Influence of the trace resolution and length in the cost optimization process in cloud computing

Javier García, Joaquín Entialgo, José Luis Díaz, Manuel García, Daniel F. García

Department of Computer Science
University of Oviedo
Campus de Viesques, 33204, Gijón, Spain
{javier, joaquin, jldiaz, mgarcia, dfgarcia}@uniovi.es

Abstract—In order to optimize costs in cloud computing deployments, Virtual Machine (VM) allocation strategies are frequently employed. To compare different strategies, workload traces are required. These traces should be of one second resolution (to take into account the per-second billing) and of one year length (to consider reserved VMs properly). However, there are no public traces with these characteristics corresponding to transactional services. To overcome this issue, in this paper we present different synthesis techniques designed to generate a trace of one second resolution based on a trace with lower resolution, usually one hour. The influence of the different synthesis techniques in the cost optimization process is analyzed, concluding that they have an appreciable effect in the optimization process. As an alternative, the execution of VM allocation strategies using traces shorter than a year (a few months, for example) and the extrapolation of the results to the whole year is also analyzed. We found that this mode of operation can generate misleading results.

Index Terms—cloud computing, trace synthesis, virtual machine allocation, cost optimization.

I. INTRODUCTION

In the field of cloud computing, cost is a crucial aspect in service deployments. In order to deal with costs issues in IaaS platforms, Virtual Machine (VM) allocation strategies are frequently employed to find allocations of VMs that support a given workload minimizing the cost. A VM allocation is a combination of VMs, indicating their types, the number of each type, and the pricing categories (on-demand or reserved) to which the VMs belong. An example of a VM type in the Amazon IaaS Platform (EC2) is t2.xlarge, which has 4 virtual CPUs and 16 GB of memory [1].

An allocation strategy must produce successive VM allocations to adapt the computational power of the deployed VMs to the level of the workload at all times. Usually, a time slot is defined to determine the periods at which the strategy is applied. Strategies focused in the short term produce allocations for the next time slot, while those focused in the long term generate an allocation for each time slot in a long period (usually, one year). Long-term strategies are aimed to take advantage of reserved VMs, which are nowadays offered by all major IaaS platforms, that is, Amazon EC2 [2], Microsoft Azure Virtual Machines [3] and Google Compute Engine [4]. The minimum reservation period in all these platforms is one year.

The length of the time slot managed by allocation strategies must be established in relation to the billing time slots used by major IaaS platforms. At the end of 2017, Google Compute Engine and Amazon EC2 started using a billing time slot of one second [5], [6]. Microsoft Azure Virtual Machines uses one minute [7]. Therefore, time slots of one second and one minute should be used by allocation strategies.

In order to compare different allocation strategies proposed by the research community, workload traces are required. These traces must have one second resolution (to take into account the current per-second billing) and one year length (to consider reserved VMs properly). However, in the field of transactional services (such as web services), there are no public traces that fulfill both characteristics simultaneously. In this paper, we propose and analyze different synthesis techniques designed to generate a trace of one second resolution based on a trace with lower resolution (one hour, for example). These synthesis techniques can be applied to the available low resolution one-year-long traces, to obtain the traces with the required properties. The synthesis techniques propose three different methods of distributing the number of requests per hour in a number of requests per second, so that the total number of requests per hour is preserved for each hour of a trace. In addition, the use of different kinds of noise (white, pink and brown) to provide variability to the synthetic trace is also explored.

In order to assess the influence of the different synthesis techniques in the optimization process, a VM allocation problem is solved for the real trace of a service and for the synthetic traces obtained applying the different synthesis techniques. A state-of-the-art virtual machine allocation algorithm is used to solve the problem, calculating the allocation costs and the solving times when the real trace and the different synthetic traces are used.

The lack of one-second-resolution traces of a whole year may lead researchers to conduct the experimentation for VM allocation problems using traces shorter than a year and extrapolating the results for the whole year. This method is also investigated in this paper, and the experimentation shows that it can generate misleading conclusions.

The paper is organized as follows. First, the related work
is studied. Then, how the optimization process works is summarized. Section IV introduces the synthesis techniques proposed, that are then evaluated in Section V. Finally, the last section summarizes the conclusions.

II. RELATED WORK

Research in cloud computing requires workload traces, but not enough attention has been devoted to obtaining them. There are two main reasons: firstly, workload traces are difficult to obtain, and secondly cloud providers are reluctant to publish their traces. As a result, only a few real traces are publicly available. This lack of traces is even higher for transactional workloads. To overcome this lack of workload traces, two main alternatives have been considered: using benchmarks and the development of workload generators.

Typical benchmarks used in computer performance evaluation have shown not to be appropriate for cloud environments due to their complexity. To tackle this problem, some new benchmarks have been proposed in order to study the behaviour of cloud systems. Cloudsuite [10] is a first example that combines several typical real workloads to produce a mix workload. This mix is used to evaluate the scale-out behaviour of different micro-architectures. BigDataBench [11] is another benchmark developed to consider the characteristics of big data applications. Another example is CloudMix [12], which produces different workloads with diverse computational behaviours. Finally, the cloud benchmark developed by SPEC for IaaS cloud services [13] and its newest version [14] combine computing and NoSQL database workloads. The difference between both versions are the type of applications considered as workload. In spite of the availability of these benchmarks, they are not suitable for obtaining one year long traces.

The second alternative, the most common, consist on the development of workload generators. These generators are normally based on the analysis and characterization of some real workloads. Rain [15] is a toolkit developed to generate workloads based on probability distributions. It supports multiple generation strategies and can be extended via user-defined requests. The authors propose a framework to analyze and extract the characteristics of multi-tenant cloud platforms; then they build some basic workload elements, and finally, using a specification language, the required workload is generated. In [16], the authors develop an extension to the cloud simulation program CloudSim. This extension generates the users’ behaviour (number of requests, resource demand and probability distribution) that represents the workload to be submitted to the cloud simulation model. CloudGen [18] represents a specific workload generator based on the Dropbox real workload and it is used to study network traffic and storage services in realistic environments. Finally, a different strategy is used in Cloud-WG [19]. In this case, the authors consider three granularity levels of cloud computing workload: users, applications and service units. Considering these three layers and the way they relate each other, they are able to generate different types of cloud workloads: bag of task, map-reduce, loosely coupled services, etc. These workloads do not address transactional systems or, in some of them, are based on statistical distributions instead of using real workloads.

In the literature there are a great amount of research papers devoted to cloud optimization. However, the number of papers that consider a real workload is smaller. An early study is represented by [20]. In it, the authors analyze a set of real workloads in order to find their characteristics, develop a performance model and study different capacity planning methods. The real workload used is six month long with a resolution of five minutes. There are several papers ([21], [22] and [23]) that take as real workload the Google trace logs [24], [25], which are only one month long and are not focused on transactional systems. In [21] the authors synthesize realistic cloud workloads from the Google trace logs and latter use them to study the performance of two cases studies with Hadoop. The work carried out by [22] characterizes and clusters the tasks of the Google workload and, based on this classification, estimates the resources required by a new incoming task. In [23] the authors extend the target of the classification in order to identify the optimal VM placement strategy. Two final examples of optimization based on real workload are [26] and [27]. In [26] the authors characterize different real workloads in order to generate synthetic workloads to improve the new workload forecast in order to obtain the best auto-scaling algorithm. In [27] the authors focus on the reduction of the placement cost from the cloud provider point of view. None of the workloads used in these works are long and have high resolution at the same time.

When we focus on the problem of IaaS cost optimization from the point of view of the user and considering transactional workloads, there are only a small number of papers. This number is even smaller when the optimization cost algorithms deal with both on-demand and reserved VMs. Some examples of papers based on real workloads to conduct the cost optimization process are [28], [29], [30] and [31]. This last work has been extended in [32] to carry out the cost optimization process of multiple applications on a multi-cloud environment. However, all these papers use traces with low resolution that are not suitable for the one-second billing currently used by the most important cloud providers. To address this issue, in the work presented here we focus on the synthesis of a workload with the required characteristics of time slot resolution and trace length.

III. OPTIMIZATION PROCESS

In order to test the influence of the synthesis techniques proposed in this work on allocation problems, the Malloovia VM allocation strategy will be used in the experimentation. Although other allocation algorithms exist, Malloovia is a state-of-the-art algorithm, allowing both short time slots and long scheduling periods, and whose implementation is open source. This section summarizes the problem that Malloovia

solves, from a high level point of view. The mathematical details of the optimization process can be found in [32].

The first input of the problem is an enumeration of the different VM types offered by cloud providers. Since the same VM type can have different price and performance depending on the region and availability zone in which is deployed, and because it can be purchased under different pricing schemas (such as reserved or on-demand), a new concept, namely Instance Class, is used to denote each case. So, for example, an AWS EC2 on-demand c3.large VM type in the us-east-1 region is considered a different instance class than the same VM type in the us-east-2 region. These Instance Classes belong to Limiting Sets, which are a generalization of regions and availability zones. Limiting Sets are used to model the existence of limits imposed by the provider on the maximum allowed number of machines running at the same time. An Instance Class, IC_i, is thus defined by a set of parameters: the price per time slot (p_i), the pricing schema (on-demand or reserved), the limiting set to which it belongs, and the limits on the number of running machines.

The second input to the problem is the performance (perf_i) of each Instance Class for the kind of workload under consideration. This performance can be obtained by benchmarking or monitoring, or inferred by some other mechanism, out of the scope of this paper.

The last input to the problem is the expected workload, l_k, for each time slot t_k, for a number of time slots N equal to the length of the reservation period.

Given these inputs, the purpose of the optimization process is:

- to obtain the number, Y_i, of reserved machines of each Instance Class, IC_i, to purchase for the whole reservation period,
- and the number, X_i,k, of on-demand machines of each Instance Class, IC_i, to activate at each time slot t_k, so that
  - the total cost over the whole period is minimized,
  - the performance given for the active machines at each time slot is at least equal to the expected workload for that time slot,
  - and the limits imposed by the limiting sets are not exceeded at any time slot.

The two first items (relative to the total cost and the performance) are formalized in the two following equations.

Minimize:

\[ \text{Cost} = \sum_i Y_i p_i + \sum_{k=1}^N \sum_i X_{i,k} p_i \]  (1)

with the restriction:

\[ \sum_i Y_i \text{perf}_i + \sum_i X_{i,k} \text{perf}_i \geq l_k \quad \forall k = 1, \ldots, N \]  (2)

The third item (relative to limits) can also be expressed as additional inequalities, but we do not include them here to avoid introducing more complexity in the notation.

Since it is assumed that there is no dependence of the solution among time slots, all time slots with the same expected workload will have the same optimal allocation. This is the key for the "load-level aggregation" made by Lloovia [31], to reduce the size of the problem.

The workload is converted into a histogram which counts the number of time slots in which each workload level is expected. Then, unknowns X_i,k are rewritten as X_i,l, representing the number of VMs of instance class IC_i which need to be active at any time slot in which the workload is l. Equation 1 can be easily rewritten to use these variables and the histogram values, and restriction 2 is enforced for each possible workload level, instead of each time slot. This technique reduces the number of variables and thus the complexity of the optimization problem.

Lloovia and Malloovia operate in two phases:

- In Phase I, an estimation of the whole reservation period is used. This estimation does not need to be perfect, but it has to be long enough, and detailed enough (one value for each time slot). It is denoted by LTWP (Long Term Workload Prediction). This phase is solved in advance, and the result is used to purchase the required number of reserved machines.
- In Phase II the number of reserved machines is fixed, from the solution of Phase I. A second prediction is required, more precise and much shorter than in Phase I. This prediction, denoted as STWP (Short Term Workload Prediction) requires only the expected workload for the next time slot, since this second phase is performed online and in real time. For each time slot, if the expected workload coincides with any value in LTWP, the optimal solution is already known (from Phase I). Otherwise a linear programming problem is created and solved to obtain the optimal number of on-demand machines for the next time slot.

Note that, if we assume that LTWP was a perfect prediction, then the second phase is not necessary because all optimal allocations are already computed in Phase I. This assumption is of course unrealistic, but can be a useful assumption to postulate an "oracle" algorithm to which compare other techniques.

IV. TRACE SYNTHESIS TECHNIQUES

As explained in Section II, there is a lack of year-long traces with a resolution of seconds. However, traces with resolution of one hour are more common. In this section, several techniques to generate traces with resolution of seconds from traces with resolution of hours will be proposed.

The input for all techniques is the total number of requests per hour during a period (for instance, one year). For obtaining the number of requests per second, the requests in an hour must be distributed among all the seconds in that hour. It is important to note that the distribution of requests should make equal the sum of all the requests per second to the total number of requests in the hour.
Three basic approaches are proposed (see an example in the top plot of Fig. 1).

The first one, called **uniform** (in orange in Fig. 1), randomly distributes the total number of requests in an hour between its seconds with a random uniform distribution. With this approach, the number of requests per second is stable inside the hour. However, as shown in Fig. 1, there are discontinuities each hour and the peaks that appear in a real trace would not appear with this approach.

The second basic approach, called **smooth** (in blue in Fig. 1), connects the middle point between two successive workload levels with the middle point of the next successive workload levels. For example, refer to Fig. 2 and consider three successive workload levels denoted by $A$, $B$ and $C$ in the figure. Each one of these represents the average number of requests per second (rps) during a period of $T$ seconds (for one hour $T$ would be 3600). The total number of request per hour (rph) would be $A \times T$, $B \times T$ and $C \times T$, respectively.

Focusing on the second hour, in which the average rps is $B$, the total number of rph ($B \times T$ as said) is the area of the grayed rectangle in the figure. We define the point $AB$ as the middle between $A$ and $B$, corresponding thus to a workload level of $\frac{A+B}{2}$ rps. Analogously, $BC$ corresponds to the workload level $\frac{B+C}{2}$ rps.

It could be tempting to take the simple approach of making the workload vary smoothly between $AB$ and $BC$, i.e., following a line which connects those points. However, the total number of requests inside the hour using such a distribution would not add to the required value for that hour, since the area under that line would be less than the area of the grayed rectangle. To preserve that area, the curve connecting $AB$ with $BC$ cannot be a straight line. The simplest approach is to consider it a polygonal line by adding a new vertex, $H$ in the figure.

The workload level at $H$ can be calculated by imposing the condition that the area under the blue line has to be equal to $B \times T$. This area is the sum of four areas: the area of the rectangle under $AB$—$AB'$ (in light pink in the figure), plus the area of the rectangle under $BC'$—$BC$ (in dark green in the figure), plus the area of the triangle $AB$—$H$—$AB'$ (in light green in the figure), plus the area of the triangle $BC'$—$H$—$BC$ (in blue in the figure). This leads to the following equation:

$$B \times T = \left(\frac{A+B}{2}\right) \frac{T}{2} + \left(\frac{B+C}{2}\right) \frac{T}{2} + \left(\frac{H-(A+B)/2}{2}\right) \frac{T}{2} + \left(\frac{H-(B+C)/2}{2}\right) \frac{T}{2}$$

Solving for $H$:

$$H = \frac{1}{4}(6B - A - C)$$

Fig. 1: Example of approaches for distributing the number of requests per hour in seconds
The shape of the blue curve can be used as a probability density function, to randomly generate visits inside that hour, until the total number of visits is equal to $B \times T$. This way it is guaranteed that the total number of visits of the synthetic trace inside each hour is equal to the actual number of the original trace per hours, and that the overall shape of the synthetic trace follows (with random noise) the blue curve, which smoothly connects the middle points of the workload between successive hours in the original trace, as shown also in blue in the example of Fig. 1.

A third basic approach, called **constant** (in black in the top plot of Fig. 1), distributes equally the requests in an hour so that each second has the same number of requests. The total number of requests in an hour is divided by 3600 (the number of seconds in an hour) and each second is assigned that number of requests. As the number of requests in a second has to be integer and the quotient might not be integer, the remainder of the integer division is distributed among the first seconds in the hour. Thus, the first seconds might have one more request than the rest.

The last strategy has the disadvantage that the number of requests per second in each hour is constant, which is unrealistic. In order to add variability, new traces can be generated by adding three kinds of noise: **white**, **pink** and **brown** (see three bottom plots of Fig. 1). White noise is sampled from a standard normal distribution and the corresponding signal has equal power in any bandwidth of the same width, while pink noise loses 3 dB per octave and brown noise loses 6 dB per octave. White noise has no memory, i.e., each value is totally independent of the previous one. On the other hand, brown noise is equivalent to the sum of random increments, so it has long-term memory. Pink noise is a case in between the other two, and has been found in many physical systems. Using these three kind of noises allows analyzing how much the shape of the variations inside the hour influence the results.

These noises have a mean of 0, so the total number of requests in the hour is not altered. In order to make the noise proportional to the workload level, it is normalized between -1 and 1 and then the constant trace is multiplied by the normalized noise of each kind. This can generate negative values in parts where there is a small number of requests in the constant trace and a big negative spike in the noise. As these negative values do not have physical sense, they are substituted with 0.

### V. Evaluation

In order to test the influence of the different synthesis techniques on the optimization process, the ideal procedure would be to compare the results obtained with the different synthetic traces and a real trace. However, as no real year-long trace with a resolution of seconds for transactional systems is available, the longest one available, which is the trace for the Wold Cup 98 [33], will be used. This trace is only three-month long, so the results might be different if a year-long trace was used. That problem will be investigated in Section V-B.

#### A. Comparison of synthesis techniques

Research in allocation algorithms has to compare the cost of the allocation they obtain and the time to obtain it, i.e., the solving time. To assess how much the trace influences the results, Malloovia will be used as allocation technique to solve, first, the real trace and, then, the different synthetic traces.

Taking into account that there is a minimum one-minute billing time in most public cloud providers, and that VMs take some seconds to start up, it is not realistic to generate a new allocation every second. Thus, the allocations will only be generated every minute. Consequently, what will be considered as real trace is the trace with a resolution of seconds resampled by minute. To resample each minute, the maximum value inside the minute is used, so that the workload is correctly served for each second.

In order to emulate the typical situation of having traces with a resolution of hours, the real trace will be resampled per hours and the resulting trace will be used as input for the different synthesis methods explained in Section IV.

To carry out an experiment, a mix of VM types from Amazon ECS has been selected (see Table I) as input to the allocation strategy. The experiments were run on an Intel Core i7-4790 with 32 GBytes of RAM running Ubuntu Server 16.04 with Python 3.6.3, CBC 2.8.12 and Malloovia 1.1.0.

As shown in Fig. 3a, the cost of the synthetic traces smooth, uniform and brown noise is similar to the one obtained for the real trace. However, the other three traces have a significantly different cost, with the constant trace having a cost of 85.4% of the real trace one, while the traces with white and pink noise having a cost of 119.1% and 113.3% of the real trace one,

### Table I: Performance and price of different VM types used in the experiments

<table>
<thead>
<tr>
<th>VM type</th>
<th>on demand ($/h)</th>
<th>reserved ($/h)</th>
<th>perf (rph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c4.large</td>
<td>0.124</td>
<td>0.079338</td>
<td>32800</td>
</tr>
<tr>
<td>c4.xlarge</td>
<td>0.249</td>
<td>0.159703</td>
<td>65600</td>
</tr>
<tr>
<td>m4.large</td>
<td>0.117</td>
<td>0.079338</td>
<td>26650</td>
</tr>
<tr>
<td>m4.xlarge</td>
<td>0.234</td>
<td>0.158790</td>
<td>53300</td>
</tr>
</tbody>
</table>
Take into account that the traces were generated to have the same total number of requests per hour as the real trace, this is a result that has to be investigated further.

Fig. 4 shows the traces aggregated by minutes. It corresponds to the same hours shown by seconds in Fig. 1. While in the traces by seconds it looks like the traces with pink and white noise have some times greater values and some times smaller values than the real trace in a similar proportion, in the trace by minutes (Fig. 4) they have almost always greater values. The reason for this is that, as shown in the traces by seconds (Fig. 1), these traces have in general bigger spikes inside the minute. Thus, when aggregated, as explained above, by taking the maximum inside of the minute to guarantee the performance in any second, the total number of requests is greater. Accordingly, the constant trace has a maximum inside each minute smaller than the real trace and, thus, the aggregated trace by minutes is smaller than the real trace. This explains the difference in cost for these traces.

Regarding the time to obtain the solution (see Fig. 3b), there are even greater differences, with the constant trace having a solving time of only 11.1% of the real trace one, while the white and pink traces requiring 163.4% and 126.5% of the solving time of the real trace, respectively. The reason for this is that the number of different workload levels generated for smooth, uniform and brown traces is similar to the number of workload levels in the real trace, while the constant trace has fewer levels and the white and pink traces have more, and the number of workload levels affects the size of the optimization problem to solve.

From this analysis it can be concluded that the synthesis method has an impact on both cost and solving time, but it is specially significant in the latter. In addition, for this trace, the best results in both metrics have been obtained with the smooth, uniform and brown noise synthetic traces.

B. Comparison of extrapolation of short traces to long traces

Traces with one second resolution for a whole year are not available, but obtaining one second resolution public traces for shorter periods is possible. This may lead researchers to carry out their experiments with shorter traces and extrapolate for the whole year. In order to test if this approach is valid, an experiment has been carried out using the Wikipedia trace [34].

This trace encompasses several years, but has one-hour resolution. The requests per hour of one year were used as base trace and, with the synthesis techniques proposed in this paper, one-second resolution traces were generated and resampled in minutes. Then, an allocation that used only three months was obtained applying Malloovia. This provided the cost and the solving time for three months. These values were extrapolated
Another interesting observation is that not even the order in solving times between traces is preserved: the trace with white noise takes longer than the trace with pink noise in the extrapolated experiment, while when using the whole year the order is reversed. In addition, the differences between methods are much greater in the extrapolated version than with the whole year trace.

This experiment shows that using short traces and extrapolating the results can generate misleading conclusions, especially when comparing the computation times of different allocation techniques.

VI. CONCLUSIONS

This paper has studied the influence of the resolution and length of workload traces in the process of optimizing transactional systems in cloud computing. First, it has been noted that there has been a recent change in the billing period of major cloud providers, which used to use one hour as billing period and have changed to charge by seconds. The study of the related work has shown that there are no public traces available with this resolution with a length of one year.

To overcome this problem, several techniques to generate synthetic traces have been proposed. The cost obtained with Malloovia, a state-of-the-art cost optimization technique for VM allocations, has been studied, showing that the synthesis technique has a significant influence on the cost obtained compared to using a real trace. In addition, the solving time, which is also an important parameter to compare different optimization techniques, has been shown to be greatly affected by the synthesis technique used.

Another approach, using a short trace of some months and extrapolating its results to a whole year, has also been studied. The experiments carried out have shown that the results for the extrapolated trace are not totally applicable, specially in the solving time, which is not extrapolated correctly and depends very significantly on the trace used.

In summary, this work has shown that real traces of one year with one second resolution are needed for comparing the different cost optimization techniques proposed in the research community. In addition, it has provided a set of synthesis techniques that can be used while real traces are not available. The code for these techniques has been open sourced and the traces used in this paper have been made publicly available2 so that researchers in the field can use them to test new cost optimization techniques for cloud computing.

As future work, we are working on obtaining a real trace of one year with one second resolution. This can help in having a real case to compare allocation techniques. Furthermore, this real trace can be used to validate more thoroughly the synthetic techniques presented here, which can be used to obtain other synthetic case studies. In addition, we will work in developing allocation techniques that take advantage of the new billing periods introduced by public cloud providers and the traces obtained by the techniques presented in this paper will be used to compare against other allocation techniques.

2https://github.com/asi-uniovi/traces-seconds