Received: 17 May 2018 | Revised: 3 September 2018 | Accepted: 8 September 2018

DOI: 10.1002/spe.2644

RESEARCH ARTICLE

WILEY

Cloud application architecture appraiser (CA3): A multicriteria approach and tool for assessing cloud deployment options based on nonfunctional requirements

Ronaldo Gonçalves Junior | Americo Sampaio | Tiago Rolim | Nabor C. Mendonça

Programa de Pós-Graduação em Informática Aplicada, Universidade de Fortaleza, Fortaleza, Brazil

Correspondence
Americo Sampaio, Programa de Pós-Graduação em Informática Aplicada, Universidade de Fortaleza, Av. Washington Soares 1321, Edson Queiroz, 60811-905 Fortaleza-CE, Brazil.
Email: americo.sampaio@unifor.br

Funding information
CNPq, Grant/Award Number: 313553/2017-3 and 207853/2017-7

Summary

Cloud computing is an emerging computing paradigm that is changing software engineering. It offers scalable virtual compute resources at low prices, thus attracting many software developers interested in reducing their infrastructure and operational costs. Even though acquiring and using cloud solutions can be simple, a common difficulty developers face is how to best configure their applications at the architectural level, given a myriad of cloud services and resource types available, specially when considering different attributes such as cost, scalability, and performance. An inappropriate architectural decision can lead to a significant cost increase or a deployment option that does not meet the minimum required performance. This paper focuses on the problem of finding the best cloud deployment options for a given application that meet a given set of well-known nonfunctional requirements. To address this problem, the paper presents Cloud Application Architecture Appraiser (CA3), a multicriteria optimization approach and tool that relies on nonfunctional requirements as key drivers for assessing and selecting the best architectural options for deploying applications in the cloud. Results from the assessment of a real web application (WordPress), deployed using several different architectural options in a popular cloud provider (Amazon), are also presented to illustrate the use and benefits of the proposed method and tool.

KEYWORDS

AHP, architectural assessment, cloud deployment option, multicriteria optimization

1 | INTRODUCTION

Cloud computing providers such as Amazon, Microsoft, and Google offer scalable low cost virtual computing resources that can be rapidly provisioned through the internet.[1] This model of computation is changing the way many organizations deliver their IT services and applications, as they can free themselves from the burden to build, manage, and continuously upgrade an in-house physical computing infrastructure.

Despite of the many benefits offered by existing cloud computing solutions and the relative simplicity of acquiring cloud resources, many cloud users still have difficulties in selecting the cloud resources and services that best suit their applications' needs.[2-4] For example, previous studies[5,6] have shown that variations in the number and types of the virtual machines (VMs) used to deploy multitiered web applications in the Amazon cloud can result in significant performance and cost differences from the application users perspective. In particular, for some specific application layers (eg, web and application service layer), clustered-based deployment architectures containing arrays of low capacity VMs can significantly outperform higher capacity VMs at a much lower cost.[5] Therefore, making the right architectural decisions when deploying applications in the cloud can have a direct impact not only in the applications' operational costs but also in many other important quality attributes.[7]

In this paper, we focus on the problem of finding the best deployment options for a given cloud application that meet a given set of quality criteria, which are usually specified in terms of well-known nonfunctional requirements (NFRs), such as cost, performance, scalability, reliability, and security.[8-11] To this end, we present the Cloud Application Architecture Appraiser (CA3) approach that relies on NFRs as key drivers for assessing and selecting the best deployment options for a given target application. The approach comprises a multicriteria optimization method (analytic hierarchy process – AHP[12]) and its corresponding implementation in the form of a GUI-based interactive tool. Besides proposing the approach, we also make the following contributions: (1) definition of new metrics for assessing NFRs in the specific context of cloud applications; (2) experimental evaluation, from the perspective of multiple NFRs, of several deployment options for a real web application (WordPress) in a popular cloud provider (Amazon); and (3) analysis of the impact of real cloud services available for architectural selection based on NFRs.

It is important to clarify that our approach focuses on assessing different architectural options for an application from a deployment perspective. This means that the developer does not need to change the logical components or code implementation. What basically changes is the possibility to deploy the same application in different ways in the target cloud. For example, suppose a multitier web application organized using the layers and model-view-controller pattern in three layers, ie, View (containing the user interface code), Business (containing the business rules models and controllers) and Data (containing the data access code that manipulates the database). This application can be deployed in different ways in the cloud, such as (1) all three layers in a single VM; (2) the View and Business layers in a single VM with a separate VM for the database; (3) the View and Business layers replicated in multiple VMs under the control of a cloud native autoscaling service with a separate VM for the database; and (4) all three layers replicated in multiple VMs. All the previous deployment alternatives are referred to in this paper as cloud deployment options (CDOs).[13,14] Depending on the number and types of VMs used for each of those CDOs, the cost, performance, fault tolerance, and other quality attributes of the target application can vary significantly. For example, a CDO with service replication, such as (3) and (4) earlier, provides much better performance and fault tolerance, however can increase costs significantly. Therefore, our approach does not aim to assess different variations on the architecture logic as classic architectural assessment methods such as ATAM[15] and others.[16] In that respect, our approach can provide invaluable feedback for developers who may be considering migrating an existing application to the cloud and need to have an informed opinion about the cost, performance, and other quality attributes related to different CDOs for their application, which is a very common demand of current cloud users.[2,3,5,7] Another difference from our approach and classic architectural evaluation methods is that our work is not limited to early development stages and can be applied to any application already in production that may need to be migrated from on premise to the cloud or between cloud providers.

Our approach also differs from others such as the work of Chung et al,[7] which used simulation for selecting between different CDOs, in that, we base our assessment of quality criteria on executions of the real application deployed in the target cloud. Approaches based on simulation suffer from the fact that the resources needs of a given cloud application are extremely difficult to characterize without actually deploying the application in the target cloud platform.[17-19] Therefore, using simulation to estimate the resources necessary for a given cloud application may involve a high risk of misprovisioning if the application's resource needs are not well known or properly characterized in advance. The use of real application tests is a practice recommended by public cloud providers such as Amazon:

> *Amazon EC2 provides you with a large number of options across ten different instance types, each with one or more size options, organized into six distinct instance families optimized for different types of applications. We recommend that you assess the requirements of your applications and select the appropriate instance family as a starting point for application performance testing. You should start evaluating the performance of your applications by (a) identifying how your application needs compare to different instance families (e.g. is the application compute-bound, memory-bound, etc.?), and (b) sizing your workload to identify the appropriate instance size. There is no substitute for measuring the performance of your full application since application performance can be impacted by the underlying infrastructure or by software and architectural*

*limitations. We recommend application-level testing, including the use of application profiling and load testing tools and services.**

Therefore, to the best of our knowledge, our approach is the first cloud deployment assessment solution to address all the issues mentioned previously.

This paper is a revised and extended version of a previous conference publication.[14] This new journal version expands and improves upon the conference version in the following ways.

- A completely new problem formulation was added. This is detailed in Section 2 and Section 3, which contain a formalization of the problem that was not in the conference version.
- The formulas used for computing the AHP scores for some quality attributes, such as scalability, were changed. We did this based on the feedback received during the conference presentation, as some of the original formulas were pointed out as needing improvements.
- The AHP score calculation and assessment conducted as part of our empirical evaluation were redone to account for the formula changes mentioned earlier. Moreover, this journal version provides a more in depth analysis of the assessment results.
- A detailed description of the tool developed to support our approach as well as a discussion of other complimentary tools that can also be used were added to the journal version. These are provided in Section 3 and Section 4.
- Finally, a new discussion on the main threats to validity that may affect the results of our empirical evaluation was also added to the paper. These are provided in Section 4.

The remainder of this paper is organized as follows. Section 2 introduces some background information related to our research problem. Section 3 describes and illustrates the use of our proposed approach. Section 4 reports on design, execution, and results of our experimental evaluation. Section 5 covers related works. Finally, Section 6 presents our main conclusions and points out directions for future work.

2 | PROBLEM STATEMENT

Nonfunctional requirements[8,9] play a critical role during software development, serving as selection criteria for choosing among alternatives of distinct architectures. Moreover, NFRs can have different priorities and affect each other, which poses more challenges to the architectural selection process. For example, improving performance may increase cost, as well as improving security may decrease performance.[20] This mutual influence and relative importance of NFRs are commonly represented as hierarchical structures in some NFR related works, such as goal graphs.[8]

The deployment architecture of applications configured to use cloud services tend to be highly influenced by several NFRs.[7] For example, considering a multitier web application such as WordPress,[6] the decision on how to best configure its deployment architecture could be influenced by cost, performance, scalability, security, and so on. Therefore, the problem of selecting CDOs based on multiple conflicting NFRs can be seen as a typical multicriteria decision problem. In this sense, NFRs are the criteria that can be refined in multiple level hierarchies (subcriteria) that can be quantitatively and/or qualitatively compared against each other. Moreover, each criteria (and subcriteria) can be assigned a preference level, indicating the relative importance of the criteria when compared against others of the same level. For example, when deciding on the CDO for one specific system, its developers may consider that efficiency is more import than cost. Figure 1 shows an example of the hierarchical organization of several NFRs as drivers for deciding among CDO candidates.

Our proposed approach (Section 3) aims to build explicit connections between NFRs and CDOs through the AHP[12] multicriteria decision method (MCDM). We have chosen AHP as we believe it is the most fitting MCDM to solve our research problem, as described earlier. The AHP aids in the ranking and evaluation of solutions alternatives based on a given number of criteria. In the AHP process, after identifying the criteria and the alternatives, the decision makers conduct pairwise comparisons to attribute preferences to all criteria and subcriteria. Once all criteria are compared, the AHP method calculates a score[12] for each alternative that ranks their overall importance. More details on how our work uses AHP are given in the next section.

*http://aws.amazon.com/ec2/instance-types/

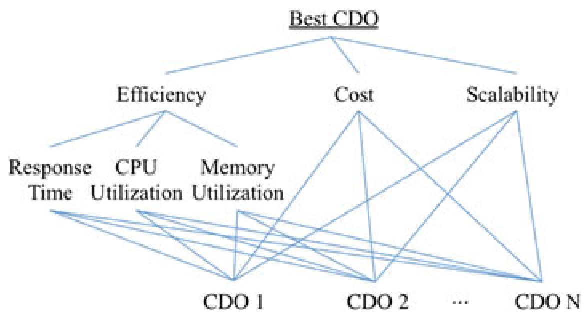


FIGURE 1 Hierarchy of nonfunctional requirements and candidate cloud deployment option (CDOs) [Colour figure can be viewed at wileyonlinelibrary.com]

3 | THE CA3 APPROACH

The CA3 approach supports software architects in analyzing the CDOs available for their applications. The goal is to enable them to choose the options that best fit a given set of NFRs of interest. In the following sections, we formalize the main concepts related to our research problem and describe the method proposed to solve it.

3.1 | Formalization

Figure 2 describes the basic hierarchical structure used by our proposed multicriteria approach, which is based on the AHP method. The approach's objective (top of the figure) is to find the best CDO for the target application. A set of criteria, derived from the objective, represent parameters related to NFRs used for decision making. The criteria themselves can be further refined into as many numbers and levels of subcriteria as necessary. At the bottom of the hierarchy are the CDOs that somehow address the criteria and subcriteria defined earlier.

Definition 1. (Hierarchy)
Let H be a set of elements. Then, H is a hierarchy if there is a partition of H into sets $L_k, k = 1, \dots, h$ for some h such that, for each element $e \in H$

- $e \in L_1 \rightarrow L_1 = \{e\}$, e is the (single) root of the hierarchy;
- $e \in L_k \rightarrow Parent(e) \in L_{k-1}, k = 2, \dots, h$, $Parent(e)$ is the set of parents of e;
- $e \in L_k \rightarrow Children(e) \in L_{k+1}, k = 1, \dots, h-1$, $Children(e)$ is the set of children of e; and
- $e \in L_h \rightarrow e$ is a leaf of the hierarchy.

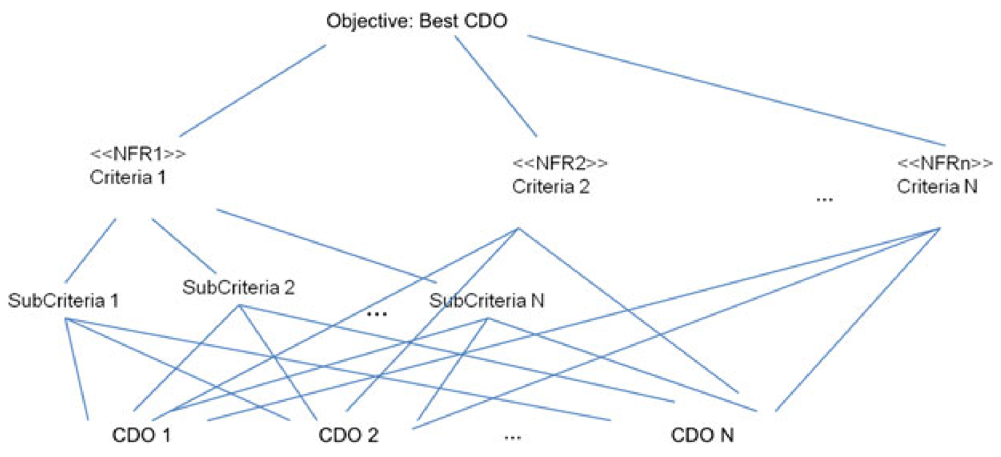


FIGURE 2 The proposed method's hierarchical decision structure. CDO, cloud deployment option; NFR, nonfunctional requirement [Colour figure can be viewed at wileyonlinelibrary.com]

Definition 2. (Objective)
The approach's *objective* is to find the best CDO among the ones evaluated. It is a unique element in the hierarchy represented by its root.

Definition 3. (Criteria)
The objective is achieved by analyzing a set of *evaluation criteria*, which represent different NFRs to be utilized as a basis for comparing CDOs. Let H be a hierarchy and let $o \in H$ be the objective (root) of the hierarchy. The set of evaluation criteria C is defined such that

- $C \neq \emptyset$; and
- $\forall c \in C, Parent(c) = \{o\}$.

Definition 4. (Subcriteria)
Each evaluation criterion can be subdivided into two or more subcriteria for better assessment by the application architect. Each sucriterion, in turn, can be further refined in terms of two or more subsubcriteria, and so on, recursively. Let C be the set of evaluation criteria and let $c \in C$ be a given criterion. The set of subcriteria S of c is defined such that

- $|S| \geq 2$; and
- $\forall s \in S$,

$$Parent(s) = \begin{cases} \{c\}, \exists c \in C | s \in Children(c) \\ \{s'\}, \exists s' \in S | s \in Children(s'). \end{cases}$$

Definition 5. (Metrics)
Each element $e \in H$ can be associated with one or more *evaluation metrics*, which enable one to assess the importance of e according to the AHP method. Let M be a set of evaluation metrics. Let $f : M \rightarrow H \times \Re^+$. Let $F_m \subset f(m)$ for $m \in M$. The function $F_m(e)$ assigns a positive real number to every element $e \in H$ according to evaluation metric m.

The nature of function $F_m(e)$ will depend on the type of metric m being considered for evaluation. For example, if m is quantifiable, eg, price, performance, and resource usage, the values returned by F can be based on one (or more) formula(s) or empirical measurements applied to e by the application architect. On the other hand, if m is not (easily) quantifiable, eg, architectural complexity, the values of m can be determined by the application architect him/herself based on external knowledge or on his/her own qualitative judgment of e in light of m.

Definition 6. (CDOs)
Each CDO to be evaluated is composed by a set of one or more VMs. Each VM in turn is configured with one or more application components or services. Let H be the set of all elements in the hierarchy. The set of CDOs $A \subset H$ is defined such that

- $|A| \geq 2$; and
- $\forall a \in A | Children(a) = \emptyset$.

Note that the hierarchy would be represented by a tree structure if the leaf nodes were to be ignored, given that all other elements have exactly one parent, with the exception of the root node, the objective, which naturally has no parents. However, when the leaf nodes (CDOs) are considered, the hierarchy is represented by a graph where each CDO is a child of every leaf node of the tree.

Definition 7. (Comparison matrix)
A *comparison matrix* is built as part of the *pairwise comparison* stage of the AHP method. The goal of this matrix is to organize the numerical values that serve as input for calculating the score of each element in the hierarchy. Let H be the set of all elements in the hierarchy. For each $e \in H$, the method compares every pair of elements $(ch1, ch2) \in Children(e) \times Children(e)$. The result is a n by n comparison matrix (see Table 1), where $n = |Children(e)|$.

Note that each child of element e is compared against its siblings, ie, the other elements that also belong to $Children(e)$. This comparison must take into account all previous elements in the hierarchy because criteria, subcriteria, and CDOs are all part of the comparison method. Moreover, the values for comparison are provided by the functions that return the score of each element according to the metrics of interest.

TABLE 1 Comparison matrix for element *e*

$Children(e)$	ch_1	ch_2	ch_3	…	ch_n
ch_1	1	x_{12}	x_{13}	…	x_{1n}
ch_2	x_{21}	1	x_{23}	…	x_{2n}
ch_3	x_{31}	x_{32}	1	…	x_{3n}
⋮	⋮	⋮	⋮	⋱	⋮
ch_n	x_{n1}	x_{n2}	x_{n3}	…	1

During the CDOs evaluation, the method calculates two types of score. *Local score* assigns a metric for one specific element of the hierarchy by comparing the values assigned to its children according to that same metric. *Global score* takes into account all elements of the hierarchy simultaneously. All calculated scores are normalized.

Definition 8. (Local score)
The local score (LS) of an element is calculated considering that element as an objective and its children is the only elements remaining in the hierarchy. Let H be the set of all elements in the hierarchy. Let o be the objective of H. The LS of an element $e \in H, e \neq o$ is defined as follows:

$$LS(e) = \frac{F(e)}{\sum_{i \in Children\,(Parent\,(e))} F(i)}.$$

Definition 9. (Global score)
The global score (GS) of an element is the value that represents the importance of that element with respect to the objective. Let H be the set of all elements in the hierarchy. Let $A \subseteq H$ be the set of CDOs. The GS of an element $e \in H$ is defined as follows:

$$GS(e) = \begin{cases} LS(e), Children(e) \subseteq A \\ \sum_{ch \in Children\,(e)} GS(ch), \text{otherwise.} \end{cases}$$

Definition 10. (Ranking)
The *ranking* is an ordering of the CDOs based on their global scores. Let H be the set of all elements in the hierarchy. Let $o \in H$ be the objective. Let $A \subseteq H$ be the set of CDOs. Let $>_o$ be a binary relation on A representing their order of preference with respect to o. Hence, for every two alternatives $a_x, a_y \in A$, $a_x >_o a_y \rightarrow GS(a_x) \geq GS(a_y)$.

3.2 | Method description

Our approach for CDO assessment consists of four main steps, as shown in Figure 3. It starts with a list of the relevant NFRs for the system, their definitions, and criteria of evaluation (Step 1). Then, the software architect defines a range of possible CDOs for the target application that take those NFRs into account (Step 2) and that are subsequently evaluated,

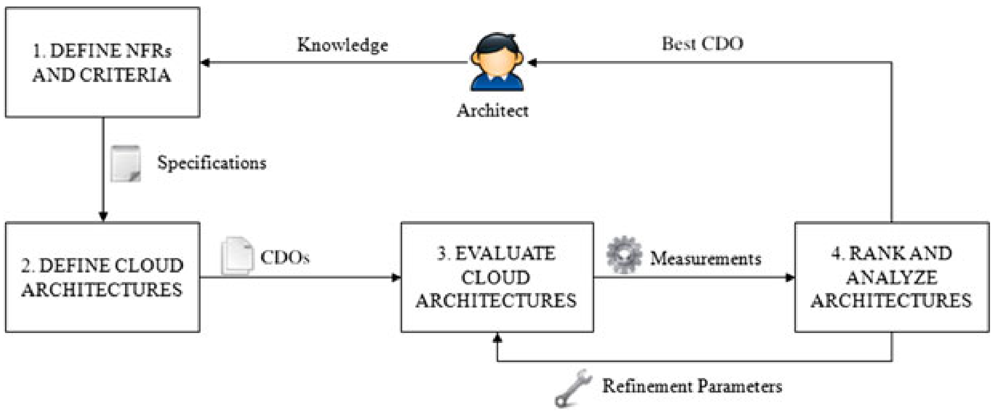

1. DEFINE NFRs AND CRITERIA
Knowledge
Architect
Best CDO
Specifications
2. DEFINE CLOUD ARCHITECTURES
CDOs
3. EVALUATE CLOUD ARCHITECTURES
Measurements
4. RANK AND ANALYZE ARCHITECTURES
Refinement Parameters

FIGURE 3 Steps for cloud deployment architecture assessment. CDO, cloud deployment option [Colour figure can be viewed at wileyonlinelibrary.com]

eg, through experimentation in the cloud (Step 3). Based on the evaluations results achieved by the different CDOs for the various NFRs, eg, response time and scalability, the architect uses the AHP method to rank the CDOs and thus identify the best deployment strategies for the target cloud application (Step 4). The following sections describe each of those four steps in more details.

3.2.1 | Define NFRs and criteria

The objective of the method (Definition 2) is to help the architect to select the best CDOs for the application. During this step, the architect will select all the relevant NFRs for the application that will be deployed in the cloud. In addition, he/she will have to define a set of criteria and, optionally, a set of subcriteria (Definitions 3 and 4) to evaluate these NFRs based on one or more metrics of interest (Definition 5).

The NFRs are very relevant for the deployment any application, specially cloud-based ones. The CDOs available for a multitier web application, for example, are highly influenced by a number of NFRs, such as cost, performance, security, and scalability. In the context of our work, we select a list of relevant NFRs and how to measure them taking as sources popular documentation such as NFR catalogs,[8] the ISO-9126-2 standard,[10] as well as our own domain knowledge of typical cloud deployment requirements. For the purpose of this paper, we decided to focus our analysis on three NFRs, ie, efficiency, cost, and scalability, as described later. Our idea was to investigate which NFR metrics are already defined in the literature and how we could reuse/adapt them for the specific case of cloud applications. Finally, it is important to highlight that the proposed method is not limited to these three NFRs and can be easily extended to incorporate other NFRs and metrics not described here.

Efficiency

This NFR is at the first level of our AHP hierarchy and represents the "time consumption and resource utilization behavior of computer system including software during testing or operations."[10] This is a fundamental, and probably the most relevant, NFR for any cloud-based system. According to the work of ISO/IEC,[10] efficiency can be further refined in two subcategories, ie, *Time Behavior* and *Resource Utilization*. We consider Time Behavior to represent the performance of the system, measured by its *Response Time* when submitted to a specific workload. For the Resource Utilization category, we decided to break it down in two specific subcriteria, ie, *CPU Utilization* and *Memory Utilization*, to assess them separately as the software architect could decide to analyze them with different preference weights.

Response time (rt). It is not relevant to the proposed approach which measurement unit for time (t) the architect decides to use as long as t represents the average amount of time taken for the system to process a request. The measurement unit is not relevant because all time units are normalized during the analysis phase. The following formula is used to calculate the response time score (rt) for a given alternative with measured response time t:

$$rt = \frac{1}{t}. \tag{1}$$

The aforementioned formula is a simple mathematical way of inverting the ordering of the CDOs in the ranking phase, for a lower response time is better than a higher one, so the former must receive a higher score.

CPU utilization (c). The measurement unit for CPU utilization is the average percentage of utilization (p) of the respective resource. The following formula is used to calculate the CPU utilization score (c) for a given alternative with measured average percentage of CPU utilization p:

$$c = \begin{cases} p, p \leq 70 \\ 140 - p, p > 70. \end{cases} \tag{2}$$

The CPU utilization score is equal to the actual value measured for this metric if that value is equal to or less than 70%. If the measured value is greater than 70%, the score is reduced so as to penalize alternatives that consume too much CPU. For example, a measured value of 85% will produce a score of 55. The reason we compute the formula this way is because, from a resource consumption perspective, consuming too little or too much CPU is equally undesirable. The former may indicate that the application is overprovisioned (ie, wasting resources), whereas the latter may indicate that the application is under provisioned (ie, with insufficient resources).[1]

This value of 70% for CPU utilization comes from previous works and practical benchmarks for applications.[21-23] Having a value much lower than that would be wasting resources. A value higher than this (closer to 100% for example) can heavily affect the application because of queuing delay, which might severely influence latency. Practice suggests a value between 70% and 75% for optimal CPU usage. In the case of our CPU score formula, the goal is to penalize any deployment option whose CPU usage is either lower or higher than that threshold, as this would indicate that CPU resources are being either overprovisioned or underprovisioned, respectively, for that particular CDO. The same reasoning applies to memory, which is also a different formula (and different criteria) dealt with by the multicriteria method. Therefore, the approach would work in any case of CPU or memory bound applications as both criteria are considered by the approach and what we aim to compare are different variations (CDOs) of the same application.

Memory utilization (m). The measurement unit for memory utilization is also the average percentage of utilization (p) of the respective resource. The following formula is used to calculate the memory utilization score (m) for a given CDO with measured percentage of memory utilization p:

$$m = \begin{cases} p, p \leq 50 \\ 100 - p, p > 50. \end{cases} \tag{3}$$

Analogously to the CPU utilization score, the memory utilization score is also reduced when the measured value is above a certain threshold, in this case, 50%, as we consider memory to be a more critical resource than CPU for most cloud applications. Of course, those two thresholds are not dictated by our proposed approach, and both can be adjusted by the software architect according to the characteristics and needs of the target application at hand.

Cost ($)

The usage of almost all kinds of cloud services and resources has a direct impact on the cost of running a given application in the cloud. This is an intrinsic characteristic of a cloud computing environment, known as the pay-as-you-go or pay-per-use business model.[1] In a practical view, it means that cost affects almost every architectural decision, even if it is a small effect.

The cost of a cloud application is the sum of the costs charged by the cloud provider for every resource used by that application. These include computing resources (eg, VMs), storage resources (eg, virtual disks and storage services), network resources (eg, data transfers), and any other paid service or resource (eg, cache, replication). The most straightforward way of calculating the cost for a running a given cloud application is to estimate the application's expected cloud resource usage. This can be done with the help of cloud calculator tools, such as Amazon Simple Month Calculator.[†] In the context of our work, the cost metric is calculated in dollars per month ($).

Scalability (s)

A software system is scalable when it maintains a satisfactory performance level even for increasing workloads. Yet, to check if a software system is indeed scalable or not is a complex activity because generating realistic workloads and measuring system performance in terms of how well the system is able to handle such workloads is not a trivial task. One practical way of evaluating the scalability of a software system is to put in place monitoring mechanisms that can reveal possible performance bottlenecks or when certain scalability assumptions are no longer valid.[24]

We propose a scalability metric that evaluates the capacity of the application to handle increasing workloads. This metric calculates the growth difference of the target application's average response time for increasing workloads. The following formula is used to compute the scalability score (s) for a given CDO when subjected to n increasing workloads, where the application's response time measured for each workload is given by $t_i(1 \leq i \leq n)$:

$$s = \frac{n-1}{\sum_{i=2}^{n} t(i) - t(i-1)}. \tag{4}$$

The rationale behind this formula is to compute the application's average response time growth as the workload increases. In order to illustrate how the scalability score is calculated, suppose two different CDOs (say $A1$ and $A2$) for a given application are each tested in the cloud with three increasing workloads (say 100, 200, and 300 concurrent users). In addition, suppose the application's measured response times for the three workloads are 100 ms, 200 ms, and 300ms,

[†] http://calculator.s3.amazonaws.com/index.html

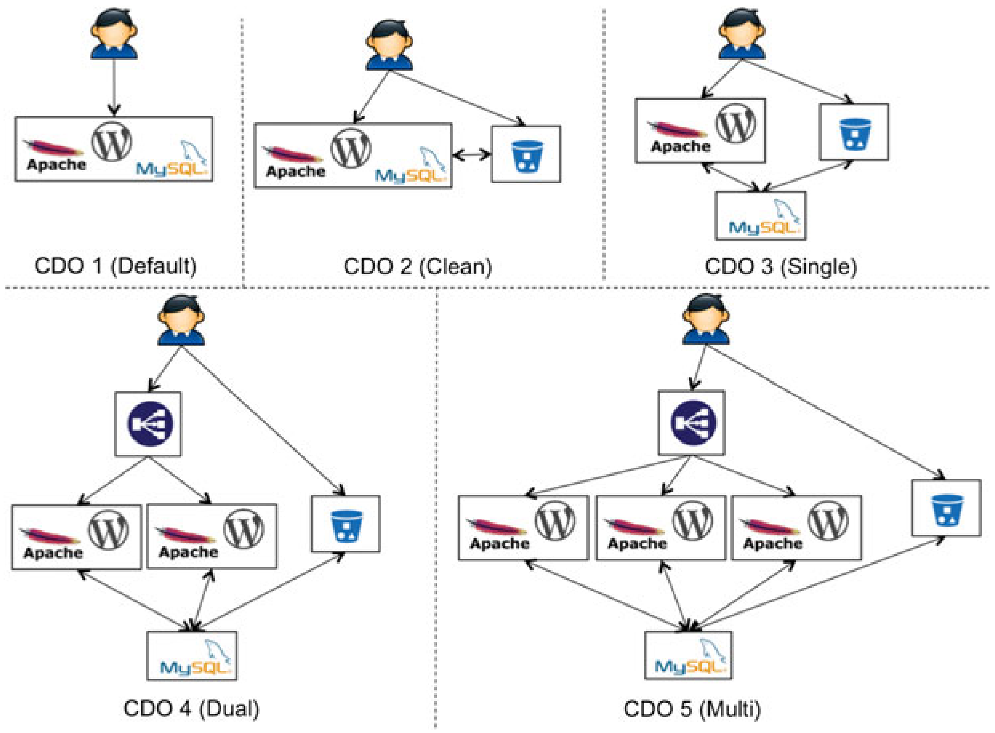


FIGURE 4 Five examples of cloud deployment options (CDOs) for WordPress [Colour figure can be viewed at wileyonlinelibrary.com]

for CDO $A1$, and 100ms, 150ms and 200ms, for CDO $A2$. Given those results, the scalability scores for $A1$ and $A2$ are calculated as follows:

$$s(A1) = \frac{3-1}{(200-100)+(300-200)} = \frac{2}{200} = 0.01 \tag{5}$$

$$s(A2) = \frac{3-1}{(150-100)+(200-150)} = \frac{2}{100} = 0.02. \tag{6}$$

Note that the scalability score of $A2$ is greater than that of $A1$, meaning that $A2$ is more scalable than $A1$ because its average response time grows more slowly than that of $A1$ as the workload increases.

3.2.2 | Define cloud architectures

This step relies both on the NFRs selected in the previous step as well as in the knowledge of the software architect on how to use cloud services to configure the target application. As an example, consider the Wordpress blogging app.[‡] Wordpress is a typical three-tier web application composed of an Apache web server, a PHP application server, and a MySQL database server. Based on the different resources and runtime configurations that can be used to deploy such an application in a public cloud provider such as Amazon Web Services (AWS), the software architecture might suggest the five CDOs illustrated in Figure 4 as possible deployment alternatives to be assessed and compared in terms of their efficiency, cost, and scalability.

The first (and simplest) CDO, called *Default*, is the one where all Wordpress components (ie, Apache, MySQL, and PHP) are deployed in a single VM. This should be the best CDO in terms of cost as it utilizes the minimum number cloud resources. The other four CDOs build upon this first one by adding further services and resources to improve application performance and scalability. The second CDO, called *Clean*, adds a cache and a content delivery network (CDN) service to improve the performance of the database server. The third CDO, called *Single*, deploys the database server in a separate VM. The fourth CDO, called *Dual*, adds a second application server instance and a load balancer. Finally, the fifth (and most complex) CDO, called *Multi*, adds a third application server instance.

[‡]Available at: https://wordpress.org/

The selection of those five CDOs of increasing complexity is justified from the software architect perspective, as this allows her/him to better assess the extent to which adding resources and services to the application would pay off in terms of its resulting cost, efficiency, and scalability.

3.2.3 | Evaluate cloud architectures

During this step, all deployment alternatives are evaluated with respect to all NFRs defined in Step 1. The goal is to verify how the system behaves with respect to the metrics previously defined for each NFR, taking into account realistic deployment scenarios. In our approach, we favor the use of real cloud experimentation,[5] as cloud simulation seems to be still unreliable for assessing efficiency-related NFRs, such as response time.[13]

Note that performing tests directly in the cloud does not necessarily limit the applicability of our approach to fully built systems, as software prototyping based on partial (but relevant) functionality is also a common industrial practice in early stages during architectural analysis.[25] Moreover, the availability of cloud benchmarking tools (eg, the work of Cunha et al[5]) that can be used to automate the experimental phase of the approach also helps to minimize the problems related to the time and cost necessary to run the tests.

Compare criteria and subcriteria

The architect must perform a pairwise comparison of all the elements (ie, a comparison of every combination of two elements) at each criteria or subcriteria level of the AHP hierarchy. As part of the evaluation process, the architect may state her/his preferences for each criteria/subcriteria by assigning them different weights. The AHP method suggests a scale for conducting pairwise comparisons that can range from 1 to 9, with the weight values indicating the relative importance of each element. For example, suppose that efficiency and cost are the only two criteria (or NFRs) being analyzed. If the architect considers those two NFRs to be of equal importance to the application, she/he can assign both weight 1. If, on the other hand, the architect considers efficiency to be more important than cost, she/he can assign the latter weight 1 and the former any weight value from 2 to 9, with 2 and 9 meaning that efficiency is, respectively, weakly more relevant and extremely more relevant than cost. This scale is not fixed by AHP (in general, it is very useful for comparing qualitative criteria) and the assigned weights can assume any value, including fractions.

This pairwise comparison stage is an essential step of the overall CDO assessment process, as it represents the point of view of the software architect regarding the expected trade-offs between the different NFRs being analyzed. That means that the same software architect might choose different NFR trade-offs for different target applications and that different software architects might choose different NFR trade-offs for the same target application.

Evaluate CDOs

Before ranking the CDOs, the architect is expected to provide values for each CDO regarding all the relevant NFRs. For example, considering the different deployment alternatives shown in Figure 4, the architect needs to test the target cloud application to determine its efficiency (in terms of response time, CPU utilization, and memory utilization), scalability, and cost for each of the five selected CDOs. Section 4 will give a concrete example on how those values can be calculated for a real cloud application based on the concepts and formulas defined in Section 3.1.

3.2.4 | Rank and analyze architectures

The next and final step of the proposed approach is the ranking and analysis of CDOs. This step consists of calculating the score vector, also known as priority vector, for each CDO. The goal is to apply the AHP multicriteria method based on the results of the previous steps to evaluate the best CDOs, considering the NFRs, criteria, and subcriteria evaluated previously. The solution is obtained with the eigenvalue formula $A.w = n.w$, where A is the comparison matrix built in Step 3, $w = (w_1, w_2, \dots, w_n)$ is the priority vector, and n is the main eigenvalue of A.

After the priority vector is calculated, the values of the scores assigned to each CDO are normalized such that their sum equals 1. This process is completely automated and carried out with the support of our AHP tool, which is described next. Once the scores are calculated, our tool ranks the CDOs so that they can finally be analyzed by the software architect to identify the most suitable deployment strategy(ies) for the target cloud application.

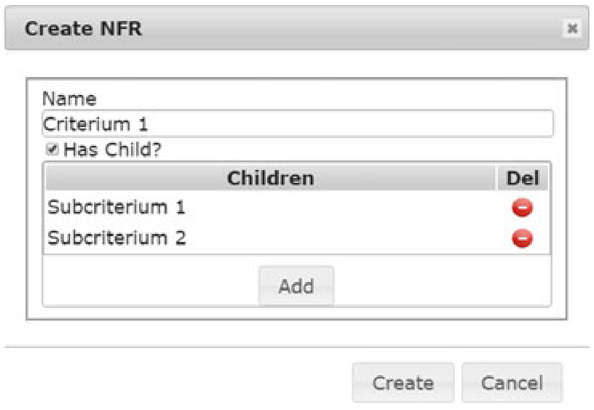


FIGURE 5 CA3 screen for defining nonfunctional requirements (NFRs) [Colour figure can be viewed at wileyonlinelibrary.com]

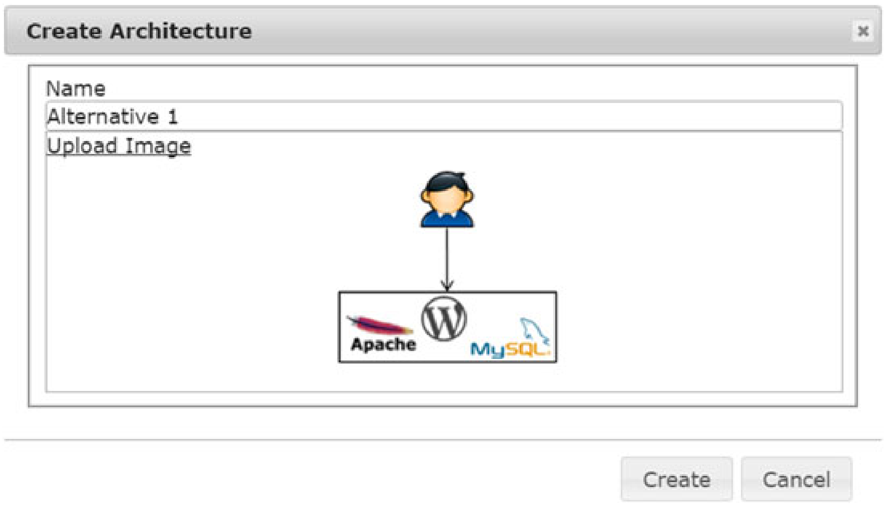


FIGURE 6 CA3 screen for defining cloud deployment options [Colour figure can be viewed at wileyonlinelibrary.com]

3.3 | Tool support

In order to aid the software architect with the execution of the proposed approach, we developed a tool, ie, *Cloud Application Architecture Appraiser (CA3)*, which helps to automate the steps described previously. In the following, we briefly illustrate how each of those steps is executed with the help of CA3.

3.3.1 | Define NFRs and criteria

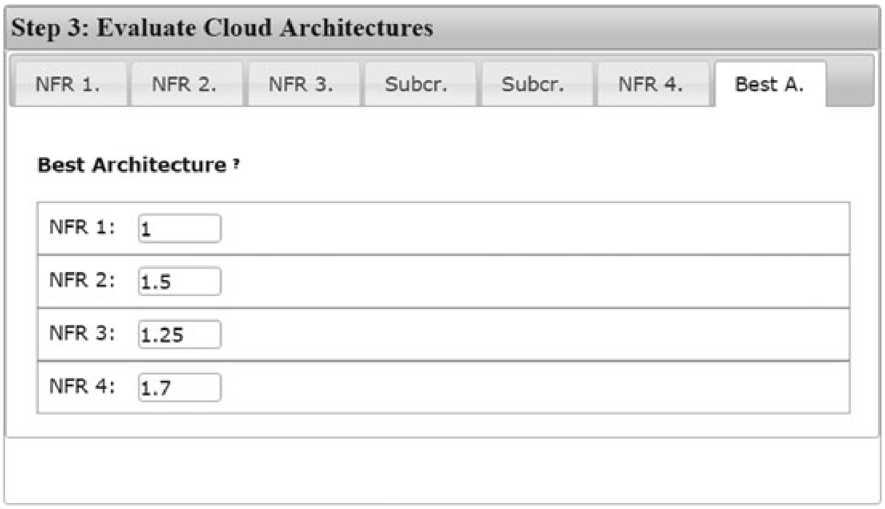
CA3 enables the architect to define all NFRs and criteria that are relevant for an application under evaluation. Figure 5 shows the tool screen for registering an NFR and defining its children (subcriteria). After the user has registered all relevant NFRs, then she/he selects which ones are going to be part of the evaluation. This step is completed when all the relevant NFRs to be evaluated are selected by the user.

3.3.2 | Define cloud architectures

At this step, the CA3 tool enables the architect to define the CDOs to be evaluated, as shown in Figure 6. The CDO illustrated in Figure 6 comprises a single VM that contains all software components necessary to run the Wordpress multitiered web application, ie, an Apache web server, a PHP application server, and a MySQL database server.

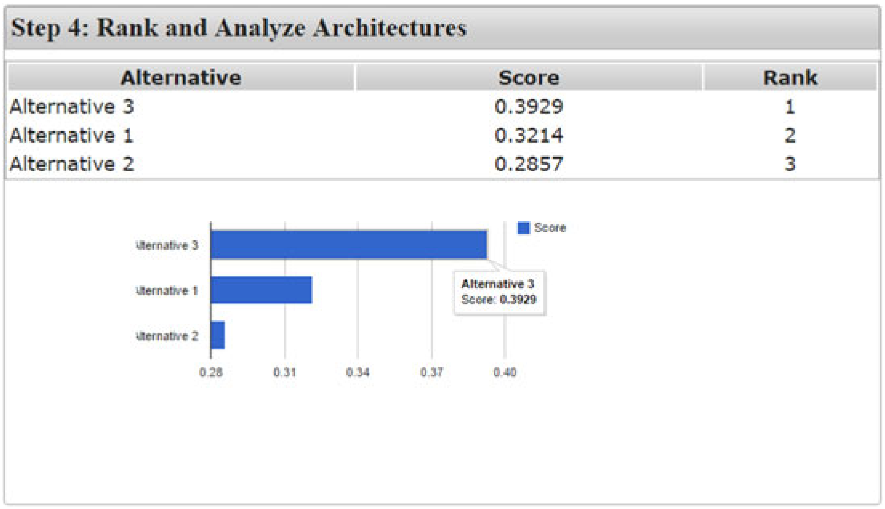
3.3.3 | Evaluate cloud architectures

At this step, CA3 enables the architect to assign different weights for the selected NFRs (see Figure 7) as well as to define the mathematical formulas necessary to calculate the scores for those criteria and subcriteria that can be measured



NFR 1:	1
NFR 2:	1.5
NFR 3:	1.25
NFR 4:	1.7

FIGURE 7 CA3 screen for defining weights for the selected nonfunctional requirements (NFRs) and their subcriteria



Alternative	Score	Rank
Alternative 3	0.3929	1
Alternative 1	0.3214	2
Alternative 2	0.2857	3

FIGURE 8 CA3 screen for ranking the selected cloud deployment options [Colour figure can be viewed at wileyonlinelibrary.com]

experimentally. Once the user provides all the values and formulas for all the elements that need to be evaluated, the tool generates the comparison matrix that serves as input data for the execution of the next step.

3.3.4 | Rank and analyze architectures

At this step, the CA3 tool computes the priority vector for the set of CDOs and ranks them according to their assigned scores, as shown in Figure 8. To this end, CA3 uses JAHP,[§] an AHP implementation in Java, and JAMA,[¶] a Java package for matrix manipulation.

3.4 | Usage and applicability

The CA3 approach/tool can be used across different phases of the software life cycle, from early design to after the application has been delivered into production.

During the design phase, the software architect can use CA3 to explore different deployment options as soon as a functional prototype of the application under development becomes available. This can help the architect to gain insights into how the application would behave in the target cloud environment with respect to multiple quality attributes, such

[§]Available at: http://www.di.unipi.it/morge/software/JAHP.html
[¶]Available at: http://math.nist.gov/javanumerics/jama/

as performance, scalability, and cost. Early feedback obtained from experimenting with this partial prototype can be used by the architect to refine and improve the application's deployment strategy while it is still being developed.

Once the application has been released into production, the architect might use CA3 to continuously reassess and possibly reconsider its deployment options, for example, by changing the current deployment topology to include load balancers and add/remove VMs to/from a replicated server. This can help the architect not only to anticipate potential quality issues that may raise at runtime but also to effectively address them once they are detected.

In terms of its applicability, our approach is particularly suited to assess stateless applications whose components/tiers are physically separated and can be independently deployed and executed. While this limits the use of our approach to distributed applications, which are stateful (eg, those which require sessions support in an application server), we believe the current trend toward the microservices architectural style[26] and DevOps,[27] in which application components are implemented and continuously delivered into production as small independent services with no shared state will have a significant impact on reducing the number of stateful monolithic applications being developed in the future. Therefore, we expect our approach to be even more useful as the industry continues to embrace microservices and their enabling DevOps practices into their software development pipeline.[28]

3.5 | Limitations and complimentary tools

Our AHP approach still requires a substantial effort from application architects, who must manually identify potential CDOs, collect metrics, and assign weight to assess a variety of NFRs, and then enter the collected data into the CA3 tool. In the following, we discuss how each of those tasks could be facilitated by extending our own tool or by reusing existing complimentary tools.

CDO identification

Currently, the architect must rely on its own domain expertise and common architectural knowledge to identity potential CDOs for the target application. As the number of potential configurations of a cloud application grows exponentially based on the number of components of the application, assessing all possible configurations may quickly become intractable. To reduce the CDO search space, we are considering extending CA3 with “smart” search-based selection mechanisms (eg, genetic algorithms) to automatically generate and eliminate (eg, based on a given pruning criteria) potential CDO candidates. The development and evaluation of such an extension is a new research topic in its own and will be addressed in our future work.

CDO assessment

To empirically assess different CDOs, architects can rely on automated cloud benchmarking tools, such as Cloud Workbench[29] and Cloud Crawler.[30] Indeed, as to what we explained in Section 4.2, in our work, we have used Cloud Crawler, which was originally developed in our research group, to automate deployment and performance testing of different CDOs for Wordpress in the Amazon cloud.

NFR metric collection

To support the collection, storage, and processing of a variety of system and application-specific NFR metrics, architects can benefit from a number of commercial and open source resource monitoring tools, such as Sensu[#] and Prometheus.[‖] Those tools are particularly suited to collect typical cloud-based metrics (eg, resource usage, performance, scalability), and thus are natural candidates to be integrated with our CA3 tool.

NFR weight assignment

Architects could use our tool to explore different weight values for each NFR of interest, as a way to assess their influence on the final CDOs ranking. In that regard, our tool could also be extended to automatically generate multiple weight

[#] https://sensu.io/
[‖] https://prometheus.io/

values (eg, within a given weight range) for each NFR so as to further facilitate this exploratory process. Again, we leave such an extension for our future work.

4 | EXPERIMENTAL EVALUATION

Experimentation is an appropriate and common approach for evaluating computational tools and methods.[31] We conducted an experimental study to demonstrate the feasibility of using our proposed AHP approach and tool to assess different CDOs for a real web application (Wordpress) deployed in a real cloud platform (AWS).

4.1 | Study planning

The goal of our study was to evaluate different CDOs for Wordpress from the perspective of the software architect responsible for deploying it in the cloud, taking into account different NFRs. The first author of this paper played the role of the Wordpress architect during the study. His goal was to assess and validate the applicability of our proposed method/tool to answer the following research questions.

RQ1 How does the proposed method/tool support the architect's decision-making process?
RQ2 How does the proposed method/tool aid in analyzing the impact of different NFRs in the selected CDOs?
RQ3 Do the results produced by the proposed method/tool reflect the expectations of the architect?

Our choice for WordPress as the target cloud application and AWS as the target cloud platform is justified as both are well known and widely used solutions in their respective domains. Wordpress is a popular multitier web blogging application that is very suitable for cloud-based studies[6] as its deployment architecture can be easily varied and tested with a range of cloud services (VMs, CDNs, load balancers, and so on). Meanwhile, AWS is the most popular public IaaS cloud provider,[32] with more than 70% market share, offering a wide range of cloud resources and services as well as a rich multilanguage support documentation for application developers.

4.2 | Study design and preparation

The study consisted of the following steps, which involved executing not only the proposed method but also the activities necessary to generate its required input data.

1. *Application deployment*. We deployed Wordpress in the US EAST region of the AWS cloud using five different CDOs, as previously shown in Figure 4. All VMs created were of type *m3.large*, which provides enough capacity (2 vCPU cores, 7.5 GB of RAM, and 32GB of SSD storage) to accommodate the needs of all Wordpress components. As explained in Section 3.2.2, those five CDOs varied in terms of the number of VMs created and the set of cloud services (such as CDN and load balancing services) used.
2. *Load testing*. We tested the performance and scalability of Wordpress using the five CDOs under three different workload levels. To this end, we used a load testing tool to create 100, 300, and 600 concurrent Wordpress clients. During the tests, each client invoked Wordpress by requesting a mix of its provided blogging operations, ie, create a post; query a post; list all posts; edit a post; and delete a post. Tests involving the same CDO and the same workload were repeated three times in different hours and days. We used Cloud Crawler,[30] a performance testing tool, to automate the deployment and execution of the Wordpress components in the Amazon cloud.
3. *Data collection*. During the tests, we collected relevant application performance information (eg, response time, CPU consumption, memory consumption) that served as input for calculating the efficiency and scalability metrics, as defined in Section 3.1. In addition, we collected price information for the AWS cloud resources and services used by each CDO, which served as input for calculating the cost metric.
4. *Method execution*. Finally, we executed our proposed AHP method by using the CA3 tool to assign scores and rank the selected Worpress CDOs based on the cost and performance data collected from the AWS cloud.

4.3 | Study execution and results

After load testing and collecting the performance and cost data for each selected Wordpress CDO in the AWS cloud, the software architect used the CA3 tool to execute the two remaining steps of our proposed AHP method, as described later.

4.3.1 | Evaluate cloud architectures

Compare criteria

Initially, the architect established his expected tradeoffs regarding the pairwise comparison of the three NFRs, ie, efficiency, scalability, and cost, by assigning them with different weights. The architect considered efficiency as the most important NFR for Wordpress. He also considered cost slightly less important than efficiency and slightly more important than scalability. Finally, he considered scalability moderately less important than efficiency. Since scalability was considered as the least important criteria, it received the minimum weight, ie 1. Cost, in turn, received weight 2 (weak importance), and efficiency received weight 4 (moderate importance).

Compare subcriteria

The software architecture then established similar tradeoffs for the three efficiency subcriteria, i.e., CPU utilization, memory utilization and response time. The architect considered CPU utilization and memory utilization of equal importance to Wordpress, with response time being slightly more important than both. Since both resource utilization subcriteria were considered the least important ones, both received weight 1. Response time, in turn, received weight 2 (weakly more important).

Compare architecture

After defining the required pairwise comparison weights for each criteria and subcriteria, the architect provided the tool with required metric values for each CDO regarding all relevant NFRs. This was done based on the performance and cost data collected from deploying and load testing WordPress in the AWS cloud, as described in the previous section.

The efficiency, scalability, and cost results observed for the five Wordpress CDOs in AWS, and their respective metric scores as calculated by the CA3 tool, are discussed later.

Efficiency results

Figure 9 shows the average response time (in milliseconds) measured for each CDO for each workload. It is easy to notice that, for 100 concurrent users, the lowest workload tested, all CDOs offer a fairly similar response time. This means that, under low demand, those five different deployment options have little influence on WordPress performance. For 300 concurrent users, we can see a difference of one second between the CDOs with the lowest and the highest response time. Lastly, for 600 concurrent users, it becomes clear that only CDOs 4 ((*Dual*)) and 5 ((*Multi*)) can handle such level of demand, with CDOs 1 (*Default*), 2 (*Clean*), and 3 (*Single*) achieving considerably higher response times.

Overall, *Default* has the highest average response time and, therefore, according to Equation 1, the lowest response time score. *Multi*, in turn, has the lowest average response time and, by the same formula, has the highest response time score, being approximately six times faster than *Default*. Figure 10 shows the pairwise response time scores calculated by CA3 for all five CDOs.

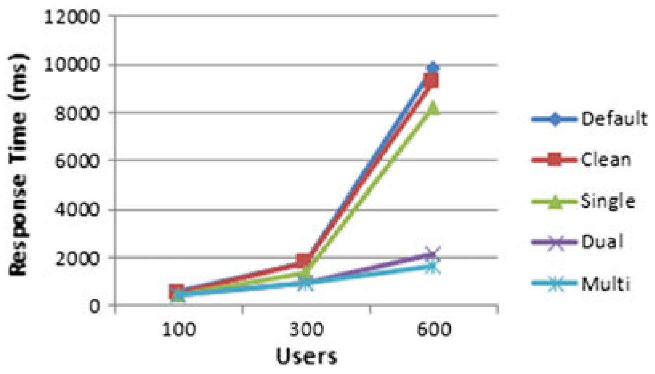
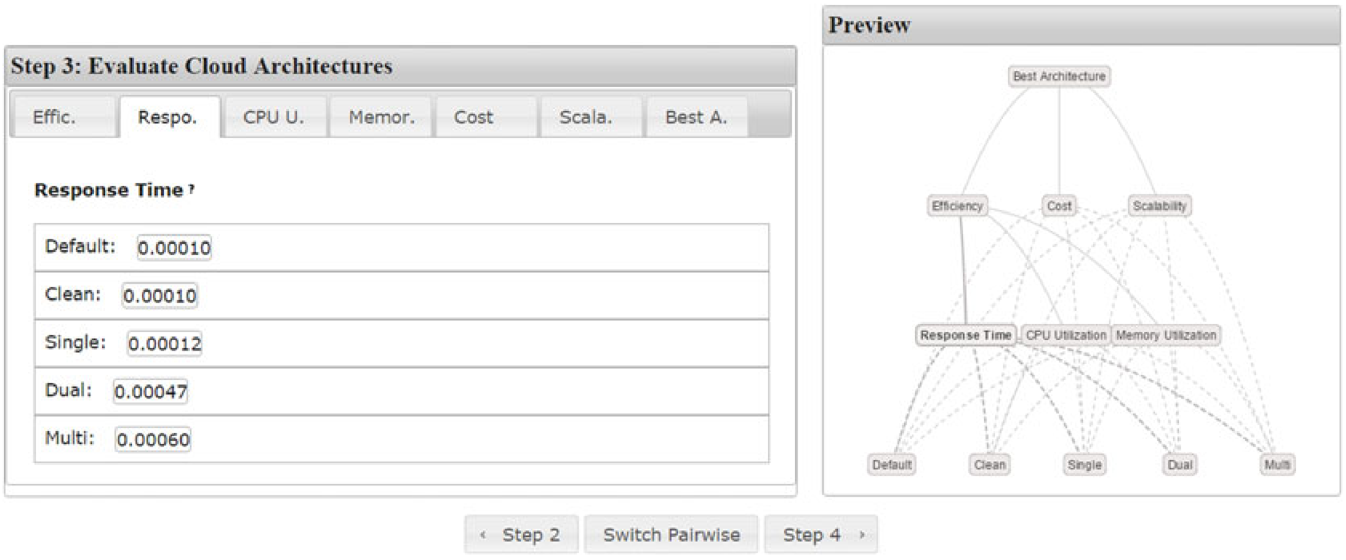


FIGURE 9 Average response times (ms) measured for the five cloud deployment options [Colour figure can be viewed at wileyonlinelibrary.com]


Step 3: Evaluate Cloud Architectures

Effic. | Respo. | CPU U. | Memor. | Cost | Scala. | Best A.

Response Time ?

Default:	0.00010
Clean:	0.00010
Single:	0.00012
Dual:	0.00047
Multi:	0.00060

FIGURE 10 Pairwise evaluation of the five cloud deployment options with respect to response time [Colour figure can be viewed at wileyonlinelibrary.com]

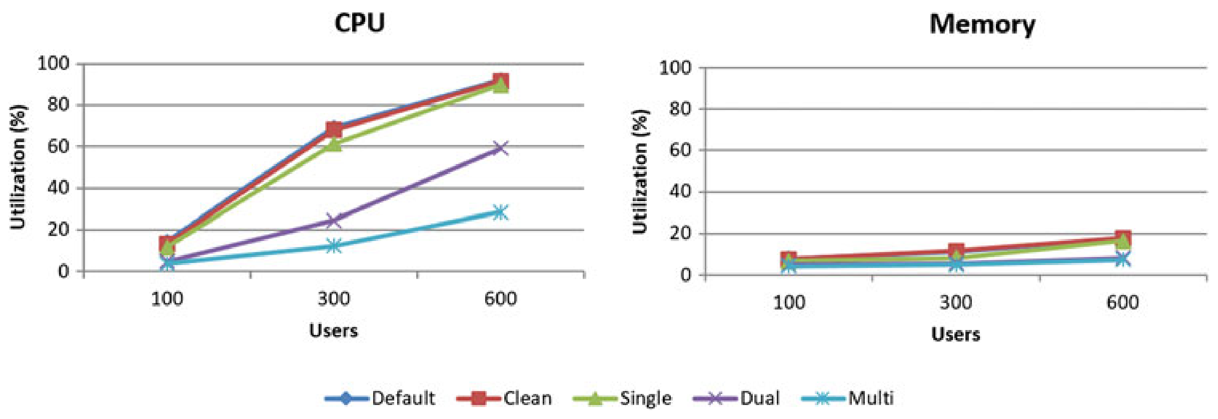
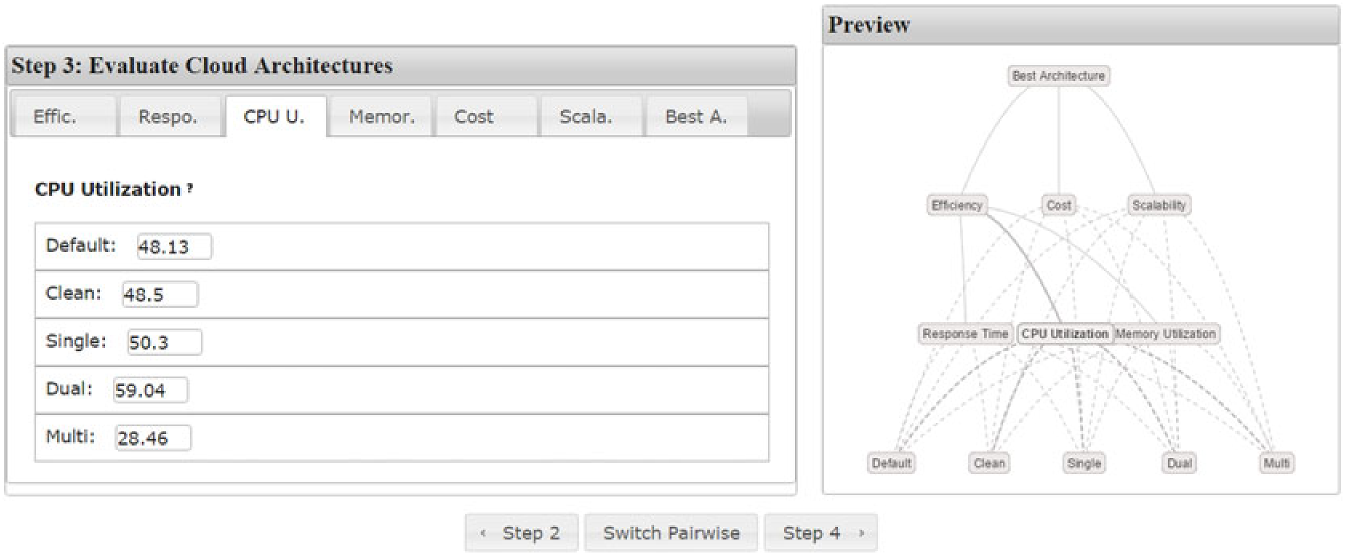


FIGURE 11 Average CPU (left) and memory (right) utilization measured for the five cloud deployment options [Colour figure can be viewed at wileyonlinelibrary.com]

Figure 11 shows the average resource (ie, CPU and memory) utilization measured for each CDO for each workload. For 100 concurrent users, we can see that all five CDOs are clearly overprovisioned, especially *Dual* and *Multi*, whose average CPU consumption was well below 10% for both metrics. For 300 users, *Default*, *Clean*, and *Single* approach a near optimal CPU usage, whereas *Dual* and *Multi* are still overprovisioned. Finally, for 600 users, *Default*, *Clean*, and *Single* all reach a CPU saturation level, as they no longer can handle the application demand, whereas *Dual* grows closer to an optimal CPU usage, with *Multi* still being overprovisioned, albeit not as much as for the two lower workloads.

In contrast to the CPU utilization results, we can see on the left side of Figure 11 that WordPress is not a memory-intensive application (at least not for serving the kinds of client requests generated during the load tests executed in the AWS cloud), as all its five deployment options consumed less than 20% of the available VM memory in the three workload scenarios evaluated. As a consequence, none of the five CDOs were penalized when computing their average memory consumption scores. Yet, if the generated client requests were changed in a way to involve more memory-intensive Wordpress operations, such as image and video processing, for example, the application's memory usage would certainly increase and perhaps have a greater impact in its observed performance results.

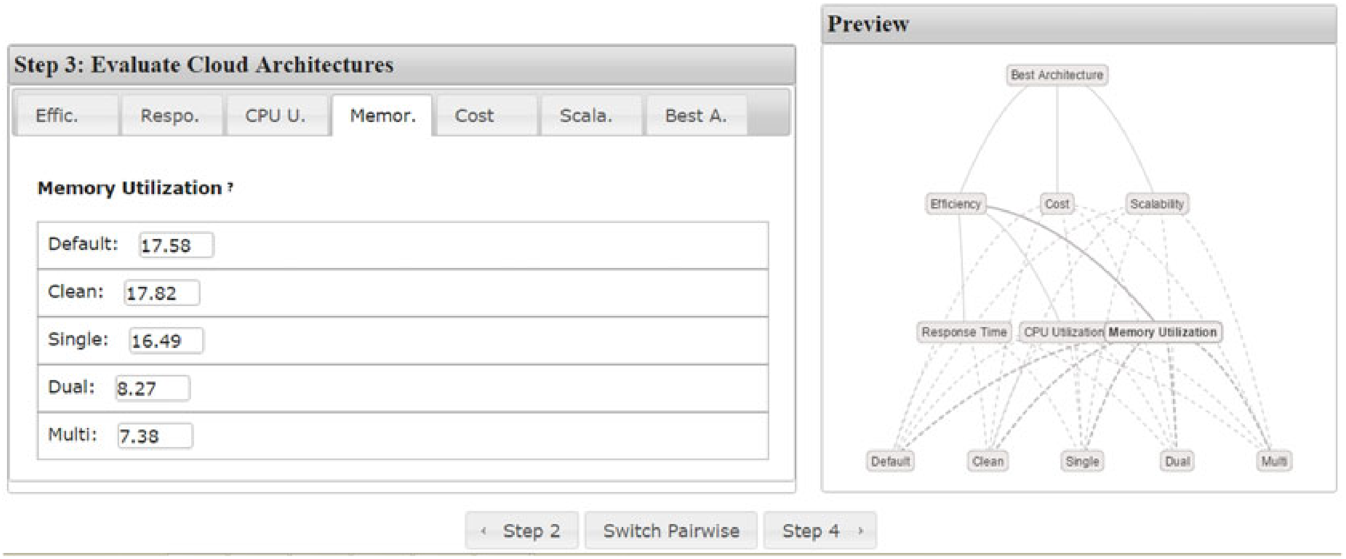
Figures 12 and 13 show the pairwise CPU and memory consumption scores, respectively, as calculated by CA3 for all five CDOs.


Step 3: Evaluate Cloud Architectures

CPU Utilization

Default:	48.13
Clean:	48.5
Single:	50.3
Dual:	59.04
Multi:	28.46

FIGURE 12 Pairwise evaluation of the five cloud deployment options with respect to CPU utilization [Colour figure can be viewed at wileyonlinelibrary.com]


Step 3: Evaluate Cloud Architectures

Memory Utilization

Default:	17.58
Clean:	17.82
Single:	16.49
Dual:	8.27
Multi:	7.38

FIGURE 13 Pairwise evaluation of the five cloud deployment options with respect to memory utilization [Colour figure can be viewed at wileyonlinelibrary.com]

Scalability results

According to Equation 6, the lower the growth differences between the average response times measured for a given CDO for increasing workloads, the higher its scalability score. Figure 14 shows the average response time growth differences for each of the five Wordpress CDOs and Figure 15 shows their scalability scores as calculated by CA3. From those figures, we can see that *Default*, *Clean*, and *Single* are by far the worst options in terms of scalability, whereas *Dual* and *Multi* are the best, with very similar results.

Cost results

There are three main resources that affect cost by deploying a typical web application in a public cloud such as AWS, ie, compute, storage, and network. In the case of the five Wordpress CDOs considered in our work, the network cost is virtually inexistent due to the nature of the requests and workloads being considered, and the current bandwidth pricing. Thus, even considering every network related attribute, only the CDN service is truly relevant for the evaluation of the cost NFR. Furthermore, there are no additional cloud services in any of the five CDOs, except load balancing, which is commonly found in the more complex WordPress setups.

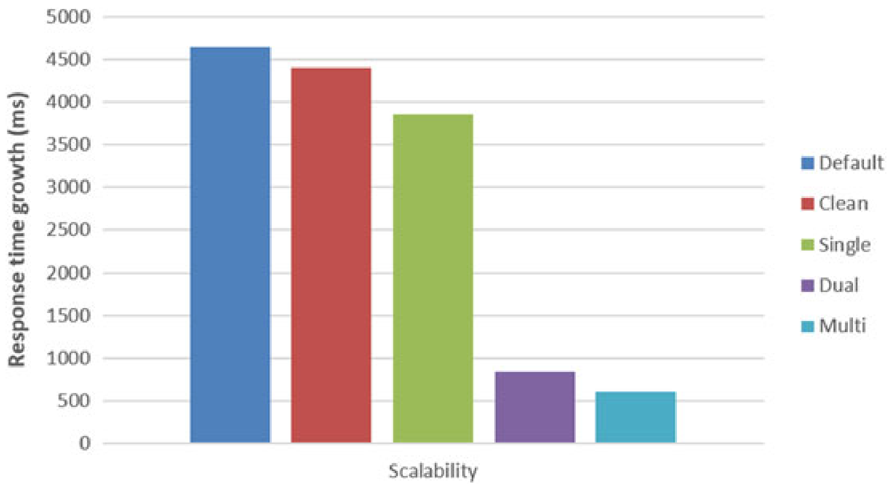
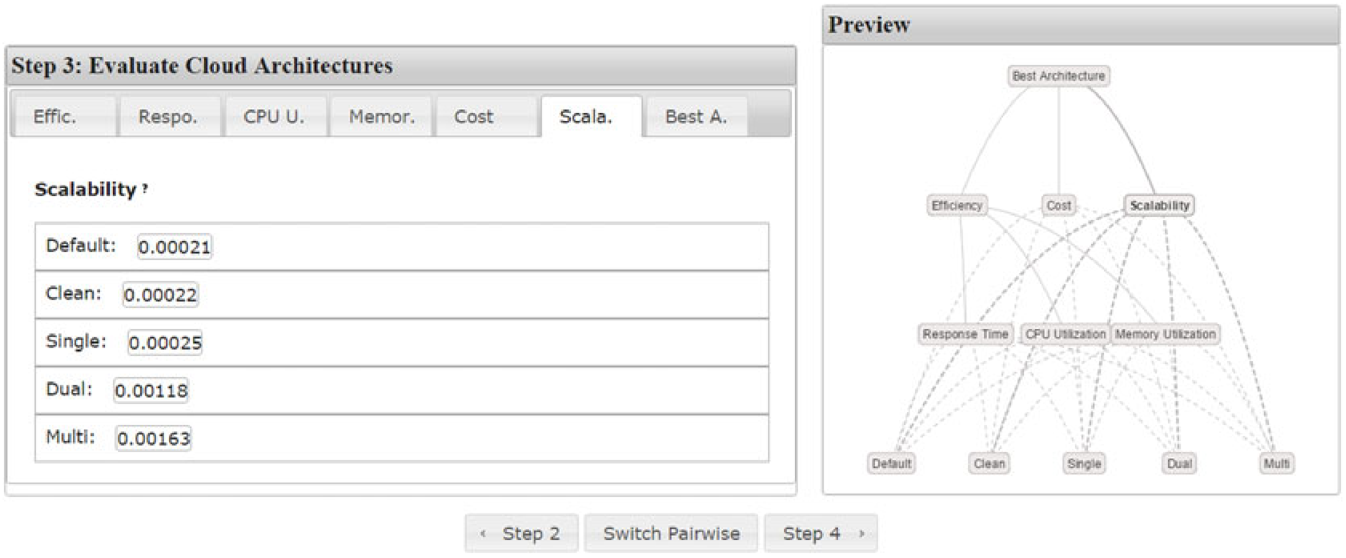


FIGURE 14 Average response time growth differences measured for the five cloud deployment options [Colour figure can be viewed at wileyonlinelibrary.com]



Scalability

Default:	0.00021
Clean:	0.00022
Single:	0.00025
Dual:	0.00118
Multi:	0.00163

FIGURE 15 Pairwise evaluation of the five cloud deployment options with respect to scalability [Colour figure can be viewed at wileyonlinelibrary.com]

Figure 16 shows the cost distribution (in US$/month) of each CDO, as calculated by Amazon's simple calculator tool. Note that the decisive factor for the total cost impact is to compute resources by a large amount, with the cost increasing from *Default* and *Clean*, the two CDOs with the least number of compute resources, to *Dual* and *Multi*, the ones with the highest number of compute resources.

Figure 17 shows the cost scores calculated by CA3 for each CDO. Note that the scores are calculated by inverting the cost estimate values, implying that deployment options with higher costs are worse alternatives from the cost perspective.

4.3.2 | Rank and analyze cloud architectures

The final step of the proposed approach is the ranking and analysis of the CDOs based on their assigned global scores. The CA3 tool calculates these scores and ranks the CDOs automatically, based on the values provided by the software architect in the previous steps. Table 2 shows the scores and ranks calculated for the five CDOs, for each of the three workloads, considering all criteria (NFRs) and subcriteria.

In order to facilitate the analysis, we discuss the results for each different workload. As we can see from Table 2, with the 100 users workload, the five CDOs are ranked in the following order (from best to worst): *Default*, *Clean*, *Multi*, *Single*, and *Dual*. This rank makes sense as the low number of users results in very short response times for both simple and

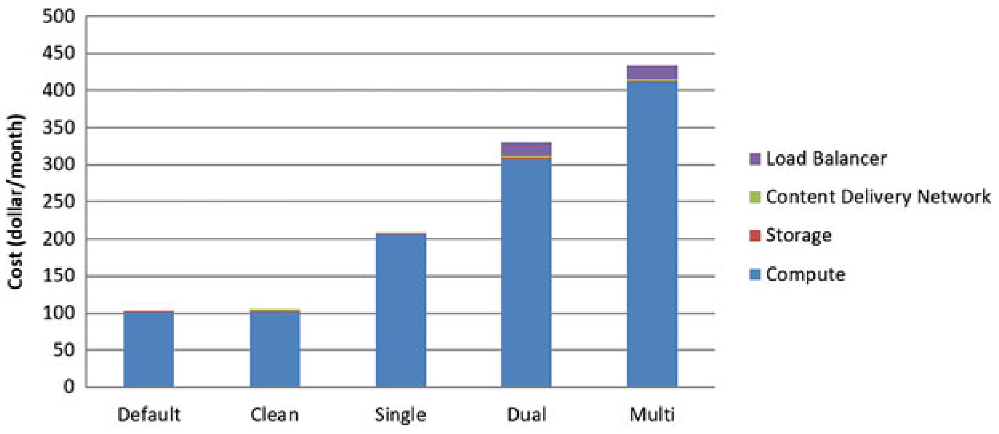
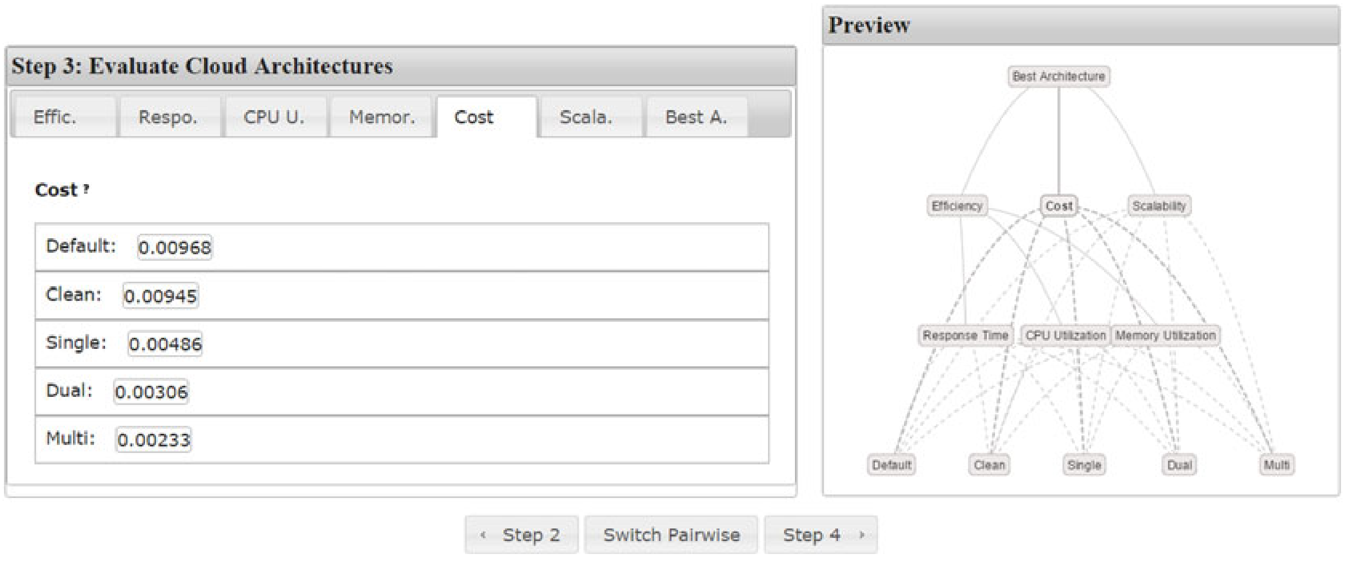


FIGURE 16 Estimate cost values for the five cloud deployment options [Colour figure can be viewed at wileyonlinelibrary.com]



Cost

Default:	0.00968
Clean:	0.00945
Single:	0.00486
Dual:	0.00306
Multi:	0.00233

FIGURE 17 Pairwise evaluation of the five cloud deployment options with respect to cost [Colour figure can be viewed at wileyonlinelibrary.com]

TABLE 2 Analytic hierarchy process results for the five cloud deployment options (CDOs) considering all NRFs. Results for the top ranked CDO of each workload are highlighted in bold face

		Efficiency						
Workload	CDO	Resp. Time	CPU Usage	Mem. Usage	Scalability	Cost	Total	Rank
100	**Default**	**0.0482**	**0.043**	**0.0356**	**0.0087**	**0.0941**	**0.2296**	**1**
	Clean	0.0539	0.0395	0.0348	0.0092	0.0919	0.2293	2
	Single	0.0563	0.0348	0.0302	0.0105	0.0472	0.179	4
	Dual	0.0624	0.0145	0.023	0.048	0.298	0.1777	5
	Multi	0.0649	0.0111	0.0193	0.0664	0.0227	0.1843	3
300	**Default**	**0.0404**	**0.0419**	**0.0387**	**0.0087**	**0.0941**	**0.2238**	**1**
	Clean	0.0402	0.0413	0.0398	0.0092	0.0919	0.2224	2
	Single	0.0526	0.0373	0.0281	0.0105	0.0472	0.1758	5
	Dual	0.0757	0.0149	0.019	0.048	0.0298	0.1873	4
	Multi	0.0768	0.0075	0.0173	0.0664	0.0227	0.1906	3
600	Default	0.0205	0.0293	0.0372	0.0087	0.0941	0.1899	4
	Clean	0.0218	0.0296	0.0377	0.0092	0.0919	0.1901	3
	Single	0.0247	0.0307	0.0349	0.0105	0.0472	0.148	5
	Dual	0.0953	0.036	0.0175	0.048	0.0298	0.2266	2
	Multi	**0.1234**	**0.0173**	**0.0156**	**0.0664**	**0.0227**	**0.2454**	**1**

complex CDOs, whereas the more complex CDOs are penalized for their higher cost and waste of computing resources (due to overprovisioning).

Note that the response time scores for *Default* (0.0482), *Dual* (0.0624), and *Multi* (0.0649) are relatively close when compared to their cost scores (0.0941, 0.48, and 0.0227, respectively). This means that the two 1more complex CDOs (*Dual* and *Multi*) are not beneficial enough in terms of their efficiency to justify their higher costs. For the two simpler CDOs, ie, *Clean* and *Default*, we can see that their scores are very close, with *Default* showing a slight advantage. This happens because the gains offered by *Default* in terms of resource utilization and cost are slightly greater than those offered by *Clean* in terms of response time and scalability. textitSingle also has a low global score, being the fourth in the rank. This also due to its higher cost when compared to the simplest CDOs even though it offers better response time and scalability.

For the 300 users workload, we can see that resource utilization (CPU and memory) increases for all CDOs except for *Dual* and *Multi*. This is because, even though there are more users accessing the applications, most CDOs remain *overprovisioned*. Therefore, simpler options such as *Default* and *Clean*, which offer better resource utilization and still maintain a relatively low response time, are good choices for this workload. The results for scalability and cost are very similar to those obtained for the 100 users workload. The final ranking is also very similar to that calculated for the previous workload, with *Default*, *Clean*, and *Multi* being the best three options in that order. However, note that, this time, *Single* and *Dual* switched places as the worst and second worst options. This happened because, while *Single* still has a higher cost, it does not provide much more in terms of efficiency when compared to the other four options.

Finally, for the 600 users workload, we can see that *Dual* gets the best score for CPU utilization because, while *Default*, *Clean*, and *Single* are penalized for being overloaded, *Multi* is still *overprovisioned*. The response time scores for *Dual* and *Multi* increase significantly as they are the only ones to provide low response times for this workload. Another interesting fact is that the *Default* and *Clean* options offer very similar results and are both good candidates for lower workloads and bad candidates for higher workloads. The final ranking for the 600 users workload is as follows: *Multi*, *Dual*, *Clean*, *Default*, and *Single*.

We can conclude that *Default* preserves a superior position compared to *Clean* for all three workloads and it is the best CDO for WordPress under low to moderate workloads (up to 300 concurrent users). However, *Default* does not sustain its position for a much higher workload (600 concurrent users), for which *Multi* becomes the best option.

4.4 | Study discussion

An experienced professional might know beforehand, if a deployment option meets the needs of a client better than other options just by studying, their measured performance results, as shown in Figures 9 and 11. However, it might not always be a trivial task to identify which CDO is more adequate, especially when the options considered are too complex or when you have too many options. Sometimes CDOs might end up with similar global scores. In this case, the architect might choose to refine the deployment configurations and start again at the previous (or even the first) step of our approach. If the results are not satisfactory yet again, this process can be executed iteratively until the results meet the architect's expectations.

Moreover, there may be situations in which the architect cannot rely on the final AHP results alone. An example would be if one has an SLA that requires the response time of all WordPress operations to be no longer than two seconds. In our example evaluation, this would imply in the removal of CDOs *Default*, *Clean*, and *Single* in the evaluation step of our approach when considering the 600 users workload.

It is also possible to combine our proposed approach with other methods, as it was designed to support the architect in her/his decision-making process. In fact, the architect is encouraged to use any other relevant information that can further enhance the results produced by the CA3 tool.

In the following, we revisit our three research questions and discuss how the results described in the previous sections have contributed to answer each of them.

RQ1 How does the proposed method/tool support the architect's decision making process?
During the evaluation, it was noticed that the process of comparing different CDOs considering multiple NFRs can become complex depending on the number of options, criteria, subcriteria, and workloads analyzed. For this reason, the method and tool become very relevant as they support the architect in performing the required pairwise comparison of each CDO for each NFR. In this regard, the architect only needs to provide the necessary input data for the CA3 tool, which then automatically calculates the scores and ranks of all CDOs based on the AHP method.

RQ2 How does the proposed method/tool aid in analyzing the impact of different NFRs in the selected CDOs?
Based on the results described in the previous section, we can conclude that employing the AHP method to analyze different CDOs from the perspective of different NFRs can go a long way in helping the architect to understand how the different options are affected by each NFR. In particular, the use of quantitative data obtained from real experiments carried out in the AWS cloud proved invaluable to reduce the subjectivity of the approach, and further helped the architect in making relevant and informed decisions.

RQ3 Do the results produced by the proposed method/tool reflect the expectations of the architect?
In general, increasing the number of computational resources, such as VMs, for a cloud application, also increases its nonfunctional quality attributes, such as response time and scalability. However, this increase in computational capacity has a direct impact on the application's operational cost. It is important to mention that, when different NFRs and deployment options are considered, it is hard to predict which will be the best deployment strategy, considering the multicriteria nature of this problem. For example, our evaluation results have shown that, from low to moderate workloads (up to 300 concurrent users), simpler (ie, less resourceful) CDOs are recommended for deploying Wordpress in the AWS cloud mainly because of their low cost and their performance being comparable to that offered by more complex (ie, more resourceful) CDOs. However, for higher workloads, more complex CDOs may be recommended as simpler CDOs tend to bottleneck.

In short, our results have shown that knowing which deployment options are the best for a given cloud application is not a trivial problem as the correct answer depends on multiple factors, including the most relevant NFRs for the application and its expected workload levels. That is why having a multicriteria based architecture assessment approach and tool, such as CA3, is fundamental to support cloud application developers in investigating the trade-offs between different deployment alternatives.

4.5 | Threats to validity

As with any empirical evaluation, the validity of our study may be threatened by a few relevant limitations. One first limitation was the fact that the main researcher responsible for designing and developing CA3 also played the role of the software architect conducting the evaluation. As we have mentioned previously in this paper, the main purpose of our study was not to perform a usability assessment of our tool, but rather to validate our proposed approach by showing its use and benefits in practice. In that regard, having the tool's main developer (its most expert user) as the architect conducting the evaluation was key for the purpose of our study.

Another threat was that we limited the number of criteria (NFRs) to three, namely, efficiency (which was further refined into response time, CPU usage, and memory usage), scalability, and cost. As discussed in the previous sections, this is not a limitation of the approach per se, as both method and tool are capable of addressing any number of NFRs and subcriteria. We focused on only those three NFRs to avoid cluttering the exposition and because they were the most commonly used for evaluating cloud based applications. Naturally, using more NFRs and subcriteria could have impacted the results obtained in our study as more options would have affected the AHP calculations and ranking.

Finally, our study was also limited by the fact that we only assessed the approach with one application. As mentioned before, the target Wordpress application is commonly recommended for cloud studies due to its multitier architecture that enables multiple deployment configurations with varying NFRs and results. As our main goal was to demonstrate the validity of our approach and tool in supporting the evaluation of different deployment options for a single application, it would have been even more interesting to have analyzed more deployment variations of Wordpress than analyzing a different application.

5 | RELATED WORK

The application of MCDMs in requirements engineering has been discussed before in perspectives different from our work. For example, the work of In et al[9] presents the multicriteria preference analysis requirements negotiation that aims at aiding requirements negotiation. The approach uses MCDM to establish preferences among functional requirements that can help to negotiate requirements among stakeholders. Analyzing architectural trade-offs based on NFRs is also a very well-known problem in software engineering[20] that we tailor to the specific cloud domain.

CloudAdvisor[2] allows cloud users to compare cloud providers in terms of their estimated price and performance for a given application workload, subject to several user preferences (eg, maximum budget, throughput expectation, energy saving). Differently from our work, CloudAdvisor focuses more on low-level characteristics of the workload and how different cloud providers match (using simulation) this workload based on the nature of their resources. Similarly, the work of Frey et al[3] simulates application performance and cost to compare how different configurations of an application can be migrated to cloud providers. The approach employs genetic algorithms to find the most optimal deployments with respect to VM configurations required for the application. The work by Garg et al[33] tackles the problem of selecting the most appropriate cloud providers that satisfy users' requirements. The approach is based on a catalog of service measurement indexes[34] that are defined for several quality attributes such as accountability, agility, performance, and security. The authors proposed a framework that also uses AHP to rank the different cloud providers based on the values attributed for the different service measurement indexes. That approach is very interesting but tackles a different problem than the one discussed in our work. While we focus on finding the best deployment options for a specific application based on multiple user-provided criteria, the aforementioned work[33] tries to find the most appropriate cloud providers based on generic service metrics.

Sodhi and Prabhakar[35,36] proposed an approach for assessing platform suitability, in early stages of development, to choose an environment for hosting an application. The approach is very generic as it tries to consider different types of clouds (IaaS, PaaS) and different types of virtualization technologies. The criteria for comparison is also very broad and considers several types of quality attributes (security, performance, fault tolerance, and others). The comparison criteria are drawn from analysis of documentations available from cloud providers and their assessment by architects. We believe that, though this approach has value for having a general understanding among different cloud options, without precise data collected from real experiments, it is difficult to achieve precise results. Moreover, trying to compare different types of clouds with the same attributes might lead to bias. The approach also does not address variations on the deployment options and its corresponding costs for variations of the application.

Lloyd et al[37,38] proposed a workload cost prediction methodology to support developers in determining cost effective deployment options for cloud-based SOA applications. Despite sharing some of our goals, that work focuses only on analyzing performance and cost related metrics, whereas we propose a more general approach to assess and rank deployment options based on multiple quality attributes. In addition, Lloyd et al used a modeling approach to estimate application performance in the cloud instead of measuring the actual application performance experimentally, as we do in our work. However, a similar modeling approach could be used to estimate the AHP score for some of the quality metrics used in our work, in cases in which full experimentation may be deemed too costly or unfeasible. Therefore, we believe that the work by Lloyd et al is complementary to our work.

The approach by Chung et al, the most similar to our work, combines goal-oriented requirements engineering with simulation techniques to evaluate system architectures for private clouds.[7] The NFR framework[8] is used to deal with stakeholder goals and to build and analyze the goal structure through the Softgoal Interdependency Graph. Based on the Softgoal Interdependency Graphs constructed, simulations are performed (using CloudSim[39]) to evaluate different design options for a system that will be deployed in a private cloud. Though interesting and similar to our work, we tackle a slightly different problem using a different approach. We focus on analyzing applications that are configured using real public cloud services instead of a private cloud. Moreover, we do not resort to simulations and prefer real experimentation as it is the most reliable way of evaluating cloud applications.[13] Finally, we also use an MCDM method to analyze that the mutual influence NFRs can have on the chosen deployment options.

6 | CONCLUSIONS AND FUTURE WORK

As more organizations consider migrating or developing systems in the cloud, a common difficulty developers face is how to best configure their applications using a myriad of cloud services. This work presented an approach that considers NFRs as key drivers for analyzing, based on a multicriteria method, the best deployment options for cloud applications. Results from a real application (WordPress) deployed on a popular cloud provider (Amazon) showed that, depending on the quality parameters considered and characteristics of the application, the results for the best deployment options can vary a lot based on the NFRs selected and also the cloud services chosen.

Regarding our future work, we plan to investigate more NFRs such as security and fault tolerance and devise specific metrics for assessing these NFRs in cloud applications. Another interesting topic for future research is investigating the extent to which our approach can be adapted or extended to accommodate more innovative deployment services (eg, containers[40]) and architectures (eg, serverless[41]). Moreover, as the number of possible deployment options even for a relatively small application can be quite large, we plan to improve our approach and tool by providing a more intelligent mechanism (eg, the work of Etemaadi et al[42]) for selecting deployment options and thus rely less on the architect's expertise. Finally, we plan to conduct a more systematic evaluation of our approach and tool with other applications and deployment scenarios, eg, multiclouds,[43] involving more (nonbiased) subjects. Potential questions to explore in that direction are as follows.

- How do rankings provided by our approach vary with respect to expert opinion or the opinion of typical software engineers?
- What is the value to practitioners of the CDO rankings provided by our approach?
- To what extent can the application of our approach be automated?
- What is the effort involved in applying our approach in other application domains and cloud providers? How does this effort relate to alternative architecture assessment approaches?
- How do our approach's effectiveness and effort vary based on the type of application being evaluated? Is our approach consistently accurate and efficient to apply regardless of the application domain and/or architectural style being considered?

ACKNOWLEDGEMENTS

The authors thank the reviewers for their invaluable comments and feedback, which greatly contributed to improve the paper. Nabor C. Mendonça is partially supported by CNPq under grants 313553/2017-3 and 207853/2017-7.

ORCID

Americo Sampaio http://orcid.org/0000-0003-0267-268X

REFERENCES

1. Armbrust M, Fox A, Griffith R, et al. A view of cloud computing. *Commun ACM*. 2010;53(4):50-58.
2. Jung G, Mukherjee T, Kunde S, Kim H, Sharma N, Goetz F. CloudAdvisor: a recommendation-as-a-service platform for cloud configuration and pricing. Paper presented at: IEEE Ninth World Congress on Services; 2013; Santa Clara, CA.
3. Frey S, Fittkau F, Hasselbring W. Search-based genetic optimization for deployment and reconfiguration of software in the cloud. Paper presented at: International Conference on Software Engineering (ICSE); 2013; San Francisco, CA.
4. Saripalli P, Pingali G. MADMAC: multiple attribute decision methodology for adoption of clouds. Paper presented at: IEEE 4th International Conference on Cloud Computing; 2011; Washington, DC.
5. Cunha M, Mendonça N, Sampaio A. A declarative environment for automatic performance evaluation in IaaS clouds. Paper presented at: IEEE Sixth International Conference on Cloud Computing; 2013; Santa Clara, CA.
6. Borhani AH, Leitner P, Lee B-S, Li X, Hung T. WPress: an application-driven performance benchmark for cloud-based virtual machines. Paper presented at: IEEE 18th International Enterprise Distributed Object Computing Conference (EDOC); 2014; Ulm, Germany.
7. Chung L, Hill T, Legunsen O, Sun Z, Dsouza A, Supakkul S. A goal-oriented simulation approach for obtaining good private cloud-based system architectures. *J Syst Softw*. 2013;86(9):2242-2262.
8. Chung L, Nixon BA, Yu E, Mylopoulos J. *Non-Functional Requirements in Software Engineering*. Vol. 5. New York, NY: Springer Science+Business Media; 2012.
9. In HP, Olson D, Rodgers T. Multi-criteria preference analysis for systematic requirements negotiation. In: Proceedings 26th Annual International Computer Software and Applications; 2002; Oxford, UK.
10. ISO/IEC. *Software Engineering – Product Quality – Part 1: Quality Model*. Technical Report ISO/IEC TR 9126- 1:2001. Geneva, Switzerland: International Organization for Standardization; 2001.
11. ISO/IEC. *Software Engineering – Product Quality – Part 2: External Metrics*. Technical Report ISO/IEC TR 9126- 2:2003. Geneva, Switzerland: International Organization for Standardization; 2003.
12. Saaty TL, Vargas LG. *Models, Methods, Concepts & Applications of the Analytic Hierarchy Process*. Vol. 175. New York, NY: Springer Science+Business Media; 2012.
13. Fittkau F, Frey S, Hasselbring W. CDOSim: simulating cloud deployment options for software migration support. Paper presented at: IEEE 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA); 2012; Trento, Italy.
14. Gonçalves Junior R, Rolim T, Sampaio A, Mendonça NC. A multi-criteria approach for assessing cloud deployment options based on non-functional requirements. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing; 2015; Salamanca, Spain. Technical Track on Requirements Engineering.
15. Kazman R, Klein M, Barbacci M, Longstaff T, Lipson H, Carriere J. The architecture tradeoff analysis method. In: Proceedings of the Fourth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS); 1998; Monterey, CA.

16. Babar MA, Gorton I. Comparison of scenario-based software architecture evaluation methods. Paper presented at: 11th Asia-Pacific Software Engineering Conference; 2004; Busan, South Korea. https//doi.org/10.1109/APSEC.2004.38
17. Jayasinghe D, Malkowski S, Li J, Wang Q, Wang Z, Pu C. Variations in performance and scalability: an experimental study in IaaS clouds using multi-tier workloads. *IEEE Trans Serv Comput*. 2014;7(2):293-306. https//doi.org/10.1109/TSC.2013.46
18. Li A, Yang X, Kandula S, Zhang M. CloudCmp: comparing public cloud providers. In: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC); 2010; Melbourne, Australia. http://doi.acm.org/10.1145/1879141.1879143
19. Malkowski S, Hedwig M, Jayasinghe D, Pu C, Neumann D. CloudXplor: a tool for configuration planning in clouds based on empirical data. In: Proceedings of the 2010 ACM Symposium on Applied Computing (SAC); 2010; Sierre, Switzerland. http://doi.acm.org/10.1145/1774088.1774172
20. Kazman R, Klein M, Barbacci M, Longstaff T, Lipson H, Carriere J. *The Architecture Tradeoff Analysis Methods*. Technical Report CMU/SEI-2000-TR-004. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University; 2000.
21. Gunther NJ. *Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services*. Vol 1. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2007.
22. Mißbach M, Gibbels P, Karnstädt J, Stelzel J, Wagenblast T. *Adaptive Hardware Infrastructures for SAP*. 1st ed. Vol 1. Galileo Press; 2005.
23. Is there an optimal cpu utilization? https://turbonomic.com/blog/on-turbonomic/optimal-cpu-utilization-depends/. Accessed May 7, 2018.
24. Weinstock C, Goodenough J. *On System Scalability*. Technical Report CMU/SEI-2006-TN-012. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University; 2006.
25. Ameller D, Ayala C, Cabot J, Franch X. Non-functional requirements in architectural decision making. *IEEE Softw*. 2013;30(2):61-67.
26. Lewis J, Fowler M. Microservices. 2014. https://martinfowler.com/articles/microservices.html. Accessed May 7, 2018.
27. Bass L, Weber I, Zhu L. *DevOps: A Software Architect's Perspective*. Boston, MA: Addison-Wesley Professional; 2015.
28. Jamshidi P, Pahl C, Mendonça NC, Lewis J, Tilkov S. Microservices: the journey so far and challenges ahead. *IEEE Softw*. 2018;35(3):24-35.
29. Scheuner J, Cito J, Leitner P, Gall H. Cloud workbench: benchmarking IaaS providers based on infrastructure-as-code. In: Proceedings of the 24th International Conference on World Wide Web; 2015; Florence, Italy.
30. Cunha M, Mendonça NC, Sampaio A. Cloud crawler: a declarative performance evaluation environment for infrastructure-as-a-service clouds. *Concurr Comput Pract Exp*. 2017;29(1).
31. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A. *Experimentation in Software Engineering: An Introduction*. Vol 6. New York, NY: Springer Science+Business Media; 2000.
32. Leong L, Petri G, Gill B, Dorosh M. Magic Quadrant for loud Infrastructure as a Service, Worldwide. Stamford, CT: Gartner. https://www.gartner.com/doc/reprints?id=1-2G2O5FC&ct=150519. Accessed April 15, 2017.
33. Garg SK, Versteeg S, Buyya R. SMICloud: a framework for comparing and ranking cloud services. Paper presented at: Fourth IEEE International Conference on Utility and Cloud Computing; 2011; Victoria, Australia.
34. CSMIC SMI. Selecting a cloud provider: Defining widely accepted measures for cloud services. Cloud Service Measurement Initiative Consortium, Service Measurement Index. https://spark.adobe.com/page/PN39b/. Accessed April 15, 2017.
35. Sodhi B, Prabhakar TV. Cloud platforms: impact on guest application quality attributes. Paper presented at: IEEE Asia-Pacific Services Computing Conference; 2012; Guilin, China. https//doi.org/10.1109/APSCC.2012.22
36. Sodhi B, Prabhakar TV. Assessing platform suitability for achieving quality in guest applications. Paper presented at: 19th Asia-Pacific Software Engineering Conference; 2012; Hong Kong. https//doi.org/10.1109/APSEC.2012.53
37. Lloyd WJ, Pallickara S, David O, et al. Demystifying the clouds: harnessing resource utilization models for cost effective infrastructure alternatives. *IEEE Trans Cloud Comput*. 2017;5(4):667-680.
38. Lloyd W, Pallickara S, David O, Lyon J, Arabi M, Rojas K. Performance modeling to support multi-tier application deployment to infrastructure-as-a-service clouds. In: Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing; 2012; Chicago, IL.
39. Calheiros RN, Ranjan R, Beloglazov A, De Rose CAF, Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw Pract Exp*. 2011;41(1):23-50.
40. Burns B, Oppenheimer D. Design patterns for container-based distributed systems. In: Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing (HotCloud); 2016; Denver, CO.
41. Roberts M. Serverless Architectures. 2016. https://martinfowler.com/articles/serverless.html. Accessed May 7, 2018.
42. Etemaadi R, Lind K, Heldal R, Chaudron MRV. Quality-driven optimization of system architecture: industrial case study on an automotive sub-system. *J Syst Softw*. 2013;86(10):2559-2573.
43. Jamshidi P, Pahl C, Mendonça NC. Pattern-based multi-cloud architecture migration. *Softw Pract Exp*. 2017;47(9):1159-1184.

How to cite this article: Gonçalves Junior R, Sampaio A, Rolim T, Mendonça NC. Cloud application architecture appraiser (CA3): A multicriteria approach and tool for assessing cloud deployment options based on nonfunctional requirements. *Softw Pract Exper*. 2018;48:2278–2301. https://doi.org/10.1002/spe.2644