# Performance Inference: A Novel Approach for Planning the Capacity of IaaS Cloud Applications

Marcelo Gonçalves, Matheus Cunha, Nabor C. Mendonça, Américo Sampaio
Programa de Pós-Graduação em Informática Aplicada (PPGIA)
Universidade de Fortaleza (UNIFOR)
Fortaleza, CE, Brazil
Email: {marcelocg,mathcunha}@gmail.com, {nabor,americo.sampaio}@unifor.br

***Abstract*—This work presents a novel approach to support application capacity planning in infrastructure-as-a-service (IaaS) clouds. The approach, called *performance inference*, relies on the assumption that it is possible to establish a capacity relation between different resource configurations offered by a given IaaS provider, enabling one to *infer* an application's performance under certain resource configurations and workloads, based upon the application's actual performance as observed for other related resource configurations and workloads. Preliminary evaluation results, obtained from testing the performance of a well-known blogging application (WordPress) in a public IaaS cloud (Amazon EC2), show that the best performance inference strategies can significantly reduce (over 80%) the total number of application deployment scenarios that need to be actually tested in the cloud, with a high (over 98%) inference accuracy.**

## I. Introduction

One of the main challenges faced by users of Infrastructure-as-a-Service (IaaS) clouds is to adequately plan the capacity of the cloud resources, in particular, virtual machines, necessary to run their applications [1]. This usually involves identifying the most effective way to deploy an application in the cloud taking into account a variety of resource options offered by current IaaS providers as well as multiple application-specific quality criteria (e.g., performance, scalability, cost, security) [2].

In general, IaaS providers charge their customers in a pay-per-use basis, with resource utilization prices varying according to the offered capacity (e.g., number of CPU cores, memory size, disk space). Therefore, in order to calculate the operational cost of running an application in the cloud it is necessary to estimate or measure how the application will respond to varying demand levels considering different deployment options. In practice, this means that it is up to cloud users to find out which resources and deployment options are the most appropriate (e.g., in terms of cost or performance) for their applications. It is worth noting that using an auto-scaling service, such as Amazon Auto Scaling,[1] which allows users to deploy an application in a cluster of virtual machines whose size can automatically be adjusted according to user-defined elasticity rules, only partially solves the problem, since the user still needs to plan the capacity of the cluster's virtual machines.

Several solutions have been proposed to support IaaS cloud users in planning the capacity of their applications. We categorize those solutions in two general approaches, which we refer to as *predictive* and *empirical*. Predictive solutions aim at predicting how a cloud application would behave under different resource configurations and workload levels without having to deploy the full application stack in the cloud [3]–[8]. This is usually done by simulating the application's behavior in the cloud (e.g., [6], [7]) or by comparing its expected usage profile with existing cloud benchmarks (e.g., [3]). Empirical solutions, in turn, aim at evaluating the performance of the application in the "real" cloud environment. To this end, most empirical solutions provide automated ways to deploy, execute and test the performance of the application in the cloud using different resources configurations under multiple artificially-generated workloads [9]–[12].

Predictive solutions have the advantage of imposing little or no extra cost for testing the cloud application. However, due the performance variability commonly observed for public cloud services [13], and the fact that some characteristics of public clouds, such as multi-tenancy and physical resource sharing, are not yet properly represented in most performance prediction models, the results obtained with current predictive solutions, specially cloud simulators [7], can be quite inaccurate. In this regard, empirical solutions are much more reliable, but tend to be considerably more demanding due to the many possible resource configurations and workload levels that have to be effectively evaluated in the cloud. Another drawback of empirical solutions is that, depending on the number of tests to be conducted, and on the duration of each test, they can incur a significant cost to cloud users.[2]

This paper proposes a new application capacity planning approach for IaaS clouds, which combines the advantages from both predictive and empirical solutions. The proposed approach, called *performance inference*, relies on a simple yet intuitive assumption that it is possible to establish a capacity relation between different resource configurations offered by a given IaaS provider, enabling one to predict — or *infer* —, with high accuracy, an application's expected performance for certain resource configurations and workloads, based upon its actual performance as observed for other (related) resource configurations and workloads. The underlying idea is to reduce the number of deployment scenarios that need to be actually tested in the cloud, thus also reducing the cost and effort typically associated with the capacity planning process. To give

[1] http://aws.amazon.com/autoscaling/.

[2] Even though many cloud providers offer their users a limited amount of free resources, those are often of very low capacity and, hence, insufficient to handle real application workloads.

an example of how the performance inference approach works, consider the case in which an application deployed in the cloud satisfies an expected quality indicator (e.g., response time) under a certain resource configuration and workload. In this case, it should be possible to infer that the application will also satisfy the same quality indicator under other higher-capacity resource configurations or lower workload levels. Analogously, if the application fails to satisfy the expected quality indicator, one could infer that it will also fail the tests under other lower-capacity resource configurations or higher workload levels.

Despite its simplicity, in practice using the performance inference approach requires a careful selection of the deployment scenarios upon which to test the application in the cloud, so as to maximize the number of related deployment scenarios for which the application performance could be inferred at no cost for the cloud user. Our preliminary evaluation results, obtained from testing the performance of a well-known blogging application (WordPress) in a public IaaS cloud (Amazon EC2), show that the best performance inference strategies can drastically reduce (over 80%) the number of application performance tests executed in the cloud, while providing a high (over 98%) inference accuracy.

The remainder of the paper is organized as follows. Section II describes our proposed application capacity planning process based on the performance inference approach. Section III reports on our evaluation method and results. Section IV compares our approach with related work. Finally, Section V presents our conclusions and directions for future work.

## II. A Capacity Planning Process based on Performance Inference

### A. Concepts and Terminology

Before presenting the proposed process, we need to define some important concepts related to the domain of application capacity planning in the cloud (see Table I). Those concepts constitute the main terminology which will be used throughout the rest of this paper.

TABLE I. Concepts and terminology used in the paper.

| Concept | Definition |
|---|---|
| *Application under Test* | A software application, possibly implemented in a multilayer architecture, to be deployed in a cloud platform and for which one wants to identify the most appropriate cloud resource configurations based on one or more *performance metrics*. |
| *Performance Metric* | A characteristic or measurable behavior of the application under test collected in an automated fashion and comparable to a *reference value*. A performance metric is the means to asses the degree of success of an execution of the application under test in the cloud. Its measurement unit is dependent on the application domain, e.g., response time, frames per second, throughput and so on. |
| *Reference Value (SLO)* | A value defined as minimally acceptable for a performance metric measured during an execution of the application under test. This value, also referred in this work as SLO (*Service Level Objective*), serves as a basis for assessing whether the application is capable of successfully executing using a given *virtual machine configuration* under a given *workload*. |
| *Workload* | Denotes a demand imposed onto the application under test in a particular execution. Its measurement unit also depends on the application domain, e.g., number of concurrent users for a web application, image size for a image conversion application, etc. |
| *Virtual Machine Type* | A label used to classify virtual machines offered by a cloud provider based on their technical characteristics (e.g., number of processing cores, memory size, disk space), such as *small*, *large*, *extra large*, and so on. Labeling VMs with named types allows the provider to offer a finite and discrete virtual machine product line. |
| *Virtual Machine Category* | A label used to classify related virtual machine types offered by a cloud provider according to their common features, such as hardware platform and/or intended usage. For example, a provider my offer virtual machine categories that prioritize processing speed, memory consumption, disk access, etc. |
| *Virtual Machine Configuration* | Denote a set of virtual machines of the same type and category. Different configurations can be used to deploy and execute the components of different architectural layers (e.g., presentation, business, persistence) of the application under test. |
| *Deployment Space* | Denotes a finite set of virtual machine configurations upon which the application under test will be deployed and executed during a capacity evaluation session. |
| *Capacity Relations* | Define a directed graph over the deployment space where vertices correspond to the deployment space's virtual machine configurations and edges represent the relative superiority or inferiority (depending on the direction of the edge) of a configuration over another in terms of expected computing power. |
| *Capacity Levels* | Establish a hierarchy over the deployment space's virtual machine configurations according to their capacity relations. Virtual machine configurations belonging to the same capacity level are considered indistinguishable in terms of computing power. |

### B. Process Input

The process requires the following data as input: a reference value (or SLO), used to asses if the application under test has achieved the expected performance level after each execution; a set of workload values, used to submit the application under test to varying demand levels; and the application's deployment space. The latter is built from three parameters also provided as input to the process: (i) an ordered set of virtual machine types offered by the target cloud provider; (ii) the maximum number of virtual machines used in each virtual machine configuration; and (iii) the criteria for establishing the capacity relations between the deployment space's configurations. Examples of criteria that can be user for that purpose are the technical characteristics of each virtual machine type (e.g., number of CPU cores or memory size) or other non-technical features such as the virtual machine types' price per hour of utilization. Section III-A will show an example of a deployment space in which capacity relations were defined based on the category, type and number of virtual machines belonging to each virtual machine configuration.

### C. Process Activities

The main activities executed as part of the proposed capacity planning process are shown in Figure 1. The activities tagged with the $\ll$A$\gg$ label (highlighted in red) are abstract, meaning that they need to be implemented or customized by the process user according to different capacity planning strategies (described in more detail in Section II-D). The other activities are concrete and thus always executed in the same (generic) way independently of any particular application or capacity planning strategy.

The process execution is cyclic and takes places in four different phases: (i) selection of an initial execution scenario; (ii) application execution; (iii) performance inference; and (iv) selection of the next execution scenario.

*1) Selection of an Initial Execution Scenario:* The first activity in this phase is the selection of a workload. As this is an abstract activity, different strategies can be employed for selecting a particular workload, such as selecting the lowest or highest workload from the set of workload values provided as input to the process. After selecting a workload, the process

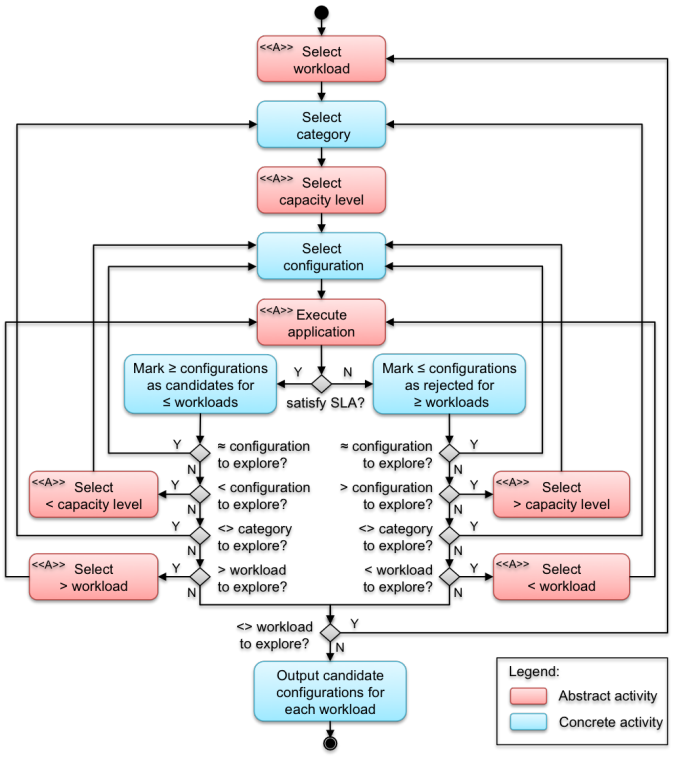



Fig. 1. Activity diagram for the proposed capacity planning process.

proceeds with the selection of a virtual machine category. This is a concrete activity since the category chosen will not influence the process outcome as all configurations of every category included as part of the deployment space will have to be evaluated. The next activity is the selection of a capacity level from the deployment space. This is another abstract activity as different strategies can be used for selecting a capacity level, such as selecting the lowest of highest capacity level in the deployment space. Finally, the process proceeds to select one of the set of virtual machine configurations belonging to the previously selected capacity level. The order of the selection is irrelevant as all configurations at the chosen capacity level will have to be evaluated.

*2) Application Execution:* Once a workload, a virtual machine category, a capacity level and a virtual machine configuration have all been chosen, the process is ready to execute the application in the cloud. This is another abstract activity as it depends on several technical factors related to either the application under test of the target cloud provider. Examples of such factors are the technologies required to deploy, configure, stress and monitor the performance of the application components in the cloud. After the application has been executed, the process proceeds to assess the collected execution results and moves on to the performance inference phase.

*3) Performance Inference:* During this phase the process reaches its first decision point. Based on the analysis of the execution results, the process determines whether the application is capable of handling the current workload using the current virtual machine configuration. If the required SLO is satisfied (or not), the process marks the current configuration,

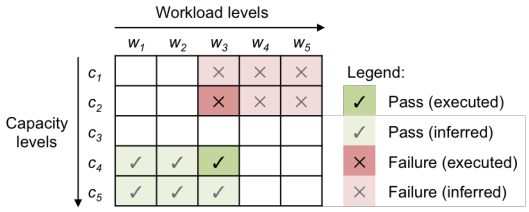



Fig. 2. Example of the performance inference approach in action.

respectively, as *candidate* or *rejected* for the current workload.

At this moment, the process triggers the performance inference approach originally proposed in this work. Using this approach, the process can *infer* (without the need of actual executions) the application's expected performance for other workloads and virtual machine configurations not yet evaluated. This is possible based on domain knowledge captured in the form of the capacity relations defined over the deployment space's configurations and simple reasoning. Specifically, if the execution results show that the application meets the required SLO for the current configuration and workload, then the application is also expected to meet that same SLO for any higher capacity configuration (as indicated by the deployment space's capacity relations) under that same workload. Similarly, the application is also expected to meet that same SLO using the same configuration under lower workloads. In this case, the process marks as *candidate* for the current workload all the other configurations considered of having higher capacity (according to the deployment space's capacity relations) than the current configuration. Finally, the process also marks the current configuration as *candidate* for all lower workloads.

The case in which the application fails to meet the required SLO using the current virtual machine configuration under the current workload is treated analogously. In this case, the process marks as *rejected* for the current workload all configurations considered of having lower capacity than the current configuration. Finally, the current configuration is also marked as *rejected* for all higher workloads.

Figure 2 shows an example of a possible outcome from the performance inference phase. In this example, the deployment space is represented as a matrix whose rows and columns correspond, respectively, to virtual machine configurations and workloads. The configurations' capacity levels grow from top to bottom while the workload values grow from left to right. The lower left part of the matrix shows the case in which the application meets the required SLO for a given execution scenario along with all the other execution scenarios for which the application's successful performance can be inferred. The upper right part of the matrix, in turn, shows the opposite case, highlighting an execution scenario in which the application fails to meet the required SLO and all the other execution scenarios for which that same negative behavior can be inferred. Note, in this example, that by testing the application in only two execution scenarios the process is able to infer its expected performance for a total of ten other different scenarios. This represents a reduction of near 90% in the overall number of execution scenarios that needed to be effectively tested in the cloud.

This example is illustrative of the great potential offered by the performance inference approach to reduce the time, cost and effort typically associated with other empirical capacity planning approaches. Section III gives further evidence of the benefits of the performance inference approach by evaluating its effectiveness in a real cloud environment.

*4) Selection of the Next Execution Scenario:* After the performance inference phase, the process either selects the elements for the next execution scenario or ends its execution in case there are no more alternative scenarios to explore. In the latter case, the process produces as outcome a list of all configurations marked as *candidate* for each given workload ordered by configuration price.

The selection of the next execution scenario involves taking either of the following decisions: selecting a new virtual machine configuration from the same (current) capacity level; selecting a new capacity level; selecting a new virtual machine category; or selecting a new workload. The selection of a new capacity level or a new workload depends on the application's execution results, as the process will attempt to decrease (increase) the current configuration's computing power or, alternatively, increase (decrease) the current workload in case the application has met (failed to meet) the required SLO. For this reason, these are also abstract activities that have to be customized by the process user when instantiating a new capacity planning strategy.

### D. Capacity Planning Strategies

All abstract activities of the process (except the Application Execution activity) need to be instantiated as part of the implementation of a new capacity planning strategy. Since those activities basically involve the selection of workloads and capacity levels, implementing different capacity planning strategies means providing different ways to explore the application's deployment space.

Choosing a proper way to explore the deployment space is an important decision that can have a significant impact on the effectiveness of the performance inference approach. To give an example, consider the case in which none of the configurations on a given deployment space are capable of satisfying the required SLO for any workload. In this case, starting the evaluation process by the lowest capacity level and highest workload would be a bad decision as the application would fail all tests, thus preventing the process from inferring the application performance for any other higher capacity configuration or any other lower workload. On the other hand, starting the process by the highest capacity level and lowest workload would be much more effective as the application failing a single test would be enough for the process to infer its failure for all other lower capacity configurations as well as for all other lower workloads.

These two extreme cases illustrate the challenges of designing effective capacity planning strategies for the performance inference approach. As a first step towards addressing these challenges, in this work we introduce the concept of a *selection heuristic*, which encompasses different *selection tactics* to be used by the process when selecting a new capacity level or workload. Initially, we defined three selection tactics, namely *optimistic*, *conservative* and *pessimistic*. These can be used to select both workloads and capacity levels. The combination of these three selection tactics yields a total of nine different selection heuristics, as illustrated in Figure 3.

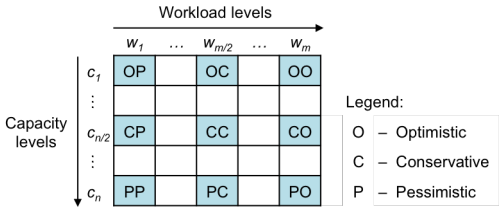



Fig. 3. Tactics and heuristics for selecting new capacity levels and workloads.

Note in Figure 3 that each selection heuristic is identified by a pair of capital letters, with each letter representing the initial of the respective selection tactic used by the heuristic. The first letter corresponds to the tactic used for selecting a capacity level (i.e., row), while the second letter corresponds to the tactic used for selecting a workload (i.e., column). From the way the letters are positioned in the deployment space matrix we can see that an optimistic tactic selects the lowest capacity level and the highest workload. A conservative tactic in turn selects an intermediate capacity level and workload. Finally, a pessimistic tactic selects the highest capacity level and the lowest workload.

Selection heuristics are (re)applied recursively, for the unexplored part of the deployment space, at each new iteration of the process. Therefore, the terms *highest*, *lowest* and *intermediate* are all relative in this context, meaning the highest, lowest and intermediate elements amongst the remaining choices of capacity levels and workloads. In this respect, choosing an optimistic or pessimistic tactic corresponds to performing a linear search in a given dimension of the deployment space matrix, while choosing a conservative tactic corresponds to performing a binary search in that dimension.

The proposed capacity planning process has been implemented in the form of an extensible performance inference framework, called *Cloud Capacitor*.[3] This framework was used to support the execution of the experimental evaluation reported in the next section. More details on the framework's design and implementation have been omitted from the paper due to space constraints.

## III. Experimental Evaluation

This section describes the method and results of an experimental investigation conducted to evaluate the effectiveness of the performance inference approach, in particular, of the nine selection heuristics introduced in the previous section.

### A. Method

The experiment consisted in using the proposed capacity planning process to systematically evaluate the performance of a real blogging application (WordPress[4]) in a real IaaS cloud

[3] A web-based version of the Cloud Capacitor framework is publicly available at http://cloud-capacitor.herokuapp.com/.

[4] https://wordpress.org/.

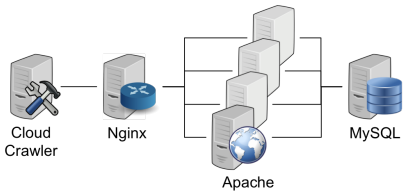



Fig. 4. WordPress's deployment architecture used in the Amazon EC2 cloud.

platform (Amazon EC2[5]). We chose WordPress and Amazon EC2 due to their huge popularity in their respective domains.

We deployed WordPress in the Amazon EC2 cloud in two architectural layers: one layer for the Apache application server, and the other layer for the MySQL relational database (see Figure 4). Due to cost and time constraints, during the experiment we varied only the type and number of virtual machines used to deploy the Apache application server, using the Nginx web server as load balancer. Finally, we used the Cloud Crawler [11] perfomance evaluation environment to automate all the tests in the cloud. These included starting and stoping each virtual machine, configuring the load generator according to the required workload, and executing and monitoring the performance of the WordPress components.

To build WordPress's deployment space we selected two virtual machine categories offered by Amazon EC2, namely *c3*, which provides CPU-optimized instance types, and *m3*, which provides general-purpose instance types. We then selected seven virtual machine types from those two categories: *c3_large*, *c3_xlarge* and *c3_2xlarge*, from category *c3*; and *m3_medium*, *m3_large*, *m3_xlarge* and *m3_2xlarge*, from category *m3*. For each of those types we created four virtual machine configurations composed of one, two, three and four virtual machine instances, respectively, yielding a total of 28 different virtual machine configurations upon which WordPress's performance would be evaluated.

We defined capacity relations over those 28 virtual machine configurations based on the category, type and number of virtual machines belonging to each individual configuration. Specifically, configurations with a higher (lower) number of virtual machines of a given type were considered of higher (lower) capacity than configurations with a lower (higher) number of virtual machines of that same type. Moreover, configurations with a certain number of virtual machines of a given type were considered of higher (lower) capacity than configurations with the same number of virtual machines of a higher (lower) type of the same category according to the provider's instance type hierarchy. For example, a configuration containing three virtual machines of type *m3.medium* was considered of higher (lower) capacity than a configuration containing two (four) virtual machines of that type. Similarly, a configuration containing two virtual machines of type *c3.xlarge* was considered of higher (lower) capacity than a configuration containing two virtual machines of type *c3.large* (*c3.2xlarge*). Note that we did not assume any capacity relation between configurations containing virtual machine types from different categories.

[5]http://aws.amazon.com/ec2.

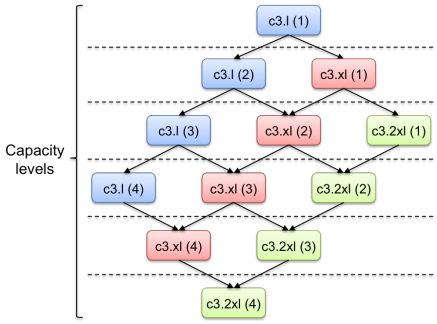



Fig. 5. Sample of WordPress's deployment space used in the experiments.

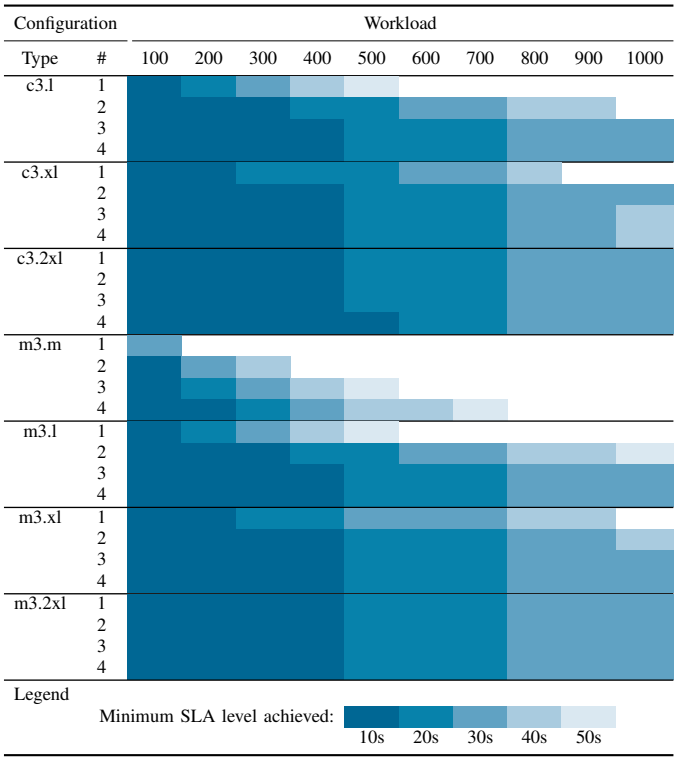
Figure 5 shows a sample of the deployment space built from the 28 virtual machine configurations selected from Amazon EC2. This particular sample only includes 12 configurations containing virtual machine types belonging to the *c3* category. Note how the configurations are hierarchically organized in the deployment space according to 6 different capacity levels, with the lowest and highest capacity configurations located, respectively, at the top and the bottom of the hierarchy. Also, note that there are no capacity relations defined amongst configurations belonging to the same capacity level, as these are considered indistinguishable from an expected computing power perspective.

To evaluate how WordPress would perform using each of the 28 deployment space configurations under varying demand levels, we defined 10 different workloads by varying the number of concurrent users that would access the application in a given execution. The number of current users defined for each workload was 100, 200, 300, 400, 500, 600, 700, 800, 900 and 1000, respectively. In each execution the load generator would create the required number of users, who would continuously access WordPress during several minutes by repeatedly issuing the following sequence of requests: *logon*; add a new post; search for the new post; update the new post; search for a previously existing post by keyword; update the existing post; *logoff*.

The performance metric used was the *total response time*, which was measured by each user as the elapsed time between issuing a logon request and receiving a response from the subsequent logoff request. A test execution would be considered satisfactory if at least 90% of all request sequences issued by WordPress users were successfully served by the application within the minimum total response time defined by the provided SLO parameter. In order to allow the investigation of the effectiveness of different capacity planning strategies under different SLO requirements, we defined five SLO levels upon which to evaluate the performance of WordPress, starting at 10s (a very strict requirement for which only a few configurations would achieve the required SLO under most workloads) and then gradually increasing the SLO in 10s increments up to 50s (a more soft requirement for which most configurations would achieve the required SLO under most workloads).

Finally, to establish a baseline upon which we could compare the effectiveness of our proposed capacity planning strategies, we deployed, executed and collected detailed performance results for WordPress using all (28) virtual machine

TABLE II. WORDPRESS'S CONSOLIDATE PERFORMANCE RESULTS IN THE AMAZON EC2 CLOUD.




configurations under all (10) workloads, yielding a total of 280 execution scenarios investigated. Table II shows a consolidate view of WordPress's performance results obtained in the Amazon EC2 cloud.

We refer to the performance results shown in Table II as the experiment's *oracle*, and to the exhaustive empirical strategy used to collect them as the *Brute Force* (BF) selection heuristic. The other 9 selection heuristics presented in the previous section were then compared with each other as well as with BF in terms of both efficiency (i.e., the fraction of execution scenarios that had to be effectively tested in the cloud) and accuracy (i.e., the fraction of execution scenarios for which the application performance was correctly inferred based on the oracle).

## B. Results

*1) Efficiency:* To assess the relative efficiency of the nine selection heuristics with respect to BF we consider two metrics: *relative execution time* and *relative cost*. Since each test scenario is executed for the same amount of time, the relative execution time of a given selection heuristic is calculated by the ratio between the number of real application executions required by that heuristic and the total number of application executions required by BF. The relative cost of a selection heuristic, in turn, is calculated by the ration between the sum of the cost of each virtual machine configuration effectively tested by that heuristic in the cloud and the sum of the cost of every virtual machine configuration tested by BF. We should note that the cost of a given virtual machine configuration depends on the number as well as on the utilization price of the virtual machine instances belonging to that configuration.

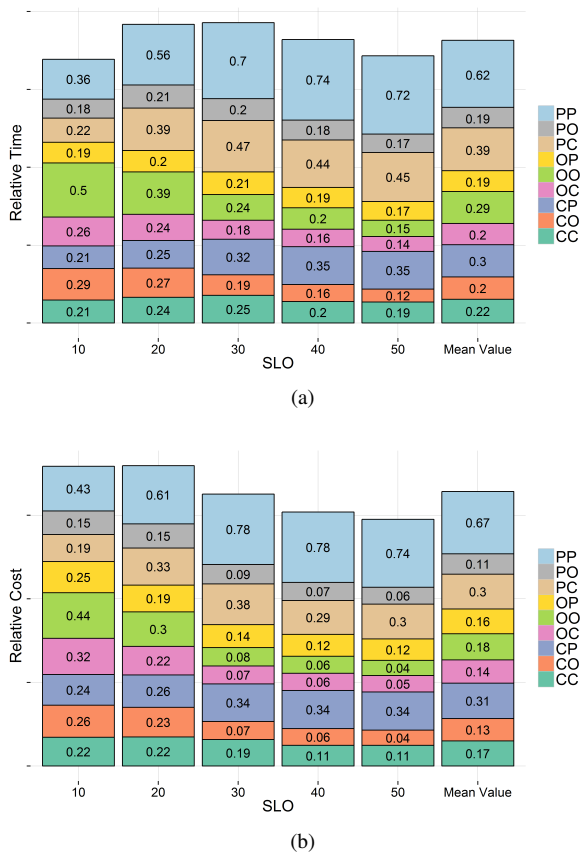



Fig. 6. Relative effectiveness of the 9 selection heuristics: (a) execution time; and (b) cost.

Therefore, since a provider can charge quite different prices for different virtual machine types, the relative cost of a given selection heuristic will be greatly influenced by the specific configurations that heuristic selects to evaluate in the cloud.

Figure 6 shows the results for both metrics considering the five SLOs investigated. An analysis of the results for the relative execution time metric, as seen in Figure 6(a), shows that under more soft SLOs the best heuristics are OC and CO, offering gains up to 86% and 88%, respectively, with respect to BF. However, under more stringent SLOs the best heuristics are PO and OP, with gains up to 82% and 81%, respectively, over BF. In fact, PO and OP, along with CC, are amongst the best heuristics overall for this metric, since their results remain stable throughout the five SLOs, as indicated by their mean values (represented in the right most column in both charts). The lowest gains for this metric are obtained with PP and CP, particularly the former, whose average gain is only about 30% with respect to BF.

Now, regarding the relative cost metric, an analysis of Figure 6(b) shows that under more soft SLOs the best results are obtained with OO and CO, which offer gains up to 96% compared with BF. However, those two heuristics do not perform so well under more stringent SLOs, where the best results are obtained with PO, PC and CC, with gains between 78% and 85% over BF. Overall, the best heuristics for this metric are PO, OC and CO, offering average gains between 86% and 89% with respect to BF. The lowest average gains

TABLE III. ACCURACY OF THE PROPOSED SELECTION HEURISTICS.

| | SLO | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 10s | | | 20s | | | 30s | | | 40s | | | 50s | | |
| Heuristic | P | R | F | P | R | F | P | R | F | P | R | F | P | R | F |
| CC | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.98 | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| CO | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.99 | 1.00 | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| CP | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.98 | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| OC | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.99 | 0.99 | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| OO | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.99 | 1.00 | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| OP | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.98 | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| PC | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.98 | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| PO | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.99 | 1.00 | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| PP | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.98 | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |

are offered by PP, CP and PC, with PP once more providing the worst results by a large margin.

An analysis of the results for both metrics reveals that the best heuristics overall are PO, OP and CC, all offering gains of at least 75% over BF throughout the five SLOs. These results reveal that the most effective strategies to explore the deployment space, at least in the case of WordPress, is to either select the highest capacity level and the highest workload; the lowest capacity level and the lowest workload; or an intermediate capacity level and an intermediate workload, in that order.

*2) Accuracy:* To assess the selection heuristics's accuracy, we compare their capacity planning results with those available in the oracle and then calculate their average values in terms of *Precision*, *Recall* and *F-Measure* [14]. Specifically, we use the oracle's performance results to assess whether each of the virtual machine configurations marked as *candidate* or *rejected* (representing positive and negative capacity planning results, respectively) by each selection heuristic for each workload under each of the five SLOs are actually true.

Table III shows the average values of Precision ($P$), Recall ($R$) and F-Measure ($F$) calculated for the nine selection heuristics under each of the five SLOs investigated. Note that all selection heuristics achieve 100% accuracy except for the 30s SLO, where their accuracy results vary between 98% and 100%.

A more thorough investigation of the oracle's performance data revealed that the heuristics' slight accuracy drop under the 30s SLO was due to a few unexpected performance variations experienced by WordPress in the Amazon EC2 cloud. Those variations caused some virtual machine configurations to perform better than other configurations that were supposed to have higher computing power according to the deployment space. This was the case of the configuration containing two virtual machines of type *c3.xlarge*, which under high demand performed better than the two configurations containing three and four virtual machines of that same type (see the corresponding results in Table II).

Even though significant performance variations are relative common in current IaaS cloud offerings [13], in our experiment such variations had a very limited impact on the effectiveness of the performance inference approach, causing an accuracy drop of less than 3% under a single SLO. These results reinforce our confidence that the proposed approach can provide high prediction accuracy even for real applications and cloud environments.

## IV. RELATED WORK

As we have mentioned earlier in the paper, we categorize existing application capacity planning solutions for the cloud domain as following two main approaches, which we refer to as predictive and empirical. We discuss each of those approaches in more details below.

Predictive solutions employ different techniques to predict the performance of applications in the cloud. Some solutions (e.g. [3]–[5], [8]) base their predictions on results of several *benchmarks* executed previously in the cloud. These results are then consolidated by the capacity planning solution and used later, by users, to predict the performance of applications by analogy between the user's application and the results of the benchmarks. Another work [7] simulates the behavior of the application in the cloud based on a simulation model without having to conduct any performance test in the cloud. The solution described in [6] monitors the application behavior (e.g., CPU, memory, disk) in a local environment by collecting the events that are relevant from a performance perspective. After the data is collected, the solution executes the same events in the cloud in a way to try to predict the behavior of the application.

Regarding costs, predictive solutions can range from virtually no cost when doing simulation, as in [7], from low to moderate costs when using analogy tools [3]–[5], [8] or event replays [6], respectively. However, one common shortcome of all predictive solutions is the limited accuracy of their results. The prediction by analogy technique requires that the user's application has a similar behavior as the benchmarks previously available and executed in the cloud. The simulation technique also suffers from possible mismatches between the user's application and the simulation model. Some results show performance discrepancies larger than 30% between existing simulation models and real executions in IaaS clouds [7]. The event replay technique [6] presents some limitations in its event capturing based on a local environment that also limits the accuracy of its predictions.

Empirical solutions (e.g., [9]–[12]), on the other hand, have a high accuracy as they execute the real application (not a benchmark) in the cloud and collect real execution data that can serve as a precise basis for capacity planning. Moreover, those solutions are much more flexible as they allow users to configure the application architecture in different ways, enabling different deployment options, configurations and workloads to be assessed. One relevant drawback of those solutions is that as the number and complexity of execution scenarios grow so does the cost incurred in the evaluation phase.

There are also a number of other so called autoscaling solutions that focus on a different perspective of short-term capacity planning (e.g., [15]). The main goal of those solutions is to dynamically adjust the cloud resources required for the application based on decisions taken while monitoring the behavior of the running application components in a cloud environment. Such decisions usually consist of scaling the application up or down based on general virtual machine parameters (e.g., CPU and memory consumption) [15] as well as on elasticity rules defined by users of the autoscaling service. Some problems related to those autoscaling solutions

are that in some cases application level parameters are more relevant to scaling decisions than generic virtual machine parameters. Another issue is that the scaling decisions are generally based on some pre-established virtual machine types (usually selected by the user) that are not always the most optimal configuration for handling specific workloads and tend to over provision the application [16]. We plan to investigate how the performance inference approach could be used to improve current autoscaling solutions and services in our future work.

Compared to those existing solutions, our work proposes a hybrid approach for application capacity planning that leverages the advantages of both empirical and predictive approaches. Its predictions are based on capacity relations defined between virtual machine configurations offered by a cloud provider and on empirical results of the execution of the application in that same provider. As shown in the previous section, the approach provides a very high prediction accuracy and at the same time manages to keep the time and costs of executing the capacity planning experiments in the cloud very low.

## V. Conclusion and Future Work

Wisely choosing an appropriate set of cloud computing resources (e.g., virtual machines), with the goal of minimizing an application's operational cost considering varying demand levels and performance requirements, is an important challenge for which there still no fully satisfactory solutions available. This paper presented a novel approach to support application capacity planning in Iaas clouds. The approach, called *performance inference*, was shown to be at the same time an efficient (in terms of both execution time and cost) and accurate (in terms of the quality of the results) solution to support IaaS cloud users in planning the capacity of their applications.

The performance inference approach described here is the result of an on-going research work and, for this reason, still has a number of important limitations that need to be overcome before the approach can gain more widespread use. In particular, our current and future work focus on improving the approach in the following ways:

- *New capacity planning strategies.* We are investigating other alternatives to explore the application's deployment space, for example, by selecting new capacity levels and workloads based on resource utilization metrics such as CPU and memory consumption. The idea is to propose "intelligent" selection heuristics that could more quickly find the optimal virtual machine configuration for the application under each given workload.
- *New capacity relations.* Another interesting line of research is to investigate more effective ways to define capacity relations, for example, by micro-benchmarking some specific configurations and then establishing their capacity relations based on utility functions derived from how well they perform for each micro-benchmark.
- *Multi-layered deployment space.* We also are investigating how to extend our approach to support multi-layered deployment spaces. The challenge here is to propose effective ways to explore the different virtual machine configurations that can be used to deploy each layer of the application. For example, if the application fails to achieve the required SLO for a given workload, one could analyze the resources utilization metrics collected for the components of each layer in order to decide if and how each layer's configuration should be changed in terms of increasing or decreasing their computational power.
- *New experiments.* Finally, we are already conducting a number of new performance evaluations experiments involving different types of applications and cloud providers. The idea is to assess the extent to which our approach and tools can be generalized to a more diverse set of cloud applications and deployment scenarios.

## Acknowledgment

This work is partially supported by Brazil's National Council for Scientific and Technological Development (CNPq), under grants 311617/2011-5 and 487174/2012-7.

## References

[1] D. A. Menascé and P. Ngo, "Understanding Cloud Computing: Experimentation and Capacity Planning," in *CMG 2009*, 2009.

[2] R. Gonçalves Junior *et al.*, "A Multi-Criteria Approach for Assessing Cloud Deployment Options Based on Non-Functional Requirements," in *ACM SAC 2015*, 2015.

[3] CloudHarmony, "CloudHarmony: Benchmarking the Cloud," 2015, http://goo.gl/IHDYxN.

[4] S. Malkowski *et al.*, "CloudXplor: A tool for configuration planning in clouds based on empirical data," in *ACM SAC 2010*, 2010, pp. 391–398.

[5] A. Li *et al.*, "CloudCmp: Comparing Public Cloud Providers," in *ACM SIGCOMM IMC 2010*, 2010, pp. 1–14.

[6] ——, "CloudProphet: Towards Application Performance Prediction in Cloud," in *ACM SIGCOMM 2011*, 2011, pp. 426–427.

[7] F. Fittkau *et al.*, "CDOSim: Simulating cloud deployment options for software migration support," in *IEEE MESOCA 2012*, 2012, pp. 37–46.

[8] G. Jung *et al.*, "CloudAdvisor: A Recommendation-as-a-Service Platform for Cloud Configuration and Pricing," in *IEEE SERVICES 2013*, 2013, pp. 456–463.

[9] D. Jayasinghe *et al.*, "Expertus: A Generator Approach to Automate Performance Testing in IaaS Clouds," in *IEEE CLOUD 2012*, 2012, pp. 73–80.

[10] M. Silva *et al.*, "CloudBench: Experiment Automation for Cloud Environments," in *IEEE IC2E 2013*, 2013, pp. 302–311.

[11] M. Cunha *et al.*, "A Declarative Environment for Automatic Performance Evaluation in IaaS Clouds," in *IEEE CLOUD 2013*, 2013, pp. 285–292.

[12] J. Scheuner *et al.*, "Cloud WorkBench – Infrastructure-as-Code Based Cloud Benchmarking," *arXiv preprint arXiv:1408.4565*, 2014.

[13] A. Iosup *et al.*, "On the performance variability of production cloud services," in *IEEE/ACM CCGrid 2011*, 2011, pp. 104–113.

[14] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[15] F. J. A. Morais *et al.*, "Autoflex: Service Agnostic Auto-scaling Framework for IaaS Deployment Models," in *IEEE/ACM CCGrid 2013*, 2013, pp. 42–49.

[16] K. Hwang *et al.*, "Cloud Performance Modeling and Benchmark Evaluation of Elastic Scaling Strategies," *IEEE Transactions on Parallel & Distributed Systems*, 2015, PrePrint. DOI Bookmark: http://doi.ieeecomputersociety.org/10.1109/TPDS.2015.2398438.