Towards a Goal-oriented Approach to Adaptable Re-deployment of Cloud-based Applications

Patrizia Scandurra[1], Marina Mongiello[2], Simona Colucci[2] and Luigi Alfredo Grieco[2]
[1]*Dip. di Ingegneria Gestionale dell'Informazione e della Produzione, Universitá di Bergamo, Dalmine (BG), Italy*
[2]*Dipartimento di Ingegneria Elettrica e dell'Informazione, Politecnico di Bari, Bari, Italy*

Keywords: Cloud Applications, Adaptation, Deployment, Goal-model.

Abstract: Due to the on-demand and dynamic nature of Cloud, there is an increasing interest for automated management of adaptation and (possibly) re-deployment of cloud applications to realize quality requirements and evolution needs autonomously at run-time. This paper proposes a fast and automated approach for adapting and re-deploying a cloud application at run-time as dictated by evolution needs and sudden changes in the operating environment conditions. The proposed approach exploits a graph-based model and an algorithm that extracts a sub-graph identifying the adaptation processes to be executed according to evolution changes. The approach is general enough to be implemented by any cloud application management framework. A TOSCA-based description of the structure and management aspects of the cloud application may be updated according to the above mentioned sub-graph. Then, this description may be processed by a TOSCA-compliant runtime environment to effectively adapt and possibly re-deploy the cloud application in an automated manner. The paper also illustrates the instantiation of this generic approach for adapting an e-commerce cloud application.

1 INTRODUCTION

Modern service-oriented software applications, like those envisioned in cloud computing scenarios, operate in highly dynamic and often uncertain and unpredictable environments that can degrade their quality of service (such as availability, reliability, performance, etc.). Moreover, *rapid elasticity* and *self-service* – to enable on demand dynamic growing and shrinking of resources and service composition depending on the user/evolution needs (e.g., extending the storage of an email account, adding temporarily a social login service to the conventional authentication system, etc.) – are becoming key goals in cloud service and application management (Lehrig et al., 2015; Dubois et al., 2015; Andrikopoulos et al., 2013).

Managing the adaptation of complex applications over clouds with high evolving resource provisioning and service compositions, while guaranteeing quality attributes, is one of the emerging problems in the cloud era, especially for data-intensive applications (Casale et al., 2015). In particular, automation of cloud application deployment and management (including adaptation) across clouds, regardless of provider platform or infrastructure, is becoming a prerequisite to realize key cloud properties (see, for example, the works (Wettinger et al., 2015; Brogi et al., 2015; Andrikopoulos et al., 2013), to name a few, addressing such topics).

In this paper, we investigate the challenge in adapting and re-deploying cloud applications at run-time according to evolution needs and operating environment conditions changes (such as brokerage of new resources, QoS optimization and tradeoffs, failures, changes in the workload and size of jobs, elasticity requirements based on application-level metrics, etc.). Cloud adaptation essentially requires configuration and elasticity changes in the application topology at any cloud layer through low-level actions at IaaS (Infrastructure as a Service) level, such as a new Virtual Machine (VM) is added or a resource-related parameter of a VM is changed (e.g., a new disk is attached), or high-level actions at the PaaS (Platform as a Service)-SaaS (Software as a Service) levels, such as moving an application service from one IaaS provider to another or adding a new service.

To this purpose, we propose an automated approach for adapting and re-deploying a cloud application at run-time as dictated by evolution needs and sudden changes in the operating environment conditions. The proposed approach exploits a goal-oriented graph-based model and a fast algorithm (based on

Scandurra, P., Mongiello, M., Colucci, S. and Grieco, L.
Towards a Goal-oriented Approach to Adaptable Re-deployment of Cloud-based Applications.
In *Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER 2016) - Volume 1*, pages 253-260
ISBN: 978-989-758-182-3

the well-known Dijkstra algorithm) that extracts a sub-graph identifying the adaptation processes to be executed according to evolution changes/events. A TOSCA-based[1] description of the structure and management aspects of the cloud application may be updated according to the above mentioned sub-graph. The description may be then processed by a TOSCA-compliant runtime environment to effectively adapt and possibly re-deploy the cloud application in an automated manner. The paper also illustrates the instantiation of the approach for a scalability adaptation scenario in an e-commerce cloud application.

The proposed approach is general enough to be implemented by any cloud application management framework. The proposed approach can be used, for example, to strengthen current management engines (e.g., current auto-scaling tool available in the cloud market) or to support adaptation decision making at run-time in more complex management frameworks (see, for example, the initiative (Brogi et al., 2015)).

This paper is organized as follows. Section 2 provides some background assumptions and concepts about TOSCA for realizing the proposed approach. Section 3 introduces a running example used throughout the paper. Section 4 presents our goal-oriented adaptation approach and exemplifies it on the running example; in particular, it presents a scalability scenario of the application example involving scaling and adding of an additional login service as adaptation actions. Section 5 reports some related initiatives, while Section 6 provides concluding remarks and challenges we want to address as future work.

2 BACKGROUND ASSUMPTIONS

We assume that a cloud application is described in terms of the OASIS standard TOSCA. TOSCA allows to express in a portable manner how to automatically deploy and manage complex cloud applications (Binz et al., 2014) by requiring developers to define an abstract topology of a multi-component Cloud application and to create plans describing its deployment and management. The proposed approach can be therefore implemented in any TOSCA-compliant cloud platform offering a TOSCA container (like OpenTOSCA) able to process an archive format called CSAR (Cloud Serivce ARchive) packaging the TOSCA Cloud application specification together with concrete implementation and deployment artifacts. TOSCA containers not only support application at deployment time, but also at run-time.

In TOSCA, the architecture (or topology) of a cloud application is explicitly modeled by a graph where nodes represent the components of the application and edges represent different kinds of relations between these components. Relations may be, for example, one component is “hosted on”, “depends on”, or “communicates with” another component. Nodes and edges in the topology may define additional properties, the management operations they offer (e.g., how to setup the component, establish a relation, deploy artifacts, scale-up, or backup), the artifacts required to run the component, or non-functional requirements. Figure 2 in Section 3 shows the topology delivering the e-commerce cloud application we consider as running example throughout the paper.

Without loss of generality, we assume that each server component is installed in a stand-alone VM. Accordingly, the scaling up/down the cloud application taken as example in this work typically involves adding/removing extra software servers, and hence extra VMs in a cloud environment.

Moreover, we assume that a TOSCA container adopts a *declarative processing mechanism* (Brogi et al., 2014), i.e. it deploys the application by trying to automatically excerpt a deployment plan from the application topology. Essentially, the CSAR engine (a) first deploys the nodes without requirements on other nodes, and then (b) until all nodes have been deployed, it searches the nodes whose requirements are satisfied (by the capabilities of the already deployed nodes) and deploys them. This way of processing allows us to easily adapt the current application deployment by adapting directly the TOSCA application topology. Alternatively, an *imperative processing* model could be adopted by taking the CSAR archive and deploying the application according to a TOSCA build plan (defined using a workflow modeling language such as BPMN[2] or WS-BPEL[3]) combining the operations offered by the nodes in the topology.

3 RUNNING EXAMPLE

In this paper, we consider a scalability scenario where an e-commerce cloud application is adapted to provide the amount of load balancing capacity required to distribute application traffic. The e-commerce cloud application is divided into multiple tiers (see

[1]The OASIS Topology and Orchestration language (http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html)

[2]http://www.bpmn.org/

[3]http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

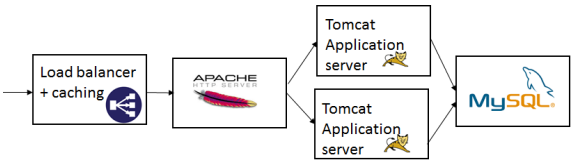

Figure 1: Basic, multi-tier e-commerce web application.

the logical structure of such application shown in Figure 1 in a free-style notation). An elastic load balancer is used as frontend for accepting and distributing end users' requests automatically across multiple application server instances (the back end) in the cloud. Moreover, it has the power to cache content according to some rules to set; this means that when a request comes for some content that exists in the cache, the load balancer can simply deliver it directly to the browser, rather than handing the request to the application server(s). The real application tiers (presentation, business and data tiers) are implemented through an Apache web server for handling http requests, a middle-tier Tomcat application server for implementing the business logic, and a backend MySQL database with data store and processing. These servers work together to handle end users' requests. The corresponding topology description in TOSCA is shown in Figure 2. The Presentation Tier and Business Tier are declared scalable (by the multiplicity 1..n), and the composite application's template includes a Load Balancer node with content caching that understands how to connect to and scale them. The DBTier is also scalable. The Presentation Tier includes the entire set of dependent service components necessary for deploying the e-commerce web application on an Apache web server (with PHP Module support, for example). It's required hosting environment also includes a Linux Operating System (OS), a VM container and a "Tier container"[4] to logically group all related components. Similarly for the other tiers.

Initially, the application is deployed (see Figure 1) on five server instances in the Cloud to support a small number of customers. Depending on the application workload, the servers at each tier can be stressed at different times and the implementation ideally needs to scale up or down the resources at the appropriate tier so as to maintain the overall quality requirements of the application while minimizing the cost of resources used. To this purpose, each tier is elastically scaled independently to adapt the application in response to load changing demand. In particular, a separate elastic scaling is adopted for the Presentation tier and the Business tier because the processing components are more computation intensive or used less frequently than the User Interface components. For example, additional server instances can be launched automatically (e.g., in the example of Figure 3 one Apache server and two Tomcat servers are added) when the load on the current instances rises, for example, to 70 percent, and then some server instances can be removed to reduce the cost of service provision when the load falls to 40 percent.

Adaptation Requirements. As types of adaptation, various reactive or anticipatory scaling actions could be done vertically (i.e., scale up/down) or horizontally (i.e., scale out/in) to handle the increased demand. The load balancer with caching, the message queues, the presentation tier and the business-tier can be scaled horizontally by adding more cache and queue devices and server instances at each tier into the computing cloud platform. The database tier can be scaled both vertically, by adding more resources to the same computing pool (e.g., more disks), and horizontally, by using a MySQL master-slaves configuration model with data replication. In the last case, a MySQL Master is initially deployed and, when the database tier is scaled up, extra MySQL Slaves are added and configured with replication from the MySQL Master. The numbers of load balancer servers and MySQL Master database do not change. These scaling actions have different prices and realization costs that may also change at run-time. Usually, vertical scaling can handle most sudden, temporary peaks in application demand on cloud infrastructures since they are not typically CPU intensive tasks. Sustained increases in demand, however, require horizontal scaling and load balancing to restore and maintain peak performance. Horizontal scaling is also manually intensive and time consuming, requiring a technician to add machinery to the customers cloud configuration, and this may be not productive since traffic may settle to its pre-peak levels before new provisioning can come on line.

In addition to scaling actions, that mostly affect the PaaS and IaaS layers, the e-commerce cloud application can be adapted by adding a new service component at SaaS level to reflect an evolution need in the application requirements. Essentially, to increase sales for a certain event calendar (e.g., in the Christmas period or Black Fridays), a new social login component (for example, Facebook) is added temporarily to the application to allow not registered users to purchase orders by using their personal social account credentials (e.g., their Fb authentication credentials) to authenticate into the system.

[4]"Tier" is a topological concept used to describe sets of nodes (or sub-topologies) that can be deployed and managed as a single group.

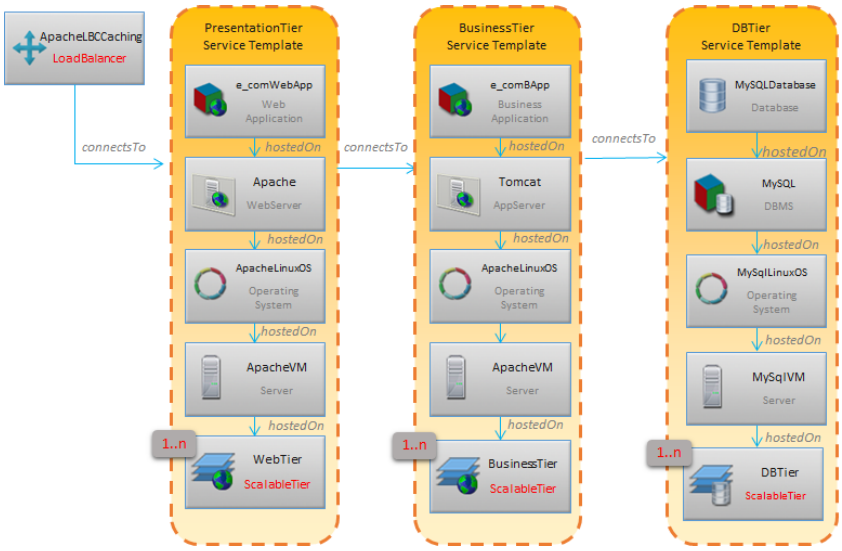

Figure 2: TOSCA topology of the e-commerce web application.

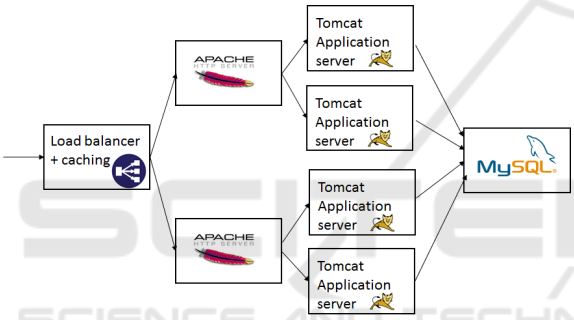

Figure 3: Scaled, multi–tier e-commerce web application.

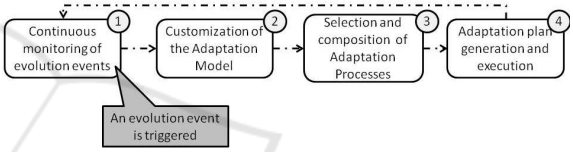

Figure 4: Generation and execution of an adaptation plan.

4 ADAPTATION APPROACH

The proposed methodology for the generation and the execution of adaptation plans is summarized in the steps shown in Figure 4. The proposed approach helps in deciding and guiding the necessary adaptation changes to carry out at different Cloud tiers of the application as dictated by evolution needs. Essentially, the management framework implementing the proposed approach observes, according to configured monitoring parameters (i.e., sampling rate), and generates real-time graphs by aggregating metrics originated from multiple sources[5]. From these graphs, appropriate adaptation processes are determined and translated on-the-fly into TOSCA-based adaptation plans.

Metrics and monitor mechanisms/frameworks for identifying the evolution events or predict future changes that may trigger the Cloud application adaptation at run-time and eventually its re-deployment are out of the scope of this paper.

[5]Metrics can be obtained either from native Cloud monitoring systems or from custom metric collectors (probes) provided by the user during deployment.

4.1 Adaptation Model

The proposed adaptation approach is grounded on a graph-based model. This model generalizes the elements involved in the composition of adaptation processes in heterogeneous scenarios of interest.

In particular, different scenarios are described in terms of three inter-related building entities: evolution *events*, *adaptation processes* triggered by events and *cloud node tiers* implementing adaptation processes. Formally, the general model is defined as an **Adaptation Graph** in the following:

Definition 1 (Adaptation Graph (AG)). *Let* $\mathbf{EV}$, $\mathbf{AP}$, $\mathbf{CT}$ *be three sets describing events, adaptation processes and cloud tiers, respectively.*

An Adaptation Graph is a weighted directed graph $G = \{\mathbf{V}, \mathbf{E}\}$, *such that:*

- $\mathbf{V} = \mathbf{EV} \cup \mathbf{AP} \cup \mathbf{CT}$, *i.e., a vertex can model an event, an adaptation process or a cloud node tier;*
- $\mathbf{E} = (\mathbf{EV} \times \mathbf{AP}) \cup (\mathbf{AP} \times \mathbf{CT}) \cup (\mathbf{CT} \times \mathbf{EV})$, *i.e., an edge connects either an event to an adaptation*

process or an adaptation process to a cloud tier or a cloud tier to an event;

- *a function* $c : E \rightarrow \Re_+ \cup \{0\}$ *labels edges in E as follows:*
  1. $c(v,w)$ *is the cost of performing an adaptation process w triggered by an event v, for* $v \in EV$ *and* $w \in AP$
  2. $c(v,w)$ *is the cost of implementing an adaptation process v in a cloud tier w, for* $v \in AP$ *and* $w \in CT$
  3. $c(v,w) = 0$*, for* $v \in CT$ *and* $w \in EV$

The model is aimed at finding the best composition of adaptation processes satisfying a goal, generated at run-time by an occurring event. In the Adaptation Graph, the adaptation cost can be computed based on the performance of the physical platform and/or defined by the system administrator depending on the user's profiles of the application.

In the following, a **Goal** is defined according to the proposed model.

Definition 2 (Goal)**.** *Given an AG* $R = (V,E)$*, defined according to Definition 1, a* Goal *G in R is an ordered pair* (s,d) *with* $s \in EV \cap V$ *and* $d \in EV \cap V$.

Intuitively, a goal is defined as a transition from an original status (modeled by the event node s) to a destination status (modeled by the event node d).

The reader may note that, given the graph structure of an AG R, if a goal (s,d) is identified in R, all paths connecting s to d need to match patterns made up by sequences of ordered quadruples $(Event, AdaptationProcess, CloudTier, Event)$. Of course, many of such paths exist. This paper only focuses on minimum cost paths and on possible **Goal-oriented Adaptation Subgraphs (GASs)** of R, defined by such paths.

Definition 3 (Goal-oriented Adaptation Subgraph (GAS))**.** *Given an AG* $R = (V,E)$*, defined according to Definition 1, and a goal* (s,d) *in R, defined according to Definition 2, a Goal-oriented Adaptation Subgraph (GAS) of R is a direct graph* $S = (V',E')$*, such that:*

- $S \subseteq R$*;*
- $s \in V'$ *and* $d \in V'$*;*
- E' *includes a path p connecting s to d, such that* $cost(p) \leq cost(r)$ *for every path r in R connecting s to d.*

Algorithm *Adapt* is now proposed for finding a GAS S of an AG $R = (V,E)$, given R and a goal (s,d) in R. In Row 2, the Dijkstra algorithm (Dijkstra, 1959), well-known form graph theory, is called. The algorithms associates to each node $v_i \in V$, the cost of the (minimum) path from s to v_i and the node v_{i-1} preceding v_i in the minimum cost path. Then (Row 3), the minimum cost path from s to d is read from the results of Dijkstra algorithm and S is consequently built in Rows 4–5. Note that Algorithm *Adapt* is based on the Dijkstra algorithm (Dijkstra, 1959), and thus, its time complexity depends on the chosen implementation. In particular, if Fibonacci Heap is adopted as priority queue, the algorithm requires $O(|E|+|V|log|V|)$ running time in the worst case.

	Data: An AG $R = (V,E)$, a goal (s,d) in R,
	Result: A GAS $S = (V',E')$ of R
1	let c be the function labeling R;
2	**solve** $DIJKSTRA(R,c,s)$
3	**read** a minimum cost path p_{min} from s to d
4	**add** all nodes in p_{min} to V'
5	**add** all edges in p_{min} to E'

Algorithm 1: *Adapt*.

At instantiation level, the model presented hereby needs to be referred to an application scenario, for which specific events, adaptation processes and cloud tiers have to be identified, together with the relations among them. This activity leads to the definition of an application-specific AG. Then, when an occurring event requires intervention, the following steps have to followed: i) a contextual AG is built by customizing the application-specific AG on the basis of run-time settings; ii) a Goal is identified, again depending on run-time settings; iii) the GAS determined by the AG and the Goal is computed.

4.2 Model Instantiation

In this section we instantiate the graph model previously defined by considering a concrete adaptation scenario w.r.t. scalability of the e-commerce cloud application taken as running example. Let us suppose the e-commerce service is initially deployed with a Load Balancer with one cache, one Apache web server instance, two Tomcat servers, one MySQL Master server instance. In order to illustrate the composition of adaptation processes required by our example, we need to refer to the AG model instance in Figure 5. Note that, in the proposed example, there is only one outgoing transition from an adaptation process to a Cloud node tier (except for the adding/removing the social login service that is linked to three social networking services); in a more realistic scenario involving a variability of alternative Cloud technologies for realizing a specific Cloud node tier, there would be different outgoing transitions with different costs.

The evolution event initiating the adaptation is an event calendar, for example Christmas days beginning (or Black Fridays). Coherently with the evolu-

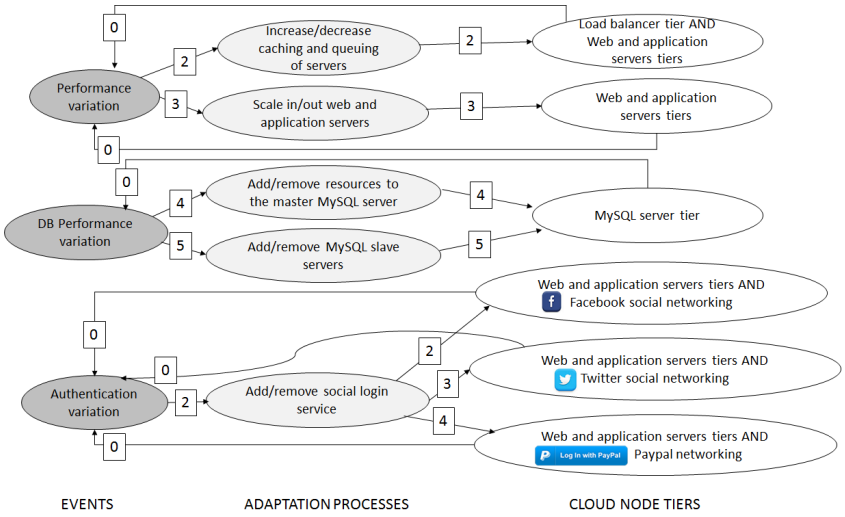

Figure 5: AG for the e-commerce application scalability scenario.

tion needs, the adaptation goal is the pair (authentication variation, authentication variation) to add/remove a new social login service to the application. To address the goal, in the AG in Figure 5 there are different paths that can be followed depending on the different cloud social networking services and their costs (e.g., utilization-level/serviceability). The *Adapt* algorithm selects the minimum cost path and therefore determines the GAS shown in Figure 6.

This originating goal may in turn increase workload significantly, thus generating new adaptation goals related to the performance of the application that need to be managed by one or more further adaption processes. In particular, in a first period the number of concurrent users is 100[6] and when the concurrent users increases, for example to 300, affecting the application performance (performance variation), a vertical scaling process is selected by increasing the capacity of the load balancer caching and of the server's message queues. The target goal is (users requests variation, performance variation). The resulting GAS extracted for this goal can be intuitively derived from Figure 5.

If the number of users increases further, e.g. to 500, and saturates the Apache and Tomcat tiers (performance variation), horizontal scaling is triggered and one Apache and two Tomcats servers are added. When the number of concurrent users increases further the adaptation cycle repeats. In contrast, when this number decreases, the application is scaled down by removing idle servers and reducing the capacity of the cache and message queues.

Data-intensive jobs are also monitored and if the number of traffic data per user increases stressing the data tier (DB performance variation), a vertical scaling of the database tier is first executed by adding resources to the MySQL master server. Further workload increasing that affects the DB performance requires horizontal DB scaling by enabling the master/slaves configuration with data replication.

Once a triggering event occurs, the customization of the application-specific AG and the composition of the adaptation processes (construction of the GAS) are completed within few milliseconds. From the GAS, an adaptation plan may be generated by updating the TOSCA description of the application and re-deploying it on-the-fly using the deployment service of the Cloud platform provider. Technical details needed to adapt the TOSCA description may be generated from accessory data attached to the AG nodes. The adaptation of the TOSCA topology from an AG model is out of the scope of this paper.

5 RELATED WORK

General purpose studies on adaptation based on QoS optimization and balancing are available in the literature. The SeaClouds project focuses on the problem of how to deploy and manage, in an efficient and adaptive way, complex applications across multiple heterogeneous cloud platforms. A key ingredient in the proposal is the use of two OASIS standards initiatives for cloud interoperability, CAMP and TOSCA, which allow for: describing the

[6]The values in the example are used only for illustration purposes. Realistic workload measures can be obtained through monitoring systems based on statistical analysis and on the amount of impact that various tuning parameters produce on the application performance.

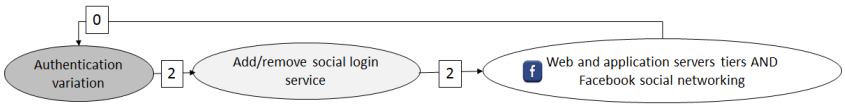

Figure 6: GAS extracted for the first goal.

topology of user applications independently of cloud providers; providing abstract plans; discovering, deploying/reconfiguring, and monitoring the applications independently of the peculiarities of the cloud providers (Brogi et al., 2015).

A Self-adaptive hierarchical monitoring mechanism for Clouds is proposed in (Katsaros et al., 2012), where a multi-layered monitoring framework measures QoS at both application and infrastructure levels. The framework targets trigger events for runtime adaptability of resource provisioning estimation and decision making.

Changed user needs, system intrusions or faults, changing operational environment, resource variability are among the main kinds of adaptation required in Service Oriented Architectures. An automatic optimization process for adaptation of service-oriented applications is proposed in (Mirandola et al., 2014). The approach is based on trade-offs between functional and extra-functional requirements. The proposed methodology relies on heterogeneous service assembly and open tools and runtime infrastructures to process architectural models that are directly tight to the real assembled components implementations and their distributed deployment.

Optimal solutions for cloud layers when resource allocation is needed and to support explicit cooperation between layers is addressed in (Scandurra et al., 2012) using Pareto multi-objective optimization.

All these approaches satisfactorily address adaptation at design and at maintenance time, but are less flexible than our one in coping with evolution needs emerging at runtime.

Several frameworks are available for providing more specific solutions to addressed problems.

c-Eclipse is an open-source Cloud Application Management Framework through which users are able to define the description, deployment and management of their cloud applications in a clean and intuitive graphical manner (Sofokleous et al., 2014).

JCatascopia is a Monitoring System capable of supporting a fully automated cloud resource provisioning system with proven interoperability, scalability and low runtime footprint (Trihinas et al., 2014).

Celar is a platform able to automatically scale applications deployed over a cloud infrastructure. Celar allow Cloud developers to build efficient services and achieve high performance by elastically managing the use of available resources (Giannakopoulos et al., 2014).

The framework proposed by (Copil et al., 2013c) is made up of the SYBL language (Copil et al., 2013b) and the Mela service (Moldovan et al., 2013). For the control of elasticity the framework use the control mechanism detailed in (Copil et al., 2013a).

Both methods and tools analyzed in this state of the art face adaptation from well-defined perspectives and propose solutions to specific need emerging in an adaptation scenario. On the contrary, our work manages adaptation goals as general evolution needs. As a consequence, the proposed approach generates adaptation plans through a workflow which is independent of the evolution need to address. This makes the whole approach general enough to be implemented in any monitoring and management cloud framework.

6 CONCLUSION

This paper proposes a goal-based approach to the run-time selection and composition of adaptation processes in cloud-based applications. The approach exploits a graph-model to generate and execute an adaptation plan on the basis of evolution needs emerging at run-time in a scenario of interest. Such a plan is derived from sub-graphs in the general model extracted by a specifically proposed algorithm. The instantiation of this generic approach in an e-commerce cloud application is given as running example.

We are currently working at the full implementation of the proposed approach, so to proceed with the evaluation of its feasibility in different scenarios involving cloud-based applications. To this purpose, we are combining open source tools such as those of the OpenTOSCA initiative and cloud middleware software stacks (like Openstack and Opscode Chef configuration management software) to develop a cloud management framework supporting our approach and evaluate an end-to-end realistic scenario. In particular, we are defining a mapping from a graph-based representation of an adaptation plan to a TOSCA-based adaptation plan.

As future work, we want to extend our approach on several directions. Essentially, we want to: cope with the management of concurrent multiple goals; investigate on the adoption of an imperative processing model for deploying a TOSCA-based cloud application through a workflow; study how to make the

system dynamically perform architecture-based adaptation to meet run-time quality requirements (such as availability, performance, resilience, and greenness). Architecture tactics could support the realization of this last aim (see, for example, the work in (Mirandola et al., 2014)), but only if embedded at design time into the TOSCA cloud topology of the application and enabled/disabled at run-time accordingly.

ACKNOWLEDGEMENTS

This work is supported in part by the EC H2020 program, project EscudoCloud (644579), by the Apulia Region initiative "Future in Research", and by the PON MIUR Edoc@work 3.0.

REFERENCES

Andrikopoulos, V., Binz, T., Leymann, F., and Strauch, S. (2013). How to adapt applications for the cloud environment - challenges and solutions in migrating applications to the cloud. *Computing*, 95(6):493–535.

Binz, T., Breitenbücher, U., Kopp, O., and Leymann, F. (2014). chapter TOSCA: Portable Automated Deployment and Management of Cloud Applications, pages 527–549. Springer, New York.

Brogi, A., Carrasco, J., Cubo, J., Nitto, E. D., Durán, F., Fazzolari, M., Ibrahim, A., Pimentel, E., Soldani, J., Wang, P., and D'Andria, F. (2015). Adaptive management of applications across multiple clouds: The seaclouds approach. *CLEI Electron. J.*, 18(1).

Brogi, A., Soldani, J., and Wang, P. (2014). Tosca in a nutshell: Promises and perspectives. In Villari, M., Zimmermann, W., and Lau, K.-K., editors, *Service-Oriented and Cloud Computing*, volume 8745 of *Lecture Notes in Computer Science*, pages 171–186. Springer Berlin Heidelberg.

Casale, G., Ardagna, D., Artac, M., Barbier, F., Nitto, E. D., Henry, A., Iuhasz, G., Joubert, C., Merseguer, J., Munteanu, V. I., Perez, J. F., Petcu, D., Rossi, M., Sheridan, C., Spais, I., and Vladuic, D. (2015). DICE: quality-driven development of data-intensive cloud applications. In *7th IEEE/ACM International Workshop on Modeling in Software Engineering, MiSE 2015*, pages 78–83. IEEE.

Copil, G., Moldovan, D., Truong, H.-L., and Dustdar, S. (2013a). Multi-level elasticity control of cloud services. In *Service-Oriented Computing*, pages 429–436. Springer.

Copil, G., Moldovan, D., Truong, H.-L., and Dustdar, S. (2013b). Sybl: An extensible language for controlling elasticity in cloud applications. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 112–119. IEEE.

Copil, G., Moldovan, D., Truong, H.-L., and Dustdar, S. (2013c). Sybl+ mela: Specifying, monitoring, and controlling elasticity of cloud services. In *Service-Oriented Computing*, pages 679–682. Springer.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271.

Dubois, D. J., Valetto, G., Lucia, D., and Nitto, E. D. (2015). Mycocloud: Elasticity through self-organized service placement in decentralized clouds. In *8th IEEE International Conference on Cloud Computing, CLOUD 2015*, pages 629–636. IEEE.

Giannakopoulos, I., Papailiou, N., Mantas, C., Konstantinou, I., Tsoumakos, D., and Koziris, N. (2014). Celar: automated application elasticity platform. In *IEEE International Conference on Big Data (Big Data)*, pages 23–25. IEEE.

Katsaros, G., Kousiouris, G., Gogouvitis, S. V., Kyriazis, D., Menychtas, A., and Varvarigou, T. A. (2012). A self-adaptive hierarchical monitoring mechanism for clouds. *Journal of Systems and Software*, 85(5):1029–1041.

Lehrig, S., Eikerling, H., and Becker, S. (2015). Scalability, elasticity, and efficiency in cloud computing: a systematic literature review of definitions and metrics. In *Proc. of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures, QoSA'15 (part of CompArch 2015)*, pages 83–92. ACM.

Mirandola, R., Potena, P., and Scandurra, P. (2014). Adaptation space exploration for service-oriented applications. *Sci. Comput. Program.*, 80:356–384.

Moldovan, D., Copil, G., Truong, H.-L., and Dustdar, S. (2013). Mela: Monitoring and analyzing elasticity of cloud services. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 1, pages 80–87. IEEE.

Scandurra, P., Raibulet, C., Potena, P., Mirandola, R., and Capilla, R. (2012). Adapting cloud-based applications through a coordinated and optimized resource allocation approach. In *CLOSER 2012 - Proceedings of the 2nd International Conference on Cloud Computing and Services Science*, pages 355–364. SciTePress.

Sofokleous, C., Loulloudes, N., Trihinas, D., Pallis, G., and Dikaiakos, M. D. (2014). c-eclipse: An open-source management framework for cloud applications. In *Euro-Par 2014 Parallel Processing*, pages 38–49. Springer.

Trihinas, D., Pallis, G., and Dikaiakos, M. D. (2014). Jcatascopia: monitoring elastically adaptive applications in the cloud. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 226–235. IEEE.

Wettinger, J., Breitenbücher, U., and Leymann, F. (2015). Compensation and convergence - comparing and combining deployment automation approaches. *Int. J. Cooperative Inf. Syst.*, 24(3).