Self-Optimizing Architecture for Ensuring Quality Attributes in the Cloud

Vivek Nallur, Rami Bahsoon, Xin Yao
School of Computer Science, University of Birmingham
Birmingham B15 2TT, UK
{*V.Nallur, R.Bahsoon, X.Yao*}*@cs.bham.ac.uk*

Abstract—**We describe various challenges in ensuring Quality Attributes (QA) of applications hosted in the cloud and hence the perceived quality of service of the cloud as a whole. We advocate a self-management/optimization architecture-driven approach to ensure that Quality Attributes are met. We discuss the limitations of current approaches to self-managing architecture. We propose a novel approach, which exploits the El Farol problem as a modelling mechanism for QAs in architectures of applications in the cloud. The approach uses Service Level Agreements (SLA) and Utility Theory to direct the self-optimization. We conclude by looking at directions for further work.**

Keywords-**cloud computing; self-organizing; web-services; architecture**

I. Introduction

Advances in networking, storage and processing technologies have given us software that is mind-boggling in size and complexity of structure. The combination of development, configuration and management complexity has resulted in software fragility and increased interest in autonomic software [1]. In the wake of organizational change, economic downturn and a demand for tightening the belt on IT costs, there is a trend toward moving large applications to the cloud. Organizations such as IBM [2] and Gartner [3] advocate cloud computing as a potential cost-saver as well as provider of higher service quality. Clouds, as made available by the major players like Amazon, Google and 3Tera, use the Software-As-A-Service or Infrastructure-As-A-Service model. This means that payment for the services of the cloud are made on the basis of cpu-hours used as well as storage used [4], which is more economical than purchasing processors and storage devices. However cloud providers make no guarantees about the Quality of Service attributes being provided by them. We envisage that Quality Attributes like performance, security, availability, reliability will be important parameters that organizations would want to monitor and optimize. It is in this context, that self-optimization of applications distributed in the cloud, becomes important.

Usually, applications in the cloud are of the following types:

1) Web-applications that cater to diverse users across the internet and demand fast response times.

2) Enterprise applications that cater to different business units across the world and require large amounts of secure, reliable data transfer and high availability (99.999%).

3) Scientific applications that need raw cpu, or enterprises that perform batch processing

The third type of application has the natural advantage of '*cost associativity*' offered by the cloud [5]. The first two types of applications represent a slice of systems that we depend on in our daily life; these need to be dependable and long-lived. By dependable, we refer to qualities like reliability, availability, and other non-functional requirements which need to be provided, maintained, evolved and monitored at runtime. These requirements often evolve in response to changes in users' needs, services, and the runtime environment. Consequently, this will necessitate dynamic change, evolution, and self-optimisation of applications hosted in the cloud with respect to the said requirements. Needless to say, evolution and self-optimisation needs to be constrained by cost, available resources, and any other operational constraints specific to the cloud.

The applicability of current research on self-managing architectures to the cloud is doubtful. This is because these approaches assume that the system is a closed-loop and self-management is done via a centralized component. As we will discuss in the section on Related Work, the environment of the cloud is sufficiently different (no control over physical topology, for instance), for centralized control and closed-loop type feedback to be inapplicable. The contribution of this paper is as follows: we highlight existing work on self-managing architectures and discusses their inapplicability to the case of the cloud. We turn to a motivating example to discuss the need for self-optimization in the cloud. We argue that Service Level Agreements (SLA) can assist the problem of self-optimization in the cloud. We introduce the idea of an application changing/optimizing its architecture by means of modifying its components and connectors, through the use of composable web-services, that automatically negotiate their cost versus feature offering via SLAs. We propose to use the emergent effect of simple self-optimizations at lower levels (feature level) to achieve a higher level optimization (application-wide) in the cloud. We model the simple self-optimizations as solutions to the *El Farol Bar problem*.

The rest of this paper is organised as follows: In section

2, we present a motivating example to elucidate some of the Quality Attributes that need optimization. In section 3 we look at related work on self-managing architecture and discuss how it does not solve the problem mentioned in section 2. In section 4, we propose the idea of self-optimization in detail. And in the final section, we discuss future work.

II. MOTIVATING EXAMPLE

We motivate the need for research in self-managing architecture of cloud-based applications by the following example:

Consider a new social networking site (myChaseBook), much like Facebook, but in addition to the Web it also incorporates new media like cellphones and online radio. myChaseBook offers the following innovative services:

- Track friends geographically via cellphone, overlaid on a city map.
- Tagging of streaming radio.
- Recommend radio snippets to friends who can then choose to listen via cellphone and buy them from online stores.
- Cellphone-based access to photos, music tracks, friends and their recommendations based on the location of the user.

Suppose that myChaseBook becomes wildly popular and the creators of myChaseBook decide to move their site onto the Cloud. Their principal requirements are:

- **Scalability:** Cloud providers charge for their service on the basis of cpu-hours used, storage and bandwidth usage. Since myChaseBook is a startup and does not have too much cash in hand, in addition to scaling up, the application should also scale down in times of low demand.
- **Availability:** Users are turned off if the service is not available due to frequent over-loading. myChaseBook should be resilient in times of increased traffic and always be available.
- **QoS:** myChaseBook provides a lot of its services by composing several other web-services, that are created and controlled by third-parties. If these web-services fail to provide proper performance, then myChase-Book's reputation plummets. Hence, myChaseBook would like QoS from these web-services to be guaranteed via Service Level Agreements (SLA)
- **Cost:** The cost of using other web-services could fluctuate depending on demand and supply. myChaseBook should be able to automatically switch to a lower cost web-service, as long as it meets other requirements.

Using web-services in a composable manner to provide functionality is a typical instantiation of Service-Oriented Architecture.

Using SOA, myChaseBook should be able to dynamically evolve its architecture by modifying the web-services to which it connects. When organizational priorities change, and myChaseBook wants to optimize on dependability (say), it could search for a web-service that offers an SLA with high dependability and use that web-service instead of the high performance web-service that it is currently using. However, decisions like changing focus from one Quality Attribute to another (performance to dependability) will probably not be optimal for the entire myChaseBook application. Each feature might want to optimize for a different Quality Attribute. For example, features like tagging of streaming radio and tracking of friends via cellphone might want to optimize for performance, while features like purchase of music tracks via cellphone might want to optimize for security. Further, all of these optimizations would have cost as a constraint as well. In this context, it is easy to see how different parts of an application might want to optimize differently, each realizing different tradeoffs.

III. RELATED WORK

There has been extensive work on self-managed architecture, as reported by Kramer and Magee [6]. Current research pursues software architectures as the appropriate level of abstraction for evaluating, reasoning about, managing and facilitating the dynamic change and evolution of complex software systems.

There have been a plethora of attempts at creating self-managing architectures. These attempts can be classified as approaching the problem from the following perspectives:

- Map system to ADL, dynamically change ADL, check for constraints, transform ADL to executable code [7].
- Create a framework for specifying types of adaptation possible using constraints and tactics thus ensuring 'good adaptation' [8], [9].
- Using formalized methods like SAM [10], control theory [11] and middleware optimizations [12].

However, all of these approaches envision a closed-loop system. That is, all of them assume that

1) The entire state of the application and the resources available for adaptation are known/visible to the management component.
2) The adaptation ordered by the management component is carried out in full, never pre-empted, ignored or disobeyed.
3) The management component gets full feedback or is able to view the results of changes made, on the entire system.

We feel that these are limiting assumptions because of the following reasons:

1) In a system of systems (say a web-application comprising several web-services), each software component will have a different objective to attain and hence need not cooperate. In a cloud, we expect a system of

systems to be implemented through composable web-services. Also, since each web-service is potentially owned or managed by a different entity, its optimization objective could be different.

2) In a truly large scale system, communication between centralised management components and distributed lower-level components may not be feasible/robust.

3) In a large scale system, global application knowledge is infeasible. If there are thousands of object instances, each with its own performance/memory issues, it is infeasible to expect a central optimizer to take into account each of those instances' states or metrics while trying to optimize in reasonable time.

IV. OUR APPROACH

A. Problem in Detail

We see that the architectural approaches proposed in the previous section do not fit the requirements of myChase-Book. The limitations elucidated preclude the achievement of the properties discussed in the motivating example. We envisage that applications that want to provide / consume services will register themselves with the cloud, which acts as the central registry or even as a broker for negotiating price and SLAs. However, different web-services would have differing agendas and different properties that they wish to optimize for. It is in this context that this paper's main contribution lies. We consider web-services running in the cloud ecosystem and consider how these services might optimize themselves and what this means for the quality of the cloud as a whole.

Each web-service will set its own priorities with regard to cost, performance, reliability and availability. Negotiation amongst these services will occur through any of a variety of resource reservation protocols as long as they conform to the *Web services agreement specification* [13]. The problem that we consider is whether such a system would reach its optimal level of cost vis-a-vis other quality attributes.

B. The El Farol solution

We view the distributed selfish self-optimization as the *El Farol Bar* problem [14] and seek inspiration in the solutions proposed in [15], [16]. In the *El Farol Bar* problem, the bar reaches its optimal level of customers, without any prior communication or coordination amongst the jazz-loving population. As long as the patrons are free to adjust their strategy about deciding whether to go to the Bar or not, the emergent result of multiple individual selfish optimizations is the optimal level of customers for the Bar, as well. Similarly, in our case, we posit that every web-service should attempt local self-optimization only, without any knowledge of the orchestrating web-application as a whole. The resulting emergent effect would be near-optimal for the web-application. *Yagan and Tham* describe a successful attempt [17] in using Reinforcement Learning to optimize
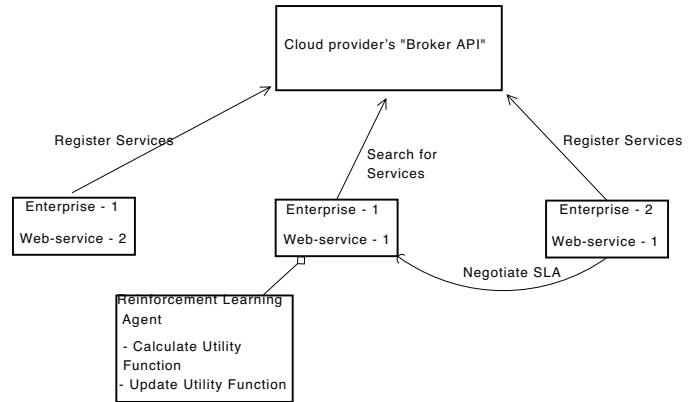

Figure 1. Interaction between autonomous web-services in the cloud

network routing. We hope that a similar mechanism would work amongst web-services as well. A software architecture can be specified as a triad of components, connectors and topology. In the particular case of the cloud, topology is opaque to the web-services and hence, the only optimization possible is in the types of components and connectors used. Thus, if a certain web-service of myChaseBook requires dependability rather than performance, it would connect to other web-services that offer dependability in their SLA. Modelling the web-services in a cloud in such a manner requires viewing the cloud as a marketplace. Each web-service can be viewed as an agent with a budget to spend and a targetted quality attribute to optimize. The cloud provider could be viewed as the 'market maker 'or the broker, that specifies the negotiation protocol to be followed when web-services bid for services. Now, each agent applies its own learning algorithm to choose amongst the strategies it has, to negotiate with other agents. Depending on the quality attribute that a particular agent is interested in optimizing, it would be prepared to pay a high price for an SLA that offers the desired guarantees. For example, an agent that is interested in optimizing performance would pay a premium for an SLA that guarantees a certain bandwidth or certain algorithmic performance. Alternatively, another agent interested in availability would pay a premium for an SLA that promises redundancy. How much premium to pay for obtaining a particular SLA is an engineering problem in decision making, with tradeoffs between multiple quality attributes and cost. A possible solution could be modelled using multi-attribute utility theory [18]. Allowing each agent to set its own parameters in a utility function creates a marketplace where each agent would set a (possibly) different price on different quality attributes. As shown in [16], if there are enough agents with differing strategies for maximising their utility, the system as a whole quickly reaches a near optimal level.

Returning to the motivating example, during a sustained period of low traffic when performance is no longer a high

priority, myChaseBook's web-services would automatically chose to connect to other web-services that are cheaper and provide the same functionality. This sort of dynamic re-arrangment of components would lead to optimal level of cost vs QAs for the application. We contend that with enough diversity in the population of web-services (and corresponding learning strategies), the cloud as a whole will reach a level of optimal cost vs. quality attribute tradeoff.

C. Applicability

Applications built in this manner should be resilient to changes in cloud conditions with regard to demand, change in types of supporting services and even organisational objectives. Thus, maintainability of applications in the cloud is greatly increased. For instance, a web-service that seeks to optimize on cost, would constantly monitor the demand being made on it and switch to cheaper composing services as long as SLAs that it agreed to, are met. Now, if the organisation's objective were to change to providing a high level of availability, the web-service would (due to a change in its utility function) switch to more expensive services as long as they promise higher availablity in their SLA.

V. Future Work

We think that modelling of the problem in greater detail would be necessary to carry out any experimental evaluation. We will propose more precise application of multi-attribute utility theory to SLA negotiation. Also, modification strategies that a web-service can learn from, will be proposed. This would enable simulations of a cloud with negotiating web-services, thus allowing us to test our idea of low-level self-optimization leading to an emergent higher level optimized application state in the cloud. If successful, this would lead to long-lived applications in the cloud being more resilient to change, and successfully adapting to changing Quality Attribute optimization needs.

Acknowledgement

References

[1] M. Parashar and S. Hariri, *Autonomic Computing: An Overview*, 2005.

[2] IBM, "Ibm perspective on cloud computing," Whitepaper published on the web, http://www-05.ibm.com/services/ch/download/ibm-cloud-computing-12-08.pdf, 2008.

[3] C. Boulton, "Gartner sees great saas enterprise app growth despite downturn," *eWeek.com*, October 2008.

[4] A. Inc, *Amazon Elastic Compute Cloud (Amazon EC2)*. http://aws.amazon.com/ec2/#pricing: Amazon Inc., 2008.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009.

[6] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," in *FOSE '07: 2007 Future of Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 259–268.

[7] E. M. Dashofy, A. van der Hoek, and R. N. Taylor, "Towards architecture-based self-healing systems," in *WOSS '02: Proceedings of the first workshop on Self-healing systems*. New York, NY, USA: ACM Press, 2002, pp. 21–26.

[8] I. Georgiadis, J. Magee, and J. Kramer, "Self-organising software architectures for distributed systems," in *WOSS '02: Proceedings of the first workshop on Self-healing systems*. New York, NY, USA: ACM Press, 2002, pp. 33–38.

[9] S. Cheng, D. Garlan, and B. Schmerl, "Architecture-based self-adaptation in the presence of multiple objectives," in *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*. Shanghai, China: ACM, 2006, pp. 2–8.

[10] J. Wang, C. Guo, and F. Liu, "Self-healing based software architecture modeling and analysis through a case study," in *Networking, Sensing and Control, 2005. Proceedings. 2005 IEEE*, 2005, pp. 873–877.

[11] J. Hellerstein, *Engineering Self-Organizing Systems*, 2007, p. 1.

[12] M. Trofin and J. Murphy, "A Self-Optimizing container design for enterprise java beans applications," in *In Proceedings of the Second International Workshop on Dynamic Analysis (WODA 2004*, 2003.

[13] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web services agreement specification," http://xml.coverpages.org/WS-Agreement-13652.pdf.

[14] B. W. Arthur, "Inductive reasoning and bounded rationality," *American Economic Review (Papers and Proceedings)*, no. 84, pp. 406–411, 1994.

[15] D. H. Wolpert and K. Tumer, *An Introduction to Collective Intelligence*, August 1999.

[16] D. H. Wolpert, K. R. Wheeler, and K. Tumer, "Collective intelligence for control of distributed dynamical systems," *Europhysics Letters*, vol. 49, pp. 708–714, 2000.

[17] D. Yagan and C. K. Tham, "Self-optimizing architecture for qos provisioning in differentiated services," in *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 346–347.

[18] D. L. Thurston, *Decision Making in Engineering Design*. ASME Press. New York, New York, 2006, ch. Multi-attribute Utility Analysis of Conflicting Preferences, pp. 125–133.