PRESS: PRedictive Elastic ReSource Scaling for cloud systems

Zhenhuan Gong, Xiaohui Gu
Department of Computer Science
North Carolina State University
zgong@ncsu.edu, gu@csc.ncsu.edu

John Wilkes
Google
Mountain View, CA
johnwilkes@google.com

***Abstract*—Cloud systems require elastic resource allocation to minimize resource provisioning costs while meeting service level objectives (SLOs). In this paper, we present a novel *PRedictive Elastic reSource Scaling* (PRESS) scheme for cloud systems. PRESS unobtrusively extracts fine-grained dynamic patterns in application resource demands and adjust their resource allocations automatically. Our approach leverages light-weight signal processing and statistical learning algorithms to achieve online predictions of dynamic application resource requirements. We have implemented the PRESS system on Xen and tested it using RUBiS and an application load trace from Google. Our experiments show that we can achieve good resource prediction accuracy with less than 5% over-estimation error and near zero under-estimation error, and elastic resource scaling can both significantly reduce resource waste and SLO violations.**

I. Introduction

Cloud computing allows tenants to rent resources in a pay-as-you-go fashion. It offers the potential for a more cost-effective solution than in-house computing by obviating the need for tenants to maintain complex computing infrastructures themselves. To achieve this benefit, the right amount of computing resources need to be given to the applications running in the cloud. The amount of resources needed is rarely static, varying as a result of changes in overall workload, the workload mix, and internal application phases and changes.

Under-provisioning resources will cause service level objective (SLO) violations, which are often associated with significant financial penalties. Over-provisioning wastes resources that could be put to other uses. To avoid both problems, the amount of resources allocated to applications should be adjusted dynamically, which brings two main challenges: (1) deciding how much resource to allocate is non-trivial since application resource needs often change with time and characterizing runtime application behavior is difficult; (2) application resource needs must be predicted in advance so that the management system can adjust resource allocations ahead of the needs. Furthermore, resource-management systems should not require prior knowledge about applications, such as application behavior profiles, and running the resource-management system itself (including its prediction algorithms) should not be costly.

The goal of our research is to develop a light-weight elastic resource allocation scheme for use by a cloud service provider that addresses these challenges, while not requiring advance application profiling, model calibration, or deep understanding of the application. We call our approach *PRedictive Elastic reSource Scaling* (PRESS). Figure 1 shows its overall architecture.

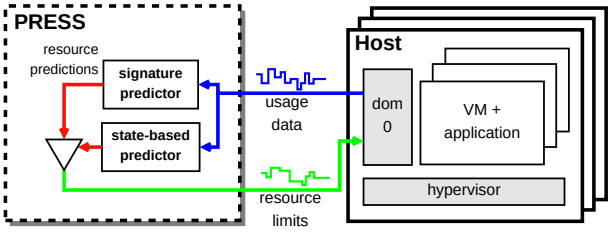


Fig. 1. Overall architecture of the PRESS system. The host system runs applications in virtual machines (VMs), and monitors their resource usage. This data is fed to PRESS, which makes predictions of future application resource needs, which are sent back to the host system, which scales the resource-limits on each VM accordingly.

PRESS strives to allocate just enough resources to applications to avoid SLO violations and minimize resource waste. It continuously tracks the dynamic resource requirements of applications in an unobtrusive way and predicts resource demands in the near futurem using two complementary techniques to do so. PRESS first employs signal processing techniques to identify repeating patterns called *signatures* that are used for its predicns. If no signature is discovered, PRESS employs a statistical state-driven approach to capture short-term patterns in resource demand, and uses a discrete-time Markov chain to predict that demand for the near future. The resource prediction models are repeatedly updated when resource consumption patterns change. PRESS gives higher priority to avoiding under-estimation than to avoiding over-estimation since the former is more likely to cause SLO violations.

This paper makes the following contributions:

- It describes PRESS, a light-weight online resource demand prediction scheme that can handle both cyclic and non-cyclic workloads.
- It compares the performance of PRESS against several other algorithms, by implementing a prototype on top of the Xen virtual machine (VM) platform, and using workloads derived from the RUBiS online auction benchmark [1] and a sample of real application load traces collected on a Google cluster [2].
- It demonstrates that PRESS does better than any of the alternate algorithms, while imposing significantly less overhead than the best of them.

Proc. 6th IEEE/IFIP International Conference on Network and Service Management (CNSM 2010), Oct. 2010 (Niagara Falls, Canada).

The metrics used for the analysis include over-and under-estimation errors, the rate of SLO violations, and the profitability of a cloud system provider operating under a number of different penalty functions.

PRESS helps applications meet their SLOs while using near-minimum resources. Most of the schemes we compared PRESS against induced more SLO violations (up to 80% of requests), or wasted resources, allocating up to 35% more than PRESS, or both. We found that PRESS achieved high prediction accuracy: less than 5% over- and near zero under-estimation errors. In contrast, most other prediction schemes had higher prediction errors on the same workloads, with up to 40% over- and 15% under-estimation errors.

Our prototype implementation shows that PRESS is feasible and imposes negligible overhead for a virtualized computing cluster.

The rest of the paper is organized as follows. Section II presents the design and algorithms of the PRESS system. Section III presents the experimental results. Section IV compares our work with related work. Finally, the paper concludes in Section V.

II. System Design

For simplicity of exposition we focus here on CPU usage, although PRESS is capable of managing memory, I/O, and network usage, too. PRESS manages an application's resource allocation by setting the CPU resource limit for the virtual machine the application is running in (we call this *scaling*). It bases that limit on a prediction derived from recent application behavior. The limit sets the upper bound for the application's CPU consumption; if the limit is higher than the actual demand of the application, the residual allocation is wasted; if it is lower, SLO violations are likely. Ideally, the limit should be set to exceed the actual demand by a small margin. To achieve this, we developed two complementary resource prediction techniques.

A. Signature-driven resource demand prediction

For workloads with repeating patterns, PRESS derives a *signature* for the pattern of historic resource usage, and uses that signature in its prediction. Such repeating resource usage patterns are often caused by repeating requests or iterative computations [3], [4].

To avoid making assumptions about the length of the repeating pattern, PRESS uses signal processing techniques to discover the signature, or to decide that one does not exist. It first employs a Fast Fourier Transform (FFT) to calculate the dominant frequencies of resource-usage variation in the observed load pattern. Starting from a resource-usage time series $L = \{l_1, ..., l_W\}$, the FFT determines coefficients that represent the amplitude of each frequency component. The dominant frequencies are those with the most signal power. If there are multiple dominating frequencies that have similar amplitude, PRESS picks the lowest dominating frequency f_d, thereby selecting the longest repeating pattern that was observed.

Since workload signatures may vary over time, this calculation is performed anew each time a prediction is made.

Given a dominating frequency f_d, PRESS derives a *pattern window size* of Z samples: $Z = (1/f_d) \times r$ where r denotes the sampling rate. It then splits the original time series $L = \{l_1, ..., l_W\}$ into $Q = \lfloor W/Z \rfloor$ pattern windows: $P_1 = \{l_1, ..., l_Z\}$, $P_2 = \{l_{Z+1}, ..., l_{2Z}\}$, ..., $P_Q = \{l_{(Q-1)Z+1}, ..., l_{i,QZ}\}$. To detect whether the time series contains repeating patterns, PRESS evaluates the similarity between all pairs of different pattern windows P_i and P_j. Two pattern windows are considered similar if their Pearson correlation[1] value is close to 1 (e.g., > 0.85) and the ratio of their mean values is close to 1 (e.g., within 0.05). If all pattern windows are similar, PRESS treats the resource time series as having repeating behavior, and uses the average value of the samples in each position of the pattern windows to make its prediction. Otherwise, PRESS falls back to an alternate, state-based scheme to predict the resource demands.

Note that the length of the time series window L is not necessarily a multiple of the signature length Z. This means that even if a signature pattern has been found, PRESS still needs to determine where the system is within the signature window to predict the next usage value. This is accomplished as follows. After PRESS finds a signature S with length Z, it retrieves the last Z measurement samples to form a time series S'. Since S and S' are similar but time-shifted, PRESS applies the dynamic time warping (DTW) algorithm [5] to determine the alignment. DTW can match two similar but warped or shifted time series. The result is the minimum distance mapping from S' to S. This allows PRESS to derive a point-to-point mapping between S' and S, which allows it to find the position of the current value (i.e., the last point on S') on the signature S.

B. State-driven resource demand prediction

For applications without repeating patterns, PRESS uses a discrete-time Markov chain with a finite number of states to build a short-term prediction of future metric values (see Figure 2). Consider a resource metric x such as memory usage or CPU usage. PRESS discretizes its values into M equal-width bins where each bin represents a distinct state. (We used $M = 40$ bins in our experiments.) To build a Markov chain model for metric x, PRESS learns the transition probability matrix P_x, which is an $M \times M$ matrix where the element p_{ij} at row i and column j denotes the conditional probability of making a transition from state i to state j. Assuming the Markov chain is homogeneous, PRESS can derive the feature value distribution of x for any time in the future by applying the Chapman-Kolmogorov equations [6]: after t time units, the probability distribution for metric x is $\pi_t = \pi_{t-1} P_x = \pi_{t-2} P_x^2 = ... = \pi_0 P_x^t$, where π_t and π_0 denote the probability distribution at time t and the initial probability distribution for the metric x, respectively. Given a current state

[1]The Pearson correlation is obtained by dividing the covariance of P_i and P_j by the product of their standard deviations.

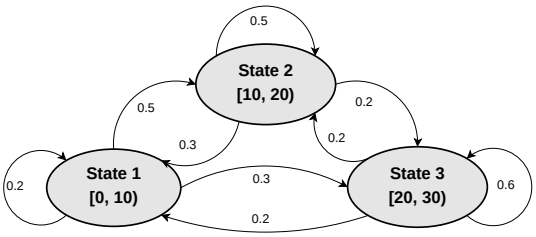


Fig. 2. A state-driven resource demand prediction model. In this example, the metric value (e.g., memory usage) ranges from 0 to 30 units, and this range is partitioned into three states, with the range corresponding to the state shown below the state name. The arcs are labeled with their transition probabilities.

i, PRESS predicts the state at a future time t by extracting the most probable future state j (i.e., the one with the largest transition probability) from the matrix P_x^t at row i.

The further t is into the future, the weaker the correlation between the model and the actual demand. But since PRESS only needs to make predictions for the near future, this is not an issue in practice.

C. Integrated runtime elastic resource scaling

PRESS integrates both signature-driven and state-driven approaches. When a new application starts, PRESS assumes zero knowledge about the application, and sets the limit to its maximum (e.g., 100% CPU). After a few (e.g., 10) resource demand samples have been acquired, PRESS starts making predictions. It starts scaling the resource allocations once it is confident in those predictions. PRESS uses the signature-based approach to make its predictions if it can find a signature; otherwise it uses the state-based approach to make them. (In the future, PRESS might choose to rely on the state-based prediction only if it had sufficient confidence in its correctness.)

PRESS can use different input window sizes for the signature and state-based schemes. Typically, the sliding window for the signature approach should be larger than for the state-based one, although for these experiments we set the window sizes to be the same to allow more direct compatrison of the different algorithms.

PRESS repeatedly updates both prediction models using new measurement samples. To minimize the overhead of doing this, the updates are triggered when the models make a few (e.g., 3) consecutive mis-predictions (under- or over-predictions of more than 10%).

If it is to observe any resource-demand increases, PRESS must not limit the resource allocation to less than the real demand. Under-estimating the application resource demand will make the scaling system set a resource cap that is lower than the real demand. This will not only cause SLO violations but also affect the accuracy of future resource demand predictions since the application's real demand is unknown to the system. To avoid this risk, and to reduce the chance of under-provisioning, PRESS pads (increases) the predicted value by a small amount (5-10%) to allocate a bit more resources than the model suggests. We find that this almost eliminates all under-estimation errors while incurring only a small cost for the additional resources.

D. Non-PRESS algorithms: alternative prediction approaches

For our experiments, we also implemented a set of alternative resource prediction algorithms to compare PRESS against, as follows. All use a sliding window of recent samples.

- *Mean*: the predicted resource demand is the mean usage over the samples in the window.
- *Max*: the predicted resource demand is the maximum resource usage over the samples in the window.
- *Histogram*: this scheme constructs a histogram from the samples in the window, using 40 equal-width bins. The prediction is the average value of the samples in the bin with the largest number of samples - i.e., an approximation to the mode. (This approach is modeled on [7].)
- *Auto-correlation*: this scheme repeatedly shifts the input resource demand time series by one step (up to half the total window length) and calculates the correlation between the shifted time series and the original one. If the correlation is higher than a fixed threshold (e.g., 0.9) after n shifts, a repeating pattern is declared, with duration n steps, and the algorithm stops. The predicted value is based on the position of the current value on the repeating pattern. (This approach is similar to [8].) If no repeating pattern is found, the algorithm falls back to using the mean value of the samples as its prediction result.
- *Auto-regression*: this scheme attempts to predict a value X_t of a system based on the previous values X_{t-1}, X_{t-2} using the formula $X_t = a_1 X_{t-1} + a_2 X_{t-2} \ldots$ The coefficients (a_1, a_2 and so on) are determined by calculating auto-correlation coefficients and solving linear equations:

$$\sum_{i=1}^{Z} a_i R(i-j) = -R(j), \text{ for } 1 \leq j \leq Z$$

where R is the auto-correlation coefficients of the time series, and Z is the length of the sample. (This scheme is similar to [9].)

In practice, we found that the prediction models can sometimes produce negative results when the recent resource demand suddenly drops (e.g., CPU = 100%, 90%, 10%). Since resource demand can never be negative, we allocate a default minimum amount (5% of the maximum resource allocation) when the prediction model generates a negative result.

The next section describes how we evaluated PRESS against these approaches.

III. Experimental Evaluation

We implemented the PRESS system and conducted extensive evaluation studies using the RUBiS [1] online auction benchmark (PHP version) and a small sample of Google cluster resource usage data [2]. This section describes those experiments.

A. Experiment setup

Our experiments were conducted on NCSU's virtual computing lab (VCL) [10]. Each VCL host has a dual-core Xeon 3.00GHz CPU and 4GB memory, and runs CentOS 5.2 64-bit with Xen [11] 3.0.3. The guest VMs also run CentOS 5.2 64-bit.

We focus in this paper on scaling CPU resources because these appeared to be the bottleneck in our experiments.

PRESS monitors an application's resource demands from domain 0, using the `libxenstat` library to collect resource usage information (CPU time, memory, network, and I/O) for both domain 0 and guest VMs. The average resource usage over the sampling duration is used as the input to the time series; it is sampled every minute.[2]

By default, PRESS makes a prediction every minute, and uses the prediction to scale the resources allocated to the application for the next minute. It does this by adjusting the CPU limits of the target VM setting controls on the Xen credit scheduler [12]. The Xen credit scheduler runs in the non-work-conserving mode so that a domain cannot use more than its share of CPU.

The prediction algorithms used by PRESS are indifferent to the sampling rate, as long as the input window size is large enough to encompass the most important patterns to track, and there aren't too many samples to analyze. Since diurnal patterns are important in our test workloads, we used a window size of 3 days, or 4320 samples (fewer during the first 3 days). Unfortunately, the auto-regression approach uses a Gaussian elimination algorithm, which has time complexity $O(n^3)$, and this took longer than a minute with this many samples, so we coalesced the input time series for this one predictor by using the average of every 3 consecutive samples, resulting in 1440 input samples. We also explored the effect of using shorter windows (32 samples) for the simpler algorithms - *mean*, *max*, and *histogram*, to explore the effects of using more recent history to explore the near future. The window sizes are shown on the graphs and tables by appending the number of samples to the algorithm name (e.g., mean-32).

We tested two variants of the PRESS algorithm: PRESS-5% pads the predicted CPU usage by 5% while PRESS-0% adds no padding.

A service provider can either rely on the application itself or an external tool to keep track of whether the application's SLO is violated [13]. In our experiments using RUBiS, we adopted the latter approach, using the RUBiS workload generator to track the response time of the HTTP requests it made. The per-hour SLO violation rate is the fraction of requests in an hour that have response time longer than 1.5 seconds, averaged over a 1 hour period.

In order to evaluate our system under workloads with realistic time variations, we used the per-minute workload intensity observed in two real-world web traces to modulate the request rate of our synthetic RUBiS benchmark. To avoid overloading our servers, the mean request rate was scaled so that the maximum rate did not exceed about 50 requests/sec. Using this approach, we constructed three workload time series:

1) Per-minute average rates observed in the ClarkNet web server trace beginning at 1995-08-28:00.00 from the IRCache Internet traffic archive [14].
2) Per-minute average rates observed in the web access logs from the World Cup 98 official web site [14] that start at 1998-05-05:00.00.
3) A hybrid, with the first 7 days the same as the ClarkNet trace, and then switching to a synthetic workload-intensity time series, whose request rate was uniformly distributed over 15-40 requests/sec.

[2]A future enhancement would be to sample the load much more rapidly (e.g., once a second) and use the 95th%ile of those measurements as the estimate of the resource demand over the 1 minute interval.

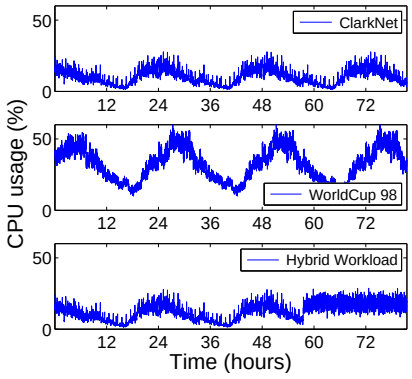


Fig. 3. CPU usage trace for our RUBiS web server driven by three different workloads.

B. Results and analysis

Figure 3 shows the *demand* (the CPU usage achieved with no resource caps) for the RUBiS Web server under three test workload patterns.

Table I shows how well the predictors do against this workload, and how good they are at predicting the actual demand. In addition to showing the total CPU allocation emitted by each predictor, the table also shows the fraction of the predictions that were below the actual demand (te rest were above it). There are quite large ranges of the ratios between the predicted load and the demand, and the frequencies of under-estimation errors: 0.54–3.35× and 0.04%-79% under for ClarkNet, 0.47–1.83× and 0.1%-95% under for WorldCup, and 0.44–2.71× and 0.1%-85% under for Hybrid.

PRESS-5% almost always gave the closest predictions of total demand, and did better than the other algorithms at avoiding under-predictions. It was the most robust of the predictors, with auto-correlation coming second. Compared to the *max-4320* algorithm, which is a close approximation to a static resource allocation, PRESS allocates up to 70% less CPU.

The poor performance of algorithms like *mean-32* comes because they don't allow the workload to exhibit it's actual demand: it is locked into running with less than it needs, and

Algorithm	ClarkNet	WorldCup	Hybrid
Demand	*498 cpu-mins*	*1579 cpu-mins*	*614 cpu-mins*
PRESS-5%	102% (43%-)	99% (51%-)	83% (56%-)
PRESS-0%	96% (52%-)	93% (69%-)	78% (63%-)
Mean-32	54% (79%-)	47% (95%-)	44% (85%-)
Mean-4320	83% (58%-)	81% (65%-)	68% (69%-)
Max-32	63% (68%-)	62% (80%-)	51% (74%-)
Max-4320	335% (0.04%-)	183% (0.1%-)	271% (0.1%-)
Histogram-32	56% (76%-)	50% (93%-)	45% (82%-)
Histogram-4320	110% (42%-)	59% (84%-)	89% (59%-)
Auto-correlation	96% (53%-)	93% (69%-)	78% (63%-)
Auto-regression	70% (74%-)	105% (35%-)	53% (82%-)

TABLE I

TOTAL CPU PREDICTIONS FOR THE RUBIS WEB SERVER WITH A PREDICTION INTERVAL OF 1 MINUTE, SHOWN AS A FRACTION OF THE TOTAL DEMAND, AND (IN PARENTHESES) THE FRACTION OF THE 4983 PERIODS IN WHICH THE PREDICTION WAS LESS THAN THE DEMAND.

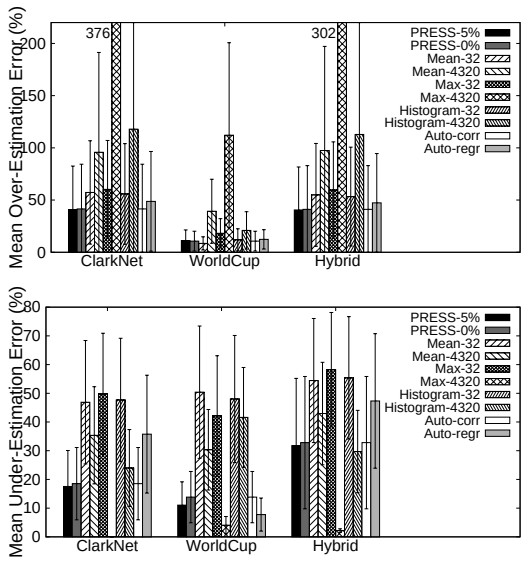


Fig. 4. CPU over- and under-estimation errors for the RUBiS web server with a 1-minute prediction interval. Error bars show the standard deviation from the mean.

they have no way of observing a larger demand that would cause them to increase their cap. Padding would clearly be beneficial here.

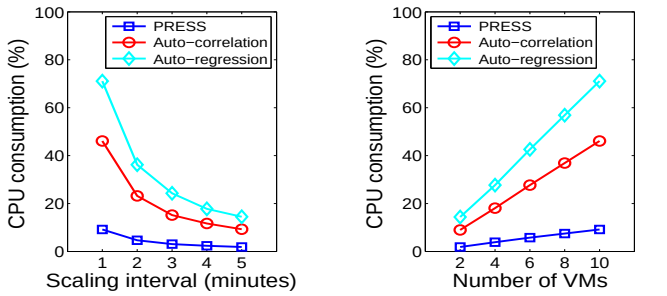
The efficacy of a predictor is also a function of the severity of the individual prediction errors. Figure 4 shows how large the over- and under- prediction errors are for the RUBiS web server workload. (The graphs are obtained by comparing the predicted values to the measured demand value for each period, with over- and under-predictions plotted separately.) PRESS consistently achieved the smallest over- and under-estimation errors. Auto-correlation was almost as good in terms of under-estimations, with auto-regression third best, but those algorithms have higher execution overheads, as we show below. The other algorithms had larger under-estimation errors in at least one case, or, like *max-4320*, allocated significantly more resources than needed. In general, using a longer training window lead to lower under-estimation errors, but at the expense of larger over-estimates.

PRESS is not only the best algorithm in terms of under-estimation errors, but it is significantly cheaper to execute than the next best candidates, as shown in Table II. The

Algorithm	Input size	CPU
PRESS	4320 samples	0.55 $\pm$ 0.01s
Auto-correlation	4320 samples	2.67 $\pm$ 0.01s
Auto-regression	1440 samples	4.26 $\pm$ 0.01s

TABLE II

MEAN AND STANDARD DEVIATION OF CPU EXECUTION COSTS FOR SOME PREDICTION ALGORITHMS, AVERAGED OVER 300 TRIALS.



(a) Prediction interval, with 10 VMs.

(b) Application VMs, with 1 minute prediction intervals.

Fig. 5. CPU overhead from scaling, including both prediction and resource cap settings, under different prediction intervals and numbers of VMs.

auto-correlation algorithm takes about five times as long to run as PRESS, and the auto-regression algorithm takes seven times longer than PRESS even with only one quarter as many samples. For comparison, it takes 120$\pm$0.55ms to apply a CPU limit to a virtual machine, and about 10ms to collect a resource usage sample. Clearly, the dominant overhead cost for dynamic resource allocation is that of making the predictions.

One way to reduce this overhead is to perform scaling less often, and make predictions for longer intervals. At the same time, most cloud service providers will probably run more than one application on a physical machine. Figure 5 shows the results of exploring this space, for the three best predictors. PRESS uses significantly fewer CPU cycles than the other two algorithms. This data suggests that a 9% investment in CPU overhead for predictions can manage up to 10 VMs with a one minute sampling period, delivering up to 70% resource savings compared to a static allocation.

Table III and Figure 6 explore the effects of performing scaling less often, using the same input data as before. Instead of executing prediction and scaling every minute, we adjusted the algorithms to run every 9 minutes, and to provide predictions of usage for each of the next 9 one-minute periods. The overall prediction for the 9-minute period is the maximum of these values, and this is used as the resource allocation for those 9 minutes.

Under these conditions, PRESS continues to perform well. Although auto-regression now achieves fewer, and slightly smaller under-estimation errors than PRESS, it over-estimates more aggressively, resulting in allocations that are 12-48% larger. This occurs because it tends to magnify the effects of sudden changes, over-predicting both upward and downward swings, to the point where its usage predictions are sometimes negative. Because predictions are combined by taking the

Algorithm	ClarkNet	WorldCup	Hybrid
Demand	*497 cpu-mins*	*1577 cpu-mins*	*613 cpu-mins*
PRESS-5%	102% (44%-)	99% (52%-)	83% (56%-)
PRESS-0%	97% (49%-)	93% (69%-)	79% (60%-)
Mean-32	55% (79%-)	48% (95%-)	44% (85%-)
Mean-4320	83% (59%-)	81% (66%-)	68% (69%-)
Max-32	66% (67%-)	66% (95%-)	54% (73%-)
Max-4320	336% (0.04%-)	183% (0.1%-)	272% (0.1%-)
Histogram-32	56% (77%-)	50% (92%-)	46% (83%-)
Histogram-4320	110% (42%-)	59% (85%-)	89% (59%-)
Auto-correlation	97% (49%-)	93% (69%-)	79% (60%-)
Auto-regression	114% (32%-)	147% (6%-)	119% (29%-)

TABLE III
CPU PREDICTIONS FOR THE RUBIS WEB SERVER WITH A PREDICTION INTERVAL OF 9 MINUTES. SEE TABLE I FOR THE DESCRIPTION.

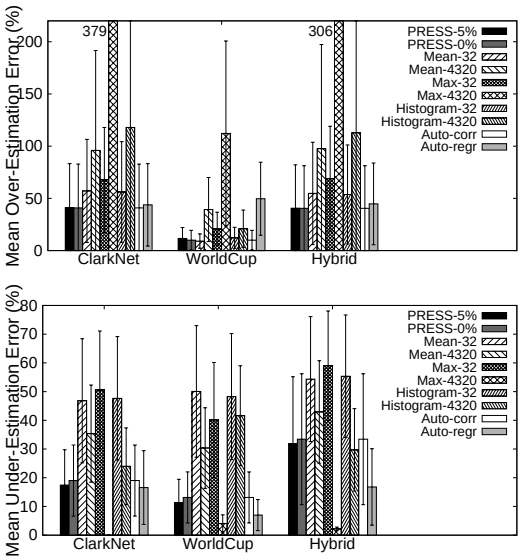


Fig. 6. CPU over- and under-estimation errors for the RUBiS web server, using a 9-minute prediction interval. Error bars show the standard deviation from the mean.

largest predicted value over a multi-minute period, the effects of over-predictions are amplified.

Although under-predictions might save resources, they can have serious consequences, including causing service level objectives (SLOs) to be missed. Figure 7 shows the average per-hour SLO violation rates over days 4 to 6 (i.e., a 72 hour portion) of the ClarkNet and World Cup traces. In this comparison, all the algorithms are using a training window size of 4320 except for auto-regression, which uses 1440, and all the prediction outputs are padded by 10%. Clearly, high SLO violation rates occur when the system under-provisions its resources (e.g., *mean* and *histogram*). As expected, *max* does well at the cost of many more resources, but both PRESS and auto-regression achieve low SLO violation rates too, although the latter is more expensive to run.

We look next at the aggregate impact of over- and under-predictions by examining the profitability of a cloud service provider using these algorithms. To compare the different prediction algorithms, we applied a cost model based on that of Amazon's EC2 [15] to our RUBiS data. We set the resource price for the customer at $1.00 per hour and the resource cost

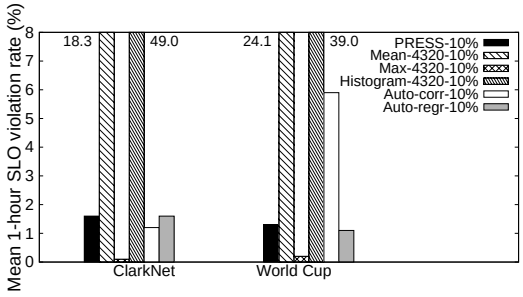
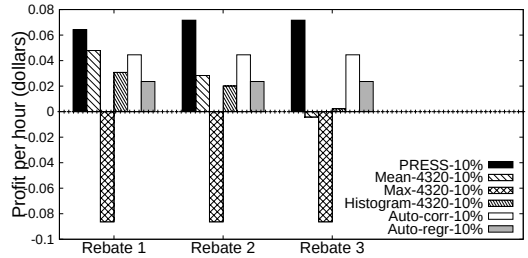


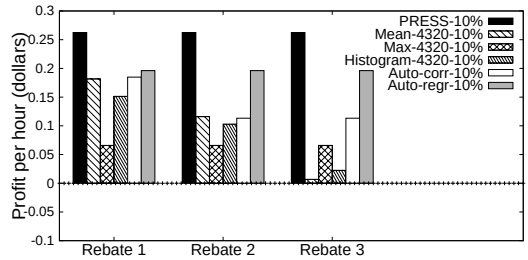
Fig. 7. Average 1-hour service level objective (SLO) violation rates for RUBiS under two different workload modulations. The SLO is "request latency $<$ 1.5 seconds".

Penalty function	Violation-rate $\rightarrow$ rebate	Violation-rate $\rightarrow$ rebate
Rebate1	$\geq$1% $\rightarrow$ 5%	$\geq$5% $\rightarrow$ 10%
Rebate2	$\geq$5% $\rightarrow$ 10%	$\geq$10% $\rightarrow$ 25%
Rebate3	$\geq$10% $\rightarrow$ 25%	$\geq$20% $\rightarrow$ 50%

TABLE IV
THREE DIFFERENT PENALTY FUNCTIONS FOR SLO VIOLATIONS.



(a) ClarkNet workload.



(b) WorldCup 1998 workload.

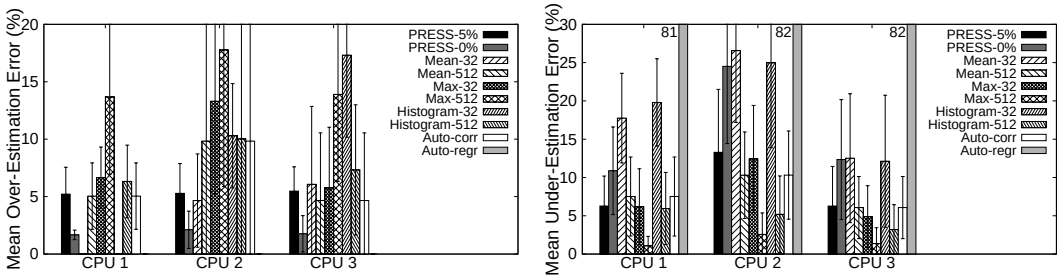
Fig. 8. Profit-rate comparisons for two different workloads over a 72-hour period.

for the service provider at $0.40 per hour.[3]

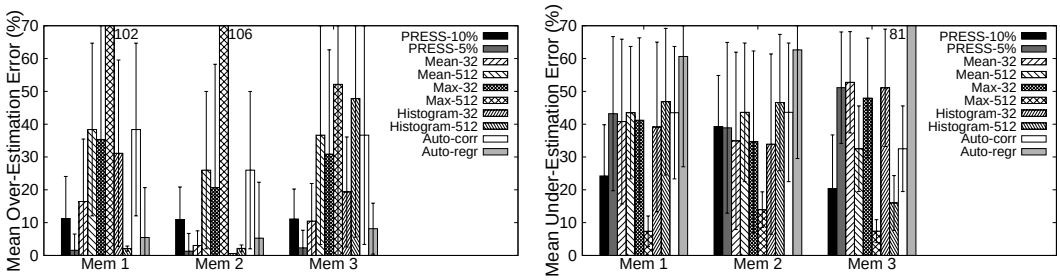
A cloud service provider needs to balance the application owner's desire to achieve a good SLO with its own desire to reduce the resources it allocates to its customers. In support of this, we used the three different penalty functions described in Table IV. As with many cloud providers, a penalty takes the form of a rebate that is a percentage of the resource price; it is paid out if enough client requests take longer than the 1.5s target latency specified by the SLO.

The profit-rate of the cloud provider is calculated by starting with the revenue obtained from renting out the resources used by (not allocated to) the application. We then subtracted from this any penalties (rebates) incurred, as well as the costs for the resources allocated to the application and required to

[3]We experimented with a resource cost of $0.60 per hour, but all algorithms except PRESS made a loss with that value.



(a) CPU resource



(b) Memory resource

Fig. 9. Over- and under-estimation errors for three tasks from the Google workload trace. Error bars show the standard deviation from the mean.

run the prediction algorithm. The profit was calculated every hour and we show the average profit over the same 72 hour period as before in Figure 8 for the three penalty functions. PRESS consistently achieves the highest profit rate. Note that our profit formula causes *max* to do poorly because we only credit the service provider with income from the application's resource usage, not the resources allocated to it.

We also evaluated our prediction algorithms using a small sample of real application workload trace data from a Google cluster [2]. We picked three tasks that ran for the duration of the trace. Because the trace is quite short (about 6 hours), we chose shorter training windows (512 samples) than for the for the RUBiS trace. We also tested very short windows (32 samples) for the *mean*, *max*, and *histogram* predictors. Memory usage for these tasks changes more rapidly than does CPU usage, so we found that it was helpful to increase the prediction padding in PRESS to 10% for memory consumption. No padding was added for the other prediction algorithms. The results are shown in Figure 9. PRESS consistently achieves lower under-estimation errors than all the other schemes (save *max-512*) with a small over-estimation error. As with the RUBiS experiments, using shorter training windows tended to increase the under-estimation errors for the *mean*, *max*, and *histogram* approaches. We omit the prediction error frequency results due to space limitations, but they show that PRESS incurred significantly fewer under-estimation errors than all the other algorithms.

IV. Related Work

Resource management has been extensively studied. Due to space limitations, we focus on work that is most closely related to ours. Previous work has used offline or online profiling [16], [17], [18], [19] to experimentally derive application resource requirements using benchmark or real application workloads. However, profiling needs extra machines and may take a long time to derive resource requirements. PRESS avoids the need to profile offline by deriving resource demands as the application runs.

Recently, research work has been done in model-driven resource management. Those approaches use queueing theory [4] or statistical learning methods [20], [21], [22], [23] to build models that allow the system to predict the impact of different resource allocation policies on the application performance. The model needs to be calibrated in advance and the system needs to assume certain prior knowledge about the application and the running platform (e.g., input data size, cache size, processor speed), which may not be feasible in the cloud system. In contrast, PRESS is application and platform agnostic.

Previous work has applied control theory [24], [25], [26] or reinforcement learning [27] to adaptively adjust resource allocations based on SLO conformance feedback. However, those approaches often have parameters that need to be specified or tuned offline, and need some time to converge to the optimal (near-optimal) decisions. In comparison, PRESS directly predicts optimal resource allocation based on historical resource demand time series.

Trace-based resource management utilizes historical application resource demands to guide various resource management decisions. PRESS is most closely related to this category of research work. Rolia et al. perform dynamic resource allocation using an estimated burst factor times the most recent resource demand [28]. In contrast, PRESS examines a window of recent resource demands to predict future resource demand directly. Our experiments have shown that such an approach is highly accurate. Gmach et al. [29] used a Fourier transform-based scheme to perform offline extraction of long-term cyclic workload patterns. In comparison, PRESS does not assume the workload is cyclic, and can predict resource demands for

both repeating and non-repeating workload patterns. Moreover, PRESS performs online signature extraction by detecting signature changes and supporting signature alignment with runtime measurement time series. Chen et al. [3] used sparse periodic auto-regression to perform load prediction. However, their approach is tailored toward long prediction intervals (e.g., hours) and assumes the repeating period is known in advance. Our experiments have shown that auto-regression is computationally intensive, which makes it impractical for short-term online VM resource scaling. Chandra et al. [9] proposed two workload prediction algorithms: auto-regression and histogram based method. We have shown in the experiments that PRESS can significantly outperform the histogram based method and has much less overhead than the auto-regression scheme.

Gmach et al. [30], [31] proposed an integrated workload placement solution using both peak demand based workload assignment simulation and fuzzy logic based feedback control guided workload migration. Buneci and Reed [8] used auto-correlations to extract repeating patterns for detecting performance anomalies. In comparison, PRESS provides resource prediction for both repeating and non-repeating patterns, and integrates online resource demand prediction with dynamic VM resource scaling. Wuhib et al. [32] proposed a gossip protocol to achieve load balancing in cloud systems. In contrast, PRESS addresses an orthogonal problem of setting minimum resource caps to different VMs while still meeting the resource demand of different applications.

V. Conclusion

We have presented the design and implementation of the PRESS system, a novel predictive resource scaling system for cloud systems. PRESS employs light-weight signal processing and statistical methods to predict dynamic resource demands of black box applications and performs elastic VM resource scaling based on the prediction results. We implemented the PRESS system on top of Xen in the NCSU Virtual Computing Lab, using RUBiS benchmarks driven by real-life workload traces, and a resource-usage trace of a Google application workload.

Our results show that PRESS's resource-usage predictions achieve high accuracy and its allocations achieve better service provider profitability than other approaches across a range of workloads. We believe PRESS is an attractive scheme for large-scale cloud computing infrastructures.

References

[1] "RUBiS Online Auction System," http://rubis.ow2.org/.

[2] "Google Cluster Data," http://googleresearch.blogspot.com/2010/01/google-cluster-data.html.

[3] G. Chen and et al., "Energy-aware server provisioning and load dispatching for connection-intensive internet services," in *Proc. NSDI*, 2008.

[4] R. Doyle, J. Chase, O. Asad, W. Jin, and A. Vahdat, "Model-based resource provisioning in a web service utility," in *Proc. USITS*, 2003.

[5] S. Salvador and P. Chan, "FastDTW: Toward accurate dynamic time warping in linear time and space," in *Proc. KDD Workshop on Mining Temporal and Sequential Data*, 2004.

[6] A. Papoulis, *Probability, Random Variables, and Stochastic Processes, 2nd ed*. McGraw-Hill, 1984.

[7] M. Cardosa and A. Chandra, "Resource bundles: Using aggregation for statistical wide-area resource discovery and allocation," in *Proc. ICDCS*, 2008.

[8] E. S. Buneci and D. Reed, "Analysis of application heartbeats: Learning structural and temporal features in time series data for identification of performance problems," in *Proc. Supercomputing*, 2008.

[9] A. Chandra, W. Gong, and P. Shenoy, "Dynamic resource allocation for shared data centers using online measurements," in *Proc. IWQoS*, 2004.

[10] "Virtual Computing Lab," http://vcl.ncsu.edu/.

[11] P. Barham and et al., "Xen and the art of virtualization," in *Proc. SOSP*, 2003.

[12] "Xen Credit Scheduler," http://wiki.xensource.com/xenwiki/CreditScheduler.

[13] D. Breitgand, M. B.-Yehuda, M. Factor, H. Kolodner, V. Kravtsov, and D. Pelleg, "NAP: a building block for remediating performance bottlenecks via black box network analysis," in *Proc. ICAC*, 2009.

[14] "The IRCache Project," http://www.ircache.net/.

[15] "Amazon Elastic Compute Cloud," http://aws.amazon.com/ec2/.

[16] B. Urgaonkar, P. Shenoy, and et al., "Resource overbooking and application profiling in shared hosting platforms," in *Proc. OSDI*, 2002.

[17] T. Wood, L. Cherkasova, and et al., "Profiling and modeling resource usage of virtualized applications," in *Proc. Middleware*, 2008.

[18] W. Zheng, R. Bianchini, and et al., "JustRunIt: Experiment-based management of virtualized data centers," in *Proc. USENIX Annual Technical Conference*, 2009.

[19] S. Govindan, J. Choi, and et al., "Statistical profiling-based techniques for effective power provisioning in data centers," in *Proc. Eurosys*, 2009.

[20] P. Shivam, S. Babu, and J. Chase, "Learning application models for utility resource planning," in *Proc. USITS*, 2003.

[21] C. Stewart, T. Kelly, A. Zhang, and K. Shen, "A dollar from 15 cents: Cross-platform management for internet services," in *Proc. USENIX Annual Technical Conference*, 2008.

[22] A. Ganapathi, H. Kuno, and et al., "Predicting multiple metrics for queries: Better decisions enabled by machine learning," in *Proc. ICDE*, 2009.

[23] P. Shivam, S. Babu, and J. Chase, "Active and accelerated learning of cost models for optimizing scientific applications," in *Proc. VLDB*, 2006.

[24] X. Zhu and et al., "1000 Islands: Integrated capacity and workload management for the next generation data center," in *Proc. ICAC*, Jun. 2008.

[25] E. Kalyvianaki, T. Charalambous, and S. Hand, "Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman filters," in *Proc. ICAC*, 2009.

[26] P. Padala and et al., "Adaptive control of virtualized resources in utility computing environments," in *Proc. Eurosys*, 2007.

[27] J. Rao, X. Bu, C.-Z. Xu, L. Wang, and G. Yin, "VCONF: A reinforcement learning approach to virtual machines auto-configuration," in *Proc. ICAC*, 2009.

[28] J. Rolia, L. Cherkasova, M. Arlitt, and V. Machiraju, "Supporting application QoS in shared resource pools," *Communications of the ACM*, 2006.

[29] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Capacity management and demand prediction for next generation data centers," in *Proc. ICWS*, 2007.

[30] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper, "An integrated approach to resource pool management: Policies, efficiency and quality metrics," in *Proc. DSN*, 2008.

[31] D. Gmach, J. Rolia, and L. Cherkasova, "Satisfying service level objectives in a self-managing resource pool," in *Proc. SASO*, 2009.

[32] F. Wuhib, R. Stadler, and M. Spreitzer, "Gossip-based resource management for cloud environments," in *Proc. CNSM*, 2010.