Smart LAMA API: Automated Capacity Analysis of Limitation-Aware Microservices Architectures*

Rafael Fresno–Aranda[0000−0001−8480−5014], Pablo Fernández[0000−0002−8763−0819], and Antonio Ruiz–Cortés[0000−0001−9827−1834]

Unidad de Excelencia Smart Computer Systems Research and Engineering (SCORE)
Instituto de Investigación en Ingeniería Informática (I3US)
Universidad de Sevilla, España
{rfresno, pablofm, aruiz}@us.es

Abstract. The use of microservices architectures (MSA) to develop web applications has experimented a considerable increase over the last year. In an MSA, developers tend to consume external services provided by third parties. These external services usually include a pricing that comprises different plans that impose capacity limits (aka limitations) for a given price. To name an MSA which consumes external APIs with limitations, we have coined the concept of Limitation-Aware Microservices Architecture (LAMA). In this situation, when developing a LAMA, it is necessary to analyse its capacity to know how many requests it is able to serve according to the subscribed plans. This is a tedious and error-prone activity, so its automation is very valuable.
In this paper we present Smart LAMA API, which is a public API that allows developers to analyse the capacity of any LAMA through three different endpoints, optimising the capacity, the cost or the time. From these endpoints, an unbounded number of other operations can be modelled.

Keywords: Automation · Capacity · Analysis · Microservices · API.

1 Introduction

The use of microservices architectures (MSA) [2] to develop web applications has experimented a considerable increase over the last years. In an MSA, more and more often, developers tend to consume external services provided by third parties, usually through public RESTful API with a series of endpoints. Commonly, external APIs include a *pricing* that comprises different *plans* that impose usage limitations for a given price, which is usually a monthly subscription. To name an MSA which consumes external APIs with limitations, we coin the concept

* This work has been partially supported by the European Commission (FEDER) and Junta de Andalucia under projects APOLO (US-1264651) and EKIPMENT-PLUS (P18-FR-2895), by the Spanish Government under project HORATIO (RTI2018-101204-B-C21), and by the FPU scholarship program, granted by the Spanish Ministry of Education and Vocational Training (FPU19/00666).

of **Limitation-Aware Microservices Architecture** (LAMA). Fig. 1 depicts a LAMA with 3 internal services (S1, S2 and S3) and 2 external APIs (E1 and E2). Edge labels represent the number of requests sent from a service to another each time the first one is invoked. E1 offers a *Basic* plan with a limitation of 15 requests per second (RPS), 1,000 requests per day (RPD) and a cost of $5 per month, and a *Premium* plan with a limitation of 25 RPS, 10,000 RPD and a cost of $8. E2 also has its own two plans with their corresponding limitations.

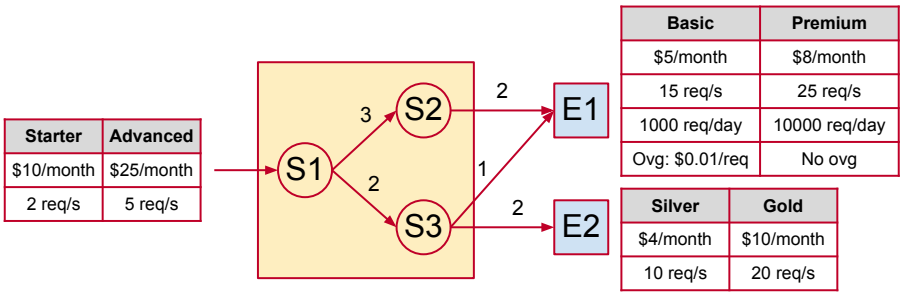

Fig. 1. A LAMA with three internal services and two external APIs. Each API has two pricing plans.

Developers who want to use the API need to subscribe to certain plans. In most cases, it is possible to subscribe to multiple plans. Furthermore, some APIs allow multiple subscriptions to a single plan. To identify each individual subscription, APIs usually provide an *API key*, that is sent within each HTTP request to identify the client that sends the request as well as its subscription, and thus apply the correct limitations.

When developing a LAMA, it is necessary to analyse its capacity to know how many requests can be served according to the subscribed plans. Furthermore, it is a fact that this analysis is tedious, time-consuming and error-prone, so its automation is very valuable.

In this paper we present **Smart LAMA API**, which is a public API that allows developers to analyse the capacity of any LAMA through three different endpoints, optimising the capacity, the cost or the time. From these endpoints, an unbounded number of other operations can be modelled. To validate the API, we provide an online Jupyter notebook [4]. The demo video is available at https://youtu.be/DE8da9JFxOk.

2 Smart LAMA API

2.1 Description of the LAMA

In order to invoke the analysis operations provided by the API, it is necessary to describe the LAMA by considering the internal services, the external APIs on

which it depends and their pricings. This description must be in **LAMA-DL**. A full example of LAMA-DL for the LAMA in Fig. 1 is included in our online notebook [4]. LAMA-DL allows every element of the LAMA to be specified. Due to lack of space, we will not explain LAMA-DL in this paper as most elements are self-explanatory.

2.2 API

Smart LAMA API [3] allows developers to register a LAMA into the system and then perform three different requests to analyse its capacity with any given values. The API transforms the LAMA into a Constraint Satisfaction and Optimisation Problem (CSOP) using the MiniZinc modelling language [1]. To manage LAMAs, the API offers the usual CRUD operations to create (POST), read (GET), update (PUT) and delete (DELETE) a LAMA. Our notebook [4] includes an example on how to create the LAMA in Fig. 1.

Once a LAMA has been registered, three different analysis operations can be performed. All of the following endpoints start with `/api/v2/lamas/<id>` to specify the LAMA under analysis and only support the GET method:

- `/operations/maxRequests`. This operation returns the maximum number of requests that can be sent to the LAMA without exceeding any external limitation. It supports various query parameters: `OpEx` (operational expenditures) can be used to indicate the maximum total cost that can be spent to subscribe to the external plans; `time` is used to specify the time window in which the operation must be calculated (in relation to seconds, e.g. a minute is specified as 60); and `K-<API>-<Plan>` can be used to indicate an exact amount of subscriptions for a specific plan (e.g. `K-E1-Basic=1`). Note that if no OpEx or subscriptions are specified, there will be no restriction to the total cost.
- `/operations/minCost`. This operation returns the minimum cost to serve a certain number of requests, indicated as `reqL` in the query, over a specific time window, indicated using the `time` query parameter.
- `/operations/minTime`. This operation returns the minimum time (in seconds) in which the LAMA is able to serve a certain number of requests, indicated as `reqL` in the query. It supports the `OpEx` and `K-<API>-<Plan>` parameters, which work exactly as described before.

Note that, by default, the API assumes that no overage requests can be used. To allow overage requests, all operations support the `useOvg` query parameter, which should be set to `true`. All of these operations return a JSON containing the result of the operation (a number) and the response of the MiniZinc model (a string containing its output, which includes the values of all variables).

3 Validation with Online Notebook

Our online Jupyter notebook [4] includes various functions that show how to analyse the capacity of the LAMA in Fig. 1. It contains wrappers to make it

easier to consume the API. The notebook lists 10 different analysis questions, grouped by their base operation (maxRequests, minCost or minTime). Some of these examples include:

- **Q2. What is the maximum number of requests that can be made in 1 minute for no more than $30?** This question can be solved through the endpoint `/operations/maxRequests?OpEx=30&time=60`.
- **Q4. What is the maximum quota per month for the Starter plan without the risk of exceeding the external limitations associated with its cost?** This is a slightly more involved question that requires some interpretation. We want to get a value for the quota of the Starter plan of the LAMA. Therefore, given that it has a cost of $10/month, we need to know the maximum number of requests per month (2592000 seconds) that can be served for no more than $10. Consequently, the endpoint `/operations/maxRequests?OpEx=10&time=2592000` gives the answer to this question.
- **Q6. What is the minimum cost to serve 35 requests in less than 5 minutes?**. The endpoint `/operations/minCost?reqL=35&time=300` provides a solution to this question.
- **Q9. What is the minimum time to serve 300 requests for less than 20 euros?** The endpoint `/operations/minTime?reqL=300&OpEx=20` gives a solution to this question.

4 Conclusions and Future Work

In this paper we introduced Smart LAMA API, a public API to analyse the capacity of a LAMA. It transforms a LAMA described using LAMA-DL into a CSOP using MiniZinc, and then provides answers to different analysis operations by interpreting them as standard operations available in a MiniZinc solver.

The current implementation has some limitations related to how MiniZinc handles variables and their domains. In the future, we want to change the CSOP solver being used by MiniZinc to overcome these issues, or even move to another CSOP modelling language. Either of these changes would be transparent to API users.

References

1. MiniZinc, https://www.minizinc.org/
2. MSA pattern, https://microservices.io/patterns/microservices.html
3. Smart LAMA API, https://smart-lama-api-beta.herokuapp.com
4. Smart LAMA API - JCIS Demo, https://deepnote.com/workspace/rafael-fresno-ab85b312-0a6a-4a0d-8975-786d40a9e336/project/Smart-LAMA-API-JCIS-Demo-39997c1c-f523-4abc-9b93-68ef1c8bf93e/%2Fnotebook.ipynb