Transforming Reactive Auto-scaling into Proactive Auto-scaling

Laura R. Moore, Kathryn Bean and Tariq Ellahi
SAP Next Business and Technology, SAP (UK) Ltd, Belfast, UK.
laura.moore@sap.com

Abstract

Elasticity is a key characteristic of cloud platforms enabling resource to be acquired on-demand in response to time-varying workloads. We introduce a new elasticity management framework that takes as input commonly used reactive rule-based scaling strategies but offers in return proactive auto-scaling. The elasticity framework combines reactive and predictive auto-scaling techniques, and we discuss the specification and performance of these individual components. We present a case study, based on real datasets, to demonstrate that our framework is capable of making appropriate auto-scaling decisions that can improve resource utilization compared to that obtained from a purely reactive approach.

Categories and Subject Descriptors C.4 [*Performance of Systems*]: Modeling techniques; G.3 [*Probability and statistics*]: Time series analysis

General Terms Algorithms, Performance, Reliability

Keywords elasticity, predictive, auto-scaling, cloud computing, platform-as-a-service

1. Introduction

Elasticity is an important feature of cloud platforms, benefitting both cloud providers and end-users. For the cloud provider, it can reduce server under-utilization whilst offering some guarantee of Quality of Service (QoS) for end-users. Elastic capabilities offered by cloud providers are typically cast as rule-based algorithms that define scaling conditions based on a target metric reaching some threshold. This paper presents details of the implementation of a real-time performance monitoring framework, Platform Insights, which is responsible for making cloud platform auto-scaling decisions. The proactive Platform Insights elasticity controller has been designed from the outset to extend the functionality offered by reactive auto-scaling rules by generating predictive models based on them. The particular contribution of the paper is the design of an elasticity controller that:

- Spawns predictive auto-scaling models based on static scale out rules
- Learns on-line and can be used right away
- Employs models operating on multiple time frames, to introduce contextualization to auto-scaling decisions

The rest of the paper is organized as follows. Section 2 presents a summary of related work in this area. Section 3 gives details on the design of Platform Insights. Section 4 discusses the performance of the models internal to Platform Insights. Section 5 describes a case study to validate the approach and evaluate the resource provisioning decisions. Concluding remarks are given in section 6.

CloudDP '13, April 14, 2013, Prague, Czech Republic.

2. Related Work

Rule-based methods for auto-scaling are offered by several cloud providers such as Amazon [1] or third party tools such as RightScale [2] or AzureWatch [3]. Moving beyond reactive elasticity, the implementation of stable and accurate proactive auto-scaling methods remains an open challenge. Typically research into predictive elasticity techniques is based on machine learning and control theory [4].

The elasticity controller implemented in Platform Insights is a hybrid approach combining reactive and predictive auto-scaling techniques. Other hybrid controllers have been proposed. In [5] the authors conclude that predictive provisioning works well on typical days and that an additional reactive controller can be used to handle flash crowds. In [6, 7] it is proposed to use a reactive controller to control scale out and a predictive controller for scale in. In [7] the authors find that SLA violations are reduced by a factor of 2 to 10 compared to a purely reactive controller. In contrast, our reactive and predictive controllers can both trigger scale in and out actions.

An autonomic controller based on Kriging surrogate models was described in [8]. The models are built off-line and subsequently updated and refined on-line. In contrast, our hybrid controller can be used without off-line training. The reactive controller can make auto-scaling decisions from the start; the predictive controller takes advantage of online

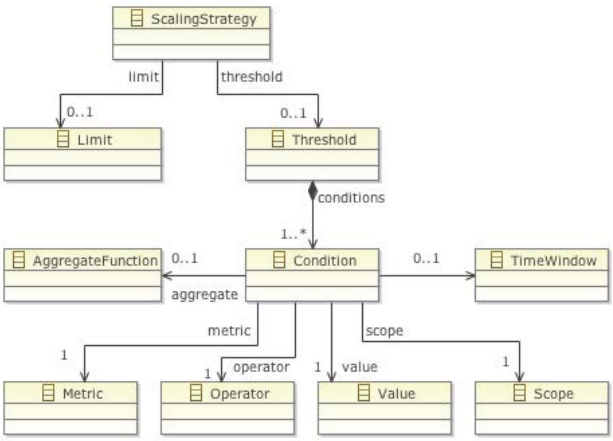

Figure 1. Scaling rule R_1 meta-model.

incremental learning techniques and over time becomes able to take over the auto-scaling decisions.

3. Design of Platform Insights

Application architects configure reactive threshold-based scaling rules based on exposed service metrics, and, upon submission to the system, Platform Insights configures and initiates a reactive controller to handle the elasticity management. If the architect configures a scale out rule to be used as the basis of a QoS condition that the system should proactively try to maintain, then Platform Insights also starts a predictive controller. In this case the architect also submits a second rule, the details of which are given below.

The reactive and predictive controllers operate in a coordinated manner; a discussion of this is presented in [9]. This paper focuses on the specification and performance of the models used in Platform Insights. Section 3.1 now describes the format of the two rules that are configurable by application architects. Following on, Sections 3.2 and 3.3 discuss the reactive and predictive controllers respectively. Model performance is discussed afterwards in Section 4.

3.1 Scaling Rules

Figure 1 shows a meta-model schematic for the scaling rule R_1. Rule R_1 is used by the system to trigger scale in or out whenever the min/max/sum ('Scope') in the tier of the average/median ('AggregateFunction') value of some 'Metric'(Q) on each instance over some 'TimeWindow'(T_1) is greater or less than ('Operator') some specified threshold 'Value'. Rule R_1 also specifies a 'Limit'on the minimum (for scale in) or maximum (for scale out) number of instances within the tier.

If the rule is a scale out rule to be used as the basis for proactive auto-scaling, then a second accompanying rule (R_2) is also specified. Rule R_2 allows the specification of a metric representing workload (W), such as the number of requests; an aggregation function (currently only sum is supported); a time window (T_2) over which to apply the aggregation and to use as the one-step ahead prediction interval; and a confidence level to compute confidence bounds on the time series predictions.

3.2 Reactive Controller

At the core of our Platform Management Framework lies a management bus, which is responsible for enabling communication among various components. The management bus is implemented using the standard Advanced Message Queuing Protocol (AMQP) [10]. In the current prototype, we have used Apache Qpid [11], which implements AMQP and provides a message broker. Lightweight probes on the various server instances publish monitoring data to the management bus. When a scaling rule (R_1) is submitted, the reactive controller subscribes for any relevant monitoring data required to evaluate the rule. A complex event processing engine is used to process the data. We use Esper [12] as it is lightweight, can be easily embedded in a Java application, and allows new queries to be registered dynamically after the engine has started so that scaling rules can be configured and set at any time. When the scaling condition is met, an alert is triggered and an auto-scaling request is sent to the Tier Decision Manager for assessment.

3.3 Predictive Controller

The predictive controller comprises 3 models: a time series forecaster and 2 incrementally updateable Naive Bayes models. All monitoring data is aggregated on a per-tier basis, as defined by R_1 and R_2, prior to use in the predictive models. Such aggregation is acceptable assuming load is well balanced across the tier. As platform components are assumed to have load balancers this is reasonable. Two types of aggregations are performed: 1) the two metrics representing workload and QoS (W and Q) are both aggregated according to R_1 to give values W_1 and Q_1, and 2) the workload metric (W) is aggregated according to R_2 to give value W_2. Esper is used to perform these aggregations with the resulting values published over the management bus and made available for the predictive models. The predictive models are implemented using the Weka machine learning library [13].

Model 1, based on a time series forecast, takes the stream of W_2 values as input. When a new value is received the model is updated and the next value is forecast. The Weka time series analysis module transforms the data by removing the temporal ordering of individual input examples to create lagged variables. After the data is transformed a support vector machine algorithm performs the regression.

Our algorithm monitors the input data stream at every step t and if the value of $W_2^t - W_2^{(t-1)}$ lies outside 4 standard deviations of the mean then it assumes a severe workload burst occurred during the previous time interval. If a burst is detected then the model will base its forecast on the 2-step ahead prediction from timestep $t-1$ rather than the 1-step ahead prediction at timestep t. The forecast value is then

compared to the 4-hour moving average value to classify the general workload trend as increasing (>10% difference), decreasing (<-10% difference), or steady. Typically enterprise workloads exhibit daily or weekly cycles [14] so changes in demand should be observable over a 4-hour period. The confidence interval of the returned forecast is scaled (to account for differing time intervals) and used as input to Model 2 (see below) to estimate bounds on the number of server instances expected in the next T_2 time interval. These bounds are used to validate auto-scaling requests output from Model 3. In summary, the output from Model 1 is the classification of the workload trend along with bounds on the number of servers expected in the next T_2 time interval.

Model 2 takes as input W_1, Q_1 and the current number of server instances (N) running in the tier. It uses Q_1 as the basis for a binary classification into one of two classes: meeting or violating QoS. We set the threshold on Q_1 that differentiates the two classes to be 95% of the target scale out value specified in rule R_1. Setting this threshold to 5% below the actual QoS target value reduces the risk of under-provisioning, an approach also adopted by others [15]. Model 2 learns the relation between the current average workload per server and the QoS value, and hence can be used to run predictions of the number of instances needed within the tier to ensure that the probability of QoS violation remains below some value.

Model 3 is similar to Model 2, except that: a) the trend output from Model 1 is used as an additional input, b) a time delay is introduced between the workload and the binary QoS classification, and c) the input value of W_1/N is adjusted to account for scaling actions that took place during this time delay. Typically a time delay of 30 minutes is used, so that the workload is classified according to the QoS value 30 minutes later. The output from Model 3 is an estimate of the number of servers that should be operational in order to meet QoS requirements over the next 30 minutes. The difference between this value and the number of currently running servers yields the auto-scaling request. A 30 minute time window allows both time to make the decision with confidence and time to provision additional servers.

The controller workflow is as follows. New W_1 values are placed in a queue to await future QoS classification. At that time Model 3 predicts the future QoS classification of W_1, using as additional input the current value of N and the current workload trend (from Model 1). If the probability of QoS violation is too high (scale out) or too low (scale in), a search on N is performed until this probability falls below 5%. The search on N is done from the current value in the relevant direction (up/down according to scale out/in) until the probability condition is met, or until the model starts to lose confidence due to lack of training data, or until the limit on the number of instances is reached. If this optimal value of N differs to the current N then an auto-scaling request is sent to the Tier Decision Manager for assessment.

Because Models 2 and 3 are both incrementally updateable and learn on-line with each new data value, they can be susceptible to outliers. If a new data point lies outside 4 standard deviations from the mean value then it is assumed to be an outlier and is not used to update the model nor make an auto-scaling decision; it is however used to update the statistics so that trend changes will not be ignored for too long. We have found this to be an acceptable policy in our research to date: if the predictive controller is uncertain of what to do then it does nothing, which is OK because the reactive controller is always running as a stand-by.

For robust and effective elasticity control, the stability of the controller is an important design aspect. One way in which this can be achieved is through the use of effective feedback control, such as proportional thresholding [16]. Our approach is to implement a threshold policy with hysteresis. More specifically, the search for a better value of N only takes place whenever either the probability of QoS violation increases above 5.5% or drops below 1%. For scale in decisions we implement a conservative policy requiring the probability of QoS violation to remain less than 1%. For scale out decisions the value of N such that the probability of QoS violation is less than 4.5% is selected. As future work we plan to investigate fuzzifying these thresholds.

The Tier Decision Manager receives all auto-scaling requests from both the reactive and the predictive controllers. It is responsible for coordinating the requests and assessing them against the current number of running instances in the tier together with any outstanding requests that are in still in the process of being executed by the platform. Successful requests are communicated to the platform for execution.

4. Prediction Model Performance

This section focuses on the performance of the individual models employed by the predictive controller using the ClarkNet August trace [17] and the bursty FIFA 1998 World Cup Access data trace [17, 18]. Figure 2 shows the time series of the number of requests per hour in the ClarkNet and the FIFA World Cup data traces, together with the predictions from Model 1. The figure shows that Model 1 is able to make reasonably accurate predictions for these traces. It does not predict the bursts in the FIFA trace but, by detecting bursts as outliers and using a 2-step or 3-step ahead prediction, nor are the forecasts unduly influenced by the burst. We find that 80% of the predictions, even on the bursty FIFA log traces, fall within 17% of the actual value.

For both the ClarkNet and FIFA World Cup traces, the Model 1 workload trend classification performance is given in Table 1. Increasing and decreasing trends are correctly classified more than 80% of the time for the ClarkNet trace. The performance is not so good on the FIFA trace with correct classifications for decreasing and increasing trends at 62.4% and 70.5% respectively. Most of the misclassifications result in trends being labelled as steady. The over-

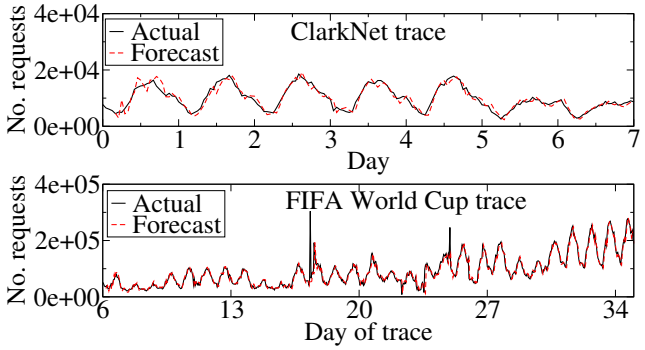

Figure 2. Hourly aggregated time series for two real data sets together with predictions from Model 1.

Table 1. Classifications of workload trends.

	Predicted Classification:		
	Decreasing	Steady	Increasing
ClarkNet Decreasing	80.5%	17.3%	2.2%
ClarkNet Steady	22.2%	59.8%	18.0%
ClarkNet Increasing	0.7%	17.7%	81.6%
FIFA Decreasing	62.4%	35.1%	2.5%
FIFA Steady	15.6%	66.8%	17.5%
FIFA Increasing	4.1%	25.3%	70.5%

laps between increasing/steady and steady/decreasing are not surprising since (in the current implementation) hard thresholds at $\pm 10\%$ are used.

This trend information is used by Model 3 to introduce contextualization in making auto-scaling decisions. For example, if CPU utilization is 70% it may be more likely to increase to 80% if the trend is generally increasing than if the trend is decreasing. Model 3 performance has been assessed within the context of an auto-scaling simulation experiment (details in Section 5 below). Figure 3 shows the distributions, at the end of the trace, of the probability that the QoS will be violated within the next 30 minutes for a range of workload values according to the workload trend. The Model 2 probability distribution of QoS violation is also shown for comparison. There is a difference between the probability distributions, and, as expected, if the trend is increasing then the 5% probability of violation occurs at a lower workload value than if it is decreasing.

Figure 4 plots the number of requests per unit time interval (2 minutes in this case) yielding a 5% probability of QoS violation within 30 minutes for the duration of the simulation. Again, Model 2 data is shown for comparison. The values vary at the start but then settle down and are reasonably steady for the last half of the simulation. We have conducted numerous experiments and checked the variation in these distributions. The distributions clearly depend on the workload trend classification and, as explained above, there can

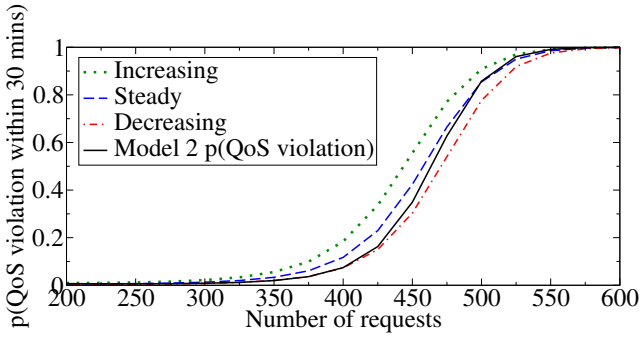

Figure 3. Model 3 probabilities of QoS violations occurring within 30 minutes by workload trend at the end of the trace.

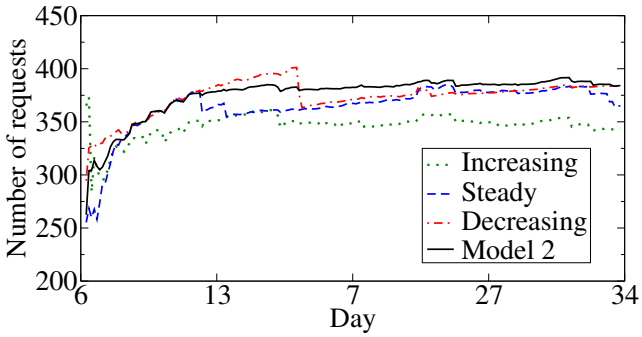

Figure 4. Number of requests per 2 minute interval giving a 5% probability of QoS violation within the next 30 minutes for different workload trends throughout the simulation.

be overlap between increasing/steady and steady/decreasing. This is frequently reflected by the steady workload trend having more variability with time. However, we consistently find that for increasing trends the 5% probability of violation occurs at a lower workload value than for decreasing trends. The discrimination between these two trends is of particular interest because it is expected that auto-scaling decisions are more likely to be made during the time periods when workload is substantially increasing or decreasing.

This section has examined the performance of individual models employed by the predictive controller. The next section presents a case study illustrating how the reactive and predictive controllers operate together to scale the platform in and out as the workload demand changes.

5. Case Study

This section presents the results of an auto-scaling simulation experiment to demonstrate the merit of our hybrid elasticity controller. The FIFA 1998 Word Cup Access logs to simulate driving the SPECjEnterprise2010 benchmark and the focus is on the platform application server tier.

The log files were summarized to extract the number of requests arriving every 2 minutes. These values were scaled by a factor of two and used to simulate driving the SPECjEnterprise2010 benchmark. The benchmark was run with

different loads and the response times were observed to be in excess of the target time of 2 seconds when the CPU utilization went beyond 80%. Rules were set as follows. Scale out: if the minimum median value of CPU utilization over the past 40 minutes is $>$ 80% then increase the number of instances by 1; Scale in: if the maximum median value of CPU utilization over the past 60 minutes is $<$ 50% then decrease the number of instances by 1. The provisioning of a new instance is assumed to take 10 minutes, a reasonable estimate given that four major public cloud providers can allocate new instances with an average scaling latency below 10 minutes [19]. The QoS condition was extracted as: the minimum median CPU utilization over a window of 40 minutes must be less than 80%.

The simulations were run using only the reactive controller and using the hybrid approach; auto-scaling results are shown in Figure 5. The reactive controller running alone is initially able to scale resource. However, from day 24 onwards, as the number of requests continues to increase, the reactive controller is less well able to scale the system in. This is because the median CPU utilization does not drop to below 50% across the whole tier. As the number of server instances in the tier increases, a larger drop in workload is required before the scale in condition will be met, resulting in less efficient use of resource. The implication is that static threshold rules for scale in can be difficult to configure to achieve efficient resource utilization. This highlights the advantage of operating a proactive controller. In scale out scenarios, a reactive controller could make the decision earlier if a modified threshold was used, but appropriate threshold setting remains trial and error. In scale in scenarios, our elasticity controller determines the optimal number of servers at any given time and is therefore not constrained to wait for utilization to drop to some value across the whole tier.

Even at high workloads, our hybrid controller continues to be capable of dynamically scaling the system in accordance with the workload. The maximum number of allowed instances was set to 40. This maximum value was never requested, even during the pronounced bursty periods, thus demonstrating the stability of the controller; the only possible exception is day 23 of the trace.

A magnification of Figure 5 at day 23 is displayed in Figure 6, revealing an unusual workload trend on this day. Most of the scale in decisions are generated by the reactive controller (decision points marked by blue circles). Therefore what looks like unstable scaling behaviour in Figure 5 is primarily the desired response according to the configured scaling rules. The shortest timespan between scaling decisions is 60 minutes, which may be acceptable given many providers charge on a per-hour basis. In the future cost-awareness will be incorporated into the scaling decisions to enable the determination of the minimum cost-effective time period over which resource may be released and re-provisioned (or vice-versa). Previous research shows this could help improve resource utilization and efficiency [20]. In this test case, the hybrid controller yielded a QoS violation rate of 0.4%, of which a third of these violations occurred during day 23.

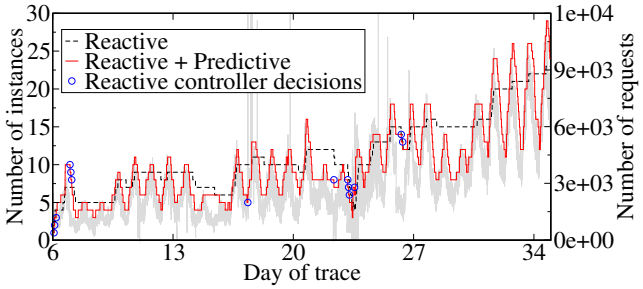

Figure 5. Auto-scaling simulation using reactive controller only and the hybrid controller. The number of requests per two minute interval is shown by the grey line (right axis).

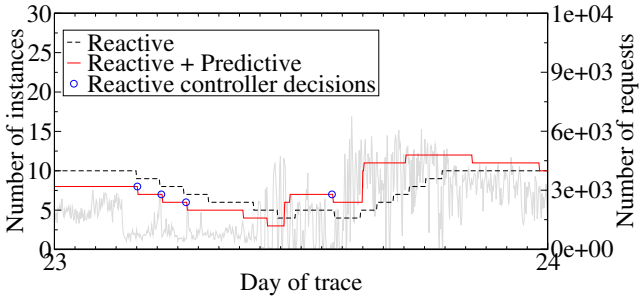

Figure 6. Zoom in on day 23 of auto-scaling simulation.

Figure 7 presents the cumulative distribution of CPU utilization across all server instances throughout the duration of the simulation. It further demonstrates the improved resource utilization achieved by the hybrid elasticity controller over the purely reactive one. Using the hybrid controller there are fewer instances overloaded with CPU utilization $>$80% and also fewer instances underloaded with CPU utilization $<$50% than using the reactive controller alone. The difference is even more stark if we consider that the reactive controller will result in 37% of instances with CPU utilization $<$60% whereas this is kept to under 13% by the hybrid controller.

6. Conclusion

In this paper we presented a new hybrid elasticity controller that extends the capabilities of commonly used reactive scaling strategies to offer proactive auto-scaling. The controller extracts the scale out condition and builds incrementally updateable predictive models to enable the system to proactively scale out before this condition is met. The predictive models are used to calculate the minimum number of server instances required during the next 30 minutes in order to have less than 5% chance of reaching the scale out (equivalently QoS) condition. This methodology allows the system to scale in as well as out.

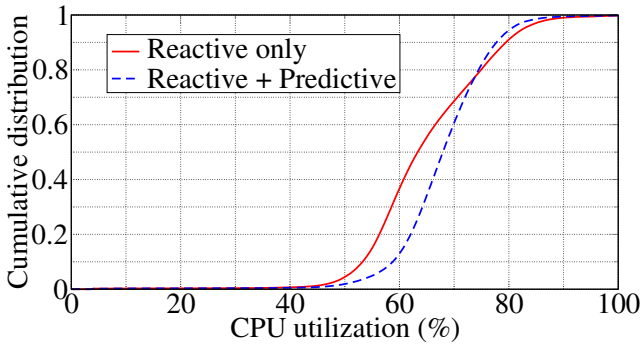

Figure 7. Cumulative distribution of CPU utilization on all server instances during the auto-scaling simulation.

The performance of the elasticity controller was presented both through discussions of individual model performance and through an auto-scaling experiment in which a real data set was used to simulate driving an enterprise benchmark. The predictive models operate over multiple time frames to introduce contextual awareness to the auto-scaling decisions, and whenever they lack certainty, auto-scaling decisions are left to the reactive controller. The case study demonstrated that, compared to a purely reactive controller, our controller is able to make better provisioning decisions for an application server tier, yielding few QoS violations and maintaining consistently high CPU utilization.

For future work we plan to integrate a controller designed specifically to handle flash crowds; handle multiple QoS objectives; test the controller with more workload profiles and on other tiers; integrate an algorithm to detect change in workload mix to enable faster learning of new relationships between the number of requests and the QoS metric; incorporate cost-awareness in the scaling decisions; and to develop a higher-level controller to correlate monitoring data and auto-scaling requests across tiers in order to provide a fully integrated elasticity management framework.

Acknowledgments

This research is funded by the CumuloNimbo project in the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement no. FP7-257993.

References

[1] Amazon Elastic Compute Cloud `http://aws.amazon.com/ec2/` Accessed 16th January 2013.

[2] RightScale `http://www.rightscale.com/` Accessed 16th January 2013.

[3] AzureWatch `http://www.paraleap.com/azurewatch` Accessed 16th January 2013.

[4] T. Lorido-Botrán, J. Miguel-Alonso and J.A. Lozano. Auto-scaling Techniques for Elastic Applications in Cloud Environments. *Technical Report EHU-KAT-IK-09-12, University of the Basque Country*, Sept. 2012.

[5] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal and T. Wood. Agile dynamic provisioning of multi-tier internet applications. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, **3** (1), 1-39, 2008.

[6] W. Iqbal, M.N. Dailey, D. Carrera and P. Janecek. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Generation Computer Systems*, **27** (6), 871-879, 2011.

[7] A. Ali-Eldin, J. Tordsson and E. Elmroth. An adaptive hybrid elasticity controller for cloud infrastructures *Network Operations and Management Symposium (NOMS)*, 204-212, 2012.

[8] G. Toffetti, A. Gambi, M. Pezzè and C. Pautasso. Engineering Autonomic Controllers for Virtualized Web Applications. In *ICWE'10 Proc. of the 10th international conference on Web Engineering*, 66-80, 2010.

[9] L.R. Moore, K. Bean and T. Ellahi. A coordinated reactive and predictive approach to cloud elasticity. Accepted for publication, 2013.

[10] Advanced Message Queuing Protocol (AMQP) `http://www.amqp.org/` Accessed 22nd January 2013.

[11] Apache Qpid `http://qpid.apache.org/` Accessed 22nd January 2013.

[12] EsperTech Event Stream Intelligence `esper.codehaus.org/index.html`, a product of EsperTech Inc. Accessed 16th January 2013.

[13] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann and I.H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, **11** (1), 10-18, 2009.

[14] D. Gmach, J. Rolia, L. Cherkasova and A. Kemper. Resource pool management: Reactive versus proactive or let's be friends. *Computer Networks*,**53** (17), 2905-2922, 2009.

[15] Z. Gong, X. Gu and J. Wilkes. Predictive elastic resource scaling for cloud systems. *2010 International Conference on Network and Service Management (CNSM)*, 9-16, 2010.

[16] H.C. Lim, S. Babu, J.S. Chase and S.S. Parekh. Automated Control in Cloud Computing: Challenges and Opportunities. In *Proc. of the First Workshop on Automated Control for Datacenters and Clouds*, 13-18, 2009.

[17] The Internet Traffic Archive `http://ita.ee.lbl.gov/html/traces.html` Accessed 16th January 2013.

[18] M. Arlitt and T Jin. Workload Characterization of the 1998 World Cup Web Site. *Technical Report HPL-1999-35R1, HP Laboratories*, 1999. The trace is available from [17].

[19] A. Li, X. Yang, S. Kandula and M. Zhang. CloudCmp: comparing public cloud providers. In *Proc. of the 10th ACM SIGCOMM conference on Internet measurement (IMC '10)*, 1-14, 2010.

[20] J.N. Silva, L. Viega and P. Ferreira. A2HA - automatic and adaptive host allocation in utility computing for bag-of-tasks. *Journal of Internet Services and Applications (JISA)*, **2** (2), 171-185, 2011.