Multi-level Elasticity Control of Cloud Services*

Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, Schahram Dustdar

Distributed Systems Group, Vienna University of Technology
{e.copil,d.moldovan,truong,dustdar}@dsg.tuwien.ac.at

Abstract. Fine-grained elasticity control of cloud services has to deal with multiple elasticity perspectives (quality, cost, and resources). We propose a cloud services elasticity control mechanism that considers the service structure for controlling the cloud service elasticity at multiple levels, by firstly defining an abstract composition model for cloud services and enabling multi-level elasticity control. Secondly, we define mechanisms for solving conflicting elasticity requirements and generating action plans for elasticity control. Using the defined concepts and mechanisms we develop a runtime system supporting multiple levels of elasticity control and validate the resulted prototype through experiments.

1 Introduction

Cloud services[1] are designed in a fashion that they typically use as many as possible resource capabilities from cloud providers and are distributed on different virtual machines consuming various types of services offered by cloud providers, possibly from different cloud infrastructures. Therefore, requirements for them would differ from the traditional applications, and potentially, they can achieve elasticity not only in terms of resources but also of cost and quality.

1.1 Motivation

In our previous work we have developed SYBL [1], a language for elasticity requirements specification which enables the user to define: (i) *monitoring* specifications for specifying which metrics need to be monitored, (ii) *constraints* for specifying acceptable limits for the monitored metrics, (iii) *strategies* for specifying actions to be taken under certain conditions, and (iv) *priorities* for the previous specifications. Listing 1.1 shows a *cost-related elasticity requirement* specified by, e.g., the service designer, using SYBL, stating that when the total cloud service price is higher than 800 Euro, a scale-in action is needed.

---

* This work was supported by the European Commission in terms of the CELAR FP7 project (FP7-ICT-2011-8 #317790).

[1] In this paper, *cloud service* refers to the whole cloud application, including all of its own software artifacts, middleware and data, that can be deployed and executed on cloud computing infrastructures.

Listing 1.1: SYBL elasticity directives

```
@SYBL_ServiceUnitLevel(Id="CloudService",strategies=
    "St1: STRATEGY CASE  total_cost>800 Euro : ScaleIn")
@SYBL_CodeRegionLevel(Id="AnalyticsAlgorithm",constraints=
   "C1: CONSTRAINT dataAccuracy>90%;
    C2: CONSTRAINT dataAccuracy>95% WHEN total_cost>400;
    C3: CONSTRAINT total_cost<800;"
priorities="Priority(C2)>Priority(C1);Priority(C3)>Priority(C1);")
```

While elasticity requirements can be specified at different levels, current elasticity control techniques do not support controlling different parts of the cloud service (i.e., elasticity requirement on service unit, on groups of service units) and from a multi-dimensional perspective. Controlling the cloud service at multiple levels enables a finer-grained control according to described elasticity requirements. On the other hand, multiple levels of elasticity requirements could give rise to conflicts on cross-level or even on the same levels. Therefore, we need solutions for overcoming cross-level conflicting elasticity requirements and generating plans for multi-level elasticity control.

1.2 Related Work

Controlling cloud services elasticity in the contemporary view has been targeted by both research and industry. Several authors propose controllers for the automatic scalability/elasticity of entire cloud services [2] or just parts of the cloud service (i.e., cloud service data-end) [3]. Guinea et al. [4] develop a system for multi-level monitoring and adaptation of service-based systems by employing layer-specific techniques for adapting the system in a cross-layer manner. Kranas et al. [2] propose a framework for automatic scalability using a deployment graph as a base model for the application structure and introduce elasticity as a service cross-cutting different cloud stack layers. Cloud providers offer tools for automatic scalability like AutoScale[2] or SmartCloud initiative[3], automatically scaling resources depending on user's detailed resource-level policies. However, these approaches do not control the cloud service on multiple levels taking into consideration the complex service structure, or the multiple dimensions of elasticity (quality, resources, and cost) [5].

1.3 Contributions

In this paper, we propose a system for multi-level cloud services elasticity control by considering the service complex structure and supporting multi-dimensional elasticity. We present the following contributions: (i) a generic composition model of cloud services for enabling the fine-grained control aware of the structure of the cloud service and (ii) a fine-grained, multiple levels automatic elasticity control of cloud services.

[2] http://aws.amazon.com/autoscaling/

[3] http://www.ibm.com/cloud-computing/us/en/index.html

The rest of this paper is organized as follows: Section 2 defines our generic composition model. Section 3 presents our techniques supporting multi-level elasticity control while Section 4 presents experiments. Section 5 concludes the paper and outlines our future work.

2 Mapping Service Structures To Elasticity Metrics

2.1 Elasticity metrics

Cloud service metrics differ on the service type, the service unit targeted by the metric, or the environment in which the service resides. *Resource-level metrics* are the most encountered in cloud IaaS APIs (e.g., IO cost, CPU utilization, disk access, memory usage). *Service unit-level metrics* refer to service units (e.g. web server, or database server) and are used for having a higher level view and being able to determine the unit's health or performance (e.g., request queue length, response time, price). Going higher into the abstraction level, when evaluating the performance of the cloud service one usually considers *cloud service-level metrics* like the whole cloud service response time or number of users per day. In elasticity control, these metrics can be associated to different cloud service parts (e.g., the whole cloud service, service unit or a group of service units), usually, metrics from higher levels (e.g., cloud service level) aggregating metrics from lower levels.

2.2 Abstracting cloud services

For obtaining highly granular control of cloud services and being aware of what service unit is being controlled, a model for structuring service-related information is needed. Our proposed model shown in Figure 1 has the form of a graph, with various types of relationships and nodes, representing both static and runtime description of the cloud service and aims at supporting different types of cloud services (e.g. queue-based applications, or web applications):

– *Cloud Service*, e.g., is a web application, or a scientific application. The cloud service represents the entire application/system, and can be further decomposed into service topologies and service units. The term is in accordance with existent architectures and standards (e.g., IBM [6] and TOSCA [7]).
– *Service Unit* [8], e.g., is a database, or a load balancer. The service units are modules or individual services offering computation or data capabilities.
– *Code Region*, e.g., is a data or a computation intensive code sequence. A code region is a code sequence for which the user has elasticity requirements.
– *Service Topology*, e.g., is a business tier, data tier, or a part of a workflow. A cloud service topology represents a group of service units that are semantically connected and that have elasticity capabilities as a group.
– *OS Process*, e.g., is a web server process or any process of the cloud service.
– *Elasticity Metric*, e.g., is cost vs. throughput, or cost vs. availability. Elasticity metrics can be associated with any cloud service part (e.g., service unit, service topology, or code region).

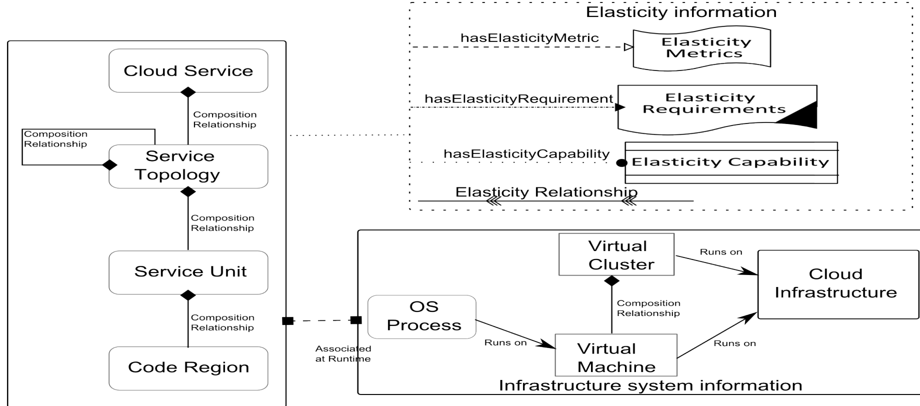

Fig. 1: Cloud service abstraction model

- *Elasticity Requirement*, e.g., is a SYBL directive. They can be specified through any language (e.g. SYBL) and are linked to any cloud service part.
- *Elasticity Capability*, e.g., is the elastic reconfiguration for higher availability, or the creation of new processing jobs for a map-reduce application.
- *Elasticity Relationship*, e.g., is a connection between any two cloud service parts, which can be annotated with elasticity requirements.

In order to describe the cloud service during runtime, a **dependency graph** (Figure 2) is used. The dependency graph is an instantiation of the described model, capturing all the information concerning structure and runtime information like metrics and associated virtual machines.

If we take the example of a Web service (the left side of Figure 2),the cloud user views his/her Web service as a set of services, the metrics targeted in users elasticity requirements being high level metrics. At runtime, the dependency graph is constructed (right part of the figure), service instances being deployed on virtual machines, in different virtual clusters, and the accessible metrics are low level ones. These two views on metrics (cloud user and control system) are mapped by our elasticity control runtime, aggregating low-level metrics for computing higher level ones.

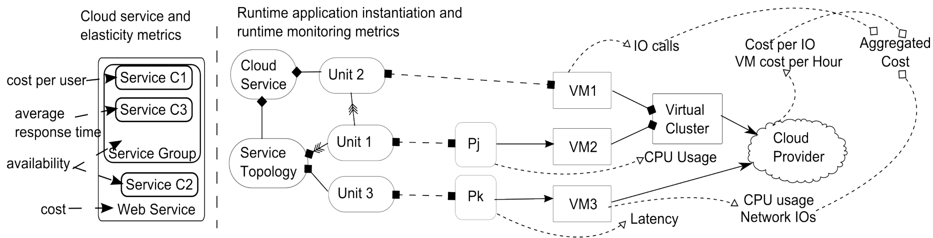

Fig. 2: Constructing runtime dependency graph

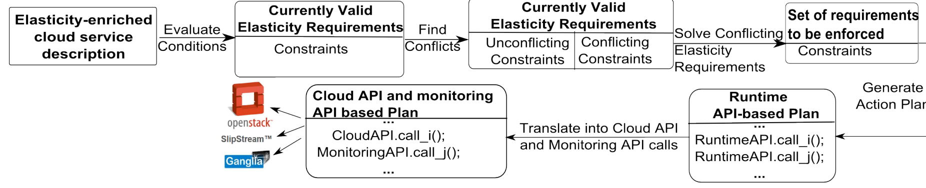

Fig. 3: Elasticity control: from directives to enforced plans

Algorithm 1 Solving single-level and cross-level elasticity requirements conflicts

1: **function** SolveSingleLevelConflicts($graph_i$)
2: **for each** l in $cloudServiceAbstractionLevels$ **do**
3: $confConstraints$= getConflictingConstraints($graph_i$,l)
4: $graph_i$.removeConstraints($confConstraints$)
5: **for each** $constraintSet$ in $confConstraints$ **do**
6: $newGeneratedConstraintsLevel$.add(constraintSolving($confConstraints$))
7: **end for**
8: $graph_i$.addConstraints($newGeneratedConstraintsLevel$)
9: **end forreturn** $graph_o = graph_i$
10: **end function**
11: **function** SolveCrossLevelConflicts($graph_i$)
12: **for each** $level1$ in $cloudServiceAbstractionLevel$ **do**
13: **for each** $level2$ in $cloudServiceAbstractionLevel$ **do**
14: **if** $level1 \neq level2$ **then**
15: $conflictingConstraints$.add(getConflictingConstraints($level1$,$level2$))
16: **end if**
17: **end for**
18: $graph_i$.removeConstraints($conflictingConstraints$)
19: $graph_i$.addConstraints(translateToHigherLevel($conflictingConstraints$))
20: **end forreturn** $graph_o$=SolveSingleLevelConflicts($graph_i$)
21: **end function**

3 Multi-level Elasticity Control Runtime

Considering the model of the cloud service described through the abstract model presented in the previous section, we enable multiple levels elasticity control of cloud services, based on the flow shown in Figure 3. The elasticity requirements are evaluated and conflicts which may appear among them are resolved. After that, an action plan is generated, consisting of actions which would enable the fulfillment of specified elasticity requirements.

3.1 Resolving elasticity requirements conflicts

We identify two types of conflicts: (i) conflicts between elasticity requirements targeting the same abstraction level, and (ii) conflicts which appear between elasticity requirements targeting different abstraction levels. For the first type, as shown in function $SolveSingleLevelConflicts$ from Algorithm 1, sets of conflicting constraints are identified and a new constraint overriding previous set is added to the dependency graph for each level (lines 3-10). In the second type of conflicts (see Algorithm 1, function $SolveCrossLevelConflicts$) the constraints from a lower level (i.e., service unit level) are translated into the higher constraint's level (i.e., service topology level), by aggregating metrics considering

Algorithm 2 Generating the action plan enforcing the constraints

Input: *graph* - Cloud Service Dependency Graph
Output: *ActionPlan*

```
1: while getNumberOfViolatedConstraints(graph) > 0 do
2:     for each level in cloudServiceAbstractionLevel do
3:         actionSet=evaluateEnabledActions(graph, getViolatedConstraints(graph,level)
4:         Action=findAction(actionSet) with max(constraints fulfilled - violated)
5:         addAction(ActionPlan,Action)
6:     end for
7: end whilereturn ActionPlan
```

the dependency graph. Since the problem is reduced to same-level conflicting directives, we use the approach for the same-level conflicting directives and compute a new directive from overlapping conditions. In both (i) and (ii) it can be the case of conflict for directives that are targeting different metrics which influence each other (i.e., cost and availability- when availability increases, the cost increases as well). However, knowing how one metrics' evolution affects the other is a research problem itself which we envision as future work.

3.2 Generating elasticity control plans

For generating the action plan, we formulate the planning problem as a maximum coverage problem: we need the minimum set of actions which help fulfilling the maximum set of constraints. Since maximum coverage problem is an NP-hard problem, and our research does not target finding the optimal solution for it, we choose the greedy approach which offers an $1 - \frac{1}{e}$ approximation. The greedy approach shown in Algorithm 2 takes as input the dependency graph and returns the action plan for enforcing the constraints. The main step of the plan generation loop (lines 2-9) consists of finding each time the action for fulfilling the most constraints. For evaluating this, each action has associated the metrics affected and the way in which it affects them (i.e., `scale out with VM of type x` increases the cost with 200 Euro). The number of fulfilled constraints through action enforcement is defined as the difference between the number of constraints enforced and the number of constraints violated.

4 Experiments

We have implemented elasticity control as a service based on SYBL engine [1] for supporting multi-level, cloud service model aware elasticity control of cloud services[4]. Figure 4 shows the elasticity requirements and the experimental cloud service which is a data-oriented application with two main topologies: a data servicing oriented topology and a data analytics oriented topology. For the YCSB [5] client we generate the workload as a continuous alternation of combinations

[4] Prototype, full paper and further details: `http://dsg.tuwien.ac.at/research/viecom/elasticitycontrol`

[5] https://github.com/brianfrankcooper/YCSB/wiki

Configuration	Controllers	DB Nodes	Total execution time	Cost
Config1	1	3	578.4 s	0.48
Config2	1	6	472.1 s	0.91
Config3	2	2	382.4 s	0.42
Config4	3	7	372.2 s	0.72

Table 1: Cost and execution time for `Data Service Topology` units

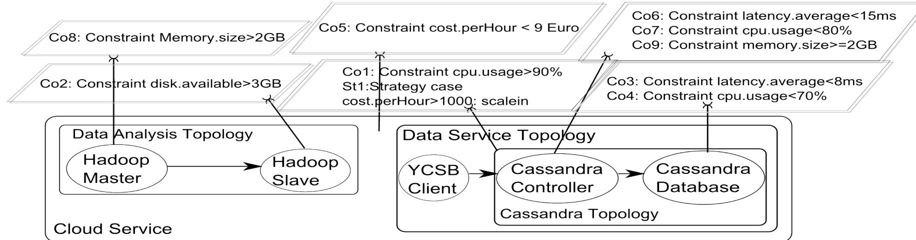

Fig. 4: Current cloud service structure and elasticity directives

of the enumerated types of workloads run in parallel. The Hadoop cluster to processes large data-sets using Mahout machine learning library[6].

For reflecting the importance of higher level elasticity control in addition to the obvious low level one, Table 1 presents performance and cost data on different Data Service Topology configurations. We assume each virtual machine costs 1 EUR/hour. Although scale out actions at service unit level do manage to increase performance (i.e., `Config2` vs. `Config1` increase in performance of 18.37%), they also enable a considerable cost increase (90 % increase in costs for `Config2` vs `Config1`). In contrast with this action level, a scale out action on Cassandra topology (`Config3`) offers a performance improvement in time of 33.88% over `Config1`, and a cost improvement of 12.03%. This is due to the fact that more controllers also increase the parallelism of requests, eliminate bottlenecks and facilitate the workload to finish in less time. However, when considering the difference of performance and cost between configurations 3 and 4, it is obvious that the dimension of the cluster and the number of clusters necessary are strongly dependent on the workload characteristics.

Figure 5 shows how the elasticity control engine can scale the Data Service Topology both at service unit and at service topology level, when directives shown in Figure 4 require such actions (e.g. scale out for Cassandra DB fixing "`Co4`" and scale out for Cassandra topology fixing "`Co4`" and "`Co7`").

5 Conclusions and Future Work

We have presented an elasticity control system which enables multi-level specification of elasticity requirements and execution of automatic elasticity of cloud services.

[6] http://mahout.apache.org/

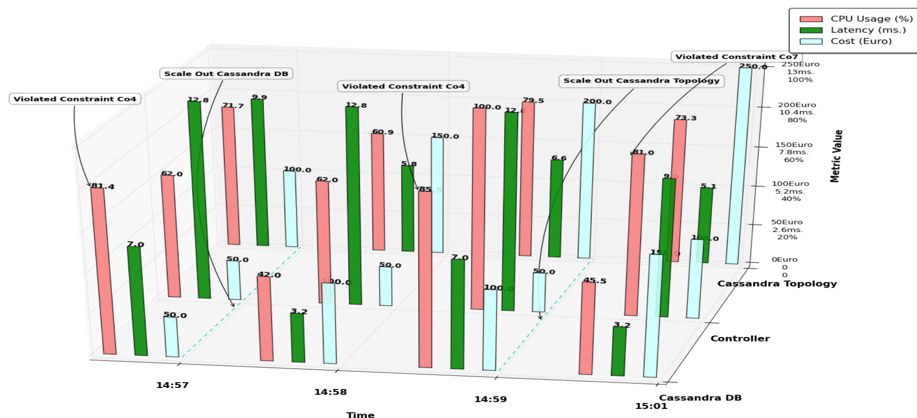

Fig. 5: Metrics (CPU usage, cost and latency) and elasticity actions for service units in `Data Service Topology`

With cross multi-level elasticity control capabilities, cloud providers could sell elasticity as a service to cloud consumers, allowing application code designers to specify elasticity in a high level manner and enforcing elasticity requirements for them while cloud consumers can deploy elastic services pre-packed with our techniques, which will automatically scale application components when needed.

References

1. Copil, G., Moldovan, D., Truong, H.L., Dustdar, S.: SYBL: an Extensible Language for Controlling Elasticity in Cloud Applications. In: 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), IEEE Computer Society (2013) 112–119
2. Kranas, P., Anagnostopoulos, V., Menychtas, A., Varvarigou, T.: ElaaS: An Innovative Elasticity as a Service Framework for Dynamic Management across the Cloud Stack Layers. In: 2012 Sixth International Conference on Complex, Intelligent and Software Intensive Systems (CISIS). (july 2012) 1042 –1049
3. Tsoumakos, D., Konstantinou, I., Boumpouka, C., University), S.S.I., Koziris, N.: Automated, Elastic Resource Provisioning for NoSQL Clusters Using TIRAMOLA. In: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), IEEE Computer Society (2013) 34–41
4. Guinea, S., Kecskemeti, G., Marconi, A., Wetzstein, B.: Multi-layered monitoring and adaptation. In: Proceedings of the 9th international conference on Service-Oriented Computing. ICSOC, Springer-Verlag (2011) 359–373
5. Dustdar, S., Guo, Y., Satzger, B., Truong, H.L.: Principles of Elastic Processes. IEEE Internet Computing **15**(5) (sept.-oct. 2011) 66 –71
6. IBM: IBM Cloud Computing Reference Architecture v3.0
7. OASIS Group: TOSCA Specification, v1.0 (2013)
8. Tai, S., Leitner, P., Dustdar, S.: Design by Units: Abstractions for Human and Compute Resources for Elastic Systems. IEEE Internet Computing **16**(4) (2012) 84–88