate directly to propagate their adaptation and coordinate with each other.

Context: This patterns has to be adopted when:

- the components are proactive;
- the components need to directly communicate one with the other to propagate adaptation.

Behaviour: Each component is managed by an internal and implicit AM. The components directly communicate with each other through a P2P communication protocol.

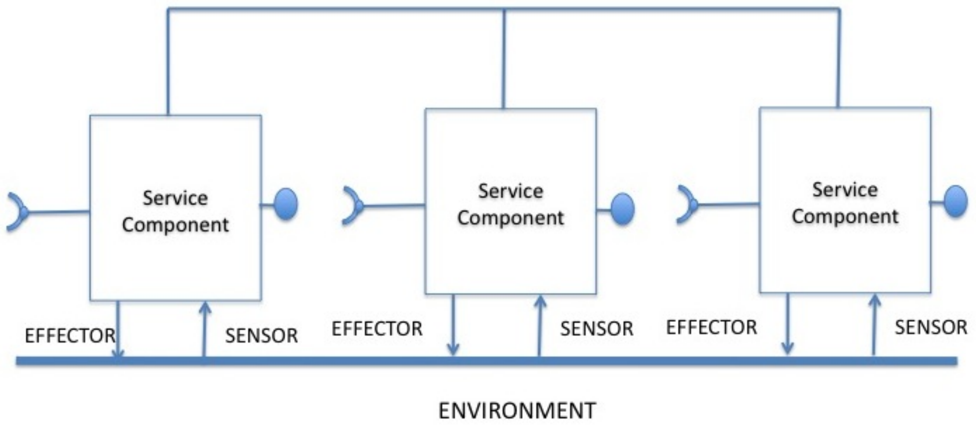

Fig. 2. P2P Negotiation Service Components Ensemble (P2PN)

Consequences: The communication between components makes it possible to share knowledge and mechanisms for adaptation.

Applications: Many case studies about intelligent transportation systems use this pattern, e.g. for a traffic jam monitoring systems [9]. The intelligent transportation system consists of a set of intelligent cameras, which are distributed evenly along a highway. Each camera has a limited viewing range and the cameras are placed to get an optimal coverage of the highway. Each camera has a communication unit to interact with other cameras. The goal of the cameras is to detect and monitor traffic jams on the highway in a decentralised way. [9] develops this case study using the MACODO system as organisation model: each camera is an agent with an internal MACODO manager (implicit AM) that makes the adaptation process.

B. Centralised AM Service Components Ensemble Pattern (CAM)

The structure of the CAM pattern is illustrated in Figure 3. The pattern is suitable when there is a limited number of simple components and they need an AM that enacts an external feedback loop to manage adaptation. In this case, a centralised feedback loop is more suitable because a single AM has a global vision on the system. Through direct communication, the components share knowledge and the same adaptation logic, so they are managed by the same AM.

Context: This patterns has to be adopted when:

- the components are simple and an AM is necessary to manage adaptation;
- needs direct communication between components;
- a centralised feedback loop is more suitable because a single AM has a global vision on the system;
- there are few components composing the ensemble.

Behaviour: This pattern is designed around an unique feedback loop. All the components are managed by a unique AM that controls their behaviour and, sharing knowledge about all the components, is able to propagate adaptation.

Consequences: An unique AM is more efficient to manage adaptation over the entire system, but it can become a single point of failure.

Applications: The case study described in [10], [11] presents an industrial assembly system to explain the MetaSelf Framework. The assembly system is an industrial installation that receives parts and joins them in a coherent way to form a final product. It consists of a set of equipment items called modules (SCs) which are each managed by an AM. The systems' modules spontaneously and dynamically select each other and their position in the assembly system layout, using direct communication. Using their AMs they also dynamically derive the micro-instructions for the robot movements. The result of this self-organising process is a new or reconfigured assembly system that will assemble the ordered product. For other application examples please refer to [7].

VI. An implementation scenario

The above patterns could be implemented in a cloud computing system. Under usual circumstances, the system would follow the P2PN pattern which uses P2P interactions to propagate adaptation. For example, in an initial simulation scenario, there are 50 nodes working together, creating a private cloud for a company. For the sake of simulation simplicity, we suppose that the load of each application shared through the nodes is the same. The number of users who can access this private cloud is around 120 users (that is the total of the employers of the company).

When experiencing disruptions and disturbances, an increasing number of nodes may go down, leading to a partial blackout of the cloud, as show in Figure 4. The horizontal axis represents time steps whereas the vertical axis shows the number of available nodes.

In Figure 5, the horizontal axis again represents time steps whereas the vertical axis shows the number of users requesting services from the cloud (applications or other resources) in different simulations.

Concerning the development of the user requests in time, we consider four different profiles or configurations A, B, C and D. In configuration A, the requests are numerous in the

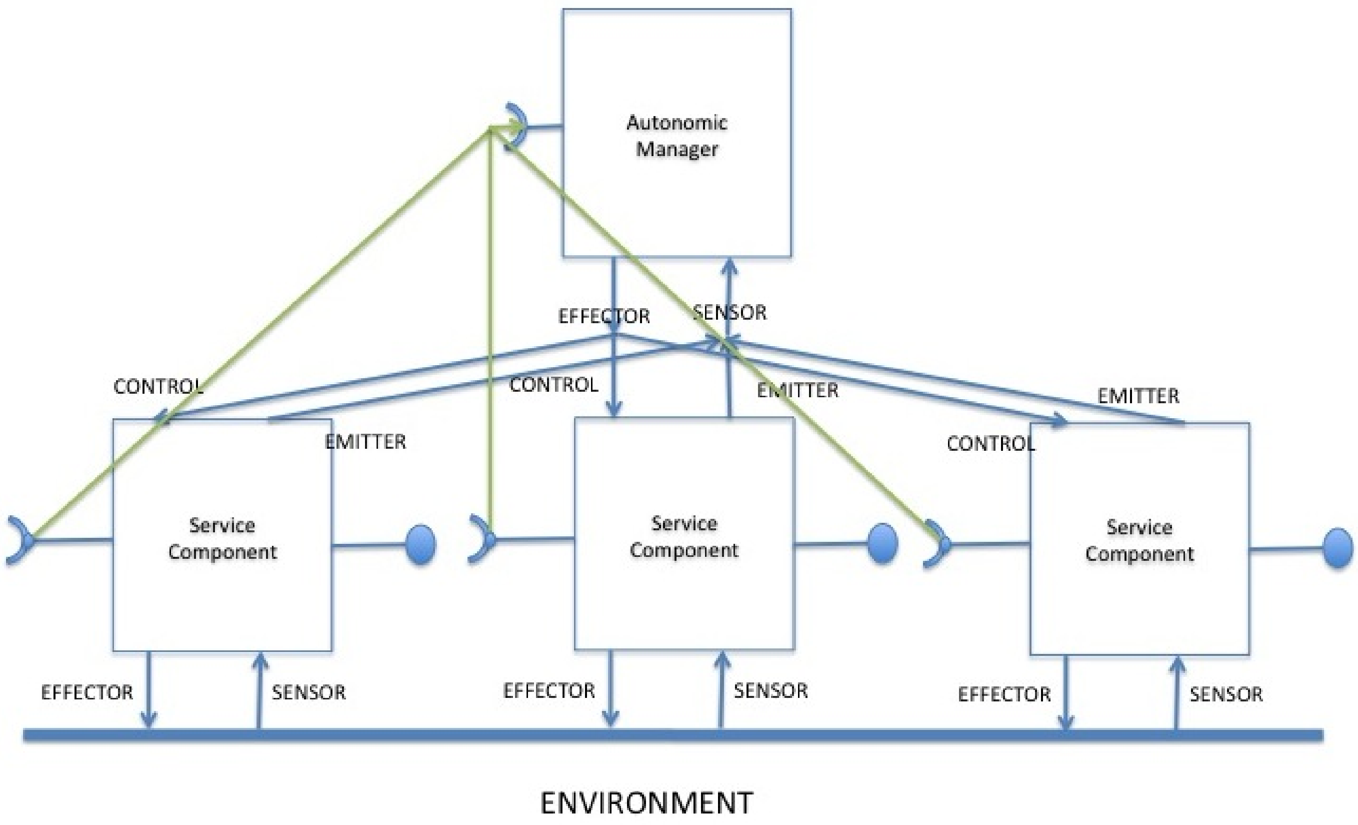

Fig. 3. Centralised AM Service Components Ensemble (CAM)

beginning but quickly decrease in time. In configuration B, the number of requests remains constant in time. In configuration C, the requests quickly increase to reach a maximal number. Finally, in configuration D, the number of requests peak around the time when the partial blackout occurs. This is a worst case scenario.

If the number of users requesting services in the cloud follows configurations A or B, the system performances remain at the desired (initial) standard (e.g. all the requests can be satisfied within a short time). This is because the service requests are of a limited number in comparison with the available number of nodes, even during the temporary server blackout. However, if the number of user requests follow configurations C or D, the average response time for resource acquisition rapidly decreases because the number of user requests increases while the number of available nodes decreases. This is not reasonable for a cloud computing system and indicates the need for an alternative cloud architecture / pattern.

A solution is that, upon passing a threshold (e.g. a specific response time while the number of nodes is less than 15 and the amount of users is superior than 40), the system self-management would make the cloud switch to the CAM pattern. This requires electing an AM for all nodes, leading to a temporary centralised structure. Nodes which are doing well will negotiate with each other to determine a temporary super node (i.e., an AM) that will manage the others during the crisis. However, the permanence of a centralised structure would introduce a single point of failure to the system, so it cannot remain any longer than strictly required.

Once the cloud is back to a state with a sufficient number of live nodes, self-management will get the system back into normal mode, based on the P2PN pattern.

In the initial cloud system, developed using the P2PN pattern, each node can communicate directly with the others and negotiate its role in the cloud (working, not working, which applications to share, which data to store and so on). If the work load is high, they negotiate for all of them to be working at operating speed, whereas if the work load decreases, they may negotiate for some of them to switch off or to decrease the number of services they provide.

In case of failures in the system, for example due to a *brownout* of some computer servers, the system is no longer able to manage all the service request within a reasonable time. In this case, the cloud system has to self-manage by changing its structure, adopting the CAM pattern. This pattern becomes suitable because of the context change that now is more similar to the one described in subsection V-B. Still using negotiation, the remaining nodes elect an autonomic manager between them. This AM, via direct communication with all the other nodes, has the authority to divide the work load between the remaining live nodes. This permits the new system to manage all user requests within a lower response time.

Usually, self-adaptation in the P2PN pattern is sufficient for a cloud system to function. However, if emergent behavior

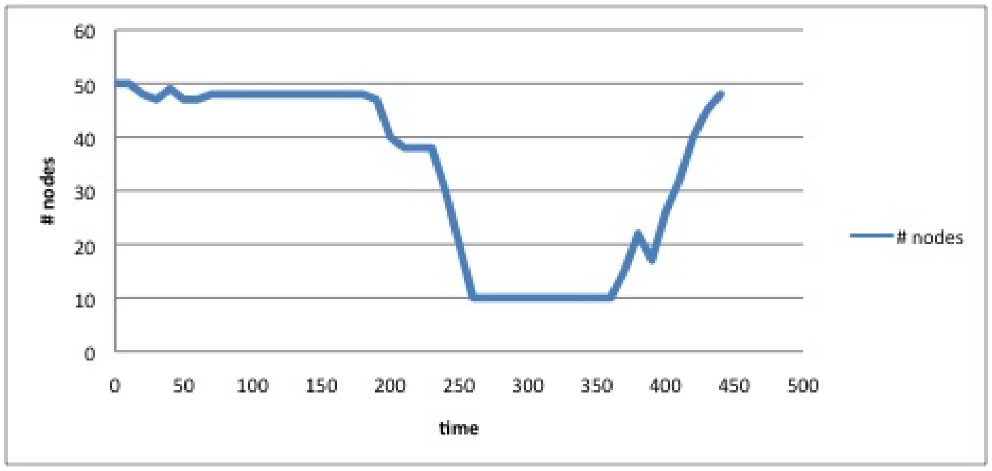

Fig. 4. Number of nodes available in the cloud

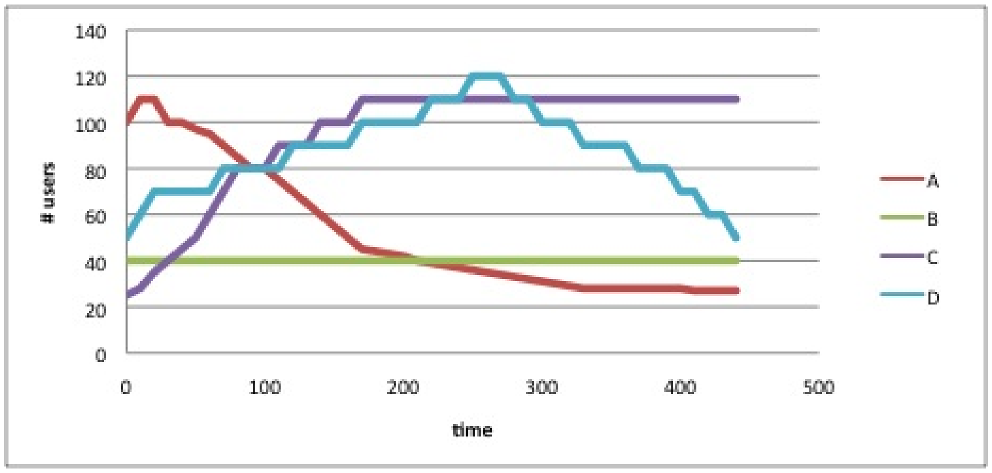

Fig. 5. Different simulations of the number of users requesting services in the cloud

leads to rapidly decreasing system performance, the cloud needs a self-management mechanism to switch form one pattern to another. These mechanisms are also called *self-expression mechanisms* [12], [13]. Self-expression is a meta-level form of structural adaptation. This makes it possible to undergo a dynamic change of adaptation pattern to increase system performance. In other words, it allows the system to dynamically self-reconfigure.

In a real system, a role based approach [14] may be adopted. It allows every component to perform different roles depending on the circumstances. In the role based approach, all components are able to become manager (and vice versa); this means that the system does not need to add another (external) component to be a manager, but it is able to reuse existing resources (e.g. nodes). The role approach also enables the separation of concerns dividing the business logic and the collaboration logic. Roles also permit to encapsulate the communication mechanisms for managing or coordinating other components. Consequently, in the role based approach, every component needs to include all possible roles and to be able to manage all of them in its internal structure.

It may be a drawback of this approach that an important amount of code is stored in the system (i.e. the nodes) but may remain unused if the system does not need to reconfigure. The advantages of such an implementation include the possibility – intrinsic to a cloud – to store all the different roles in a node, and to rapidly communicate a newly adopted role through the cloud network. The challenges lie in the implementation of the different roles, allowing different nodes to use and share the same role.

VII. Conclusion

Cloud computing is a very powerful emerging technology. Among the challenges still to be tackled is the question of how to deal with partial blackouts while assuring the best possible cloud performance regarding service request response time. This paper presents two adaptation patterns and suggests when and how to switch between them, depending on system conditions. Under regular circumstances, the system will adopt a peer-to-peer structure with inherent adaptivity. When under stress – many failing nodes and many service requests – the system will switch to a temporary centralised structure. To achieve this, the live nodes will negotiate with each other and collectively define a temporary manager. A precondition for this approach to be possible is that each node is able to perform different roles: the one of independent peer, the one of temporary manager, and the one of temporary follower.

Initial simulations have shown that this is a promising approach, but further research into a real implementation is necessary to reach a final conclusion and to evaluate the use of different adaptation patterns for self-management in different situations.

Acknowledgement

Work supported by the ASCENS project EU FP7-FET, Contract No. 257414.

References

[1] P. Horn, “Autonomic computing: Ibm’s perspective on the state of information technology,” 2001.

[2] M. Parashar and S. Hariri, “Autonomic computing: An overview,” in *Unconventional Programming Paradigms*. Springer, 2005, pp. 257–269.

[3] Y. Brun, “Improving impact of self-adaptation and self-management research through evaluation methodology,” in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2010, pp. 1–9.

[4] J. Kramer and J. Magee, “Self-managed systems: an architectural challenge,” in *Future of Software Engineering, 2007. FOSE’07*. IEEE, 2007, pp. 259–268.

[5] K. Herrmann, G. Muhl, and K. Geihs, “Self management: the solution to complexity or just another problem?” *Distributed Systems Online, IEEE*, vol. 6, no. 1, 2005.

[6] T. Patikirikorala, A. Colman, J. Han, and L. Wang, “A multi-model framework to implement self-managing control systems for qos management,” in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2011, pp. 218–227.

[7] L. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, “A break in the clouds: towards a cloud definition,” *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2008.

[8] Q. Zhang, L. Cheng, and r. Boutaba, “Cloud computing: state-of-the-art and research challenges,” *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[9] I. Brandic, “Towards self-manageable cloud services,” in *Computer Software and Applications Conference, 2009. COMPSAC’09. 33rd Annual IEEE International*, vol. 2. IEEE, 2009, pp. 128–133.

[10] J. Kephart and D. Chess, “The vision of autonomic computing,” *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[11] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, and K. Goeschka, “On patterns for decentralized control in self-adaptive systems,” in *Software Engineering for Self-Adaptive Systems II*, ser. LNCS, R. Lemos, H. Giese, H. Mueller, and M. Shaw, Eds. Springer Berlin Heidelberg, 2013, vol. 7475, pp. 76–107.

[12] M. Puviani, G. Cabri, and L. Leonardi, “Is self-expression useful? evaluation by a case study,” in *Proceedings of the 22nd IEEE WETICE conference - 3rd Track on Adaptive and Reconfigurable Service-oriented and conmponent-based Applications and Architectures (AROSA))*. IEEE, June 2013.

[13] P. Mell and T. Grance, “The nist definition of cloud computing (draft),” *NIST special publication*, vol. 800, p. 145, 2011.

[14] P. Mayer and C. Kroiss, “Jv: Specification: The science cloud case study – overview and scenarios,” Technical Report of the ASCENS project, Tech. Rep., 2012.

[15] B. Rimal and E. Choi, “A service-oriented taxonomical spectrum, cloudy challenges and opportunities of cloud computing,” *International Journal of Communication Systems*, vol. 25, no. 6, pp. 796–819, 2012.

[16] S. Bhardwaj, L. Jain, and S. Jain, “Cloud computing: A study of infrastructure as a service (iaas),” *International Journal of engineering and information Technology*, vol. 2, no. 1, pp. 60–63, 2010.

[17] A. Beloglazov, J. Abawajy, and R. Buyya, “Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing,” *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[18] B. Rimal, E. Choi, and I. Lumb, “A taxonomy and survey of cloud computing systems,” in *INC, IMS and IDC, 2009. NCM’09. Fifth International Joint Conference on*. Ieee, 2009, pp. 44–51.

[19] M. Puviani, “Tr 4.2: Catalogue of architectural adaptation patterns,” ASCENS Project, Tech. Rep., 2012. [Online]. Available: http://mars.ing.unimo.it/wiki/papers/TR42.pdf

[20] Y. Brun, G. Di Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, “Engineering self-adaptive systems through feedback loops,” in *Software Engineering for Self-Adaptive Systems*, ser. Lecture Notes in Computer Science, B. Cheng, R. d. Lemos, P. Inverardi, and J. Magee, Eds. Springer, 2009, vol. 5525, pp. 48–70.

[21] R. Haesevoets, D. Weyns, T. Holvoet, and W. Joosen, “A formal model for self-adaptive and self-healing organizations,” in *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS’09. ICSE Workshop on*. Vancouver, BC, Canada: IEEE Computer Society, May 2009, pp. 116–125.

[22] G. Di Marzo Serugendo, J. Fitzgerald, A. Romanovsky, and N. Guelfi, “MetaSelf - a framework for designing and controlling self-adaptive and self-organising systems,” BBKCS-08-08, School of Computer Science and Information Systems, Birkbeck College, London, UK, Tech. Rep., 2008.

[23] G. Di Marzo Serugendo, J. Fitzgerald, and A. Romanovsky, “MetaSelf - an architecture and development method for dependable self-* systems,” in *Symp. on Applied Computing (SAC)*, Sion, Switzerland, 2010, pp. 457–461.

[24] F. Zambonelli, N. Bicocchi, G. Cabri, L. Leonardi, and M. Puviani, “On self-adaptation, self-expression, and self-awareness, in autonomic service component ensembles,” in *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2011 Fifth IEEE Conference on*. Ann Arbor, Michigan, USA: IEEE Computer Society, October 2011, pp. 108–113.

[25] G. Cabri, L. Ferrari, and F. Zambonelli, “Role-based Approaches for Engineering Interactions in Large-scale Multi-Agent Systems (invited chapter),” in *Software Engineering for Multi-Agent Systems II”*, ser. LNCS, C. Lucena, A. Garcia, A. Romanovsky, J. Castro, and P. Alencar, Eds. Springer, April 2004, vol. 2940, pp. 243–263.