U.P.B. Sci. Bull., Series C, Vol. 76, Iss. 3, 2014 ISSN 2286-3540

PREDICTIVE PROVISIONING OF WORKLOADS FOR DYNAMIC APPLICATION SCALING IN CLOUD ENVIRONMENTS

Octavian MORARIU[1], Cristina MORARIU[2], Theodor BORANGIU[3]

The large scale emergence of cloud platforms induce the tendency to virtualize application workloads that traditionally ran on physical machines. At the same time, cloud providers advertise unlimited resources available to the customers at any time for a fixed price. These factors create the opportunity for customers to easily scale up and down the infrastructure depending on the real time requirements, reducing the overall costs for providing the service. Cloud platforms today provide a threshold trigger mechanism that can trigger provisioning or de-provisioning of additional resources.This paper argues that the threshold approach is not enough for some real life application scaling requirements and introduces a predictive mechanism that allows accurate and proactive provisioning of workloads. The prediction algorithm is based on the observation that for some applications a usage pattern exists, and this usage pattern is repetitive. This paper presents the usage pattern identified in a large scale travel booking application and the execution of the algorithm on this data. The algorithm tested using IBM CloudBurst 2.1 deployment using a benchmark application and results are discussed.

Keywords: Cloud computing, scalability, usage patterns, predictive provisioning, threshold provisioning

1. Introduction

Cloud computing paradigm promises to resolve the problem of system capacity and at the same time to keep the resource utilization at maximum. Historically the system capacity was fixed and determined by the estimated peak load that the system should support. This approach works well when the behavior of the user base is generally constant in regards to the load applied to the system. However in real world scenarios this is rarely the case and has generally caused poor resource utilization and even system outrages or loss of service if the estimated peak load has been exceeded. Elasticity can be defined as the capability

[1] Eng., Faculty of Automatic Control and Computer Science, University POLITEHNICA of Bucharest, Romania, e-mail: octavian.morariu@cimr.pub.ro
[2] Eng., Cloud Research Department, Cloud Troopers Intl., Cluj-Napoca, Romania
[3] Prof., Faculty of Automatic Control and Computer Science, University POLITEHNICA of Bucharest, Romania, e-mail: theodor.borangiu@cimr.pub.ro

of a system to automatically increase and decrees its capacity based on real time load without the intervention of the system administrator.

The innovations in virtualization technologies together with the real time monitoring capabilities implemented by cloud providers allow development of cloud based elastic systems. System elasticity has been studied as part of self-optimization problems in autonomic computing area [2] focusing on the system design, which has a great impact on elasticity.

Commercial cloud providers offer various techniques to support elasticity for customer applications. Amazon E2C cloud offers a service called Elastic Load Balancing [5] that abstracts the complexity of managing, maintaining, and scaling load balancers. The service is designed to automatically add and remove capacity as needed, without needing any manual intervention. The Amazon Elastic Load Balancing service architecture has two logical components: load balancers and a controller service. The load balancers are resources that monitor traffic and handle requests that come in through the Internet. The controller service monitors the load balancers and adds and removes capacity based on the load. Also the controller service monitors the real time behavior of the load balancers. The scaling of the capacity in Elastic Load Balancing can be configured by defining rules that operate on the following metrics gathered through Amazon CloudWatch [4]: Latency, Request count, Healthy hosts, Unhealthy hosts, Backend 2xx-5xx response count and Elastic Load Balancing 4xx and 5xx response count.

Unlike Amazon E2C, RackSpace does not provide any built in auto-scaling mechanisms. Instead RackSpace provides an API for monitoring and control of the hosted workloads. The responsibility for monitoring and scaling the service in this case is with the customer. For this, a workload management API for provisioning and de-provisionig is offered by RackSpace to its customers. External tools like Scalr [6], an open source project, are filling in the missing functionality by handling scaling of cloud applications hosted by RackSpace. Scalr is using a set of user defined metrics and rules to scale up and down. Generally this approach requires more configurations from the customer side, but can offer better reactivity from the application on the real time load. A complete comparison on the commercial and open source cloud implementation for scaling capabilities is presented in the INRIA Research Report 2012 [7].

Private cloud implementations like IBM CloudBurst 2.1[16] show that scalability is implemented based on predefined thresholds [3]. The thresholds can be defined on low level metrics like CPU usage, Memory usage or disk usage. In CloudBurst 2.1 this is implemented by IBM Tivoli Monitor[] that uses a Linux based OS agent to collect real time metrics and monitor predefined thresholds. When a threshold is reached, a new workload instance can be provisioned or de-provisioned based on the predefined work-flow. The workflow is executed by

IBM Tivoli Provisioning Manager [9] and interacts with the underlying virtualization technology, in this case VMware VCenter[10].

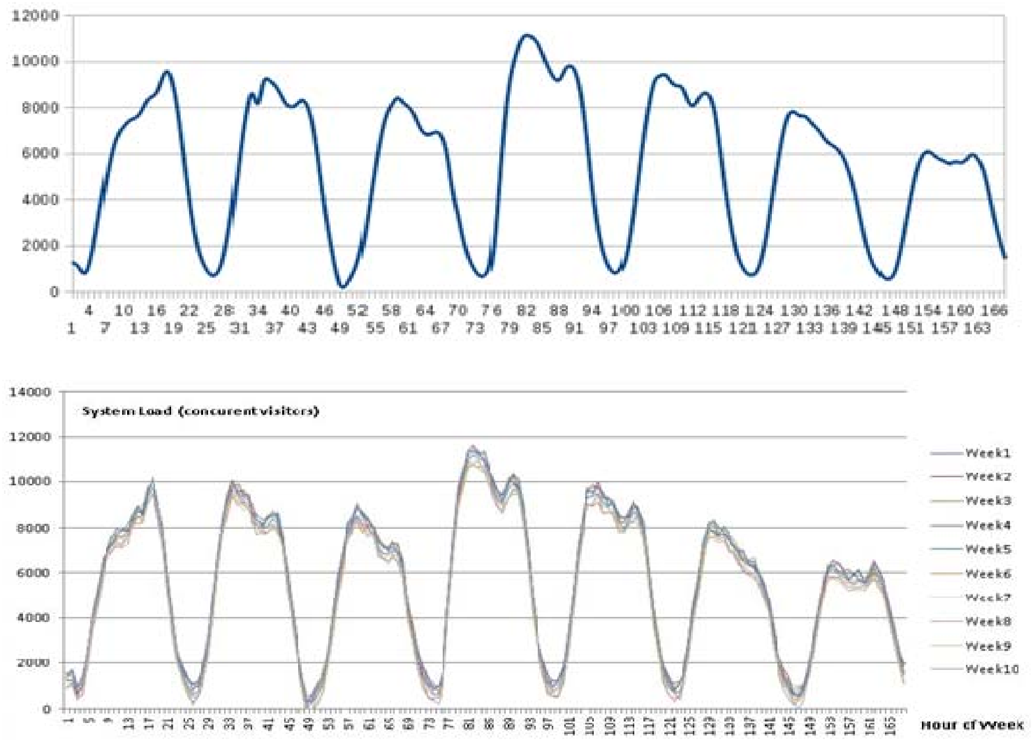


Fig. 1. Usage patterns (one week and ten week average)

One characteristic common to the implementations described is that these are reactive in nature. In other words the decision to scale up and down is taken based on some predefined rules that are evaluated against real time metrics. This approach is flexible and works well for generic applications. However, for real life applications with a more specific purpose, an advanced predictive scalability model based on repetitive usage patterns can assure better resource utilization.

This paper presents such an approach that augments the threshold mechanism, with information based on the historical repetitive usage patterns. In the following section the paper presents an industry example focusing on repetitive usage patterns and the deficiencies of the threshold model. Finally the paper presents the predictive mechanism developed based on usage pattern and the experimental results obtained during tests performed on IBM CloudBurst 2.1 running DayTrader[11] benchmark application against various load tests.

2. An Industry Example for Usage Patterns

By employing web application monitoring tools like Google Analytics[12], service providers can now determine in real time the load and usage of the applications. In this section a real life industry use case is presented. The application considered is a web based travel booking application. The implementation is based on Drupal[13], Ajax[14] and Symfony[15] PHP based frameworks. The end user population is US based. The weekly load on the application is illustrated in Fig. 1 (up).

The above image illustrates the number of concurrent users accessing the application during 168 hours of the week starting with Sunday. As the user base is regional, or in other words is located in a single time-zone, the first observation is that during the night time the load on the application is clearly lower than the usage during the day time. Also, is easy to observe that the patterns for Monday, Tuesday, Wednesday and Thursday are similar, showing a first peak during the morning hours, a decline during noon and a second smaller peak during the afternoon hours. A special pattern can be seen on Sunday, with a single peak during the late afternoon hours. Similarly on Friday the pattern show a single peak during the morning hours and a steady decline during the afternoon. Saturday shows yet a different pattern, with two equal peaks during the morning hours and the afternoon hours and a longer decline during the noon hours. To determine if these behaviors are repetitive, a comparison has been done against the hourly average of the previous ten consecutive weeks. The results are presented in Fig. 1 (down).

The comparison clearly shows that both the usage patterns and the amplitude for each day are very similar and have a strong repetitive characteristic when considering a weekly interval. It has been calculated that the maximum deviance from the average is under 200 users. This can be explained by the user behavior in regards to a travel booking application by analyzing the actions performed and the reasons why these actions and travel decisions are done at given times during the week.

For the application providers the important aspect is that these patterns exist and can be used to provision with great accuracy the capacity required at each hour during the week. A threshold mechanism has certain limitations when the granularity of the application, given by the number of application instances is high. In [1] the following example is proposed: considering a linear scalable application, when going from 1 to 2 instances, the capacity is increased by 100% but going from 100 to 101 machines increases capacity by only 1%. The relative capacity increase when adding one additional resource is not constant when the granularity is high. This leads to the conclusion that a static threshold approach would not be appropriate for high granularity applications. To overcome this, a proportional thresholding mechanism is introduced in [8].

Another important factor to consider is the time taken to provision an additional resource. This typically includes the provisioning of the workload in a virtualized environment, the startup of the workload, dynamic configuration of services and load balancing and startup of the application. This provisioning time cannot be ignored when planning capacity requirements and elasticity, because it is significant even for simple applications. The time taken for provisioning and de-provisioning an application instance for the application considered in this industry study is represented in Fig. 2.

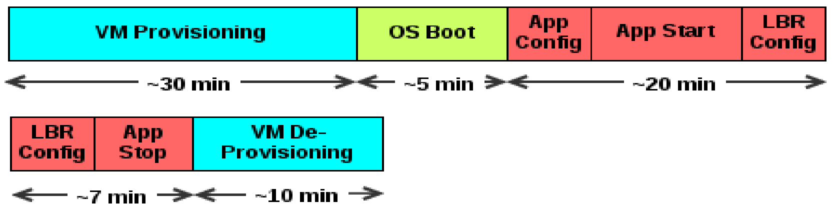


Fig. 2. Provisioning and de-provisioning time

The travel booking application considered in this industry example is hosted in RackSpace and the total provisioning time of a new instance averages at 55 minutes with a 25% variation depending on the time of day when the operation is performed. As expected the de-provisioning time is considerably smaller as it involves LBR re-configuration, graceful application instance shutdown, including session and cache replication and the actual VM de-provisioning. For this application the de-provisioining averages at 17 minutes with a 10% variation.

In practice, this provisioning time introduces the need to set thresholds at a lower level then actual dictated by the capacity requirements, in order to allow time for the new instance to become active before the capacity is exceeded by the user load. This approach has many disadvantages. One of the most obvious is that it introduces the risk of false positives that lead to poor resource utilization by unnecessary provisioning of additional resources. When using simple thresholds along with a small granularity of the application scalability the effects are amplified. The following section describes the proposed predictive mechanism that augments the generic threshold based implementation, by recognizing repetitive pattern in the application usage.

3. Related Work

The research presented in this paper can be included in the broader scope of autonomous computing and more specifically in the self-optimization systems sub category as classified by IBM in the report The Vision of Autonomic Computing [17, 18]. Several efforts have been made in this direction in terms of predictive resource provisioning.

PRedictive Elasticre Source Scaling (PRESS) [18] is using signal processing techniques to identify repeating patterns if possible. As an alternative for the situation in which patterns cannot be discovered, PRESS uses a statistical state-driven approach to capture short-term patterns in resource demand by using a discrete-time Markov chain. The resource prediction models are updated when a change in the resource consumption patterns is detected. The Signature-driven resource demand prediction in PRESS is based on a Fast Fourier Transformation (FFT) to calculate the dominant frequencies of resource-usage variation in the observed load pattern. Although PRESS proves an efficient approach when using a single metric for determining the patterns, it fails short in scenarios where the use behavior is different during the day, for example scenarios where users are performing mainly searches in the morning hours and heavy reporting during the evening hours. The solution presented in this paper is superior in the fact that it can use multiple metrics at the same time to determine a more meaningful user behavior patterns such as CPU Usage, Disk I/O and so on.

Resource overbooking based on application profiling [19] is another approach available in the literature. While is a valid approach for grids, where there is an implicit control on the operating system, so that profiling can be added to the kernel, is not feasible in cloud environments. Even in private clouds the workloads are usually controlled by the client and intrusive profiling would not be possible. Another profiling based approach is presented in [20] that focus on determining the resource requirements in a sandbox environment by employing benchmark tests. The approach can determine some valid patterns, however is static in nature and won't evolve with the application once this changes or new features are added. At the same time this solution would require an initial benchmark in simulated conditions.

4. Predictive Capacity Provisioning Mechanism

The capacity supported by one instance is an intrinsic property of the given application. For example the amount of concurrent users for one instance can be determined by load testing. The results obtained with load tests serve as baselines for determining the initial thresholds for the deployment and configuring the dynamic scalability of the applications.

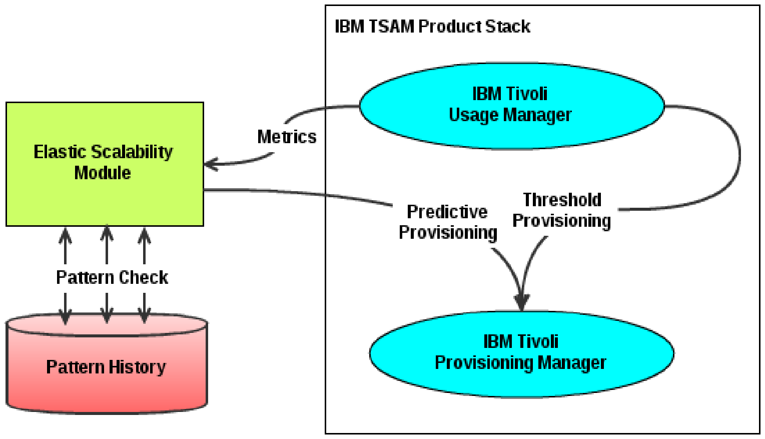


Fig. 3. Architecture of ESM for IBM Cloud Burst 2.1

To augment this we propose a daily and weekly usage pattern model that would predict the future required capacity and would act before the threshold is reached, allowing setting higher levels for thresholds and so avoid false positive triggers.

The implementation described in this paper is targeted at IBM CloudBurst 2.1 on System x, but the concept does apply to any API based cloud platform, either public or private. To encapsulate the predictive algorithm we introduce the Elastic Scalability Module that augments the CloudBurst 2.1 threshold mechanism. The architecture of the Elastic Scalability Module proposed for IBM CloudBurst 2.1 is illustrated in Fig. 3. The ESM works by collecting real time usage metrics provided by IBM TUM. The ESM has two operational phases: the learning phase and the driving phase. In the learning phase, the ESM stores in the metrics provided by IBM TUM per day for each day of the week in a relational database together with the threshold trigger events generated by TUM. This is done several times until a pattern is established. The ESM determines a pattern for a day of the week by comparing the metric variation against an average of previously recorded metrics for the same day of the week. If the difference for each hourly average is smaller than a preconfigured threshold, the pattern is considered valid. The algorithm for the learning phase is presented below:

```
function validatePattern(hourOfDay, currentLoad) is
  saveInDatabase(hourOfDay, currentLoad)
  if hourOfDay=0 then
    weekPatternFound = true
    for i=0 to 23 do
      avg = computeAvgDayWeekHour(i, 10)
      if abs(avg/currentLoad) > patternThreshold then
          weekPatternFound = false
      endif
  endif
  if weekPatternFound = true then
      swithToDrivingPhase()
      setHigherThresholds()
  endif
end function
```

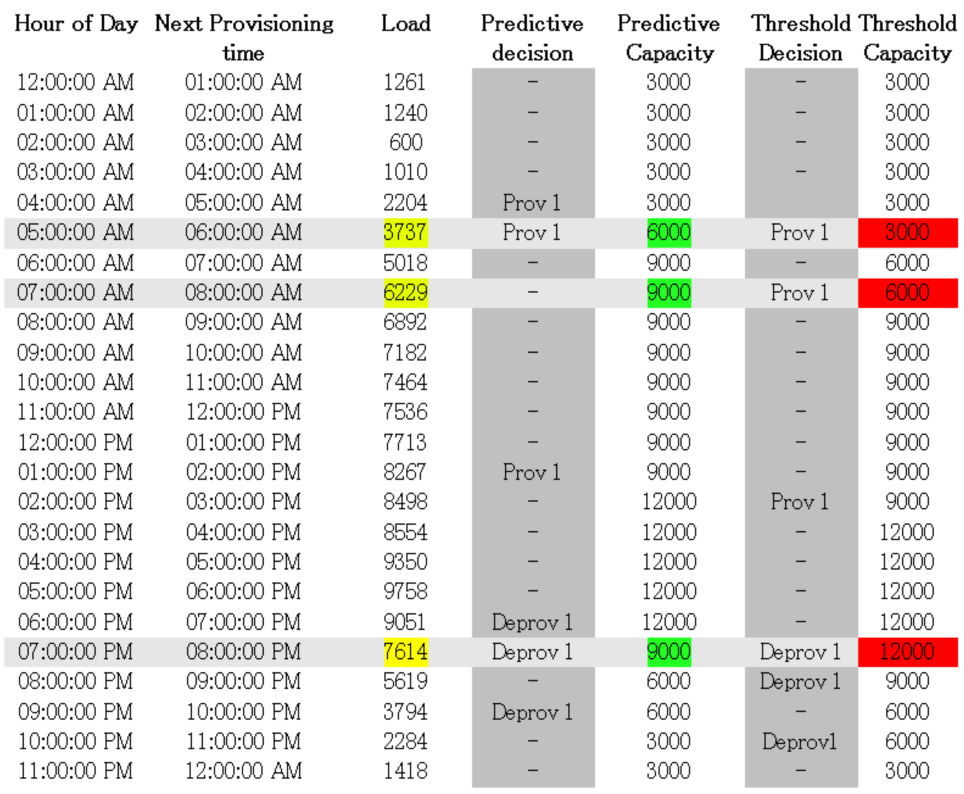
Once the pattern is validated, the ESM begins to function in the driving phase, by sending predictive provisioning instructions to IBM Tivoli Provisioning Manager (TPM), eventually replacing the trigger based behavior of TUM. Also at this point, the thresholds are set to higher values, to avoid false positives. Even in the driving phase the current usage load is validated against the pattern stored in the database and if the pattern validation fails, the ESM will transition back to learning phase and will set relaxed values for the thresholds. This behavior helps in exceptional scenarios like special days of the year or one off situations in which the system load is unexpected. In this case, the system would fall back on threshold based provisioning and de-provisioning.

```
function predictiveSetCapacity(dayOfWeek, hourOfDay) is
  loadDayPattern(dayOfWeek)
  nextProvisioningTime = hourOfDay + provisioningDelay
  estimatedCapacity = getPatternCapacity(nextProvisioningTime)
  instances = (currentCapacity - estimatedCapacity)/instanceCapacity
  if (instances > safeCapacityThresholdProv) then
    provision(instances)
  endif
  if (instances < safeCapacityThresholdDeprov) then
    deprovision(abs(instances))
  endif
end function
```

The predictive provisioning algorithm is invoked each hour and based on the provisioning delay it computes the next provisioning time. The next provisioning time represents the time when a new machine would be active, if the decision to provision it is taken now. Based on the daily pattern, the estimated capacity is retrieved. The difference between the current provisioned capacity and the estimated capacity divided by the instance capacity represents the number of instances that need to be provisioned or de-provisioned, depending if the difference is positive or negative. The result is compared with a safe capacity threshold and the decision is made to provision or de-provision additional instances. The ESM considers the average provisioning time in the *provisioningDelay* constant. The overall goal followed by ESM algorithm is to keep the capacity as close as possible with the average user load using historical usage patterns.

Table 1

Algorithm execution for Sunday pattern vs. threshold approach time



Hour of Day	Next Provisioning time	Load	Predictive decision	Predictive Capacity	Threshold Decision	Threshold Capacity
12:00:00 AM	01:00:00 AM	1261	-	3000	-	3000
01:00:00 AM	02:00:00 AM	1240	-	3000	-	3000
02:00:00 AM	03:00:00 AM	600	-	3000	-	3000
03:00:00 AM	04:00:00 AM	1010	-	3000	-	3000
04:00:00 AM	05:00:00 AM	2204	Prov 1	3000		3000
05:00:00 AM	06:00:00 AM	3737	Prov 1	6000	Prov 1	3000
06:00:00 AM	07:00:00 AM	5018	-	9000	-	6000
07:00:00 AM	08:00:00 AM	6229	-	9000	Prov 1	6000
08:00:00 AM	09:00:00 AM	6892	-	9000	-	9000
09:00:00 AM	10:00:00 AM	7182	-	9000	-	9000
10:00:00 AM	11:00:00 AM	7464	-	9000	-	9000
11:00:00 AM	12:00:00 PM	7536	-	9000	-	9000
12:00:00 PM	01:00:00 PM	7713	-	9000	-	9000
01:00:00 PM	02:00:00 PM	8267	Prov 1	9000	-	9000
02:00:00 PM	03:00:00 PM	8498	-	12000	Prov 1	9000
03:00:00 PM	04:00:00 PM	8554	-	12000	-	12000
04:00:00 PM	05:00:00 PM	9350	-	12000	-	12000
05:00:00 PM	06:00:00 PM	9758	-	12000	-	12000
06:00:00 PM	07:00:00 PM	9051	Deprov 1	12000	-	12000
07:00:00 PM	08:00:00 PM	7614	Deprov 1	9000	Deprov 1	12000
08:00:00 PM	09:00:00 PM	5619	-	6000	Deprov 1	9000
09:00:00 PM	10:00:00 PM	3794	Deprov 1	6000	-	6000
10:00:00 PM	11:00:00 PM	2284	-	3000	Deprov1	6000
11:00:00 PM	12:00:00 AM	1418	-	3000	-	3000

5. Experimental Results

To validate the functional characteristics of the ESM module implemented, a simulation environment has been created using IBM CloudBurst 2.1 system. The simulation environment is built using DayTrader benchmark application running on a virtual image based on CentOS 5.5 operating system. DayTrader is a benchmark application designed to simulate an online stock trading system. The application was originally developed by IBM for WebSphere and was known as the Trade Performance Benchmark Sample. In 2005, IBM donated the DayTrader application to the Apache Geronimo community. The user base was simulated using Apache JMeter load testing tool, by feeding the real life usage data presented in Section 2 of this paper. Each instance of the application was configured to have a limited capacity of 3000 concurrent users. The load test was performed on a day pattern considering three different patterns identified in practice. The pattern analyzed in details is the Sunday specific usage pattern. Table 1 presents the prediction based on the pattern and the actual capacity compared to the results obtained with a simple threshold algorithm.

The threshold algorithm works with simple 80% hardened thresholds when in driving phase and with relaxed predefined incremental thresholds in learning phase. In the above example, the threshold was configured to provision an additional instance when the load reaches 80% of the capacity. Similarly when the load drops under 80% of the system capacity one instance will be de-provisioned. One important thing to notice in the experimental results is that with this threshold the capacity is exceeded twice, causing loss of service for small time intervals. Also, the overall resource utilization is better when using predictive model based on usage patterns.

Fig. 4 illustrates the provisioned capacity against the load of the system for the same pattern, showing a better allocation for the predictive provisioning compared to the threshold approach. The vertical lines represent the point in time when the decision to provision (+1) or to de-provision (-1) has been made. The capacity increase will be visible only after the provisioning time. For simplicity the provisioning time has been rounded at 1 hour and the de-provisioning time at 15 minutes.

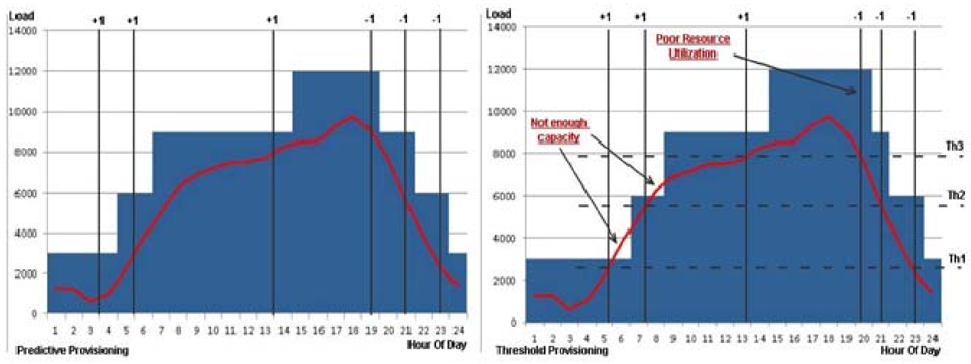


Fig. 4. Predictive and Threshold models for Sunday pattern

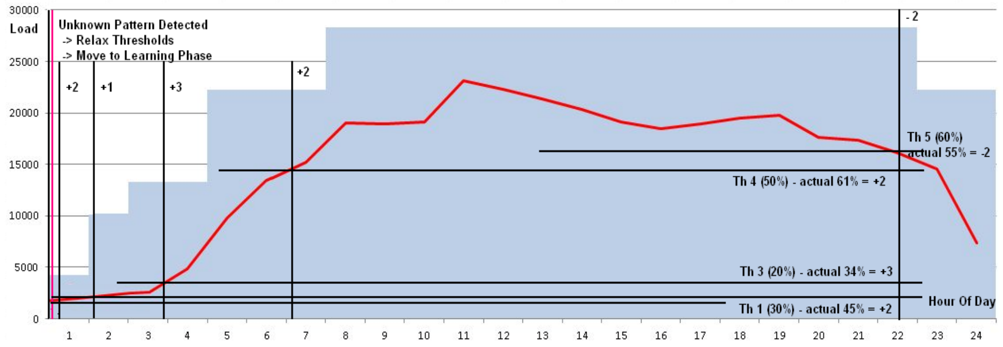


Fig. 5. Black Friday Scenario

The dotted horizontal lines represent the three thresholds configured as relative 80% ratio between capacity and load.

To validate the prototype functionality in exceptional situations, or in other words in scenarios where the user load does not match the normal usage patterns, the data recorded during Black Friday was feed to the load test. The usage during that day is illustrated in Fig. 5 above. The system behavior is to move from driving phase to learning phase within the first hour, as the load was not matching the Friday pattern. At that time the thresholds, which were set at 80% of capacity during driving phase, were relaxed to 30 % for the first threshold, 25% for the next threshold, 20% for the third one and 50% for the fourth one. The reason for this threshold setting is to keep up with the steep increase of site visitors during the morning hours considering also the provisioning time required to increase capacity. We could see that in these conditions the system performed well in regards to assuring there is enough capacity provisioned. However, the resource utilization was poor, especially on the descending side after the peak at noon hours.

A study has been done to determine the actual gain in resource utilization during one normal week, or in other words, a week that maps closely to the repetitive behavior observed. When the ESM module was operating normally in driving phase for a complete week, the resource utilization in regards to provisioned capacity and the number of users on the site was better with 23 % compared with the relaxed threshold approach on the same input data for the load test. This was achieved with an increase of 3 % in provisioning and de-provisioning operations. Although the increase of 3% in the provisioning and de-provisioning operations is important, as it does have an associated cost which cannot be neglected, the increase in resource utilization is significant.

6. Conclusions

By analyzing the usage of a real life application in the travel booking industry a set of weekly usage patterns have been discovered. Arguably each web application exposed to a large user base would show a weekly usage pattern that is repetitive. Starting from this observation this paper argues that these patterns can be used as a source for predictive scaling of the application using cloud based technologies. The algorithm proposed does not replace the threshold based mechanism completely, but rather it augments it by reusing historical information about repetitive usage patterns.

The experimental results demonstrate that in normal conditions, where a repetitive pattern can be identified, a predictive provisioning mechanism is superior to a simple threshold based mechanism, in both resource allocation and resource utilization.

However, one important limitation is the ability to react to unexpected peak loads on the system, where the threshold mechanism is better. The ideal solution for this kind of applications is to use a combined solution, in which the predictive mechanism controls the capacity in normal operation mode and a threshold mechanism can override in case of unexpected peak loads on the system. One important advantage of the predictive mechanism is that it allows setting higher level thresholds, as these will only be hit when a significant deviation from the pattern is encountered, thus avoiding false positives triggered during threshold evaluation.

This combined approach is better suited for applications where the instance granularity is high and provisioning overhead is very small, as it would allow near optimal resource utilization. In essence the predictive mechanism described represents a proactive approach for scalability, while the threshold based mechanism represents a reactive approach. When the provisioning time becomes comparable to the load variation in time in regards to instance capacity this proactive approach is mandatory.

Future work is focused on extending the ESM module to consider multiple factors as inputs for generating and storing usage patterns. It has been observed that not only the site visits are following a pattern, but also the operations of the users. These enhancements would help to better define the required capacity based on user operations and the resource costs associated and so would be a more accurate driver for dynamic scalability of the system.

REFERENCES

[1] *H. Lim*, "Automated control in cloud computing: challenges and opportunities.", Proceedings of the 1st workshop on Automated control for datacenters and clouds. ACM, 2009.

[2] *D. Azbayar, J. Chase and S. Babu*, "Reflective control for an elastic cloud application: an automated experiment workbench.", Proceedings of the 2009 conference on Hot topics in cloud computing (HotCloud'09). 2009.

[3] *A. Lemos, R. Moleiro, P. Ottaviano, F. Rada, M. Widomski, B. Braswell*, "IBM CloudBurst on System x", IBM International Technical Support Organization, IBM RedBooks April 2012

[4] *** Amazon CloudWatch, http://aws.amazon.com/documentation/cloudwatch/

[5] *** Best Practices in Evaluating Elastic Load Balancing, Amazon E2C, http://aws.amazon.com/articles/1636185810492479

[6] *** Scalr Web Site, http://sclar.com, November 2012.

[7] *E. Caron, L. Rodero-Merino, F. Desprez , A. Muresan*, "Auto-Scaling, Load Balancing and Monitoring in Commercial and Open-Source Clouds", UMR CNRS - ENS de Lyon - INRIA, Research Report, January 2012

[8] *P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem.* "Adaptive control of virtualized resources in utility computing environments", in Proc. of EuroSys, 2007

[9] *A. Olivieri, A. Pintus, A. Mallozzi, C. Santucci*, "IBM Tivoli Provisioning Manager V7.1.1: Deployment and IBM Service Management Integration Guide", IBM RedBooks, IBM Form Number SG24-7773-00, 2009

[10] *** VMware VCenter,

http://www.vmware.com/products/vcenter-server/overview.html , accessed December 2012

[11] *** Apache Geronimo Sample Applications, DayTrader,

https://cwiki.apache.org/GMOxDOC30/daytrader-a-more-complex-application.html, accessed October 2012

[12] *** Google Analytics,http://www.google.com/analytics, accessed December 2012

[13] *** Drupal Framework, http://drupal.org/about, accessed October 2012

[14] *D. Paul, H. M. Deitel*, "AJAX, rich internet applications, and web development for programmers." Prentice Hall PTR, 2008, pp25-50

[15] *T. Bowler, B. Wojciech*, "Symfony 1.3 web application development", Packt Pub Limited, 2009

[16] *J.O. Kephart*, "Research challenges of autonomic computing." Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on. IEEE, 2005.

[17] *J.O. Kephart, D. M. Chess.* "The vision of autonomic computing." Computer 36.1 (2003): 41-50

[18] *G. Zhenhuan, X. Gu, J. Wilkes*, "Press: Predictive elastic resource scaling for cloud systems." Network and Service Management (CNSM), 2010 International Conference on. IEEE, 2010.

[19] *B. Urgaonkar*, *P. Shenoy*, *T. Roscoe*, "Resource overbooking and application profiling in shared hosting platforms." ACM SIGOPS Operating Systems Review 36.SI (2002): 239-254.

[20] *T. Wood*, "Profiling and modeling resource usage of virtualized applications." Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware. Springer-Verlag New York, Inc., 2008.