Autonomous Adaptation of Cloud Applications

Everton Cavalcante[1], Thais Batista[1], Frederico Lopes[1], André Almeida[1], Ana Lúcia de Moura[2], Noemi Rodriguez[2], Gustavo Alves[1], Flavia Delicato[3], and Paulo Pires[3]

[1] UFRN – Federal University of Rio Grande do Norte, Natal, Brazil
evertonrsc@ppgsc.ufrn.br, thais@ufrnet.br, {fred.lopes,gustavoalvescc}@gmail.com, andre.almeida@ifrn.edu.br

[2] PUC-Rio – Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil
{amoura,noemi}@inf.puc-rio.br

[3] UFRJ – Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
{fdelicato,paulo.f.pires}@gmail.com

Abstract. Cloud-based applications are built from services offered by distinct third-party cloud providers. Most cloud-related information (service properties such as price, availability, response time, etc.) can change at any time during application execution. Therefore, it is essential to support the adaptation of applications in such dynamic conditions in order to ensure that the cloud services currently provided to deployed applications adhere to the established requirements. In this paper we present an autonomous adaptation process for cloud-based applications by replacing a service by an alternative one that fulfills the application needs. We discuss the factors that trigger an adaptation in a Cloud Computing scenario and describe the adaptation process within *Cloud Integrator*, a service-oriented middleware platform for composing, executing, and managing services provided by different cloud platforms.

Keywords: Cloud Computing, Cloud services, *Cloud Integrator*, Semantic workflows, Autonomous adaptation.

1 Introduction

The growing interest in the Cloud Computing paradigm is grounded on its utility model in which *computing services* are delivered through a pay-per-use model. By exploiting this model, applications can be composed of services provided by distinct third-party cloud providers. The selection of the proper cloud services that fit the application needs is based on non-functional information, i.e. properties of the services such as price, availability, response time, etc., and applications can rely on a middleware that abstracts away the burden of directly dealing with underlying mechanisms for service selection and communication with the cloud providers.

In this context, our previous work introduced *Cloud Integrator* [1–3], a service-oriented middleware platform for composing, executing, and managing services provided by different Cloud Computing platforms, so that *Cloud Integrator* works as a *mediator* between the service providers and the applications (clients). Moreover, it provides an environment that facilitates the development and execution of fault-tolerant applications that use such services by composing *semantic Web services* [4] in a *semantic workflow* [5] composed of a sequence of abstract *activities* that must be performed by concrete cloud services in order to achieve the application's business goal. In order to execute such semantic workflow, it is necessary to create at least one *execution plan* containing a set of concrete Web services that perform each one of the activities specified in the workflow.

By using *Cloud Integrator*, an application can specify the set of cloud services that it needs as well as the properties of each service, and the selection mechanism provided by the middleware platform is able to choose the cloud services provided by the integrated platforms that fulfill the application requirements. However, most cloud-related information (service properties) can change at any time during application execution, so that it is essential to support the *adaptation* of applications in such dynamic conditions to ensure that the cloud services currently provided to the applications adhere to the established requirements.

In this paper we present an adaptation process to coordinate the autonomous adaptation of cloud applications based on the replacement of services by alternative ones that fulfill the application requirements, in case of service failure or when any change in the properties of a cloud service (e.g. quality parameters) can potentially affect the running application(s). Section 2 details our adaptation process, which is evaluated in Section 3. Finally, Section 4 presents final remarks.

2 Adaptation of Cloud Applications

The service composition model adopted by *Cloud Integrator* is based both on the *functionality* of each service and on its *metadata* (such as QoS parameters and prices), thus enabling a better choice of the available services. Moreover, this composition mechanism enables *Cloud Integrator* to deal with situations in which a service that is available at composition time becomes unavailable at runtime or in case of quality degradation of any service that is included in the composition. If two or more services with similar functionality are available, then *Cloud Integrator* builds different execution plans for the current workflow, each of them using one of these alternative services. Thus, in case of service failure or quality degradation, another execution plan that contains the service with similar functionality can replace the current one in order to ensure the quality and availability of the running application. In this section we describe the *adaptation process* that supports this capability. Herein, an *adaptation* of an application means to replace a running execution plan by another one that performs the same activities. The adaptation process performed by *Cloud Integrator* currently addresses the replacement of *application services*, which may be SaaS and/or PaaS cloud services or even other traditional (non-cloud) services.

2.1 Factors That Trigger an Adaptation

There are three classes of changes in cloud environments that may trigger the adaptation process. The first one is the *failure of one or more services* due to loss of connection between *Cloud Integrator* and a service provider or internal service errors. In both cases, the service provider becomes unable to respond to requests and then the current execution plan that contains such service must be immediately replaced by an alternative execution plan that contains a service with similar functionality, when possible. For instance, suppose that the execution plan $A \rightarrow B \rightarrow C_1 \rightarrow D$ is being executed and the service C_1 becomes unavailable. In this case, the adaptation mechanism must select an alternative execution plan with the equivalent service C_2 (for instance, the execution plan $A \rightarrow B \rightarrow C_2 \rightarrow D$), thus avoiding the failure of the whole application.

The second class of events that may trigger an adaptation is the *quality degradation of one or more services*. Cloud environments can present highly dynamic execution conditions in which fluctuations in the network and high service usage may affect the quality of executing services. In this case, an alternative execution plan containing a service that provides similar functionality to the degraded service can be adopted if its utility[1] [3] is significantly higher than the utility of the current execution plan. Finally, the third case is the *arising of new services*, which can be dynamically discovered. Since these new services may provide more advantageous alternative execution plans for running applications, it is necessary to analyze the need and convenience of replacing a current execution plan with an alternative one that contains one or more new services. Similarly to quality degradation, it is also important to consider the utility gains and the impact regarding the replacement of the current execution plan. In the current development state of *Cloud Integrator*, the adaptation process is only triggered in case of service failures. Adaptation triggered by quality degradation or discovery of new services will be addressed in future works.

2.2 Factors That Affect the Adaptation Process

When the adaptation process replaces an execution plan regarding an application, its major concern is to ensure that the application requirements continue to be satisfied. Although the quality of alternative execution plans is an essential factor to be considered, it is not sufficient to ensure an efficient adaptation. Moreover, since replacing an execution plan may lead to the re-execution of services or other costly actions, the decision about such possible replacement must consider the *adaptation cost*, which is the *overhead* imposed by actions that must be performed in order to resume the application after replacing the current execution plan with an alternative one. As an example, when some of the services included in the current execution plan have already been executed, it may be advantageous to select the alternative execution plan that has more

[1] As presented in our previous work [3], the computation of the utility of an execution plan takes into account both QoS parameters and the monetary costs of the services that compose it.

services in common with the current one. Therefore, such similar plan can offer a lower adaptation cost by reusing the outputs produced by the executed services, thus avoiding the need of actions that may have considerable impact.

The computation of the adaptation cost regarding alternative execution plans must consider the following factors: (i) *reuse of executed services*; (ii) the *rollbacks* required to return services that have already been executed to a previous execution state, and; (iii) *compensatory actions* taken in order to restore an previous execution state when a service needs to return to a previous state and it does not support rollbacking. In this perspective, the computation of the *adaptation cost* of an execution plan p starts with the computation of its *absolute adaptation cost* $c_{abs}(p)$ (Eq. 1), which is defined as the sum of the number of services to be executed after the replacement of the current execution plan (e), the number of services that require rollbacks (r), and the number of services that require compensatory actions (a):

$$c_{abs}(p) = e + r + a \tag{1}$$

In turn, the adaptation cost $c(p)$ regarding an execution plan p is calculated through a vector normalization of its absolute adaptation cost $c_{abs}(p)$ (Eq. 2):

$$c(p) = 1 - \frac{\frac{1}{c_{abs(p)}}}{\sqrt{\sum_{r \in EP} \left(\frac{1}{c_{abs}(r)^2} \right)}} \tag{2}$$

in which $c_{abs}(r)$ is the absolute adaptation cost of each execution plan r in the set of available execution plans EP.

2.3 The Adaptation Process

The adaptation process performed by *Cloud Integrator* was inspired in *Adapt-UbiFlow* [6], an adaptation mechanism proposed in the context of ubiquitous applications. The selection of an alternative execution plan considers two essential parameters: (i) the *initial utility* (u) of the candidate execution plans, which expresses their properties in terms of price and quality parameters, and; (ii) the *adaptation cost* (c), which weights the actions that must be performed when replacing the application's execution plan. When the adaptation process is triggered, the initial utility of the alternative plans is computed by using the same criteria (QoS parameters and prices) that were originally used for selecting the application's current execution plan [3]. Next, the *adaptation cost* is computed for each alternative plan, as shown in Equation 2. In the adaptation process, the selection of an execution plan is based on the computation of the *adaptation utility* $\mu(p)$ of each alternative execution plan p defined as a weighted sum of the initial utility $u(p)$ and the adaptation cost $c(p)$. Equation 3 shows how the adaptation process computes the adaptation utility $\mu(p)$ for each alternative execution plan p, so that the alternative plan with maximum adaptation utility is selected to replace the current execution plan. If two or more execution plans have maximum utility, the adaptation process selects one of them at random.

$$\mu(p) = [u(p) * w_{EP}] + [c(p) * w_{AC}] \tag{3}$$

The weights w_{EP} and w_{AC} in Equation 3 are respectively assigned by the user to the initial utility and the adaptation cost, with w_{EP}, $w_{AC} \in [0, 1]$ and $w_{EP} + w_{AC} = 1$. The values assigned to these weights are defined in terms of five different configurations called *adaptation profiles*, as shown in Table 1.

Table 1. Adaptation profiles for the definition of the weights assigned to initial utility and adaptation cost

Adaptation profile	**Description**	**Assigned weights**	
		w_{EP}	w_{AC}
maximum initial utility	exclusive priority to the initial utility of the execution plan	1.00	0.00
high initial utility	prioritizes the initial utility of the execution plan while considering the adaptation cost	0.75	0.25
balanced	default configuration, with equal weights	0.50	0.50
low adaptation cost	prioritizes the adaptation cost, while also considering the initial utility of the execution plan	0.25	0.75
minimum adaptation cost	exclusive priority to the adaptation cost	0.00	1.00

After selecting an alternative execution plan, the required rollbacks and compensatory actions are transparently performed in order to replace the current execution plan and resume the execution of the application.

3 Evaluation

A preliminary evaluation was performed aiming to assess the time spent by the adaptation process triggered in case of failure of a service involved in the execution plan that is being executed, so that such failure is detected when a timeout in which the service does not respond to the request is reached. In this perspective, services involved in the selected execution plan for executing the application were forced to fail in order to trigger the adaptation process. As shown in Table 2, the time spent in milliseconds to perform the adaptation process (and that covers the selection of an alternative execution plan to replace the current one) is significantly small, thus not significantly impacting on the application execution. More details about the performed evaluation can be found at `http://consiste.dimap.ufrn.br/projects/cloudintegrator/dais2013`.

Table 2. Minimum and maximum values, average and standard deviation regarding the time (in milliseconds) spent by the adaptation process

Execution plans	Minimum	Maximum	Average	Standard deviation
2	0.2754	0.5950	0.4694	0.0708
4	0.4598	0.9522	1.2352	0.1626
8	0.7716	1.9416	1.7846	0.2574
12	1.4680	3.7006	3.2596	0.5168
18	4.0158	11.7134	7.2724	1.7644

4 Final Remarks

In this work we discussed the events that can trigger the need for adaptation in a cloud-based environment, as well as the factors that should be taken into consideration when choosing the best way to react to changes in the runtime environment in order to ensure the quality and availability of the application. We described the adaptation support designed for *Cloud Integrator* in which service failures and quality degradation are addressed with an algorithm that takes the adaptation cost (incurred in the replacement of services) into account. A preliminary evaluation of such adaptation process in case of service failures has shown that the adaptation process does not significantly impact in the application execution. In addition, the proposed adaptation process works with minimal user awareness, thus promoting the autonomy of the application in case of failures or other conditions that may trigger an adaptation.

References

1. Cloud Integrator, `http://consiste.dimap.ufrn.br/projects/cloudintegrator/`
2. Cavalcante, E., et al.: Cloud Integrator: Building value-added services on the cloud. In: First International Symposium on Cloud Computing and Applications, pp. 135–142. IEEE Computer Society, USA (2011)
3. Cavalcante, E., et al.: Optimizing services selection in a cloud multiplatform scenario. In: 2012 IEEE Latin America Conference on Cloud Computing and Communications, pp. 31–36. IEEE Computer Society, USA (2012)
4. Martin, D., et al.: Bringing Semantics to Web Services: The OWL-S Approach. In: Cardoso, J., Sheth, A.P. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 26–42. Springer, Heidelberg (2005)
5. Abbasi, A., Shaikh, Z.: A conceptual framework for smart workflow management. In: 2009 International Conference on Information Management and Engineering, pp. 574–578. IEEE Computer Society, USA (2009)
6. Lopes, F., et al.: AdaptUbiFlow: Selection and adaptation in workflows for Ubiquitous Computing. In: 9th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, pp. 63–71. IEEE Computer Society, USA (2011)