A Branch-and-Bound Algorithm for Autonomic Adaptation of Multi-Cloud Applications

André Almeida\textsuperscript{1,2}, Francisco Dantas\textsuperscript{3}, Everton Cavalcante\textsuperscript{2}, Thais Batista\textsuperscript{2}
\textsuperscript{1}Federal Institute of Education, Science and Technology of Rio Grande do Norte, Parnamirim, Brazil
\textsuperscript{2}Federal University of Rio Grande do Norte, Natal, Brazil
\textsuperscript{3}State University of Rio Grande do Norte, Natal, Brazil
e-mail: andre.almeida@ifrn.edu.br, franciscodantas@uern.br, evertonrsc@ppgsc.ufrn.br, thais@ufrnet.br

Abstract—Adaptation is an important concern in cloud-based applications composed of services provided by different cloud providers since cloud services can suffer from Quality of Services (QoS) fluctuations. Other conditions that can also trigger an adaptation process at runtime are the unavailability of services or the violation of user-defined policies. Moreover, the detection and reaction on such changes must be done in an autonomic way, without the need of user intervention. This paper presents a dynamic adaptation approach for multi-cloud applications supported by a Branch-and-Bound (B&B) algorithm in order to optimize the adaptation process itself when selecting the services to be deployed within the application. Computational experiments comparing the B&B algorithm with another algorithm that evaluates all possible configurations may be prohibitive as it hampers the scalability of the adaptation process, which can deal with large configurations of multi-cloud applications composed by a plethora of cloud services.

Keywords-Multi-Cloud Applications; Dynamic Adaptation; Optimization; Branch-and-Bound Algorithm; Scalability

I. INTRODUCTION

With the availability of a variety of Cloud Computing platforms, a new type of application composed by multiple services of distinct third-party cloud platforms is emerging. Dynamic adaptation is an important concern in this scenario, as the cloud services can suffer from instability and Quality of Services (QoS) fluctuations. Thus, it is necessary to adapt the application upon the detection of QoS violations that affect the application requirements.

When dynamically adapting a multi-cloud application, it is necessary to select which cloud platforms/services must be used according to user-defined requirements and/or constraints. However, in a heterogeneous cloud scenario with several options of platforms and services that can be used by an application, evaluating all possible configurations may be prohibitive as it hampers the scalability of the adaptation process itself. Therefore, an optimized approach for selecting the suitable application configurations for adaptation is highly desirable.

In this paper we tackle the above problem and present an autonomic adaptation process for multi-cloud applications that relies on a Branch-and-Bound (B&B) algorithm to optimize the adaptation process itself when selecting the services to be deployed within the application. The role of this algorithm is to select the best configuration when adapting a multi-cloud application by choosing the services that properly fit the quality properties specified by the application. Although we describe the whole adaptation process, in this paper we focus on presenting the B&B algorithm in the context of the dynamic process. We also present the results of computational experiments performed in order to assess its performance and to evaluate if it scales in case of applications with multiple alternatives services to be chosen during adaptation.

Our adaptation process encompasses two main phases. The first one is a modeling phase, performed at design time, in which the application is modeled as a set of services (including cloud services) and a set or requirements for each service. This modeling phase is supported by a customized feature model borrowed from the Software Product Line (SPL) development paradigm. The second phase is an autonomic adaptation process that is responsible for the dynamic adaptation of the application at runtime by changing services when the stated requirements are violated.

This paper is structured as follows. Section II presents the basic concepts of multi-cloud applications and B&B algorithms. Section III describes our autonomic approach for the dynamic adaptation of multi-cloud applications and the application that we use as an example in our experiments. Section IV presents our optimized adaptation approach by detailing the customization of the B&B algorithm to the multi-cloud scenario. Section V presents the experiments comparing the new approach with a previous version with an exhaustive search algorithm. Section VI contains related work. Section VII presents final remarks.

II. BACKGROUND

A. Multi-Cloud Applications

Cloud Computing is a paradigm that enables access to a shared pool of configurable computing resources (e.g., networks, servers, storage facilities, applications and services) that can be rapidly provisioned and released in an elastic and pay-per-use way to the user. As this paradigm is still an emergent area, it does not have a common, standardized technological model, so that each cloud platform has its own APIs, development tools and virtualization mechanisms. Consequently, the developed applications are often tightly coupled to a single cloud provider and its services and constraints, thus leading to a problem known as cloud lock-in.
[2]. However, an application can be deployed by using services provided by multiple, independent cloud platforms, thus being called a multi-cloud application [3]. Therefore, application developers are able to choose the services to be used, although they must directly negotiate with and consume the desired services from each used cloud platform.

There are some reasons to create multi-cloud applications, such as security or dimension concerns (e.g., cost, quality, etc.), or even when a specific cloud does not provide all required services [3]. However, there are at least four challenges to support the deployment of multi-cloud applications: (i) to provide some mechanism to address the heterogeneity of each cloud platform since it may lead to the abovementioned cloud lock-in problem; (ii) to take into account the monetary cost of all service options for each required feature to minimize the total deployment cost since each cloud has its own cost model; (iii) to take into account the quality properties of all service options for each required feature in order to maximize the overall quality of the application, and; (iv) to support dynamic adaptation (or reconfiguration) of the applications in terms of replacing a service by an equivalent one in case of quality degradation, high cost or service unavailability. There are other challenges (such as security and privacy issues, for example), but they are out of the scope of this paper.

B. The Branch-and-Bound algorithm

Computational optimization techniques have been applied to a range of applications from different domains [7, 8, 9]. In this study, a Branch and Bound (B&B) algorithm [20, 21] was used as an optimization technique for defining the configuration that better suits the user requirements of a multi-cloud application. This technique was chosen because it is at least competitive with other methods [11, 12, 13] for minimizing the solution search space. In addition, there are public reports of its adoption in optimization problems of different domains [7, 8].

By using a B&B algorithm, the solution space is gradually reduced and thus the definition of a given configuration can be facilitated when compared with conventional techniques, such as exhaustive search algorithms [13]. A B&B algorithm consists in finding optimal solutions for a given problem by means of two procedures called branching and bounding. The branching procedure consists in splitting the original solution set into subsets that are easier to solve. In turn, the bounding procedure will determine which subsets will be expanded (branching operation) and which ones will be bound (bounding operation). The bounding process aims to reduce the number of generated subsets in a B&B tree by calculating lower and upper bounds based on the current solution. These operations are iteratively applied to active subsets and unpromising subsets can be eliminated, thus reducing the search space.

Finally, the use of B&B algorithms enables to determine the bounds as tight as possible and thus the optimal solution of the problem, so that the computation time is kept at minimum. In Section IV.B we detail a B&B algorithm for an autonomic adaptation process of multi-cloud applications.
In order to illustrate this approach, we have developed HW-CSPL (Health Watcher Cloud Software Product Line), an SPL developed from the Health Watcher (HW) [14] Web-based system. HW enables citizens to consult information about the public health system of a city and to register complaints in terms of ingestion of contaminated food, mistreatment of animals or diseases transmitted by contaminated animals, and cases such as hygiene problems in restaurants, sewage leaks, etc. The commonalities were proposed from the requirements and features in the original HW system and the different service facilities provided by cloud platforms led to the features that represent the variabilities.

Figure 1 illustrates the HW-CSPL extended feature model. It contains mandatory features representing commonalities: (i) Persistence, which is the persistence mechanism of the application; (ii) Log System, which is the infrastructure used for storing log information, and; (iii) File Storage, which defines how files (e.g., images related to the application data) are managed in the application. Each one of these top-features has properties regarding the services represented by their alternative feature groups. For instance, the Persistence feature has three dynamic properties (price, availability, and responseTime) and offers two options for application’s data persistence, respectively represented by: (i) the Relational Amazon RDS feature, which is related to the Amazon RDS database service provided by Amazon Web Services (AWS) [15], and; (ii) the Relational HP Cloud feature, which is related to the relational database service provided by the HP Cloud [16] platform.

B. SPL-Based Adaptation of Multi-Cloud Applications

QoS parameters and other dynamic information regarding the used cloud services may change over time, thus affecting the deployed applications that make use of such services. These dynamic properties, which are modeled as properties of a feature model, must be continuously monitored and analyzed at runtime in terms of user-defined policies, so that any change on these policies must be reflected at the running application without the need of user intervention.

Our previous work [4] introduced an approach for the dynamic adaptation of multi-cloud applications based on the Mape-K [27] autonomic control loop, as depicted in Figure 2. In the Monitoring phase, values gathered by the Feature Monitoring Agent component are stored in a database managed by the Knowledge component, which is responsible for storing all information used in our strategy to achieve the adaptation of the application. In the Analysis phase, the Decision Maker component generates the product description (i.e., the configuration of the application to be deployed or reconfigured) by parsing the feature model and evaluating the decision criteria. Afterwards, the generated configuration description is stored in the Knowledge component and it serves as input for the Planning phase, in which the Application Composer component parses the configuration description generating an abstract model used by the Reconfigurator component on the Execution phase.

The following subsections describe the Knowledge component and each phase of the control loop.

1) Knowledge Component

This component is responsible for storing all the information used in the adaptation process. One of this information is the definition of which Cloud Computing platforms and the respective services can be used in the application considering that different cloud platforms can provide a same service required by an application. The service provider is selected based on a Variability Description, which describes the services to be used by the application and their quality parameters and the possible options of cloud platforms (service providers) for each application service. The Variability Description is implemented by a XML representation of the feature model and contains information about the relationship between commonalities and variabilities (in terms of the variation points in the application base code and the implementation of the associated variabilities).

2) Monitoring

In the Monitoring phase, the Feature Monitoring Agent is responsible for gathering the values of the properties defined in the feature model regarding each Cloud Computing platform/service. For instance, as shown in Figure 1, the Log System feature has two properties, namely availability and

![Figure 1. HW-CSPL feature model.](image-url)
responseTime, and two associated variabilities, namely HP Storage and Amazon S3, which regard to the services provided by the HP Cloud and AWS platforms, respectively. The values of these properties are measured for the two platforms and then stored in the database managed by the Knowledge component in order to be used in the Analysis phase. The monitor encapsulates the algorithms used to evaluate the properties defined in the feature model [17], except the price property, whose value is gathered by querying a data-price chart (maintained by our solution) that be periodically updated by the application manager.

3) Analysis
In the Analysis phase, the Decision Maker component must select the configuration that better suits user requirements (as he/she is the main stakeholder [29]) in terms of the properties defined in the feature model. Such requirements are specified in terms of the Decision Criteria, which are built in two parts: (i) the definition of multi-objective functions by using the max and min functions and logical operators (and, or), the Selection Function, and; (ii) a set of constraints specified by using comparison operators (=, >, >=, <, and <=), the Product Constraint Function. For instance, considering the feature model described in Figure 1, the Decision Criteria that selects a configuration that maximizes the availability for the File Storage feature, minimizes the price for the Persistence feature, but also establishes that the response time for the Log System feature cannot be greater than 10 ms can be written as follows:

\[
\begin{align*}
\max(\text{FileStorage}.\text{availability}) & \quad \text{and} \\
\min(\text{Persistence}.\text{price}) & \\
\text{LogSystem}.\text{responseTime} & < 10
\end{align*}
\]

In our previous work [12], we have proposed a conventional algorithm that evaluates all possible configurations for adaptation, as depicted in the Algorithm 1. This algorithm takes as input two parameters: (i) the Variability Description (VD), which is the XML file that represents the feature model, and; (ii) Product Constraint Function (PCF) and (iii) the Selection Function (SF), which are part of the Decision Criteria. In line 1, a list containing all possible products (LP) is generated based on the variability description. In line 2, a list containing all objective functions (OF) is generated based on the decision model description. In line 2, a list containing all objective functions (OF), which represent the functions to maximize or minimize the dynamic properties defined on the Decision Criteria, is generated based on the Selection Function (SF). From lines 2 to 17, the defined Product Constraint Function (PCF) is applied for each generated product. If a product p satisfies the PCF, then all the objective functions are evaluated considering the best available product (BProduct) and the current product (p). For the max function, if its value for BProduct is greater than its value for p, then the foundBestSolution variable is set to false, thus indicating that the current analyzed product does not maximize the value of the objective function. For the min

![Diagram](image-url)
function, the values are multiplied by -1 and the same procedure for the max function is performed. If the foundBestSolution variable value remains true, then the best product is updated. At the end, the algorithm returns as output the best available product (BP\text{Product}).

Algorithm 1. Conventional algorithm to select the best configuration [12]

<table>
<thead>
<tr>
<th>Inputs:</th>
</tr>
</thead>
<tbody>
<tr>
<td>VD – Variability Description (feature model)</td>
</tr>
<tr>
<td>SF – Selection Function (first part of the Decision Criteria)</td>
</tr>
<tr>
<td>PCF – Product Constraint Function (second part of the Decision Criteria)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output:</th>
</tr>
</thead>
<tbody>
<tr>
<td>BP\text{Product} – best selected configuration</td>
</tr>
</tbody>
</table>

1: $LP \leftarrow \text{GenerateProductList}(VD)$
2: $\mathbf{OF} \leftarrow \text{GenerateObjectiveFunctions}(SF)$
3: $\text{BP}\text{Product} \leftarrow \infty$
4: for each product $p \in LP$ do
5: if $\text{PCF}(p) = \text{true}$ then
6: $\text{foundBestSolution} \leftarrow \text{true}$
7: for each objective function $\text{obfunc} \in \text{OF}$ do
8: if $\text{obfunc}(\text{BP}\text{Product}) > \text{obfunc}(p)$ then
9: $\text{foundBestSolution} \leftarrow \text{false}$
10: end if
11: end for
12: if $\text{foundBestSolution} = \text{true}$ then
13: $\text{BP}\text{Product} \leftarrow p$
14: end if
15: end if
16: end for

Despite its usefulness, this algorithm has a major drawback as it evaluates all possible products and the number of these products can exponentially grow when more features/variabilities are introduced in the feature model, thus negatively affecting the scalability of this solution.

C. Planning

The Planning phase is a preparation phase in which the configuration description is analyzed according to the programming technique chosen to implement the variabilities associated to the feature model. In our previous work [4], we have used Dynamic Aspect-Oriented Programming (DAOP) [18] for this purpose. However, this programming technique may not be suitable for all types of applications and developers since the developer must have knowledge about the AOP paradigm, use third-party software to perform the dynamic weaving, and refactor the application to achieve the separation of concerns by using aspects.

In turn, an Object-Oriented Programming (OOP) technique typically uses components to modularize the application by establishing a clear separation of responsibilities. In our OOP approach, there are components requiring services provided by other components (cloud platforms/services) that must be wired at runtime, so that a dynamic reconfiguration support for components is needed, similarly to the approach used by FraSCAti [19]. FraSCAti is a framework based on Service Component Architecture (SCA) that enables the description of components and the connections between them and the capability to dynamic (un)wire components.

Finally, considering a team of developers that do not intend to rely on third-parties to implement all the possible configurations for a multi-cloud application, they can manage it by using the Flag Object-Oriented Programming (F-OOP) technique to define which service/cloud platforms will be used. This technique relies on the definition of a set of <key, value> pairs that represent the flags (written in a configuration file) can be evaluated at runtime by using conditional statements, so that a value for a given flag is checked in order to include/remove/change a specific variability.

In the Planning phase, the Application Composer component is responsible for parsing the configuration description generated by the Decision Maker component, thus allowing the Reconfiguration Manager to act to reconfigure the application in the Execution phase.

D. Execution

Finally, in the Execution phase of the autonomic loop, the Reconfiguration Manager is responsible for reconfiguring the application according to the respective programming technique that was chosen. If the AOP technique was chosen, a dynamic weaving process is performed, so that the aspects that implement the selected variabilities are weaved into the application base code at runtime. Alternatively, by using the OOP technique, the solution relies on dynamic reconfiguration of components’ connections by associating a component that requires a service (commonality) with the component that provides a service (variability). Finally, by using the F-OOP technique, the reconfiguration is performed by the application itself since it relies on checking flag values within the application base code.

IV. OPTIMIZED ADAPTATION OF MULTI-CLOUD APPLICATIONS

This section presents an optimized process used in the Analysis phase of the autonomic control loop (Section III.B) in order to select the best configuration for adapting a multi-cloud application. Section IV.A states the problem formulation by defining the mathematical functions associated with the problem. Section IV.B presents B&B algorithm proposed to optimize the products selection.

A. Problem Formulation

As described in Section II.B, the problem described in the Analysis phase can be viewed as a multi-objective problem because there are more than one criteria to be simultaneously satisfied. However, a B&B algorithm uses a single objective function for maximization or minimization. In order to transform the multi-objective problem into a mono-objective one, it is used a variation of the weighted sum approach [21] by defining an utility function (UF), as expressed by Equation 1:

\[
\text{maximize } \left( \sum_{i=1}^{n} \sum_{j=1}^{l} \text{norm}(p_{ij}) + \sum_{j=1}^{m} \sum_{k=1}^{o} \text{dist}(p_{ek}, c_{k}) \right)
\]

in which the $p_{ij}$ variable represents a property regarding an commonality $i$ and its variability $j$ in the feature model. Ac-
According to the Selection Function regarding this property, its value is normalized by applying the norm function as expressed by Equations 2 (for minimization) and 3 (for maximization):

\[
\text{norm}(p_y) = \begin{cases} 
\frac{p_y - p_{y,\text{min}}}{p_{y,\text{max}} - p_{y,\text{min}}}, & p_y - p_{y,\text{min}} \neq 0 \\
1, & p_y - p_{y,\text{min}} = 0
\end{cases} \quad (2)
\]

\[
\text{norm}(p_y) = \begin{cases} 
\frac{p_y - p_{y,\text{max}}}{p_{y,\text{max}} - p_{y,\text{min}}}, & p_y - p_{y,\text{max}} \neq 0 \\
1, & p_y - p_{y,\text{max}} = 0
\end{cases} \quad (3)
\]

Since the Decision Criteria is also composed of set of constraints, the second part of Equation 1 is related to the normalized distance of the current value associated to a property from the threshold established by the constraints associated to this property. The dist function, as expressed in Equation 4, takes the value of the property \(y\) indicated by the constraint \(y\) and the variability \(k\) and the threshold value \(c\):

\[
\text{dist}(p_y, c) = \begin{cases} 
\text{norm}(c_y - p_y), & \text{if the operator is } <, \leq \text{ or } = \\
\text{norm}(p_y - c_y), & \text{if the operator is } >, \geq \text{ or } \neq
\end{cases} \quad (4)
\]

### B. The B&B Algorithm

Algorithm 2 presents a high-level description of the proposed B&B algorithm, which is used for selecting the best configuration when adapting multi-cloud application. The algorithm takes as input the Variability Description (feature model) and the Utility Function (Equation 1), which is created based on the Decision Criteria (Section III.B), and returns the best solution that fits the specified criteria.

**Algorithm 2. The B&B algorithm for multi-cloud applications.**

**Inputs:**
- \(VD\) – Variability Description (feature model)
- \(UF\) – Utility Function (Equation 1)

**Output:** \(BSolution\) – best solution

1. DiscardedFeatures ← \(\emptyset\)
2. \(BSolution\) ← RandomSolution(VD)
3. \(CurrentSolution\) ← \(BSolution\)
4. while \(SolutionSpaceSize(VD, DiscardedFeatures) \neq 1\) do
5. \(PreviousSolution\) ← \(CurrentSolution\)
6. \(CurrentSolution\) ← NextNode(\(CurrentSolution\), VD)
7. if \(UF(BSolution) > UF(CurrentSolution)\) then
8. DiscardedFeatures ← DiscardedFeatures ∪ \{CurrentSolution, AddedFeature\}
9. \(CurrentSolution\) ← PreviousSolution
10. else
11. \(BSolution\) ← \(CurrentSolution\)
12. end if
13. end while

In order to apply the B&B technique, the problem is described as a tree in which each node represents a partial solution. The B&B algorithm runs as follows. Firstly, the DiscardedFeatures set is initialized an empty set (line 1), which will be used for storing the features that are discarded during the B&B execution. A feature is discarded when it does not improve the value of the utility function (\(UF\)) for the current solution, so that any solution derived from the discarded feature must be also discarded. In line 2, a random solution is generated as the root of the B&B algorithm tree by randomly selecting one variability of each common feature in order to compose a solution. A solution is a set \(S = \{f_1, f_2, \ldots, f_n\}\) composed of the variabilities \(f_1, f_2, \ldots, f_n\) that are part of the selected configuration. It is also important to notice that the random initial solution directly affects number of possible solutions to be analyzed.

In the main loop (line 4), the \(SolutionSpaceSize\) function calculates the number of the candidate solutions based on (i) the Variability Description (taken as input) and (ii) the discarding of the solutions that use features included in the DiscardedFeature set (taken as input). In line 5, the PreviousSolution variable stores the value of the CurrentSolution variable. In line 6, the NextSolution function generates another available solution from the current one by analyzing which feature (from left to right in the feature model) can be considered in the new solution. For instance, considering a current solution represented by \(S = \{\text{RelationalAmazonRDS, AmazonSimpleDB, HPStorage}\}\) (see Figure 1), the next available solution could be \(S' = \{\text{RelationalHPCloud, AmazonSimpleDB, HPStorage}\}\). However, the solution \(S'\) will be used only if it has a greater value for the \(UF\) function since our problem is a maximization one. Otherwise, all sub-nodes derived from it will be discarded and the relational HPCloud variability will be added to the DiscardedFeatures set.

In line 7, the \(UF\) function is evaluated for both best and current solutions. If the current solution has a greater value for the \(UF\) function, then the added feature will be discarded and the algorithm goes back to the previous solution. Otherwise, the best solution is updated and the algorithm continues to process the sub-nodes. The algorithm ends when the solution space is reduced to one, which means that the best solution was found and there are no more possible solutions to be analyzed.

Figure 3 presents the B&B combinatorial tree generated during the search process for the feature model presented in Figure 1. In Figure 3, \(f_p, f_a, f_i\) and \(f_l\) respectively represent the Persistence, Storage and Log System features and the \(a\) and \(h\) indexes represent the variabilities. For instance, \(f_{pa}\) represent the AmazonRDS variability regarding the persistence service provided by the AWS platform. The first node \(\{f_{pa}, f_{fa}, f_{fl}\}\), which was randomly generated, has \(UF = 1\) for its utility function. The edges in the tree represent which feature is being changed. The \(f_{fa}\) label means that the Persistence feature is changing from \(f_{fa}\) to \(f_{fa}\), i.e., the persistence service is changed from the one provided by the AWS platform to the provided by the HP Cloud platform. If the value for \(UF\) of the next possible node (in this example, \(\{f_{pa}, f_{fa}, f_{fl}\}\)) is greater than the previous one, then the best solution is updated and the space search is also updated. As the replacement of \(f_{fa}\) to \(f_{fa}\) has led to a better solution, any solution using the \(f_{fa}\) is discarded, thus reducing the search space.

Continuing the branching process, other generated node (solution) is \(\{f_{fa}, f_{fa}, f_{fa}\}\) with \(UF = 1\). This means that replacing the Storage feature from the storage service provided by AWS to the one provided by HP Cloud does not improve
the solution. Therefore, the variability \( f_{sb} \) is discarded as well as all the possible solutions derived from this node. The last node to be evaluated is \( \{f_{ph}, f_{sa}, f_{lh}\} \) with \( UF = 2 \), which also does not improve the best solution found so far.

V. EVALUATION

The goal of our evaluation was to compare the performance of the previous version of the algorithm (as presented in Section III.B) with the B&B algorithm (presented in Section IV.B). The main concern was to evaluate if the B&B algorithm scale out when the number of features and/or variabilities of our feature model increases, thus leading to increase the number of possible solutions. For this purpose, three different versions of our feature model (Figure 1) were designed resulting in 8, 27 and 54 possible configurations. The feature model with 27 possible configurations was built by making use of the RackSpace platform [26] to provide the respective services for the Persistence, Storage and Log System features. The feature model with 54 configurations was obtained by introducing a common feature called Messaging, which enables the application to send messages to registered user regarding any public health notification. To implement the Messaging feature, two different services/platforms were used, namely Amazon SNS and HP Cloud Messaging.

As our algorithm relies on the values gathered by the Feature Monitoring Agent, we have provided dummy values (using previous stored values of the properties) and run each algorithm 1000 times for each configuration scenario (with 8, 27 and 54 configurations). Since the solution was implemented using the Java language and the Java Virtual Machine (JVM) may influence the execution time of a program, it was needed to run the algorithms more than once aiming at more reliability for the results. The evaluation experiments were performed on Microsoft Windows 7 Professional 64-bit operating system with an AMD-8 1.6 GHz processor and 12 GB of RAM.

Figure 4 shows a chart with the average execution time (in milliseconds) versus the number of possible configurations by evaluating the following Decision Criteria:

\[
\begin{align*}
&\text{min}(\text{Persistence.responseTime}) \text{ and } \\
&\text{min}(\text{FileStorage.responseTime}) \\
&\text{LogSystem.availability} > 0.97
\end{align*}
\]

Such Decision Criteria define that the configuration to be selected must minimize the response time for the Persistence and FileStorage features and the availability related to the Log System must be greater than 97%.

As it is possible to see in Figure 4, the B&B algorithm has a faster execution time than the conventional algorithm. In the worst case, with 54 configurations, the former was, on average, faster than the latter in 43.42%. It is important to notice that as the feature model tends to grow, the B&B algorithm tends to run even faster than the conventional algorithm. When the number of configurations varies from 27 to 54, the conventional algorithm presented a growth of 36.66% regarding the execution time, against 17.17% of the B&B algorithm on the same change. The main benefit from the B&B algorithm is the capability to narrow down the space search by using an effective branch function. In this version of the B&B algorithm, the Utility Function (see Equation 1 in Section IV.A) and the management of the features to be discarded enables the algorithm to quickly remove nodes from the space search tree. Considering the tree search presented on Figure 3, when the variability \( f_{pa} \) is discarded, four possible solutions that use this variability are also discarded. Therefore, the solution space is narrowed down by 50% already at first iteration. On the other side, the conventional algorithm exhaustively performs the search process throughout the entire solution space.

Although the experimental set was small, the collected results indicated a greater capability of scaling when using the B&B algorithm rather than the conventional one for large configurations and options of services within the adaptation process. The analyzed configurations and all the source code are publicly available at [28].

In our study, the main threat to validity of the performed evaluation is the medium size of the feature models. It is important to notice that a feature model used in context of this work is to support the description of a multi-cloud scenario and not a full featured SPL. Another threat to validity includes the unknown factors that may have influenced the experimental results [25]. To reduce this threat, we have selected a subject program whose developers had no
knowledge that this study was being performed for them to artificially modify their coding practices. Additionally, our results can be influenced by the performance, in terms of precisions and recall, of the used tools. We have tried to limit the number of false positive through a manual validation.

VI. RELATED WORK

As far as we are concerned, the literature does not present any work that uses optimization models for autonomically adapting Cloud Computing applications, and few works in the literature use some optimization technique in the context of self-adaptation of systems. In this section, we briefly discuss some of these works in the context of the adaptation of component-based and/or service-based systems.

Mirandola and Potena [22] introduce a framework based on an optimization model that dynamically enables the adaptation of service-based systems while minimizing the cost due to this adaptation process and considering quality properties. The adaptation process envisioned by the proposed framework can be triggered both by a user request and/or by the framework itself in cases of runtime violation of system quality constraints (detected by an internal monitoring module) or the arising/disappearing of services into the environment. The core of the framework is represented by the Generator and Evaluator module, which is responsible for generating a mathematical based optimization model from other types of system models (e.g., component diagrams, sequence diagrams, activity diagrams, etc. generated from the system implementation by another module of the framework). This generated optimization model is then evaluated by the LINGO solver [23] in order to produce optimal adaptation actions, which represent suggested changes to be applied to the system architecture with minimum cost while preserving constraints defined in terms of quality attributes. However, the authors do not present further details of the framework nor any instantiation of the proposed model. Moreover, the framework is limited when dealing with large computational models (for instance, when the number of available services is large), which might be, according to the authors, addressed with metaheuristic techniques combined with simulation techniques.

In the context of component-based software development (CBSD), Tang et al. [24] propose an Integer Programming approach based on optimization models for assisting developers when selecting components to be assembled to a system or used in case of system adaptation. This mathematical approach aims to address the selection of available components to be deployed into applications while minimizing the incurred adaptation costs and taking into account a compatibility matrix in order to express compatibility relationships between components. The authors use a genetic algorithm for solving the component selection problem foreseeing the combinatorial explosion in large instances of the problem. However, the authors do not consider functional or non-functional requirements of the components, e.g., quality parameters. Furthermore, as genetic algorithms are heuristic (approximate) techniques, they might not guarantee the optimal solution.

The abovementioned works show the importance of taking into account some optimization technique in order to provide scalability to software adaptation processes. It is also important to highlight the usefulness of exact algorithms (such as the mathematical programming technique used by Mirandola and Potena [22] and our B&B algorithm), which are able to find optimal solutions, i.e., the best configuration(s) that will be used in the adaptation process to be taken. However, with the myriad of component/service options for reconfiguring applications, the search solution space still may grow and exact techniques may not perform well and even be able to find the optimal solutions in an acceptable computational time. In this case, heuristic techniques (such as the genetic algorithm introduced by Tang et al. [24]) can be used for finding near-optimal solutions in suitable computational times.

VII. FINAL REMARKS

This paper proposes a dynamic adaptation process for multi-cloud applications managed by an autonomic control loop based on Mape-K [27]. One of the main phases of the control loop is the Analysis phase, which is based on the monitored values of the dynamic properties that compose the feature model, in which the Decision Criteria is applied in order to select the best configuration for the application. In a previous work [4], we designed a conventional algorithm to analyze these Decision Criteria, and as it evaluates all possible configurations, it is not suitable for large feature models. In this work, we have presented a Branch-and-Bound (B&B) algorithm in order to optimize the selection of the configuration used in the adaptation process.

Computational experiments comparing the B&B algorithm with the conventional version for different configuration scenarios (with 8, 27 and 54 possible configurations) have shown that the B&B performs well and faster than the other algorithm. For the feature model that can generate 8 possible configurations (the best case), the B&B algorithm is 22.47% faster than the conventional one, on average, while for the feature model that can lead to 54 possible configurations (the worst case), the B&B algorithm is 43.42% faster than the other one, on average. Therefore, we can conclude that the proposed B&B algorithm has a greater capability of scaling for large configurations and options of services within the adaptation process.

This work can be further improved in several ways: (i) to extend our approach in order to consider historical information regarding, e.g., the cost of a specific adaptation in order to identify its influence on future dynamic adaptations and how we can prevent failures in the user defined policies regarding QoS; (ii) to evaluate our solution by using different types of applications and Cloud Computing platforms/services, and; (iii) to describe the user requirements in terms of QoS at a high level, flexible way, especially in terms of the constraints specified in the Product Constraint Function.

ACKNOWLEDGMENTS

André Almeida and Everton Cavalcante are partially supported by CAPES under grants 11096/2013-6 and
11097/2013-2, respectively. Everton Cavalcante is also partially supported by the Brazilian National Agency of Petroleum through the PRH-22/ANP/MCTI Program. Thais Batista is partially funded by CNPq under grant 485935/2011-2.

REFERENCES


[26] Public cloud hosting, computing, storage and networking by RackSpace: http://www.rackspace.com/cloud

