Applying QoS in FaaS applications: a Software Product Line approach

Pablo Serrano-Gutierrez[1], Inmaculada Ayala[1,2], and Lidia Fuentes[1,2]

[1] Departamento de Lenguajes y Ciencias de la Computación,
Universidad de Málaga, España
pserrano@lcc.uma.es

[2] ITIS Software, Universidad de Málaga, España
{ayala,lff}@lcc.uma.es

Abstract

A FaaS system offers numerous advantages for the developer of microservices-based systems since they do not have to worry about the infrastructure that supports them or scaling and maintenance tasks. However, it is not easy to apply quality of service (QoS) policies in these kind of applications. The high number of functions that an application can have and its various implementations introduce a high variability that requires a mechanism to decide which functions are more appropriate to achieve specific goals. We propose a Software Product Line based approach that uses feature models that model the application's tasks and operations, considering the family of services derived from the multiple functions that can perform a specific procedure. Through an optimization process, the system obtains an optimal configuration that it will use to direct service requests to the most appropriate functions to meet certain QoS requirements.

1 Introduction

Functions as a service (FaaS) are a type of service based on cloud computing, which allows the development of systems based on functions that are managed by the platform itself, freeing the developer from the tasks of scaling and maintenance. In a FaaS approach, Software systems are developed using a set of independent functions that perform the tasks required by the application. These are serverless [2] functions that may behave as microservices or nano-services, but with the advantage of being fully managed.

Another great advantage of FaaS applications is the possibility of choosing different functions to perform the same task or operation without having to modify the application. These functions can be implemented by independent teams or based on different algorithms. Thus, deciding which available function is more suitable is essential because function performance at runtime can differ depending on the operational context. It is necessary to consider both functional and non-functional requirements since different implementations of a function may be decisive in the result of the execution of the application. Therefore, function selection is an essential concern in FaaS applications.

Although many frameworks are available to implement FaaS systems, they do not provide mechanisms to apply quality of service (QoS) policies beyond controlling certain function-level aspects, such as scaling. So it is the developer who must decide which functions are optimal to achieve the results required by the application and how to compose them. This task is arduous and not readily adaptable to changes in QoS requirements. We use cost and response time in this work, but we can apply this approach to other QoS parameters such as security or energy consumption.

Software Product Lines [8] is an approach for software development that focuses on the development of a family of software systems using reusable assets. To define the common and variable elements of these families of software systems, a widely used approach is Feature Models [7]. A feature model contains an explicit representation of the configuration space using features. A feature model organizes features into a tree and includes the corresponding tree and cross-tree constraints representing dependencies among features. In addition, it can consist of information about what features are optional or mandatory in a final system. A valid configuration is a particularization of the feature model that complies with the imposed constraints.

In this work, we will use feature models to model the composition of the tasks of a FaaS application. Our feature models include the alternative functions to perform the operations that made a task and their constraints. Using these models with the Z3 solver, we will obtain valid and optimal configurations of our FaaS application, taking into account QoS policies.

The main objectives of this work are:

1. Manage FaaS application QoS at runtime. At any time, the application could make a request to the system to adjust the QoS parameters pursued.

2. Generate the optimal configurations dynamically or, at least, those that meet restrictions. The system recalculates the configurations every time there is a change in the QoS objectives.

3. Decouple the serverless implementation from a specific serverless framework. The system is not integrated into any framework but communicates with it through REST requests. This type of interaction makes it possible to use our proposal with most existing frameworks. At the same time, this system does not require any extra learning from the developer, who can continue working similarly.

This paper is structured as follows: Section 2 introduces our approach; Section 3 presents our case study on Reservation systems; Section 4 discuses some related work and Section 5 presents some conclusions to the paper.

2 Our Approach

Our solution consists of a system (see Figure 1) that, based on FaaS, allows to choose at runtime the best functions that perform the operations needed by the application to offer a specific QoS. The proposal is based on the Software Product Line approach, using feature models to specify the variants of a particular service or task of a workflow. Our system performs this task as transparent as possible to the developer. So, there is no need to worry about knowing the different implementations of the functions and their characteristics.

Based on a feature model of the application, the proposed system calculates the optimal configuration of the tasks and functions that it is more convenient to execute to meet specific QoS. This optimization is carried out using the Z3 solver, which uses the model and the characteristics of each available function to carry out the different tasks necessary for the application.

The application workflow is made up of a series of tasks that are modeled in a feature model. At the same time, each task is performed through a certain number of operations necessary to complete it. These operations are the ones that we will execute through FaaS functions in our system. There may be different implementations of the functions that carry out each of these operations. Each function has specific associated characteristics, such as execution times, costs, security, etc. The system considers these characteristics for each task to generate a services feature model representing the family of functions associated with it. The application feature

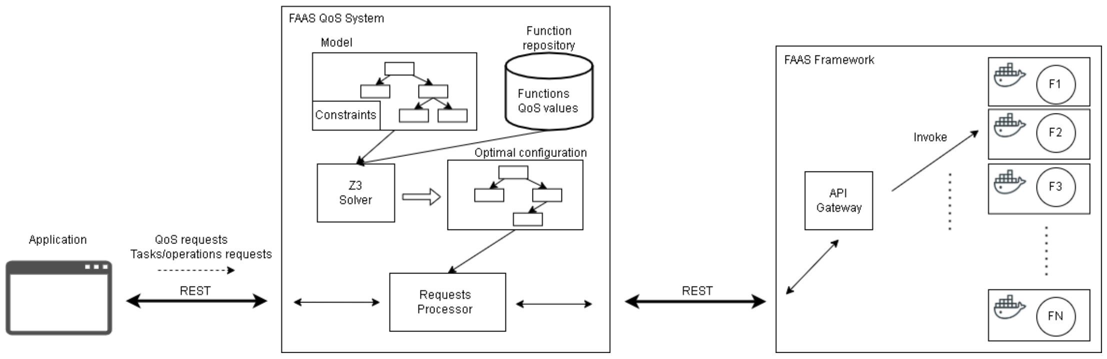

Figure 1: Proposed FaaS QoS system

model, along with the functions feature models of the tasks, results in a model that defines the valid functions configurations for the application. In this model, an optimization process can be carried out considering a set of restrictions related to the QoS to be achieved. Through this process, a valid configuration is obtained, whenever possible, that adjusts to the referred feature models, and that will be the one that allows the desired QoS to be achieved.

As part of our approach, a repository contains the list of functions that the application can use to perform the operations performed in a task. Each time a new implementation is made for one of these functions, it is only required to add the corresponding information to the repository. Then, this information is automatically integrated into the generated models.

A traditional FaaS application makes function calls through a framework used to implement FaaS. These frameworks use an element called API Gateway, through which they receive requests, sending them to the corresponding functions, which are placed in containers located somewhere in the cloud. The system delivers the execution results to the calling application through this same Gateway. Of course, these systems consist of other elements, such as an orchestrator, which is responsible for deploying and maintaining the containers. Communication with the API Gateway is often done through REST requests, which is why it has also been chosen as the communication mechanism for our system so that it is as similar as possible to working directly with the FaaS framework. The difference is we work with tasks and operations instead of specific functions. So, suppose we want to perform an operation that compress an image. In that case, we will use a generic name such as *Compress* instead of calling a function that performs the compression using a specific algorithm. Our system will be the one that makes the appropriate call to the FaaS framework so that a compressing function implemented through a particular algorithm is executed.

3 Illustrating case study

As a case study, we propose an application to make travel bookings. Our case study considers that a booking is made of three different elements. Firstly, the hotel reservation, which will be regarded as mandatory for all travel bookings, and, on the other hand, the transportation reservations (that are optional), which may be flight or train reservations. If there is a transportation reservation, the application could choose either of the two to complete the trip reservation. Therefore, In BMPN 2.0 notation, the workflow of the application (see Figure 1), which represents its functional requirements, will have two gateways. A first gateway indicates

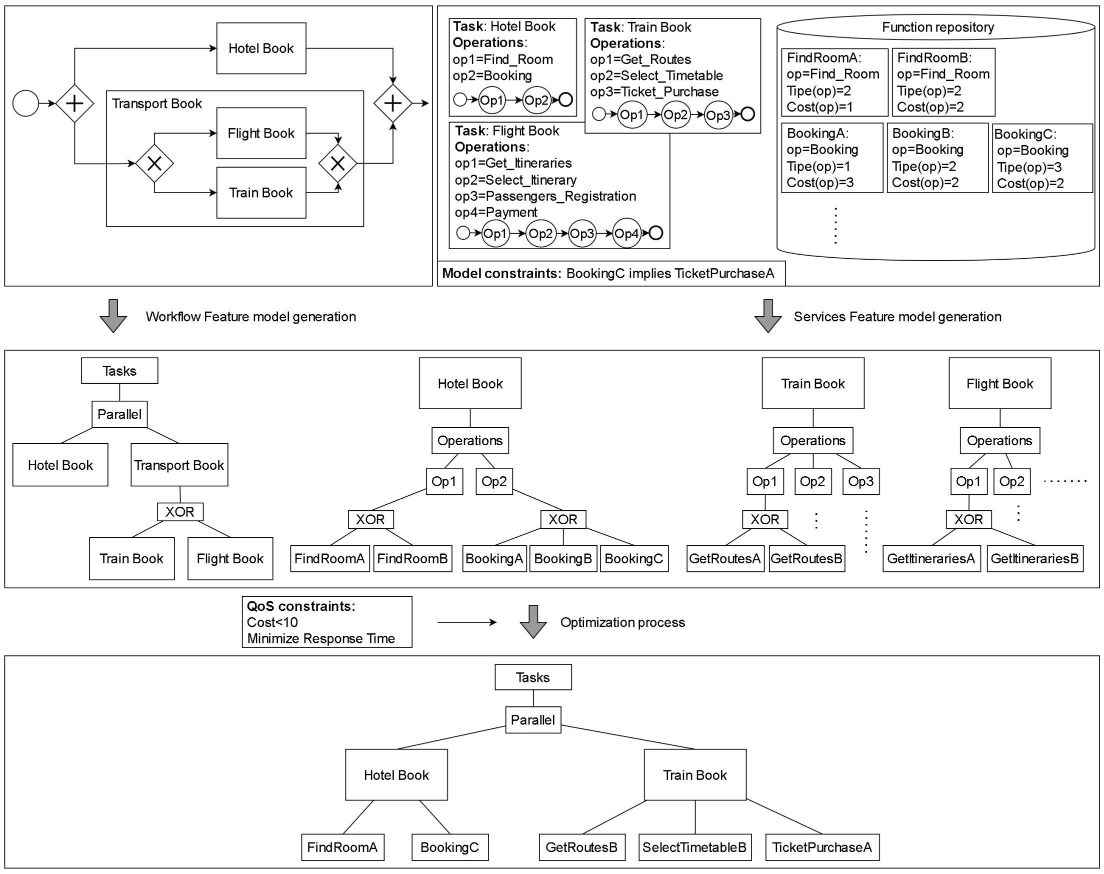

Figure 2: Feature model and configuration generation

that the booking application can perform hotel and transport reservation tasks in parallel. A second gateway shows the two possible alternatives in the transport branch, as shown in Figure 1. The correspondence of this workflow with the associated feature model is trivial. Each task is represented as a feature of the feature model. The parallel, exclusive and inclusive gateways will be modeled as feature relationships of type AND, XOR, and OR, respectively. Thus, AND will represent feature groups with two or more obligatory children, XOR will encompass feature groups with exclusive alternative children, and OR will be feature groups with alternative children. As we can see, the system is easily adaptable to the characteristics of the application. For example, if we decide to consider an alternative to the hotel for the accommodation, such as an apartment. Then, it would be enough to modify the model indicating that *Hotel Book* is not mandatory but optional and integrate it in an XOR branch together with the new *Apartment* Book task.

Each of these three tasks mentioned is performed by carrying out a series of operations, as can be seen in the task diagrams in Figure 1. In the case of hotel reservations, the best available price for a room is obtained, and, subsequently, the reservation is made. Likewise, *Train Book* and *Flight Book* must carry out a series of operations to complete the train and flight reservation, respectively. Each of these operations is performed by a FaaS function that corresponds with a function listed in the repository. As can be seen, there are alternative functions to perform each operation, and they are labeled with cost and execution time values. For example, in the case of the *Booking* operation, we find three alternatives, *BookingA*, *BookingB* and *BookingC*, in which option *BookingB* is the fastest but at the same time has the highest cost. However, *BookingA* is slower but has the lowest cost. We can model additional restrictions, for example, *BookingC implies TicketPurchaseA*, which means that if the reservation is made with the *BookingC* function, it is necessary to purchase the train ticket with the function *TicketPurchaseA*. In a real case, it can correspond to the condition of making two payments using the same banking service to obtain lower surcharges.

With the workflow and the information about the available tasks and functions, we model the feature model that appears in the central box of Figure 1. This feature model represents all the variability introduced by the tasks and the different implementations of the functions. This feature model is the main input of an optimization process that finds the best configuration of the FaaS application meeting restrictions related to the QoS. In the example, the aim is to minimize the response time but at the same time restrict the cost, imposing that it must be less than 10. It is necessary to consider that the response time will correspond to that of the slowest branch of the tree, while the cost refers to the set of all tasks performed. The Z3 solver works with the feature model and the restrictions mentioned above to obtain a valid configuration of functions that achieves the desired QoS. In this case, we get a combination of hotel and train reservations. We can see how the *BookingB* option has not been chosen despite being the fastest. In this case, it is because the cost of the system would then exceed ten units in our example.

Using this calculated configuration, our system is ready to receive requests from the application and to process them according to it. Thus, when it gets a *FindRoom* request, it will proceed to call the *FindRoomA* function through a request to the FaaS framework and return its result to the application.

4 Related Work

There are only a few works that consider QoS parameters in FaaS environments. Some of them focus on the performance of individual functions due to the performance of the framework itself [6]. Other articles, such as [5], consider the global performance of the application but, in this case, working at the FaaS platform level, controlling the location of resources. Also noteworthy is [9], in which Sheshadri K et al. present a system that allows specifying a QoS for a FaaS application and that has a QoS aware resource manager that intervenes in deployment decisions. It also works on resource requirements, trying to use them efficiently. Our system uses a complementary approach to these, working on the composition of application functions, so that any of these efficient resource allocation methods could be applied together.

On the other hand, numerous works can be found that use Software Product Lines to model the variability of services. In [10], the authors use feature models to recalculate service composition after a failure. Also, M. Abu-Matar et al. [1] model the variability of web services using Software Product Lines.

Likewise, numerous articles deal with the selection of services considering QoS. One of them is [3], which shows how to compose web services considering different quality attributes. In the area of microservices, [4] uses a model of the service workflow to select microservices utilizing an algorithm based on list scheduling. Nevertheless, unlike our proposal, they do not use a Software Product Line approach that models the variability due to multiple function implementations.

5 Conclusions

The system presented in this article allows applying QoS parameters to a FaaS application based on feature models that model the variability of systems with multiple implementations of functions that can be chosen to perform a certain operation. Our proposal selects the best available functions that allow achieving a QoS fulfilling a set of restrictions imposed by the user. This allows developers to make generic function requests, avoiding the need to know each of the implementations and their performance at the time of coding. Thanks to the use of a function repository, we can introduce new implementations of a certain function at runtime. It is also possible to change QoS requirements on-the-fly, automatically generating a new selection of functions. We plan to use this system to perform self-adaptive tasks based on changing conditions, as the infrastructure on which functions are deployed can also have a significant effect on QoS.

Acknowledgements

This work is supported by the European Union's H2020 research and innovation programme under grant agreement DAEMON 101017109, by the projects co-financed by FEDER funds LEIA UMA18-FEDERJA-15, MEDEA RTI2018-099213-B-I00 and Rhea P18-FR-1081, the PRE2019-087496 grant from the Ministerio de Ciencia e Innovación and by the project DISCO B1-2012_12 funded by Universidad de Málaga.

References

[1] Mohammad Abu-Matar and Hassan Gomaa. Variability modeling for service oriented product line architectures. In *2011 15th International Software Product Line Conference*, pages 110–119, 2011.

[2] Sarah Allen, Chris Aniszczyk, Chad Arimura, et al. Cncf serverless whitepaper, 2018.

[3] Danilo Ardagna and Barbara Pernici. Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering*, 33(6):369–384, 2007.

[4] Zhijun Ding, Sheng Wang, and Meiqin Pan. Qos-constrained service selection for networked microservices. *IEEE Access*, 8:39285–39299, 2020.

[5] MohammadReza HoseinyFarahabady, Young Choon Lee, Albert Y. Zomaya, and Zahir Tari. A qos-aware resource allocation controller for function as a service (faas) platform. In Michael Maximilien, Antonio Vallecillo, Jianmin Wang, and Marc Oriol, editors, *Service-Oriented Computing*, pages 241–255, Cham, 2017. Springer International Publishing.

[6] Anisha Kumari, B. Sahoo, Ranjan Kumar Behera, Sanjay Misra, and Mayank Mohan Sharma. Evaluation of integrated frameworks for optimizing qos in serverless computing. In *ICCSA*, 2021.

[7] Kwanwoo Lee, Kyo Kang, and Jaejoon Lee. Concepts and guidelines of feature modeling for product line software engineering. pages 62–77, 04 2002.

[8] Klaus Pohl, Günter Böckle, and Frank Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques.* 01 2005.

[9] Sheshadri K R and J Lakshmi. Qos aware faas platform. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 812–819, 2021.

[10] Jules White, Harrison Strowd, and Douglas Schmidt. Creating self-healing service compositions with feature models and microrebooting. *Int. J. Business Process Integration and Management Int. J. Business Process Integration and Management*, 1:0–0, 01 2009.