Automated Analysis of Cloud Offerings for Optimal Service Provisioning

José María García, Octavio Martín-Díaz, Pablo Fernandez, Antonio Ruiz-Cortés, and Miguel Toro

Universidad de Sevilla, Seville, Spain
{josemgarcia,omartindiaz,pablofm,aruiz,migueltoro}@us.es

Abstract. Cloud computing paradigm has brought an overwhelming variety of cloud services from different providers, each one offering a plethora of configuration and purchasing options for them. Users may have certain requirements and preferences not only concerning service configuration, but also with respect to their usage schedule. In this situation, an appropriate provisioning plan considering all restrictions would help users to achieve their goals while taking into account the different available providers, their pricing and even the usage discounts they provide. In this work, we describe an automated solution that analyzes user needs that include scheduling restrictions to obtain optimized provisioning plans for different cloud providers, which allow users to compare several offerings that possibly consider volume or usage discounts. We validate this solution against a realistic use case, while also providing a prototype implementation in the form of publicly available microservices.

Keywords: Cloud services · Pricing · Provisioning · Analysis

1 Introduction

The emergence of cloud computing have brought a significant shift in the IT industry economics for service providers and consumers alike [1,2]. Cloud services such as Amazon Elastic Computing Cloud (EC2) or Google Compute Engine offer virtual processing and storage resources (commonly referred to as Infrastructure as a Service, or IaaS in short), so that customers can purchase them as a way to reduce operational costs if compared with the procurement of on-premise, private computing infrastructures. However, the myriad of cloud service providers, as well as their overwhelming variety of configuration and purchasing options [3], result in a highly complex provisioning scenario for service consumers.

In this setting, there are major heterogeneity issues that make the comparison among providers rather difficult, e.g. different variables for configurations, additional purchasing variants apart from the usual pay-as-you-go option, billing and charge processes, and particular discount rules, to name a few. Furthermore,

users may also find convenient to specify their needs for cloud services provisioning including specific scheduling restrictions. These restrictions provide additional beforehand information concerning not only the number of instances of particular configurations that are needed at a certain time, but also the amount of time they are going to be used.

There are some on-line tools that allow consumers to search for an optimal configuration, such as Cloudorado.com and CloudScreener.com, according to their particular needs. However, these tools do not take into account scheduling. In this work, we present an automatic analysis framework that analyzes and compares cloud service offerings from multiple providers to obtain an optimal provisioning plan according to user needs. This plan specifies the amount and type of instances that have to be purchased and when they have to be initiated and terminated in order to fulfill user needs. We have developed a prototype implementation that has been validated in a particular scenario with two different providers.

The rest of the paper is structured as follows. Section 2 introduces a case study that further motivates our work. Next, Sect. 3 describes the conceptual model of the provisioning process and our solution to obtain optimal plans. Then, Sect. 4 presents the architecture of our solution, and Sect. 5 showcases our validation results. Section 6 discusses the related work. Finally, Sect. 7 concludes the paper and outlines our future work.

2 Motivation

There are several service provisioning scenarios where the usage schedule is known *a priori*. Thus, users can specify their needs including scheduling information so that a corresponding provisioning plan can be derived from it. We can characterize these service scenarios depending on the complexity of the usage scheduling and the configuration of services needed. On the one hand, the usage scheduling may consist on a simple interval when the service will be needed, or rather a complex schedule that includes several intertwined temporal slots. On the other hand, needed services complexity may range from a single service with a particular configuration, to a number of highly configurable services [4].

In the following we focus on a case study on the *virtualization of laboratory classes* in the context of our Software Engineering courses, which falls on the most complex scenario since there may be several different software needs for each course with varying scheduling needs. Furthermore, laboratory classes may have a dynamic evolution from two viewpoints: (1) the software being used on those classes may evolve, usually requiring increasing computing resources, and thus possibly rendering the corresponding hardware obsolete at short notice; and (2) the demand, due to the number of students, may vary along the academic year. In order to increase flexibility and save costs, these classes can be virtualized by purchasing cloud infrastructure to support their dynamic environment.

As an example, let us consider that we need to provide infrastructure for the laboratory classes of a year course beginning on Monday 19th September

2016, which requires a very simple hardware configuration of a two-core CPU with 4 GB RAM. The usage scheduling contemplates weekly, 2-hour sessions for several groups of varying number of students during each semester, which comprises 15 weeks, in addition to open classrooms and specific examination days.

In order to actually provision the infrastructural needs for these laboratory classes, we need to carry out corresponding provisioning actions against a cloud infrastructure service provider. Thus, our solution analyzes user needs, derives their associated provisioning plans aggregating the necessary provisioning actions according to the scheduling restrictions, and searches for suitable service offerings to obtain a corresponding charge plan that sums up the total cost, hence allowing the user to choose the best option in each case. Note that we are not considering additional costs, such as communication expenses, due to the difficulty to estimate *a priori* these aspects.

3 From User Needs to Cloud Services Provisioning Plans

In order to automatically generate a plan that specifies the provisioning events that fulfill certain user needs, we first need to model the relevant descriptions so that our solution can analyze and transform them into the resulting plan. User needs specifies the client's requirements on particular services (in our case study cloud infrastructure services, or IaaS in short). These requirements mainly state (1) the configurations which are needed to execute the client's software, and (2) the expected usage schedule.

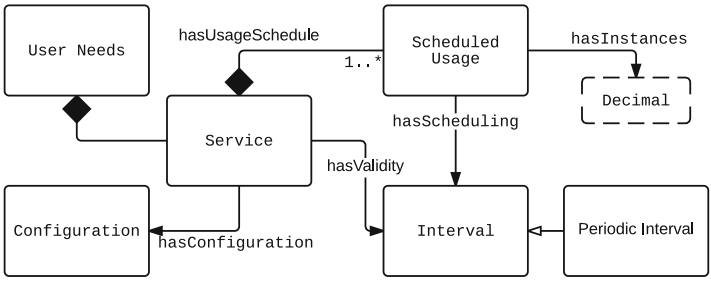

Fig. 1. Conceptual model for user needs

Figure 1 shows our conceptual model representing user needs. User needs are composed of a series of services that represent the different software components that the client needs to deploy to the cloud. In our example, each course is interpreted as a different service, which consists in a virtual machine containing all the relevant software for that course. According to this, each service is associated with its required configuration, which describes the hardware requirements for the requested cloud service instance. Thus, a configuration in case of an IaaS

may contain requirements about CPU, memory, IO performance, and storage, among others [3].

Regarding the expected usage schedule, each service enumerated in the user needs is associated with one or more scheduled usage items, which are *temporal composites* that detail the number of instances of the same configuration and the time interval when they are needed. Additionally, a global validity interval can be also specified. Unlike the latter, usage intervals may be periodic and disjoint or overlapped with others. In our motivating example, each course is given in several groups possibly with different timetables. Therefore, each group corresponds to a scheduled usage that specifies both the time interval when the course is given and the number of service instances that are needed, which depends on the group size.

Starting from the user needs, a provisioning plan that contains the actions to fulfill them is generated. It is optimal since (1) each chosen service is the best fit for the configuration expressed in the user needs, and (2) it minimizes the number of instances for each configuration for the whole validity period according to the usage schedule, favoring reserved instances, and hence decreasing the operational costs of cloud infrastructure.

The first step involved in the optimal plan generation is the optimization of usage scheduling. We analyze each service to be deployed separately, since we aim at minimizing the total number of instances needed for each configuration. Our solution takes the scheduling of every service and removes overlaps between time intervals. This is achieved by normalizing and coalescing the time sequence of the scheduling, which are well-known operations in the context of temporal databases [5]. Note that overlapping intervals leads to a higher number of instances to be run simultaneously, while disjoint intervals enables reusing of instances from one interval to the next, increasing their usage percentage so that reservation becomes a better purchasing option, hence diminishing the overall operational costs.

Once the optimal usage scheduling for each service is computed, the second step searches for the optimal service configuration from different providers. Different approaches can be applied to discover a suitable configuration from the pricing lists advertised by various IaaS providers. Our approach looks for the instance configuration from each available provider whose parameters are the closest to those stated in the user needs as in [3].

From the usage scheduling, our solution finally generates a particular provisioning plan for each cloud service provider, describing the minimum number of deployment actions that fulfills the usage schedule. Furthermore, purchasing options are also optimized so that the best purchasing type is chosen for each service instance in order to minimize the total cost of the cloud infrastructure to provision. As explained above, if the expected usage for an instance is long enough then it will be better to make a reservation as long as the provider offers such option. Otherwise, the instance will be used on-demand or pay-as-you-go basis. Thus, our approach enables the comparison of several offerings from different providers, taking into account their available configurations and pricing options.

4 Solution Architecture

In order to realize our approach, we developed a prototype solution that is based on the models described in Sect. 3 and implemented within a microservice architecture integrated in the Governify service management platform[1]. Currently, the prototype implementation supports two widely used cloud providers: Amazon EC2 and Google Compute Engine, but the architecture is designed to provide a systematic extension mechanism by means of adding new RESTful services that share a common interface.

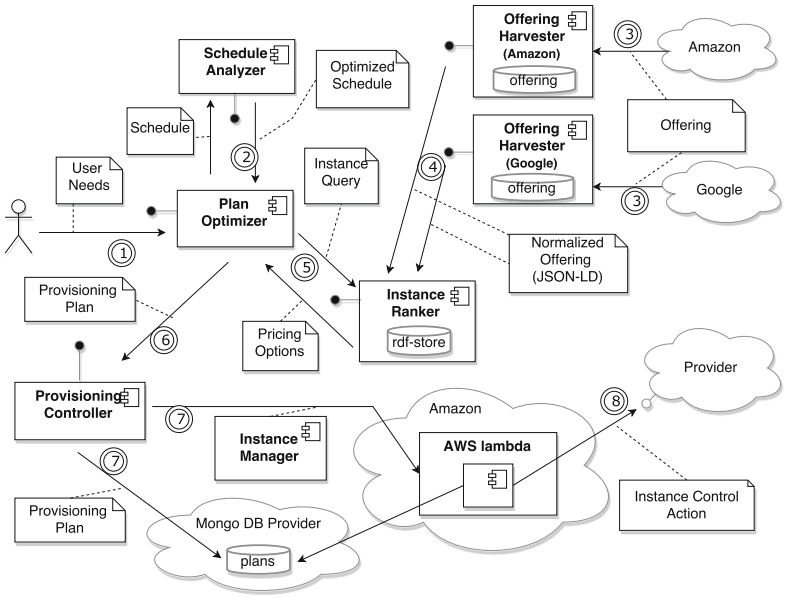

Fig. 2. Microservice architecture of the prototype.

On the one hand, pricing listings are automatically imported and standardized in our system using a JSON-LD [6] parser that takes JSON files published by service providers, such as Amazon[2] or Google[3], and annotate some properties to identify common properties (such as base price, CPU, or memory) using JSON-LD facilities. We use some cloud computing ontologies previously developed [7] as the fundamental schema to annotate configuration and pricing information from these providers, enabling interoperability of their original JSON schemas.

[1] https://governify.io.
[2] https://pricing.us-east-1.amazonaws.com/offers/v1.0/aws/AmazonEC2/current/index.json.
[3] https://cloudpricingcalculator.appspot.com/static/data/pricelist.json.

Then, annotated JSON pricing listings are parsed in order to populate the catalog of service offerings from different providers.

On the other hand, user needs are instantiated according to the model discussed in Sect. 3. First, the scheduling items are analyzed to optimize the usage schedule, and then the provisioning plan optimizer component analyzes user needs with its scheduling restrictions in order to obtain a specific provisioning plan that optimize costs for each provider.

From a deployment standpoint, Fig. 2 depicts the microservices architecture of the prototype describing the responsibility distribution and data interaction among services. As a high level overview, the flow starts when some user needs are sent (1) to the *Plan Optimizer* that acts as the main façade of the overall pricing analysis. This element interacts with the *Schedule Analyzer* microservice in order to obtain (2) an optimized version of the usage schedule as described in Sect. 3. Next, the *Plan Optimizer* uses the *Instance Ranker* microservice to develop a global search over different providers in order to obtain (5) a sorted set of instance type pricing options based on the preferences in the user needs (i.e. configurations). This search is based on the data maintained by the *Offering Harvesters* that gather service offerings from different cloud providers (3) to obtain a unified view annotated with JSON-LD that is fed (4) to the *Instance Ranker*. Finally, the *Plan Optimizer*, based on the ranking of instance types pricing and the optimized schedule, generates the actual provision plan with lower cost and send it (6) to the the *Provisioning Controller*. This controller is in charge of both (7) storing the plan in the appropriate persistence layer on a MongoDB provider and deploying (7) a new *Instance Manager* in an AWS Lambda platform that is responsible for executing the provisioning plan by means of actions (such as start or stop an instance) over the provider control API (8).

The implemented services are publicly available[4] with their interface documented following the Open API Initiative Specification[5]; in order to test the services, they all integrate an interactive testing on-line tool based on the Swagger[6] framework. We also developed a GUI for an end-user consisting on a wizard prototype[7]. This tool provides a user-friendly interface for defining needs and launch the appropriate microservices in a user-friendly way.

5 Case Study Validation Results

In order to validate our solution, we carried out the case study described in Sect. 2 using the implemented prototype. We considered service offerings from Amazon and Google. As stated by our user needs, our tool searched for the closer configuration to "`Cpu:2 Mem:4`" in their catalogs, using the approach presented in [3]. According to the *on-demand* purchasing type, the results of the search return a `t2.medium` configuration for Amazon (with a base price of 0.052$/h),

[4] https://pricing.governify.io/.
[5] https://openapis.org/.
[6] https://swagger.io/.
[7] https://designer.governify.io/demo/PlanOptimizer/wizard.

while in case of Google the most suited configurtion corresponds to `n1-std-2-pr` machine with a base price of 0.020$/h, as of price listings retrieved on October, 1^{st} 2016.

Based on these configurations, our solution generates a different provisioning plan for each provider. These plans allow a fine grained analysis of operational costs. Concretely, a comparative study of corresponding charge plans can be carried out, including the different expected charges per month along with the total cost for the whole provisioning plan. Table 1 shows the optimal charge plan, including discounts, for each provider derived from the provisioning plan generated by our prototype. It is interesting to note that in the Amazon case the maximum savings are derived from a full upfront (advanced payment of reserved instances) at the beginning which results in a considerable initial charge.

Table 1. Charge plans for our case study, discounts applied.

Amazon			Google		
Date	Type	Cost	Date	Type	Cost
Sep 19 2016	Upfront	12080.0$	*Sustained-use discounts are being applied.*		
Oct 01 2016	On-demand	20.12$	Oct 02 2016	On-demand	180.52$
Nov 01 2016	On-demand	67.08$	Nov 02 2016	On-demand	601.72$
Dec 01 2016	On-demand	67.08$	Dec 02 2016	On-demand	601.72$
...			...		
Total cost		**12884.96**$	**Total cost**		**7220.66**$

As a consequence of our analysis, we can determine that in our case study Google is the best option in terms of costs. Moreover, based on a preliminary analysis we realize that Amazon reserved instances prove to be competitive only if their usage is greater than approximately 75% of daily-usage, for a full year. Alternatively Google provides usage-sustained discounts of 25% of monthly-usage starting at the first month without the need for longer reservation periods as in the Amazon case.

The performed experiment validates that our proposal actually optimizes cloud provisioning, automatically generating plans from user needs while considering pricing models and discount rules of several cloud service providers. Although our use case has been kept deliberately simple for the sake of clarity, we can extend the scenario to include multiple courses in order to reach a higher usage ratio. We have already made some initial experiments on this matter, resulting in different comparative results. In particular, we found that Amazon provides more cost-effective options when the instance usage ratio is significant.

6 Related Work

Optimization of cloud provisioning can be considered from different perspectives. From an economic perspective, pricing models are extensively discussed

in [8,9]. In [10] authors present a comprehensive method to calculate the total cost of ownership of a cloud infrastructure. In [11] the search is modeled as a multi-objective optimization problem to minimize the overall cost due to data storage, communication, and execution. In [12] different approaches are presented to compute the pricing in the context of offering REST APIs to multiple customers. Modeling pricing and scheduling aspects for edge devices can also establish sharing economy principles in edge and cloud computing [13].

Regarding scheduling, there are some approaches to the provisioning scenario which takes into account the scheduling restrictions for optimization issues using different techniques [14,15]. In [16] authors present a service management which takes into account the optimal trade-off between cost and QoS in the context of elasticity of highly variable workloads. Ran et al. apply a probabilistic model for determining the amount of the reserved instances to minimize the total cost while keeping QoS [17], as in our approach. Note that over-provisioned instances may lead to a low usage ratio and a greater cost, while a scarce reservation will have a poorer waiting time that leads to QoS degradation. Similarly, in [18] authors also get the optimum number for long-term reservation of resources in order to minimize provisioning costs.

7 Conclusions and Future Work

Provisioning plans are of utmost importance when trying to optimize computational resources required to fulfill some user needs during specific time periods. This article presents a solution to automatically derive provisioning plans from user needs specification including scheduling restrictions. After modeling user needs for a particular scenario, our prototype implementation searches for appropriate service configurations from different providers and generates corresponding provisioning plans, optimized both in terms of scheduling and purchasing options, for each provider. Then, our provisioning controller realizes the chosen plan, which is comprised of events that contain the necessary actions needed to fulfill user requirements. As future work, we plan to automatically crawl pricing and service configuration options from other cloud providers, to support multi-cloud provisioning plans, which may provide better performance and cost minimization in certain scenarios, as well as to analyze log files to improve the optimization and execution of existing provisioning plans.

Acknowledgments. Authors would like to thank Felipe Serafim and Daniel Arteaga for their support on the prototype implementation. This work has been partially supported by the EU Commission (FEDER), Spanish and Andalusian R&D&I programmes under grants TIN2015-70560-R, and P12-TIC-1867.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM **53**(4), 50–58 (2010)

2. Ma, R.T., Lui, J.C., Misra, V.: On the evolution of the internet economic ecosystem. In: Proceedings of the 22nd International Conference on World Wide Web, WWW 2013, pp. 849–860. ACM (2013)
3. García-Galán, J., Trinidad, P., Rana, O.F., Cortés, A.R.: Automated configuration support for infrastructure migration to the cloud. Future Generat. Comp. Syst. **55**, 200–212 (2016)
4. García-Galán, J., García, J.M., Trinidad, P., Fernandez, P.: Modelling and analysing highly-configurable services. In: Proceedings of the 21st International Systems and Software Product Line Conference, SPLC 2017. vol. A, pp. 114–122. ACM (2017)
5. Jensen, C.S., et al.: The consensus glossary of temporal database concepts — February 1998 version. In: Etzion, O., Jajodia, S., Sripada, S. (eds.) Temporal Databases: Research and Practice. LNCS, vol. 1399, pp. 367–405. Springer, Heidelberg (1998). doi:10.1007/BFb0053710
6. Lanthaler, M., Gütl, C.: On using JSON-LD to create evolvable RESTful services. In: Proceedings of the Third International Workshop on RESTful Design, WS-REST 2012, pp. 25–32. ACM (2012)
7. García, J.M., Fernandez, P., Pedrinaci, C., Resinas, M., Cardoso, J., Ruiz-Cortés, A.: Modeling service level agreements with linked USDL agreement. IEEE Trans. Serv. Comput. **10**(1), 52–65 (2017)
8. Al-Roomi, M., Al-Ebrahim, S., Buqrais, S., Ahmad, I.: Cloud computing pricing models: a survey. Int. J. Grid Distrib. Comput. **6**(5), 93–106 (2013)
9. Gamez-Diaz, A., Fernandez, P., Ruiz-Cortes, A.: An analysis of RESTful APIs offerings in the industry. In: Maximilien, M., et al. (eds.) ICSOC 2017. LNCS, vol. 10601, pp. 589–604. Springer, Cham (2017)
10. Li, X., Li, Y., Liu, T., Qiu, J., Wang, F.: The Method and tool of cost analysis for cloud computing. In: 2009 IEEE International Conference on Cloud Computing, pp. 93–100 (2009)
11. Wen, Z., Cala, J., Watson, P., Romanovsky, A.: Cost effective, reliable and secure workflow deployment over federated clouds. IEEE Trans. Serv. Comput. (2016). In press
12. Vukovic, M., Zeng, L.Z., Rajagopal, S.: Model for service license in API ecosystems. In: Franch, X., Ghose, A.K., Lewis, G.A., Bhiri, S. (eds.) ICSOC 2014. LNCS, vol. 8831, pp. 590–597. Springer, Heidelberg (2014). doi:10.1007/978-3-662-45391-9_51
13. García, J.M., Fernandez, P., Ruiz-Cortés, A., Dustdar, S., Toro, M.: Edge and cloud pricing for the sharing economy. IEEE Internet Comput. **21**(2), 78–84 (2017)
14. van den Bossche, R., Vanmechelen, K., Broeckhove, J.: Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads. In: 2010 IEEE International Conference on Cloud Computing, pp. 228–235 (2010)
15. Netjinda, N., Sirinaovakul, B., Achalakul, T.: Cost optimal scheduling in IaaS for dependent workload with particle swarm optimization. J. Supercomput. **68**(3), 1579–1603 (2014)
16. Björkqvist, M., Spicuglia, S., Chen, L., Binder, W.: QoS-aware service VM provisioning in clouds: experiences, models, and cost analysis. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 69–83. Springer, Heidelberg (2013). doi:10.1007/978-3-642-45005-1_6
17. Ran, Y., Yang, J., Zhang, S., Xi, H.: Dynamic IaaS computing resource provisioning strategy with QoS constraint. IEEE Trans. Serv. Comput. **10**(2), 190–202 (2017)
18. Hwang, R., Lee, C., Chen, Y., Zhang-Jian, D.: Cost optimization of elasticity cloud resource subscription policy. IEEE Trans. Serv. Comput. **7**(4), 561–574 (2014)