Minimizing Resource Rent Loss while Maximizing User Availability in Cloud Applications through Online Switching of the Scaling Method

Marc E. Frincu  
Research Institute e-Austria and West University of Timisoara  
Timisoara, Romania  
Email: mfrincu@info.uvt.ro

Adrian F. Spataru  
West University of Timisoara  
Timisoara, Romania  
Email: spataru.florin@info.uvt.ro

Abstract—Clouds offer access to virtually unlimited resources. Applications requiring a variable number of resources (i.e., elastic) are hence ideal for such a scenario. As cloud resources are usually paid by the hour applications need efficient ways of predicting their requirements in order to reduce costs. In addition we need to maximize user satisfaction through high application availability. Finding the best balance between them is a crucial task. In this paper we consider the case of commercial web applications and focus on the hit rate as scaling factor. Since it is known that no scaling algorithm can have a definite edge over the rest we propose three strategies for online scaling strategy switching. The first one focuses on analyzing the probability distribution of the historic hit rate and determining the best scaling method for it. Finally, the third one relies on changing the method without needing to know the distribution through random choices. Results show that given the fluctuations and uncertainty of cloud environments and user behavior the most suited method for online selection of the scaling strategy is the first one based on the best strategy.

Keywords—cloud computing; elastic applications; scaling methods; adaptive scaling

I. INTRODUCTION

Cloud systems offer access to virtually unlimited resources. Hence they are ideal for applications needing to scale (i.e., elastic applications) based on the resource demand. These applications rely on scaling methods to efficiently allocate on demand resources.

As depicted in Sect. II many scaling algorithms exist. Their functionality depends on various scaling factors that determine the scaling degree of an application. These factors are usually derived from a historical trace of events. The trace can contain resource characteristics (e.g., CPU usage, number of used processor, cores, etc.), the number of rented virtual machines, information on start time, duration, failures, etc. This data has been already used in many papers involving scheduling algorithms. It can be extracted or derived from repositories such as the Grid Workflow Archive [1] and the Failure Trace Archive [2], from actual production clouds [3] or can be simulated based on statistical data [4]. Another possible relevant trace information, which has been somewhat neglected is related with the user hit rate. This is especially important since clouds are becoming home to many elastic web applications. We consider the hit rate as the primordial factor which influences both resource usage and virtual machine (de)allocation. Web applications especially commercial ones rely heavily on the hit rate for their income. The traffic generated from the hit rate is composed of a fairly cyclic pattern given by loyal visitors and also of abrupt shifts caused by (un)foreseen events (e.g., ads, promotions or special events). It is the unpredictable pattern which really tests the efficiency of a scaling algorithm.

Two main scaling strategies exist: forecast and reactive based. Each of them has one major drawback that can be seen in most algorithms that were proposed over the years: the delayed reaction to abrupt changes. While it can be argued that reactive methods can respond instantaneously, the time needed to start a virtual machine can take up to several minutes. In case of abrupt traffic surges [5] this is translated into losing users due to response delays. In scenarios involving monetary costs – e.g., commercial web sites – any delays can lead to less profit being made and allocations occurring after the hit rate has diminished are useless and only cause loss due to futile resource rent.

To make things even more complicated Wolpert [6] has shown that there are no prior distinctions between learning algorithms. The previous sentence can be generalized for forecast scaling methods and argue that no method has a definite edge over the rest. Thus if the distribution of the learning data changes the scaling algorithm will probably require to be changed as well in order to keep the monetary loss due to resource renting low. Keeping the loss low is however not sufficient since we also need to take into account the user satisfaction which indicates the degree in which the users using the applications have fast (in terms of response times), reliable, secure and accurate access to it [7].

In this paper we plan to analyze several methods for online switching of scaling algorithms so that we minimize the application owner loss while maximizing user satisfaction. Since we target mainly web applications we will deal with the number of serviced requests from the total required as indicator of the user satisfaction.
The rest of the paper is structured as follows: in Sect. III we analyze several scaling methods and compare their efficiency on some real and generated traces. Then we address the issue of dynamically switching between scaling methods at runtime in order to minimize rent costs. As seen in Sect. IV three methods are proposed and discussed. Section V analyzes the efficiency of the proposed method while Sect. VI concludes the work with some general conclusions and future work.

II. RELATED WORK ON SCALING ALGORITHMS

Many scaling algorithms have been studied over the last years. As mentioned in Sect. I they can be divided into two categories: predictive and reactive. Kupferman et al. [8] present three algorithms: linear regression and autoregressive of order 1 from the first category; and Rightscale from the second one. The first two rely on a history window to approximate the next value while Rightscale relies on a voting mechanism mixed with a threshold approach when deciding whether to scale or not.

Although regressive methods offer a solution to the scaling problem they suffer from a delay in detecting inflection points as they require to reach those points before being able to consider them. A better approach would be to attempt to match the current history window against similar historical patterns. Finger et al. [9] and Caron et al. [10] present pattern based methods for determining the resource consumption forecast. In [9] a method relying on unsupervised learning (called Use Pattern Analysis) is presented while in [10] the authors propose a prediction algorithm based on the Knuth-Morris-Pratt string matching algorithm. Both papers show advantages in using pattern based prediction but cannot definitely argue that their methods are better [10].

Supervised machine learning has also been considered. Islam et al. [5] propose an Error Correction Neural Network based scaling method and compare it with a simple linear regression. Tests are performed on CPU usage and results show the benefits of using neural networks.

Espadas et al. [11] propose a different approach relying on a tenant-based resource allocation. The authors show the method to be efficient in reducing the underutilization but state that no statistically improvement of the over utilization was noticed under the t-student tests.

Besides predictive and reactive ones we also have hybrid solutions in which the general trend is given by the predictive method while the adjustment is done by the reactive one. So we can predict using a regression model and adjust any wrong estimates by using Rightscale for instance. These approaches are however not problem free as they suffer from the same issue as the reactive ones: the time needed to start up the virtual machine.

III. ANALYSIS OF VARIOUS PREDICTIVE SCALING ALGORITHMS

In this section we present a behavioral study of several scaling policies when confronted with various trace data. Our goal is to study the shape of the real traces, derive synthetically generated ones and check whether the “best” scaling algorithm for the real trace holds also for the synthetic ones. By “best” algorithm we refer here to the one that gives the less profit loss due to renting of resources while offering high user satisfaction as measured by the UCSB metric [8] (cf. Sect. III-B).

A. Input Data Analysis

Our real trace repository consisted of four traces belonging to various websites. In order to catch a wide range of hourly hit rates we used a small traffic website (S1) with 2,044 hours of traces, two medium sized websites (called M1 and M2 respectively), both with 165 hours of traces, and a large traffic website (L1) containing 175 hours of traces.

Figure 1 depicts the hit rate traces for the four websites. All traces exhibit a cyclic behavior with a period of approximately 8 hours. Additionally L1 shows an apparently descending pattern (cf. Fig. 1(d)) with an abrupt fall in the hit rate around time 117. M1 and M2 are particular interesting as they show two (cf. Fig. 1(b)) respectively one (cf. Fig. 1(c)) unexpected spikes.

Besides the trace information we also considered the average page size in order to determine the rate of concurrent requests a web server can accommodate. These are of 3kB (S1), 102kB (M1), 49kB (M2) and 89kB (L1). These values were considered to be the size of the HTTP response (including headers), while the HTTP request size was neglected for simplicity.

Each trace follows a certain probabilistic distribution which can be found by either trying to fit the data to a certain distribution or by constructing a distribution function with the desired shape [4]. For our experiments we opted for the former approach and used Q-Q plots to determine the best fitting distribution. Figure 2 depicts the Gaussian kernel density estimations of the four traces. It can be noticed the long tails present in all traces except L1. Such skewed distributions tend to contain data that dominates the calculation of large moments \( \mu_k = 1/n \sum_{i=1}^{n} x_i^k \), where \( n \) represents the number of trace elements \( x_i \). These extreme values from the distribution’s tail are unlikely to occur if a different time frame would be picked. Thus we need to select a sample set that is representative for the entire distribution. This approach is however undesired in case we want to catch unexpected (rare) events such as the ones in the extreme tail.

Analyzing the Q-Q plots with the full traces lead however to inconclusive results for S1. As a result we eliminated the extreme dominant values and repeated the analysis. We managed to fit the distribution of S1 to a Gamma distribution with a shape of 2.14 and a scale of 0.11. The rest of...
the distributions did not show any significant changes as depicted in Table I. The Q-Q plots used for analysis were created in Wessa [12].

It can be noticed that each trace has a different distribution. This is true even for websites with a relative identical amount of average hit rates (i.e., M1 and M2).
Table I

<table>
<thead>
<tr>
<th>Trace</th>
<th>With elimination of extremes</th>
<th>Without elimination of extremes</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Gamma (2.14, 0.11)</td>
<td>inconclusive</td>
</tr>
<tr>
<td>M1</td>
<td>Weibull (5.12, 230.59) or Poisson (211.48)</td>
<td>Poisson (220.43)</td>
</tr>
<tr>
<td>M2</td>
<td>Lognormal (5.54, 0.54)</td>
<td>Lognormal (5.63, 0.64)</td>
</tr>
<tr>
<td>L1</td>
<td>Poisson (197,468.93)</td>
<td>Poisson (197,468.93)</td>
</tr>
</tbody>
</table>

B. Analysis of the Scaling Methods

To determine the “best” scaling method for a given data distribution (cf. Table I) we performed a series of 10 repeated tests on each synthetic trace. Each test considered four scaling methods selected from those presented in Sect. II: linear regression (LR) [8], autoregressive of order 1 (AR1) [8], last 4 (L4) [9] and pattern based (PB) [10]. Furthermore we also tested the methods on our initial traces: S1, M1, M2 and L1.

For each method, except L4 which does not have any parameters, a parameter sweep was performed in order to determine the best initial configuration. For the LR tests the history window size varied from 2 days to 40 days. AR1 uses two parameters, history window size and adaptation window size, which varied between 3 and 40, and 1 through 3 respectively. Finally PB varied its history window size from 2 to 15 and distance acceptance threshold from 1 to 100.

The efficiency of the prediction methods was measured by two metrics: UCSB [8] and Loss.

The UCSB score is given by

$$UCSB = \frac{(A_{log})^{\gamma} C - \gamma C/A_{log} + \beta}{\gamma C/A_{log} + \beta},$$

where $\alpha = 2$, $\gamma = 1$, $\beta = 50$, $C = noCPU/\text{per Hour} \times 0.085$ (rent cost per hour) and $A_{log} = -log(1 + \delta_0 - A)$. $A_{log}$ signifies that a scaling’s method value increases exponentially with respect to its availability $A = no\text{ServicedRequests} / no\text{TotalRequests}$. The value for $\delta_0 = 10^{-9}$, $\alpha$, $\gamma$ were taken from [8].

The Loss metric considers only the $ loss inflicted due to over provisioning.

In this work we use a combined metric that aims at maximizing UCSB while minimizing $ loss.

For simplicity we assumed a worse case scenario in which an application is executed on the least powerful machine available on Amazon’s EC2 [13]. This machine is hosted on the US East Virginia region, runs a Linux OS, has 1.7GB RAM and 1 CPU core. We assumed the network link to have 100Mh. Although the price for this machine is small, $0.085, its characteristics do not allow a normal amount of concurrent requests. Assuming – given the average page size – that this number is limited by the minimum between the capacity of the network link and the memory size we obtain a maximum of 4,266 users for S1, 125 for M1, 261 for M2 and 143 users for L1. These numbers also provide an insight on the average number of machines needed every hour for each trace. While the small and medium websites can accommodate the average request rate with just one respectively 2 machines the large traffic site requires 1,428 machines if all requests are constant and equal with the average. Even by increasing in the network link to 1Gb we would still require an average of 138 machines.

Results, summarized in Table II have confirmed our assumptions. In case of the Gamma distribution (corresponding to S1) the PB method which is the best method for S1 gave the best results in 100% of tests. For the Poisson distribution (corresponding to L1) tests also concluded that the best scaling method for L1 (i.e., AR1) gives the best results in 70% of cases. For the other 30% the LR method was the best. For the Weibull and Lognormal distributions corresponding to M1 respectively M2 the results were somewhat inconclusive as the results shown an average of 60% success for the LR method which was best in the real trace tests. The other 40% gave AR1 as the method providing the best result. This is however of no surprise as the results in the case of the real traces show a small difference between the results of the two scaling methods. Given these small numbers either of the two methods could be consequently chosen without a major impact on the outcome. The difference between scaling methods in terms of inflicted loss will be exploited in Sects. IV and V when a several online selection methods will be analyzed.

IV. Methods for Online Scaling Policy Switching

As seen in Sect. III-B the shape of the trace’s distribution is essential in determining the “best” scaling algorithm. Due to the dynamic nature of user hit rates especially on elastic commercial websites hosted on clouds is prone to changes. In order to maintain the losses due to resource provisioning low while keeping user satisfaction high we require a mechanism to dynamically switch the scaling method in order to fit the new distribution shape.

In what follows we propose three methods for online scaling method switching. Online switching can be achieved...
by either a mechanism for selecting the “best” scaling method for a given access pattern or by employing learning techniques. These mechanism were also investigated by the first author for the case of scheduling algorithms for grid systems [14] and their efficiency was proven. To our knowledge no study exists for the case of the scaling algorithms. In this paper we focus on several variations for selecting the scaling algorithm based on the “best” provided solution. We also explored the possibility of using supervised learning techniques such as neural networks but we did not observe any improvements on the results. For this reason we depict in what follows the three variations of the “best” selection approach.

The first method determines at each time step the “best” method by performing a parameter sweep on all methods and selecting the one that provides the lowest loss – i.e., *always best*. While this method is quite efficient it exhibits a problem in that it relies on a proper selection of the parameter sweep window. This could be set to a predetermined interval but it does not guarantee that the achieved loss would converge to a global minimum, i.e., it is possible that there is another parameter sweep input configuration for which the loss is even smaller. This method also requires to be executed throughout the life cycle of the application at each time step.

The second one is based on periodically recomputing the shape of the distribution. Once a new shape is found the new “best” scaling algorithm would be determined by running a parameter sweep on all the available methods and determining the one that provides the lowest loss – i.e., *always best periodical*. While this method is also quite efficient it also depends on several nondeterministic variables: the interval for determining the new distribution and the range for the variables used in the parameter sweep (cf. Sect. III-B). The method exhibits the same problems as the first one but in case of long relatively stable hit rates in terms of the distribution shape it does not need as many “best” algorithm selections.

The third method relies on determining at each time step the “best” method through guessing – i.e., *always guess*. While this method would not be as efficient as the previous ones it would provide some loss minimization which would be otherwise hard to achieve. To prove the previous statement we performed a series of parameter sweep tests on L1 as well as on synthetically generated traces based on L1’s distribution (i.e., Poisson cf. Table I).

V. Tests

The tests we performed aimed at comparing the first method and third method – i.e., *always best* and *always guess* – against the cases in which a single method is used throughout the application life cycle. The reason for omitting the *always best periodical* method was due to its similarity with the *always best* approach. Therefore we assumed its results to compare with the latter in an optimal scenario when distribution computation is done exactly at the moment it changes. Since this moment is hard to predict online the *always best periodical* method could very likely degrade to the *always best* one where the check is done at each time step.

To validate our tests we compared the results for the L1 – S1, M1 and M2 were omitted since they give insignificant $S$ losses – with the synthetic traces produced by its derived distribution (i.e., Poisson). In addition we also used a trace following a scaled Weibull distribution derived from M1.

The results depicted in Fig. 3 are the averages of a series of 100 repeated parameter sweep tests. The historical window sizes – in hours – used for LR1, AR1 and PB ranged from a 2 up until a value equal with 25% of the total size of the trace at hand.

Our experiments considered a real case scenario in which the real data is always used in the historical window since it is always quite possible to obtain actual data starting from one hour in the past and going further back.

The experiments we performed aimed at determining both the $S$ loss due to resource renting and the user satisfaction measured in terms of availability (i.e., percentage of satisfied requests).

A. Test Results

The test results are depicted in Fig. 3. Each line corresponds to one trace and shows the loss and availability as given by each method. As it can be seen for all three trace scenarios (L1, Poisson & Weibull based) the method that produces the best availability is the *always best* method. The most obvious difference is shown in Fig. 3(f) where the difference to the second best method in terms of produced availability is of 0.3%. It can be noticed that while this method is not giving the smallest loss it is the most stable when compared with the cases in which a single scaling method is used throughout the application execution time (cf. Figs. 3(a), 3(c) and 3(e)). Furthermore it is second only to a single method that depends on the trace. Since the *always best* method always selects the “best” method out of LR, AR1 and PB at each time step it is only natural that it should perform at best as the most efficient one. However due to the small time needed to tests all three methods at each time step (i.e., less than 1 minute) this approach could be preferred to one using a single algorithm for all the life cycle of the application. The motive is obvious if we consider a not so well choice of the scaling method and consequently the extra losses it inclicts.

Considering now the *always guess* method we notice that although it is not as efficient as the always best method in terms of availability and loss it is still quite stable in its results, without showing any major fluctuations with regard to the rest. Due to its random approach it is only natural that its results should vary between the ones produced by
the methods it produces. This variation is less obvious in cases in which all methods give similar results. However such cases should not be considered frequently and hence the efficiency of this method should be treated accordingly.

From the results we also notice the variations in terms of loss given by LR, AR1 and PB. For example we can take the case of LR and AR1 in Figs. 3(a) and 3(c). We notice here that in the former figure LR is the loser while in the latter one it is clear the winner in terms of loss. This is just another example of how dependent are the methods on the shape of the trace’s distribution.

When comparing the overall results for availability and loss for the cases of LR, AR and PB we notice in addition that the method offering the smallest loss does not offer the highest availability. This is not the case of always best or always guess. As such we need to consider properly selecting the method which offers the best balance. This is not always feasible and results could change when the trace’s distribution changes. So we consider that the always best to be the best method given the wide range of uncertainty found when considering scaling scenarios. This is backed by the fact that the time needed to run the parameter sweep for
determining the best historic window for all methods and so “best” method at that particular time step is under one minute – taking into account that rent is on an hourly basis and abrupt and sudden spikes are not frequent. Nonetheless if desired the interval between to executions of the always best can be augmented by studying the shape of the trace distribution. This leads as to the always best periodical method which is the topic of future work.

VI. CONCLUSIONS

In this paper we have analyzed various methods for predictive scaling in cloud applications. The reason behind our work is the well known fact that the efficiency of the learning method depends on the input data characteristics. Since the usage of cloud based application is subject to user characteristics (e.g., age, interests, gender, work schedule, etc.) these properties of the input data changes in time. So we require methods that are independent on these factors. We therefore proposed and tested three methods to switch between various scaling method at runtime. Efficiency was seen as the methods capability to maximize the user availability while minimizing the cost due to over provisioning. Results have shown that if we chose at each step the “best” algorithm, i.e., always best method, the average cost due to over provisioning is almost equal with the algorithm which gives the best result on that specific trace. The advantage of choosing at each step the “best” is its stability in terms of $S$ loss and its large availability rate. In addition results on another method, always guess has an average loss and an average availability between the values of the basic algorithms, being stable through different trace distributions. This result can be used in adaptive cloud platforms that need to provide the best scaling decision for their web applications when the hit traces is known.

Future work will involve integrating the adaptive scaling methods studied in this paper in the scalability module of the AMICAS platform [15]. We also plan on further investigating the possibility of using unsupervised learning techniques such as those based on clustering in order to see whether or not any improvements on our results can be made. Finally we intend to find a method to dynamically adjust the interval between sweeps in case of always best periodical.

ACKNOWLEDGMENT

The work of the first author has been partially funded by the Romanian AMICAS project grant of the Romanian National Authority for Scientific Research, CNCS UEFISCDI, project number PN-II-ID-PCE-2011-3-0260.

REFERENCES


