


Applying QoS in FaaS Applications: A Software Product Line Approach

Pablo Serrano-Gutierrez ✉ 

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Spain

Inmaculada Ayala ✉ 

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Spain

Lidia Fuentes ✉ 

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Spain

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, applying quality of service (QoS) policies in this kind of application is not easy. The high number of functions 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 optimisation process, the system obtains an optimal configuration that it will use to direct service requests to the most appropriate functions to meet specific QoS requirements.

2012 ACM Subject Classification Computer systems organization → Reconfigurable computing; Computer systems organization → Cloud computing

Keywords and phrases FaaS, Serverless, QoS, Software Product Line, Feature Model

Digital Object Identifier 10.4230/OASICS.Microservices.2020-2022.9

Funding 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, IRIS PID2021-122812OB-I00 and by the project DISCO B1-2012_12 funded by Universidad de Málaga.

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 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 modifying the application. These functions can be implemented by independent teams or based on various 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.



© Pablo Serrano-Gutierrez, Inmaculada Ayala, and Lidia Fuentes;
licensed under Creative Commons License CC-BY 4.0

Joint Post-proceedings of the Third and Fourth International Conference on Microservices (Microservices 2020/2022).

Editors: Gokila Dorai, Maurizio Gabbrielli, Giulio Manzonetto, Aomar Osmani, Marco Prandini, Gianluigi Zavattaro, and Olaf Zimmerman; Article No. 9; pp. 9:1–9:15



Open Access Series in Informatics

OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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 the developer 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 [10] is an approach for software development that focuses on developing 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 [9]. A feature model contains an explicit representation of the configuration space using features. A feature model organizes features into a tree, including the corresponding tree and cross-tree constraints representing feature dependencies. 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:

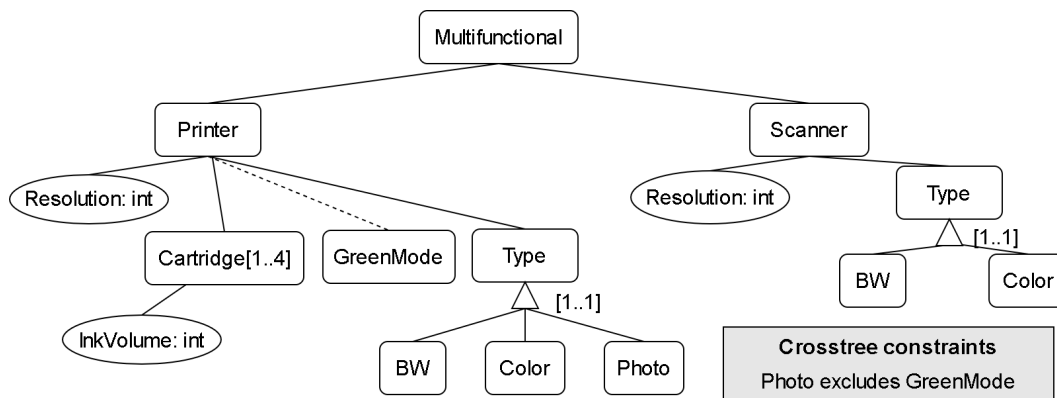
1. *Manage FaaS application QoS at runtime.* The application could request the system to adjust the QoS parameters pursued at any time.
2. *Generate the optimal configurations dynamically or, at least, those that meet restrictions.* The system recalculates the configurations every time there is a change 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: in Section 2 we present a brief introduction on feature modeling, Section 3 discusses some related work; Section 4 introduces our approach; Section 5 presents our case study on Reservation systems; Section 6 exposes the results to the experiments carried out; and Section 7 presents some conclusions to the paper.

2 Background on feature modeling

A feature model [9] is a hierarchical specification of systems through a set of features, which are elements or properties that may or may not be present in a concrete system. Features are organised in tree structures representing the dependencies between them, so selecting a leaf for the final system depends on selecting its ancestor features. In addition, it is possible to define explicit constraints, also known as cross-tree constraints. For example, if we consider a multi-functional system composed by a printer and a scanner (see Figure 1), a Printer with the feature *Photo* may not be compatible with *GreenMode*, in that case, we can add the cross-tree constraint *Photo excludes GreenMode*.

Features can be divided into Boolean features (e.g., *Printer*), numerical features (e.g., *resolution:int*), and variability sub-tree (e.g., *Cartridge[1..4]*). Boolean features represent yes/no decisions or features that can appear in the final system or not. Numerical features



■ **Figure 1** Feature Model generation.

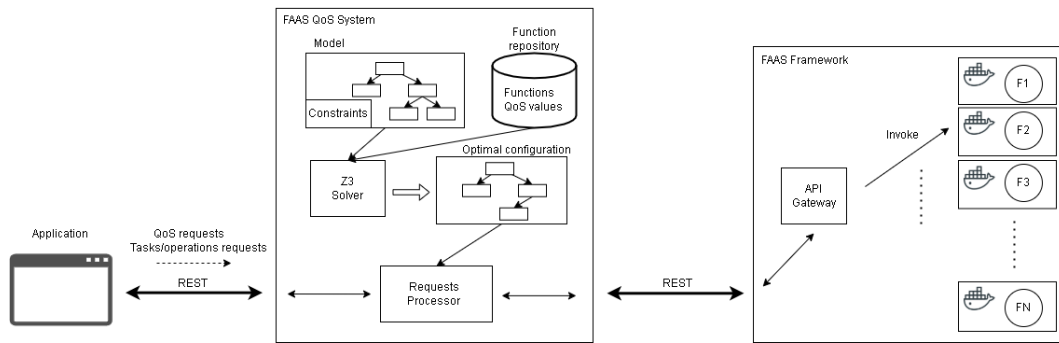
are features that require a value to be resolved. Variability sub-trees mean creating instances or clones and providing per-instance resolution for features in its sub-tree. Cardinality features or feature relations (e.g., *Type*) apply restrictions over the number of children of a feature that can be chosen. The appearance of a feature in a selection can be mandatory, linked using solid lines, or optional, linked using dashed lines (e.g., *GreenMode*). A feature model or product configuration is a selection of features that respect the tree and cross-tree constraints defined in the feature model. A configuration that complies with explicit and implicit restrictions of a feature model is considered valid.

3 Related work

There are not many jobs that consider QoS parameters in FaaS environments. Some of them focus on the performance of individual functions due to the work of the framework itself [8]. Other articles, such as [7], 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 [11], 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 efficient resource allocation methods could be applied together.

On the other hand, numerous works that use Software Product Lines to model the variability of services can be found. In [12], 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 dealing with the selection of services consider QoS. One is [4], which shows how to compose web services considering different quality attributes. In microservices, [6] 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.



■ **Figure 2** Proposed FaaS QoS system.

4 Our approach

Our solution consists of a system (see Figure 2) that, based on FaaS, allows us 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 transparently as possible to the developer. So, there is no need to worry about knowing the functions' different implementations and characteristics.

Based on a feature model of the application, the proposed system calculates the optimal selection of the tasks and functions that is more convenient to meet specific QoS. This optimisation 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.

4.1 The application model

Before optimising the feature set, our system needs to obtain a model of the application to work on. Initially, it is necessary to build a model of the application workflow. This is made up of a series of tasks that are modelled as features 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. Different implementations of the functions may carry out each of these operations. Each function has specific associated characteristics for each operation, such as execution times, costs, security, etc. As a second step, we use this information to automatically generate a collection of service feature models representing the variability of each executing task. In addition, our system takes into account another source of variability. Some functions may have parameters that vary their behaviour, sometimes affecting in some way the operation QoS. These parameters are specified as attributes associated with the corresponding feature. For example, a video playback function may operate with different resolution values and several possible frames per second. For this reason, in the third step, we will automatically generate function feature models, incorporating these new features derived from appropriate options that can modify the function's behaviour in terms of QoS.

Therefore, at this point, we have a *feature model of the application*, a *set of service feature models* related to the tasks, and a *set of function feature models* associated with each serverless function. These models constitute the global application model that our system will use.

4.2 Analysis system of the application model

Based on the feature models obtained, our system performs an analysis to generate an optimal configuration. To accomplish this task, we use a function repository that contains information about the QoS values of the different configurations of the FaaS functions, each implemented by a set of operations. Finally, an optimisation process can be carried out considering a set of restrictions related to the QoS to be achieved. A valid configuration that adjusts to the referred feature models is obtained through this process whenever possible. This configuration will be the one that allows the desired QoS to be achieved and will enable the system to choose the tasks and functions to be executed.

4.3 Valid Configuration Generation

With the models obtained in the steps described above, and applying Software Product Lines refactoring techniques [3], we generate a Feature Model with the information obtained from the entire application. To determine the valid configurations, we use a mathematical model built from the application's Feature Model and add existing and user-specified restrictions. The conversion of the features to this model is done automatically. The main component of this conversion is a recursive process which traverses a tree that represents the relationships contained in the feature models and obtains all the logical connections between features. When a node is reached, the set is assigned the AND, OR, or XOR logical relationship indicated in the parent node. The process ends with obtaining a logical expression that represents the entire tree. To denote it, the following expression will be used:

$$L(t_1, \dots, t_{|T|}) \quad t_i \in Bool \quad (1)$$

Where T is the set of tasks and t_i is a variable that identifies the task i , which is a Boolean value that indicates if the task is executed or not. L represents a function that relates these t_i through the logical expression that represents the task tree of the application, obtained through the previous procedure. Next, we must add the necessary equations to model the variability of the operations, taking into account that each can be carried out through a set of different functions. The following equation expresses this:

$$\sum_{i=1}^n f_{ijk} = 1 \quad \forall k \in [1, |T|], \forall j \in [1, |O_k|], f_{ijk} \in \{0, 1\} \quad (2)$$

Where f_{ijk} is associated with the serverless function F_{ijk} that implements some of the operations used by some of the tasks that are part of the application. It can take a value of 0 or 1 that will indicate, respectively, if that function is part of the configuration or not. On the other hand, O_k is the set of operations that make up a given task t_k . Therefore, $\{f_{1jk}, \dots, f_{njk}\}$ represents the set of n functions that can perform the same operation o_j related to the task t_k .

The equations (1) and (2) are used to identify a configuration. If, in addition, we add the constraints defined on the tasks or the functions, we will obtain a set of valid configurations. We will call CT this set of constraints, which will be represented as logical functions ct_i that relate tasks, functions and parameters, as defined in the following equation:

$$ct_i(T, F, P, V) \quad \forall i \in [1, |CT|] \quad (3)$$

Where F is the set of functions, P is the set of function parameters, and V is the set of values that the different parameters can take.

4.4 Optimisation Process

We use the equations defined in the previous section and the model constraints to carry out the optimisation process. We add to these equations another set of equations to associate the parameters related to functions' QoS with those of the complete system. These equations differ depending on the parameter considered since execution time differs from cost or security. In some cases, it is a matter of adding all the functions' values. In some cases, we must consider the tasks performed; in others, the values greater or lesser must be considered, etc. Finally, the constraints specified by the user are added. To obtain a valid configuration, the system applies the optimisation function provided by Z3 using the complete set of equations obtained. This configuration is optimal according to the specified parameters and complies with the user restrictions.

Our system can optimise using one or several QoS parameters: latency, execution time, security, cost and energy. We can easily extend our approach for new parameters by generating a specific set of equations. Some functions may have different associated values for the same QoS attribute. In this scenario, it is necessary to consider the feature model to calculate the associated QoS values. The set of equations to optimise cost or energy are the following:

$$\begin{aligned}
 Min. : & \sum_{k=1}^{|T|} \sum_{j=1}^{|O_k|} \sum_{i=1}^n t_k \cdot f_{ijk} \cdot c_{ijk} \\
 s.t. : & L(t_1, \dots, t_{|T|}) \quad t_i \in Bool \\
 & \sum_{i=1}^n f_{ijk} = 1 \quad \forall k \in [1, |T|], \forall j \in [1, |O_k|] \\
 & f_{ijk} \in \{0, 1\} \\
 & ct_i(T, F, P, V) \quad \forall i \in [1, |CT|] \\
 & \sum_{m=1}^{nv} fp_{jklm} = 1 \quad \forall k \in [1, |T|], \forall j \in [1, |O_k|], \\
 & \quad \forall l \in [1, |P_{jk}|] \\
 & fp_{jklm} \in \{0, 1\} \\
 & pv_{jkl} = Param(p_{jkl}, m) \forall p_{jkl} \in P_{jk}, \forall fp_{jklm} = 1 \\
 & c_{ijk} = Cost(F_{ijk}, \{pv_{jk1}, \dots, pv_{jkn}\}) \forall F_{ijk} \in F
 \end{aligned} \tag{4}$$

Where P_{jk} represents the set of l parameters of the operation j related to task k , and fp_{jklm} is a variable that indicates the value assigned to the parameter p_{jkl} from a set of nv possible values, it can take values 1 or 0, meaning that the value m is selected or not, respectively. For example, considering a rendering function whose result depends on the output resolution, which can take three possible values, such as $480px$, $640px$ and $800px$, and that *Render* is the first operation of the second task, fp_{1212} indicates whether the value $640px$ is chosen or not for its first parameter *Resolution*.

Cost is a function that returns the cost of the function F_{ijk} considering a set of values $\{pv_{jk1}, \dots, pv_{jkn}\}$ assigned to its parameters. $Param(p, i)$ is a function that returns the value for the parameter p that is determined by the index i from the list of possible values for this parameter. So, following with the last example, $Param(P_{121}, 1) = 640px$, and $Cost(RenderA, 640px, 1)$ will return the cost of the operation $Cost(Render)$ with $640px$ as the value for the input parameter, if the function *RenderA* is used to perform that operation. As mentioned before, this could be equally used for energy. In that case, the function *Cost* would be *Energy* and would return the Energy consumed by the execution of the function F_{ijk} .

If we want to optimise execution time or latency, the considered equations are:

$$\begin{aligned}
Min. : \quad & \max(\{s | s = t_k \cdot f_{ijk} \cdot s_{ijk} \forall k \in [1, |T|], \\
& \quad \forall j \in [1, |O_k|], \forall i \in [1, |F_j|]\}) \\
s.t. : \quad & L(t_1, \dots, t_{|T|}) \quad t_i \in Bool \\
& \sum_{i=1}^n f_{ijk} = 1 \quad \forall k \in [1, |T|], \forall j \in [1, |O_k|] \\
& f_{ijk} \in \{0, 1\} \\
& ct_i(T, F, P, V) \quad \forall i \in [1, |CT|] \\
& \sum_{m=1}^{nv} fp_{jklm} = 1 \quad \forall k \in [1, |T|], \forall j \in [1, |O_k|], \\
& \quad \forall l \in [1, |P_{jk}|] \\
& fp_{jklm} \in \{0, 1\} \\
& pv_{jkl} = Param(p_{jkl}, m) \forall p_{jkl} \in P_{jk}, \forall fp_{jklm} = 1 \\
& t_{ijk} = Time(F_{ijk}, \{pv_{jk1}, \dots, pv_{jkn}\}) \forall f_{ijk} \in F
\end{aligned} \tag{5}$$

Where *Time* is a function that returns the execution time of an operation performed by a determinate function. Since these values are used comparatively, the best solution is reached when mean values are considered. In the same way as before, we can substitute the *Time* function for *Latency* to optimise this parameter.

On the other hand, if we want to optimise some measurable aspect related to security, we will have:

$$\begin{aligned}
Max. : \quad & \min(\{s | s = t_k \cdot f_{ijk} \cdot s_{ijk} \forall k \in [1, |T|], \\
& \quad \forall j \in [1, |O_k|], \forall i \in [1, |F_j|]\}) \\
s.t. : \quad & L(t_1, \dots, t_{|T|}) \quad t_i \in Bool \\
& \sum_{i=1}^n f_{ijk} = 1 \quad \forall k \in [1, |T|], \forall j \in [1, |O_k|] \\
& f_{ijk} \in \{0, 1\} \\
& ct_i(T, F, P, V) \quad \forall i \in [1, |CT|] \\
& \sum_{m=1}^{nv} fp_{jklm} = 1 \quad \forall k \in [1, |T|], \forall j \in [1, |O_k|], \\
& \quad \forall l \in [1, |P_{jk}|] \\
& fp_{jklm} \in \{0, 1\} \\
& pv_{jkl} = Param(p_{jkl}, m) \forall p_{jkl} \in P_{jk}, \forall fp_{jklm} = 1 \\
& s_{ijk} = Secur(F_{ijk}, \{pv_{jk1}, \dots, pv_{jkn}\}) \forall f_{ijk} \in F
\end{aligned} \tag{6}$$

Where *Secur* is a function that returns the security level of the aspect considered, provided by a serverless function. Each implementation is assigned a value that will be higher the more secure the function is considered. For example, if we have a function to login and it does not use SSL, the security level could be 1, which would correspond to a low level; in case of using it, we could assign a value of 2, and if, in addition, it uses two-step verification, a value of 3 could be considered, corresponding to a high level of security. In this case, if the system uses an insecure function, the whole system became insecure, so in this optimisation process we will try to use the highest secure level of the aspect considered, for all the functions.

4.5 Function selection

As stated in the introduction, a FaaS application is built based on numerous independent functions handled by a framework. These serverless functions are deployed in containers and called by the application when needed, i.e., they behave as self-managed stateless microservices. These functions, being elements external to the application, can be altered without intervening in its functional behaviour. Therefore, it is possible to change the implementation of a function without modifying or rebuilding the application. So, it is common to have several function implementations that can perform the same operation.

We have implemented our function selection component using REST requests to use this system with different serverless frameworks. REST makes possible a natural integration of this component in FaaS platforms because they are commonly used to call serverless functions. Specifically, these frameworks use a service called API Gateway. Using this service, they receive REST requests of functions from the FaaS application and send them to the implementations of the corresponding functions deployed in containers in the cloud. Our system, programmed in Python, attends to calls made like they would be made to a FaaS framework.

The system receives a generic request and transforms it into a specific request for a function managed by the FaaS framework. For example, suppose we want to perform an operation that compresses an image. In that case, we use a generic name like *Compress* in the code instead of calling a specific function that compresses the image using a particular algorithm. Our system will transform this request into the corresponding call to the FaaS framework so that a compression function implemented using a specific algorithm is executed.

4.6 Requests processing

The system receives the requests from the application, both for functions and adjustment of QoS parameters. When the request is for the execution of a function, in the first step, it processes, if necessary, the input parameters to the function following the configuration obtained by the analysis module and reconstructs the request. The application can avoid assigning these optimal parameters to the function by passing the desired values in the request. In the next step, the processed request is passed directly to the function selector, which will redirect the request to the appropriate implementation deployed by the FaaS framework. On the other hand, the platform admits the entry of requests related to the QoS to which it is desired to adjust the operation of the application. To distinguish feature requests from QoS adjustment requests, the system listens on two different ports that can be previously configured. These requests also follow the REST style and are confirmed in the same way. Once the function is executed, the response of the function is passed directly to the application in response to the execution request. This also occurs in the form of a REST request.

We have considered two possible modes of operation to meet the needs of different types of applications. On the one hand, it is possible to work entirely transparently. In this case, the application that uses our platform must only follow the process described, and it makes QoS requests when needed, and every time it has to perform an operation, it makes a request for a said operation to the system. The developer, therefore, programs the application in the same way as if we were working directly on a FaaS framework. This way, it would also apply to already-built FaaS applications.

On the other hand, it is possible to work interactively. In this mode, the application can query the system for information on the optimal configuration. The operation is the same as we have described, with the addition of requests related to some of the tasks or operations.

Thus, if the application asks for the task T , it will be able to know if it is included in the optimal configuration calculated by the analysis module. In this case, the application does not work like a traditional FaaS application since it must interact with the platform.

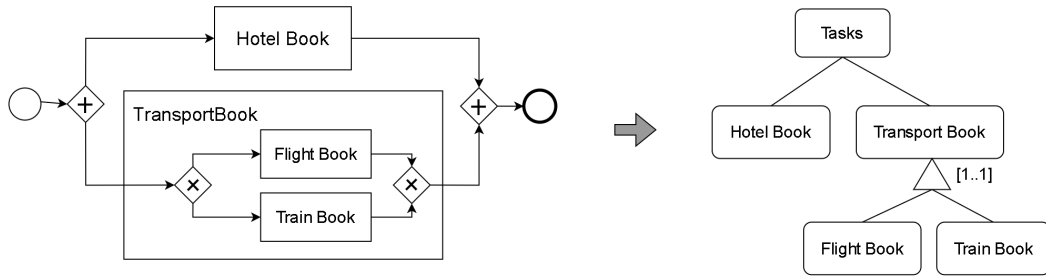
As part of our approach, a repository contains the list of functions the application can use to perform the operations performed in a task. Each time a new implementation is made for one of these functions, it is only required to add the corresponding information to the repository. Then, this information 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 API Gateway element, through which they receive requests, sending them to the corresponding functions, which are placed in containers 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 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 compresses 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.

5 Case Study

To illustrate our proposal, we use an application to make travel bookings. The system 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 two possible 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 BPMN 2.0 notation, the application workflow (see Figure 3), which represents its functional requirements, will have two gateways. A first gateway indicates 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 3. 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 modelled 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. For example, for the tasks in our case study, we would get: *Hotel_Book AND (Flight_Book XOR Train_Book)*. As we can see, the system is easily adaptable to the characteristics of the application. For example, if we 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 is performed by carrying out a series of operations, as seen in the task diagrams in Figure 4. 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,



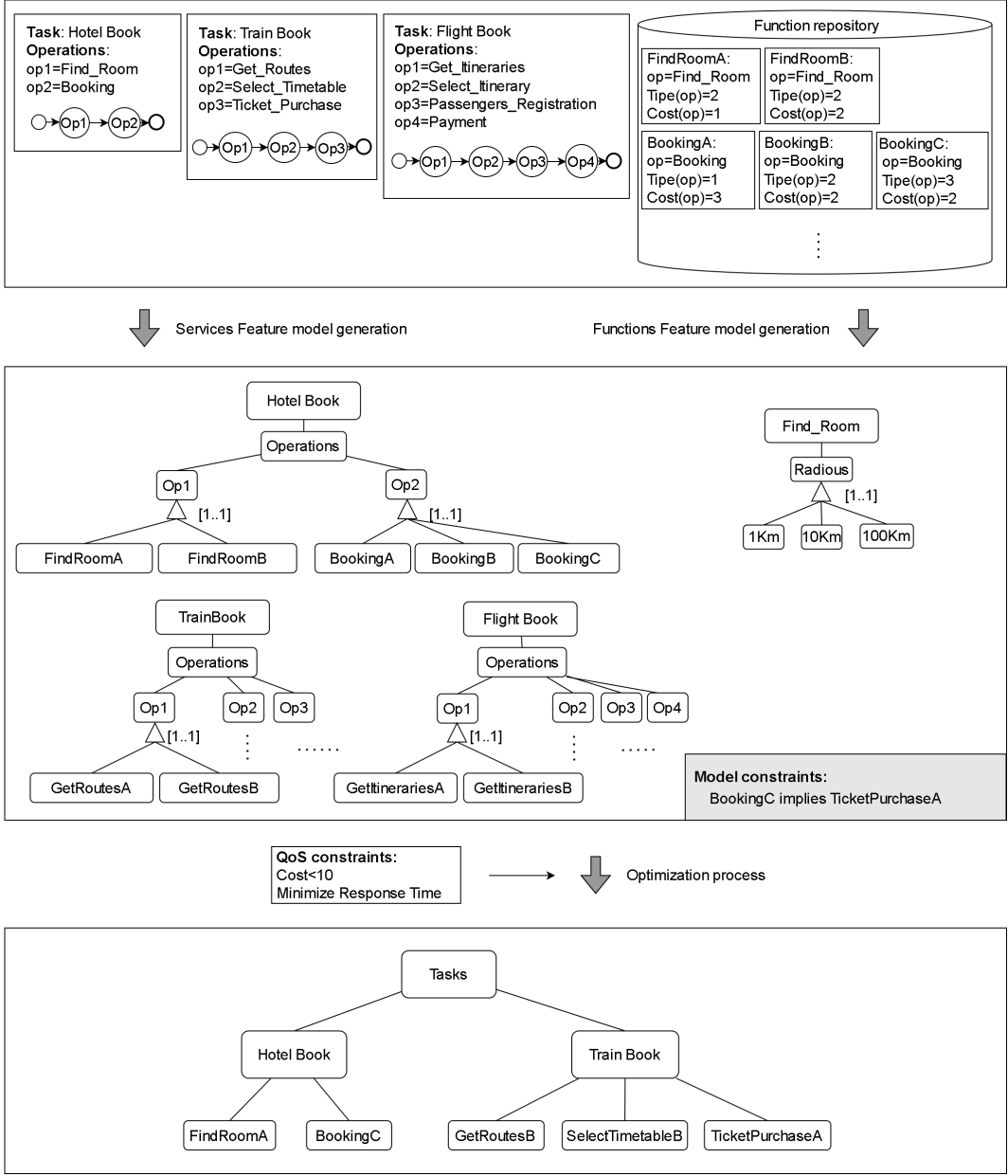
■ **Figure 3** Workflow of the application and Tasks Feature model.

respectively. Each of these operations is performed by a FaaS function corresponding to a function listed in the repository. There are alternative functions to perform each operation, and they are labelled with cost and execution time values. In order to compare results, we have measured these values in the same conditions and stored them in the repository. As can be seen in this example, the system can support the use of additional factors to perform the optimisation, since it is enough to include the values associated with each function in the repository. Thus, in this case, we not only take into account the performance of the functions but we add the cost of the reservation as a factor to take into account. 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 price. 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. For our case study, these constraints are those shown in the upper box of Figure 4.

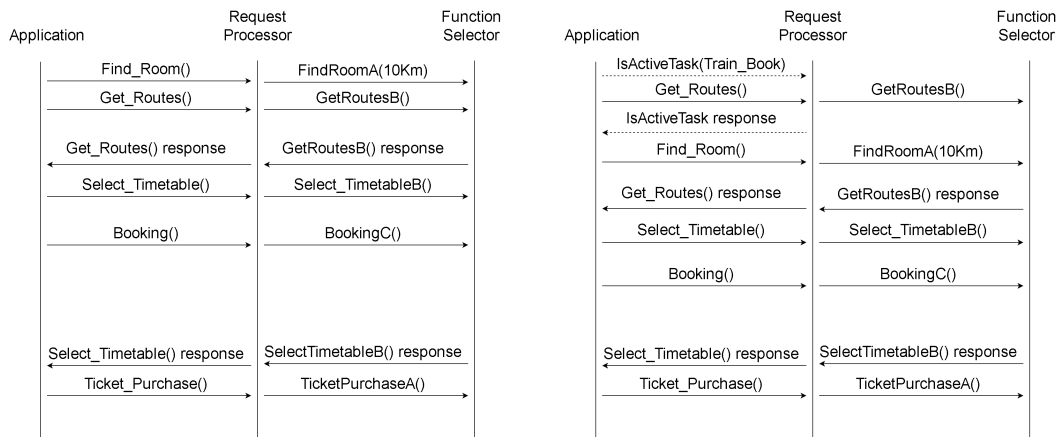
With the available tasks and functions information, we model the feature models that appear in the central box of Figure 4. These feature models represent all the variability introduced by the tasks and the different implementations of the functions. These feature models are the primary input of an optimisation process that finds the best configuration of the FaaS application meeting restrictions related to the QoS. In the example, the aim is to minimise the response time while restricting 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 above mentioned restrictions to obtain a valid configuration of functions that achieves the desired QoS. In this case, we get a combination of hotel and train reservations. Despite being the fastest, we can see how the *BookingB* option has not been chosen. 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 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.

To illustrate the difference between the two modes of operation (see Figure 5) described in section 4.6, we can think of a trip reservation application in which a hotel and transport, which can be a train or plane, must be reserved, the objective of which is to complete a reservation with the lowest possible cost. If the user chooses hotel and train, the system will



■ **Figure 4** Feature model and configuration generation.



■ **Figure 5** Sequence diagrams of the interaction with the request processor in transparent mode (left) and interactive mode (right).

generate an optimal configuration for that combination. However, we can also consider a different behaviour where the user does not select anything. In that case, the system finds the best alternative to reduce the cost as much as possible: hotel and train or hotel and plane combinations. Then, the application must know which of the two transport-related tasks it should perform, the interactive mode described above is then necessary.

6 Experimental results

A series of experiments have been carried out to evaluate the performance of our platform. In addition to the case study, larger workflows and many implemented functions have been considered. The system is programmed in Python v.3.10.4 with Z3 v.4.8.15, and the hardware used is a PC Intel i5-7400, 3.00 GHz, 24 GiB of RAM.

6.1 Case study

Our case study comprises three tasks with two, three and four operations. The number of functions considered is 20, that is, between two and three for each operation. We have carried out 10,000 executions to obtain a reliable average value of the different time measurements. The results show the system is fast for this simple case, getting times less than two milliseconds.

We have considered three aspects to evaluate the performance of the designed system. Firstly, the processing of information on the different QoS aspects related to the functions extracted from the information in the functions repository. Secondly, the generation of the feature model from the workflow and the QoS data calculated in the previous step. And finally, the duration of the optimisation process carried out by Z3. Running our case study resulted in a mean total time of 1.89 milliseconds, a mean QoS data processing time of 0.91 milliseconds, a mean modelling time of 0.45 milliseconds, and a mean optimisation time of only 0.47 milliseconds.

6.2 General performance and scalability

To carry out a study of scalability, it is necessary to ask first what are the acceptable values for the parameters that influence the model's variability. This is important because the five levels that can be present in our system mean that a slight increment in these parameters supposes a substantial increment in the total number of possible configurations, which can lead to an exponential growth in complexity.

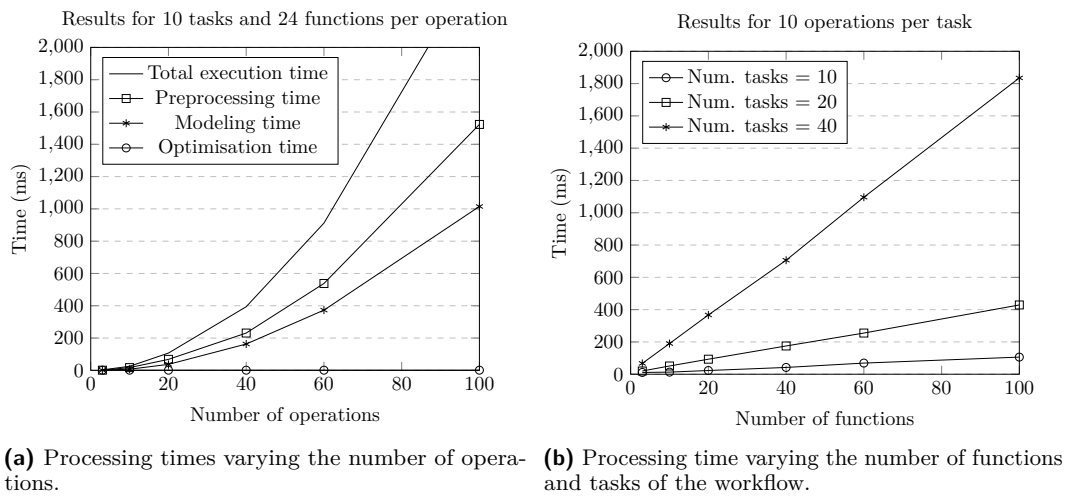
The first level to consider is the number of tasks. However, this will usually not be a very high number, usually less than ten [5] due to the high granularity of these systems, which means that the weight falls mainly on the number of operations performed by the tasks since they are the ones associated with the functions that are executed. At the same time, each operation can be performed by a different function implementation. Also, if a function can admit other parameters, they must be considered. From the point of view of analysis, to prepare a study that does not depend on so many variables and, therefore, can be more easily represented, we will include the effect of the parameters together with the multiplicity of functions. If an operation admits two configurable parameters and each of them can have three possible values, an effect similar to that of multiplying the variability by a factor of 6 can be considered, so if, for example, we have four possible implementations per operation, it would be somewhat comparable to having 24 implementations. This allows us to compare the effects of the diversity of implementations with those of the diversity of parameters and values supporting such functions in the same graph. Therefore, we have included this number when making the measurements, even though we will not usually find such a large number of implementations that perform the same operation. We must consider that the number of implementations of the same function will not usually be significant. An average of 3 could be a reasonable value, considering that not all the operations will have different implementations of functions to perform them.

The tests are performed considering the worst case. It occurs when there are no defined user restrictions or constraints. This is because constraints reduce the variability tree (i.e., the number of possible configurations of the variability model), and complexity decreases. For this test, a value of 100 executions has been chosen to obtain the average values.

As we can observe in the graphic in Figure 6a, total execution times remain low even considering unrealistic values of the parameters that have been adjusted. Although it is observed that the growth of the times is exponential, even in a case with ten tasks, 100 operations per task and 24 functions per operation, the measured time barely exceeds two seconds. For the most common cases, the times are reduced to tens of milliseconds, which is quite a good performance.

Observing each of the times of the different processes that are carried out, we see that the behaviour is similar except in the case of optimisation, in which very similar values of a few milliseconds are obtained. Therefore, the preprocessing and modelling processes are the most time-consuming, with preprocessing slightly above. However, as mentioned, the total times are very short for real cases.

On the other hand, tests have been conducted considering ten operations per task to appreciate the effect of varying the number of functions and tasks. As shown in Figure 6b, the behaviour is practically linear when the number of functions is in this range. If we increase the number of tasks, total times also increase exponentially. However, even with 40 tasks, results remain contained and are lower than two seconds. Again, we see how the measured times are a few tens of milliseconds for more realistic values.



■ **Figure 6** Scalability tests results.

7 Conclusions

In this paper, we have presented a system that allows applying QoS parameters to a FaaS application. The proposed system uses feature models to model the variability of systems with multiple implementations of functions that can be chosen to perform a specific operation. Our proposal selects the best available functions to achieve a QoS, fulfilling a set of restrictions the user imposes. This will enable developers to make generic function requests, avoiding the need to know each of the implementations and their performance during coding. Thanks to a function repository, we can introduce new implementations of a particular function at runtime. Changing QoS requirements on the fly is also possible by automatically generating a new selection of functions. We plan to use this system to perform self-adaptive tasks based on changing conditions and consider the influence of other parameters external to the application, such as the infrastructure on which functions are deployed, that can also significantly affect QoS.

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. doi:10.1109/SPLC.2011.26.
- 2 Sarah Allen, Chris Aniszczyk, Chad Arimura, et al. Cncf serverless whitepaper, 2018.
- 3 Vander Alves, Rohit Gheyi, Tiago Massoni, Uirá Kulesza, Paulo Borba, and Carlos Lucena. Refactoring product lines. In *Proceedings of the 5th International Conference on Generative Programming and Component Engineering*, GPCE '06, pages 201–210, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1173706.1173737.
- 4 Danilo Ardagna and Barbara Pernici. Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering*, 33(6):369–384, 2007. doi:10.1109/TSE.2007.1011.
- 5 A. Dietzsch. Ratios to support the exploration of business process models in business process management, 2003.
- 6 Zhijun Ding, Sheng Wang, and Meiqin Pan. Qos-constrained service selection for networked microservices. *IEEE Access*, 8:39285–39299, 2020. doi:10.1109/ACCESS.2020.2974188.

- 7 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. doi:10.1007/978-3-319-69035-3_17.
- 8 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. doi:10.1007/978-3-030-87007-2_20.
- 9 Kwanwoo Lee, Kyo C. Kang, and Jaejoon Lee. Concepts and guidelines of feature modeling for product line software engineering. In Cristina Gacek, editor, *Software Reuse: Methods, Techniques, and Tools*, pages 62–77, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. doi:10.1007/3-540-46020-9_5.
- 10 Klaus Pohl, Günter Böckle, and Frank Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, Berlin, Heidelberg, jan 2005. doi:10.1007/3-540-28901-1.
- 11 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. doi:10.1109/CCGRID51090.2021.00099.
- 12 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, jan 2009. doi:10.1504/IJBPIIM.2009.026984.