On Autonomic Platform-as-a-Service: Characterisation and Conceptual Model

Rafael Tolosana-Calasanz[1], José Ángel Bañares[1], and José-Manuel Colom[1]

COSMOS Group - Aragón Institute of Engineering Research (I3A)
Universidad de Zaragoza, Spain
{rafaelt,banares,jm}@unizar.es

Abstract. In this position paper, we envision a Platform-as-a-Service conceptual and architectural solution for large-scale and data intensive applications. Our architectural approach is based on autonomic principles, therefore, its ultimate goal is to reduce human intervention, the cost, and the perceived complexity by enabling the autonomic platform to manage such applications itself in accordance with high-level policies. Such policies allow the platform to (i) interpret the application specifications; (ii) to map the specifications onto the target computing infrastructure, so that the applications are executed and their Quality of Service (QoS), as specified in their SLA, enforced; and, most importantly, (iii) to adapt automatically such previously established mappings when unexpected behaviours violate the expected. Such adaptations may involve modifications in the arrangement of the computational infrastructure, i.e. by re-designing a different communication network topology that dictates how computational resources interact, or even the live-migration to a different computational infrastructure. The ultimate goal of these challenges is to (de)provision computational machines, storage and networking links and their required topologies in order to supply for the application the virtualised infrastructure that better meets the SLAs. Generic architectural blueprints and principles have been provided for designing and implementing an autonomic computing system. We revisit them in order to provide a customised and specific view for PaaS platforms and integrate emerging paradigms such as DevOps for automate deployments, Monitoring as a Service for accurate and large-scale monitoring, or well-known formalisms such as Petri Nets for building performance models.

1 Introduction

In the early 60's, at the dawn of the computer era, the conception of computing being organised as a public utility was already envisioned –just as the telephone system or the electrical power networks. John McCarthy, speaking at the MIT Centennial in 1961 [1], imagined that such computing infrastructure would also be integrated with a different and disruptive business model, thereby "*(...) each*

[1] http://www.technologyreview.com/news/425623/the-cloud-imperative/

subscriber needs to pay only for the capacity he actually uses, but he has access to all programming languages characteristic of a very large system (...) Certain subscribers might offer service to other subscribers (...)". Hence, McCarthy anticipated that the computing utility would become "*the basis of a new important industry*". Half a century later, after significant advances in computing as well as with the required maturity and developments in technology, such a view is becoming a reality with Cloud computing [3, 9].

Nowadays, there is a number of public Cloud providers, such as Amazon [2] or Softlayer [3], that allow users to rent virtualised resources on demand, namely computers, storage and network links, and to pay per their usage in accordance with a number of Service Level Agreements (SLAs) and enforcement guarantees. Moreover, Cloud computing can also offer additional abstraction levels, by providing not only a virtualised infrastructure (Infrastructure-as-a-Service, IaaS), but also applications (Software-as-a-Service, SaaS) or even actual software development and deployment platforms (Platform-as-a-Service, PaaS). PaaS abstracts the underlying computing infrastructure and provides the user with a language interface so that both the program logic and the SLAs can be expressed. Essentially, such specifications need to be infrastructure-agnostic, that is, without referring to specific details of a particular infrastructure, so that they can subsequently be interpreted, mapped and deployed to a computing infrastructure. Nevertheless, open challenges are emerging with the advent of big data applications, as these applications often have to deal with significants amounts of data that need to be processed *continuously* in (near) real-time, and their data generation rates are often bursty and subject to unpredictability. Such requirements lead to the need for large number of computational distributed resources and the complexity of the management inherently increases, provoked by the appearance of failures, unexpected performance degradation and, in general, unknown behaviours. In such a context, there is some approaches [2, 7] that have studied and considered the introduction a number of policies and mechanisms into PaaS in order to offer some Quality of Service (QoS) guarantees.

In this position paper, we envision a PaaS conceptual and architectural solution for large-scale and data intensive applications. Our architectural approach is based on autonomic principles, therefore, its ultimate goal is to reduce human intervention, the cost, and the perceived complexity by enabling the autonomic platform to manage such applications itself in accordance with high-level policies. Such policies allow the platform to (i) interpret the application specifications; (ii) to map the specifications onto the target computing infrastructure, so that the applications are executed and their Quality of Service (QoS), as specified in their SLA, enforced; and (iii) to adapt automatically such previously established mappings when unexpected behaviours violate the expected. Such adaptations may involve modifications in the arrangement of the computational infrastructure, i.e. by re-designing a different communication network topology that dictates how computational resources interact, or even the live-migration to

[2] http://aws.amazon.com/
[3] http://www.softlayer.com/

a different computational infrastructure. The ultimate goal of these challenges is to (de)provision computational machines, storage and networking links and their required topologies in order to supply for the application the virtualised infrastructure that better meets the SLAs. Generic architectural blueprints and principles have been provided for designing and implementing an autonomic computing system [12, 4, 8, 5, 6]. We revisit them in order to provide a customised and specific view for PaaS platforms and integrate emerging paradigms such as DevOps for automate deployments, Monitoring as a Service for accurate and large-scale monitoring, or well-known formalisms such as Petri Nets for building performance models. The rest of this paper is organised as follows. Section 2 describes the conceptual architectural model for Autonomic Platform-as-a-Service. Section 3 provides a brief related work discussion on PaaS. Conclusions are provided in Section 4.

2 Characterisation & Conceptual Model of an Autonomic PaaS

There is a number of well-established and generic architectural blueprints and principles for the design of autonomic systems [12, 4, 8, 5, 6]. When designing an autonomic system, a number of challenges emerge: (i) *Autonomic System Specification* –it relates to the appropriate specification and formulation of the autonomic elements, so that the description of what the system needs to do is captured and understood. The specification involves functional and non-functional requirements as well as autonomic behaviours, expressed in terms of high level policies, content and context driven definition, execution and management. (ii) *Autonomic System Design* –which includes the definition of appropriate abstractions, methods and tools for specifying, understanding, controlling, and implementing autonomic behaviours; the provision of models for negotiation among architectural autonomic elements and for detecting, predicting, and correcting potential problems. (iii) *Integration and Consistency of Autonomic Behaviours* –it is related to the autonomic elements constituent of the system, and on how they behave in isolation and in co-operation. It is essential to support the specification of individual and global autonomic behaviours, so that they can be implemented and controlled in a robust and predictable manner. (iv) *Middleware challenges* – the system needs to exploit middleware services for realising autonomic behaviours, including discovery, messaging, security, privacy, trust, etc. Therefore, it will be fundamental to properly identify which functional and non-functional services the middleware layer needs to provide.

In this section, we are characterising and customising such elements for an Autonomic PaaS platform that executes large-scale data-intensive applications. Figure 1 depicts an architectural blueprint for our proposal.

2.1 Application Specification for the Autonomic PaaS

Users need a specification language for expressing their functional and non-functional requirements. Hence, as an input, the PaaS platform receives users'

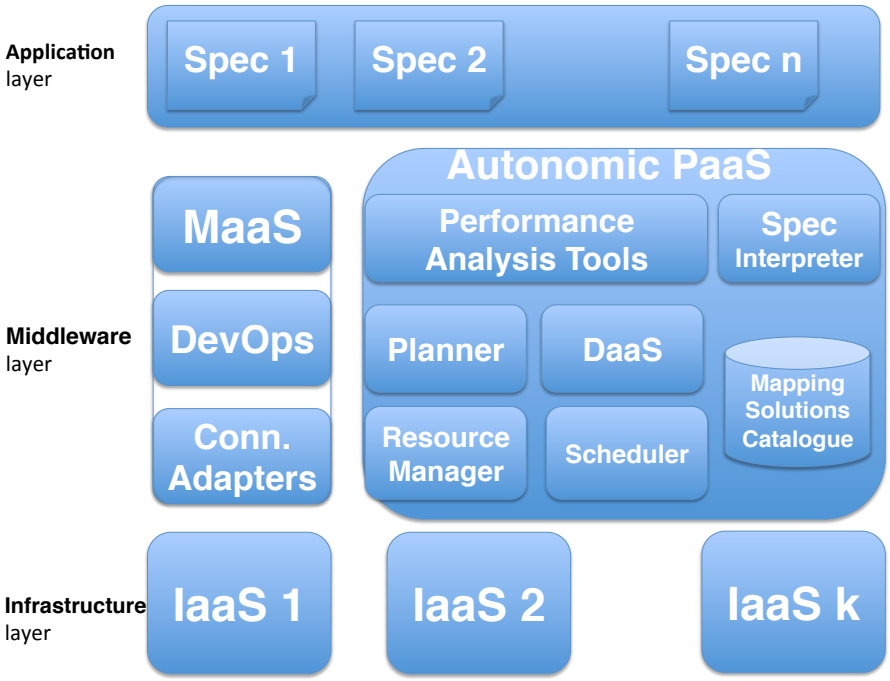

Fig. 1. Conceptual Architecture for an Autonomic Platform as a Service

specifications and maps them accordingly to the computing infrastructures. The characteristics of such specifications are, on one hand, application dependent. In the case of data-intensive applications, they may involve, from a functional point of view, compositions of computational processes and data transmission processes. The semantics in how these compositions are accomplished will determine the particular model of computation. Therefore, there is a need for supporting different of models of computation ranging from simple parallelism to data streaming [13]. From a non-functional point of view, users can be interested in considering a number of attributes such as throughput / processing time, economic cost (given by the pay-per-usage model of the Cloud), or resilience. Additionally, the desired autonomic behaviour while executing the application can be (partially inferred from the non-functional requirements), or alternatively, high-level policies, which determine the behaviour of the PaaS when executing the application, can also be supported On the other hand, the design of the specification also depends on inherent characteristics of PaaS platforms and their ultimate goal, that is essentially, such specifications need to be infrastructure-agnostic, that is, without referring to specific details of a particular infrastructure, so that they can subsequently be interpreted, mapped and deployed to a computing infrastructure. There is a number of open standards proposed for autonomic computing that can be prescribed for the specification such as OASIS Web Services Distributed Management (WSDM) [4], for the interfaces and their semantics, or Common Information Model – Simplified Policy Language [5] (CIM-SPL) for the specification of policies. In addition to specification languages, and also of key importance, is the specification interpreter of the autonomic PaaS platform, which appears on the right of Figure 1.

2.2 Autonomic PaaS Design

From the functional and non-functional requirements captured from the user, we envision that a performance engineer needs to conduct an analysis and to generate possible mapping solutions between the application and a computing infrastructure. At such a step, different configurations, i.e. computational machines linked by means of different network topologies, distinct number of computational resources, etc. may be considered. For all of them, the engineer must analyse and identify the boundaries of QoS parameters and how they affect each other. We propose that such a process is accomplished in a two main steps. First, by realising the analysis with generic and abstract resources, studying concurrency from the application and the consumption of resources. Second, by refining and customising the models with the constraints of specific infrastructures. As a result, a *catalogue* of potential application / infrastructure mapping solutions is identified, each with different QoS attribute values and guarantees. Although there is a number of formalisms and generic approaches in the literature, on performance engineering, we developed further that particular idea. We proposed a

[4] https://www.oasis-open.org/committees/wsdm/

[5] http://www.dmtf.org/documents/policy/cim-simplified-policy-language-cim-spl-100

Petri net-based, model-driven and stepwise refinement methodology for streaming applications over Clouds [14]. In such a case, the complexity of streaming applications arises from the confluence of concurrency, transmission of data and the use of distributed resources. The central role in this methodology is assigned to a set of Petri Net models describing the behaviour of the system including timing and cost information. The goal is to use these models in an intensive way before the deployment of the application in order to understand its behaviour and to obtain properties of the different solutions adopted. In some cases, the observations may induce or recommend changes into the application with the purpose of modifying parts of the design and assure agreed specifications. The consideration of Petri Nets is based on the natural descriptive power for the concurrency, but also for the availability of analytic tools coming from the domain of Mathematical Programming and Graph Theory. These tools are based on a structural analysis that support the reasoning on properties without the construction of the state space –which for such a class of systems is prohibitive. a methodology for the construction of this kind of applications is proposed based on the intensive use of formal models. Petri Nets are the formalism considered here for capturing the active entities of the system (processes), the flow of data between the processes and the shared resources for which they are competing. For the construction of a model aimed at studying different aspects of the system and for decision-taking design, an abstraction process of the system at different levels of detail is needed. This leads to several system models representing facets from the functional level to the operational level. Petri Net models are used to obtain qualitative information of the streaming application, but their enrichment with time and cost information provides with analysis on performance and economic behaviours under different scenarios. In consequence, a set of performance analysis tools need to be incorporated in the autonomic PaaS architecture.

Moreover, such a catalogue of performance models constitutes an essential element of the system *Knowledge*, and can be subsequently exploited by the autonomic elements of the architecture for their self-management actions. Whenever a violation of the QoS occurs, the platform must decide an action (or combination of actions) that corrects such a deviation. The catalogue is essential for this process, as well as obtaining real-time information from the infrastructure or even from other infrastructures that can be potentially used alternatively. Then, once an action is selected, it has to be planned and accomplished (executed). It may involve a complete re-configuration of the infrastructure –i.e. a complete different number of machines with different network topologies and storage; or it may require the shift to an alternative computing infrastructure. This raises a number of challenges such as the autonomic deployment of the application (Deployment as a Service, DaaS in Figure 1) in a new different infrastructure without interruptions in the execution and enforcing the expected QoS.

2.3 Middleware Layer Requirements

Perhaps the monitoring service from the computing infrastructures is the most important one from the middleware layer, but a deployment automation mid-

dleware service is also mandatory, so that the required computing infrastructure can be virtualised on-demand. Additionally, connection wrappers and adapters for interacting with different IaaS providers are required. On this regard, the possibility of an Inter-Cloud organisation of the infrastructure may also be considered [1].

Monitoring as a Service Big data analytics and applications often involve a large number of distributed computational resources for their execution and, therefore, monitoring states of resources (i.e. performance, running time, cost, etc.) often requires collecting values of various attributes from a vast amount of nodes. Although there can be recognised a core set of attributes common to most applications, in general terms, the specific requirements of each application determines the monitoring needs, which can even vary in time, i.e. as a result of changing conditions in the application requirements or in the infrastructure. Monitoring as a Service [10, 11] should support not only the conventional state monitoring capabilities, such as instantaneous violation detection, periodical state monitoring, and single tenant monitoring, but also performance-enhanced functionalities that can optimise on monitoring cost, scalability, and the effectiveness of monitoring service consolidation and isolation.

Deployment as a Service Deployment of a mapping solution in the Cloud can be a tedious task for a human being and it is also expected to be fast feedback to new occurring eventualities. Thus, it is a critical competitive advantage to be able to respond quickly. For these reasons, tooling is required to implement end-to-end automation of deployment processes and DevOps [15–17] as an emerging paradigm, which integrates software developers with operational personnel, can be a solution to consider. Automation is the key to efficient collaboration and tight integration between development and operations. The DevOps community is constantly pushing new approaches, tools, and open-source artifacts to implement such automated processes.

2.4 Integration and Consistency of Autonomic Behaviours

In addition to defining the global behaviour of the PaaS platform when executing an application, it is important to identify the autonomic elements forming part of the PaaS architecture as well as third-party autonomic elements, such as autonomic middleware components. Individual policies should dictate their behaviour, but their co-operation and their interactions should also be regulated. Hence, it is essential to support the specification of individual and global autonomic behaviours, so that they can be consistently integrated and the PaaS platform can be controlled in a predictable manner. A number of architectural components in the PaaS platform are likely to be autonomic, such as the planner, the scheduler or the resource manager. On the other hand, some of these internal autonomic components need to interact with third-party autonomic components as well. For instance, due to the complexity of monitoring, the MaaS middleware component may also be designed with autonomic principles.

3 Related Work

PaaS aims at application developing and subsequent deployment. Hence, it typically provides a complete set of tools and programming models and interfaces for processing the logic and automatically deploying and executing them into de underlying infrastructures. According to [2], PaaS providers include Google AppEngine, Microsoft Azure, Bungee Labs, Coghead, Etelos, Google, LongJump, Rollbase, or Salesforce.com, etc. Nevertheless, there is limited support for QoS guarantees in these solutions. The requirements for supporting QoS guarantees in PaaS architectures were analysed in [2]. In this paper, we have revisited these concepts, integrating new emerging paradigms and technologies such as DevOps or MaaS, an paradigms like autonomic computing with well-known formalisms such as Petri nets.

In a previous work [14], a first approach to this architectural model was proposed, but without the autonomic requirement as a final result. There, the goal was to manage the complexity in the construction of applications in the Cloud, but without considering dynamic aspects. Thus, the so called *functional* level there corresponds in here to the *specification* level, and the final result in [14] was a model representing the behaviour of the application without considering a specific infrastructure or third-party applications needed. The operational level, on the other hand, considers a generic infrastructure over which the behaviour of the application is studied. The operational level is described by another model that in combination with the functional level model gives us an overall model for qualitative and quantitative analysis of QoS attributes. Therefore, the approach in [14] represents the required previous step for accomplishing the autonomic PaaS architectural model presented in here.

4 Conclusions

With the advent of large-scale big data applications, which for their execution require complex management of significant number of distributed computational resources. In this position paper, we envision a PaaS conceptual and architectural model. Our architectural approach is based on autonomic principles, aiming at reducing human intervention, the cost, and the perceived complexity by enabling the autonomic platform to self-manage such applications. From the functional and non-functional requirements captured from the user, we envision that a performance engineer needs to conduct a Quality of Service (QoS) analysis (i.e. involving a combination of QoS attributes that includes performance, economic cost or resilience) to explore and assess all the possible mapping solutions of the application to abstract and generic resources. The ultimate goal is to identify the boundaries of QoS parameters and how they affect each other. Subsequent incorporation of the constraints from actual computational infrastructures will refine such models and update the derived QoS boundaries to estimate realistic behaviour of the infrastructure. As a result, a *catalogue* of potential application to infrastructure mapping solutions is constructed, each with different QoS

attribute values and guarantees. The autonomic PaaS platform with this knowledge information in combination with accurate monitored real time information of the computational infrastructures can perform autonomic deployments and adaptations of previous ones, so that applications can executed in a flexible and adaptive manner. Our model integrates emerging paradigms such as DevOps for automate deployments, Monitoring as a Service for accurate and large-scale monitoring, or well-known formalisms such as Petri Nets for building performance models.

5 Acknowledgements

This work was supported by the Spanish Ministry of Economy under the program "Programa de I+D+i Estatal de Investigación, Desarrollo e innovación Orientada a los Retos de la Sociedad", project id TIN2013-40809-R

References

1. Assis, M., Bittencourt, L.F., Tolosana-Calasanz, R.: Cloud federation: Characterisation and conceptual model. In: 3rd International Workshop on Clouds and (eScience) Applications Management (CloudAM 2014) (2014)
2. Boniface, M., Nasser, B., Papay, J., Phillips, S., Servin, A., Yang, X., Zlatev, Z., Gogouvitis, S., Katsaros, G., Konstanteli, K., Kousiouris, G., Menychtas, A., Kyriazis, D.: Platform-as-a-service architecture for real-time quality of service management in clouds. In: Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on. pp. 155–160 (May 2010)
3. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Gener. Comput. Syst. 25(6), 599–616 (2009)
4. Corporation, I.: An architectural blueprint for autonomic computing. Tech. rep., IBM (Jun 2005)
5. Hanson, J.E., Whalley, I., Chess, D.M., Kephart, J.O.: An architectural approach to autonomic computing. In: Proceedings of the First International Conference on Autonomic Computing. pp. 2–9. ICAC '04, IEEE Computer Society, Washington, DC, USA (2004)
6. Huebscher, M.C., McCann, J.A.: A survey of autonomic computing –degrees, models, and applications. ACM Comput. Surv. 40(3), 7:1–7:28 (Aug 2008)
7. Keller, E., Rexford, J.: The "platform as a service" model for networking. In: Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking. pp. 4–4. INM/WREN'10, USENIX Association, Berkeley, CA, USA (2010)
8. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer 36(1), 41–50 (2003)
9. Marinescu, D.C.: Cloud Computing: Theory and Practice. Morgan Kaufmann (2013)
10. Meng, S., Kashyap, S.R., Venkatramani, C., Liu, L.: Resource-aware application state monitoring. IEEE Transactions on Parallel and Distributed Systems 23(12), 2315–2329 (2012)

11. Meng, S., Liu, L.: Enhanced monitoring-as-a-service for effective cloud management. Computers, IEEE Transactions on 62(9), 1705–1720 (Sept 2013)
12. Parashar, M., Hariri, S.: Autonomic computing: An overview. In: Banâtre, J.P., Fradet, P., Giavitto, J.L., Michel, O. (eds.) Unconventional Programming Paradigms, Lecture Notes in Computer Science, vol. 3566, pp. 257–269. Springer Berlin Heidelberg (2005)
13. Pautasso, C., Alonso, G.: Parallel computing patterns for Grid workflows. In: Proceedings of the HPDC2006 Workshop on Workflows in Support of Large-Scale Science (WORKS06) June 19-23, Paris, France (2006)
14. Tolosana-Calasanz, R., Bañares, J.Á., Colom, J.M.: Towards petri net-based economical analysis for streaming applications executed over cloud infrastructures. In: Economics of Grids, Clouds, Systems, and Services - 11th International Conference, GECON 2014, Cardiff, UK, September 16-18, 2014. pp. 189–205 (2014)
15. Wettinger, J., Gorlach, K., Leymann, F.: Deployment aggregates - a generic deployment automation approach for applications operated in the cloud. In: Enterprise Distributed Object Computing Conference Workshops and Demonstrations (EDOCW), 2014 IEEE 18th International. pp. 173–180 (Sept 2014)
16. Wettinger, J., Breitenbücher, U., Leymann, F.: Devopslang – bridging the gap between development and operations. In: Villari, M., Zimmermann, W., Lau, K.K. (eds.) Service-Oriented and Cloud Computing, Lecture Notes in Computer Science, vol. 8745, pp. 108–122. Springer Berlin Heidelberg (2014)
17. Wettinger, J., Breitenbücher, U., Leymann, F.: Standards-based devops automation and integration using tosca. In: Proceedings of the 7th International Conference on Utility and Cloud Computing (UCC 2014). pp. 59–68. IEEE Computer Society (2014)