Workload Classification for Efficient Auto-Scaling of Cloud Resources

Ahmed Ali-Eldin[1], Johan Tordsson[1],
Erik Elmroth[1], and Maria Kihl[2]

[1]Department of Computing Science, Umeå University, Sweden
{ahmeda, tordsson, elmroth}@cs.umu.se
[2]Department of Electrical and Information Technology,
Lund University, Sweden
Maria.Kihl@eit.lth.se

May 21, 2013

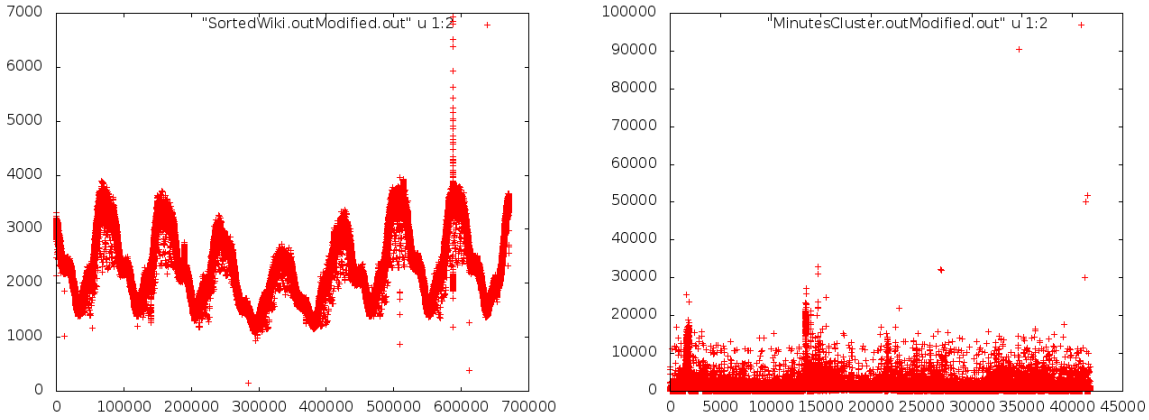
Abstract

Elasticity algorithms for cloud infrastructures dynamically change the amount of resources allocated to a running service according to the current and predicted future load. Since there is no perfect predictor, and since different applications' workloads have different characteristics, no single elasticity algorithm is suitable for future predictions for all workloads. In this work, we introduce WAC, a Workload Analysis and Classification tool that analyzes workloads and assigns them to the most suitable elasticity controllers based on the workloads' characteristics and a set of business level objectives.

WAC has two main components, the analyzer and the classifier. The analyzer analyzes workloads to extract some of the features used by the classifier, namely, workloads' autocorrelations and sample entropies which measure the periodicity and the burstiness of the workloads respectively. These two features are used with the business level objectives by the classifier as the features used to assign workloads to elasticity controllers. We start by analyzing 14 real workloads available from different applications. In addition, a set of 55 workloads is generated to test WAC on more workload configurations. We implement four state of the art elasticity algorithms. The controllers are the classes to which the classifier assigns workloads. We use a K nearest neighbors classifier and experiment with different workload combinations as training and test sets. Our experiments show that, when the classifier is tuned carefully, WAC correctly classifies between 92% and 98.3% of the workloads to the most suitable elasticity controller.

1 Introduction

Elasticity or auto-scaling can be defined as the ability of a cloud infrastructure (datacenter) to dynamically change the amount of resources allocated to a running application. Resources should be allocated according to the changing load



(a) Workload trace for the Wikipedia workload.

(b) Workload trace for the Google Cluster workload.

Figure 1: Workload traces of the Wikipedia workload and the Google cluster workload.

allowing the addition and removal of resources to preserve Quality of Service (QoS) requirements at reduced cost. Typically, a variety of different applications with different workloads run in a cloud [1]. Even when a single application is running on the infrastructure, in the case for Software-as-a Service [2], different users usually have different usage patterns.

Some workloads have repetitive patterns. For example, the Wikipedia workload shown in Figure 1(a) has a diurnal pattern where the request arrival rate is higher during the day than at night. Other workloads have seasonal patterns, e.g., the workload of an online store may increase drastically before Christmas [3]. Some uncorrelated spikes and bursts can occur in a workload due to an unusual event, e.g., when Michael Jackson died 15% of all requests directed to Wikipedia where to the article about him [4] causing a significant spike. On the other hand, some workloads have some weak patterns or no patterns at all such as the workload shown in Figure 1(b) for a Google cluster workload. Cloud infrastructure providers do not usually know the characteristics of the workloads their customers are planning to run.

Elasticity controllers are used to predict future workload and provision resources based on the prediction [5, 6, 7, 8, 9]. In our previous work, we have designed three adaptive autonomic elasticity controllers and tested them with three different workload traces; the FIFA worldcup 1998 workload trace [10, 11], Wikipedia traces from 2009 [2, 12] and a recently released workload from a production Google cluster [13, 14]. The proposed controllers showed variations in performance with different workloads. These variations are attributed to the different characteristics of different workloads.

Since there are no perfect controllers [15] or perfect estimators [16], designing a general elasticity controller for all workloads and scenarios running on a datacenter is infeasible. Elasticity controllers' performance varies with the different workloads and changing system dynamics. A controller tuned for certain workload scenarios can become unrobust if the conditions change [17] leading to wrong predictions and thus wrong capacity provisioning decisions. There is

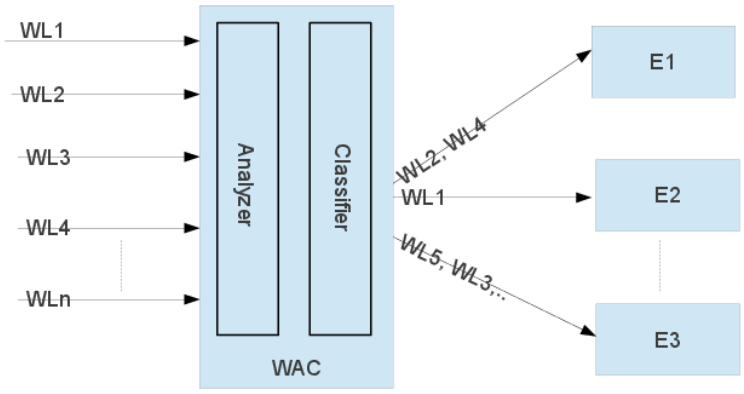


Figure 2: WAC: A Workload Analysis and Classification Tool.

a cost if a controller adds more resources to a service than actually required by the service's workload since these extra resources are not utilized, although paid for by the customer. Similarly, if the resources provisioned for a service are not sufficient for its load, the service performance deteriorates and part of the workload is dropped. The service can become unresponsive or crash. Thus, analyzing workloads is an important first step towards choosing the appropriate elasticity controller for each workload. Based on this analysis, the service provider can assign the workload to the most suitable elasticity controller.

This work presents WAC, a Workload Analysis and Classification tool for cloud infrastructure providers. The tool analyzes workloads of different applications and based on the analysis it classifies the workloads into a set of predefined classes. Each class is then assigned to the most suitable elasticity controller for that class, reducing the risk of wrong predictions and improving the Quality of Service (QoS) provided then. It is assumed that there is some historical data available for the workloads. While not the case for all applications, many applications will have this data if they have been running in the past.

Figure 2 shows the two main components of WAC; the analyzer and the classifier. The analyzer analyzes historical data of a workload to discover some of its statistical properties that can be used by the classifier to find the most suitable elasticity controller for the workload of an application. There are many properties that can be used to characterize a workload such as the mean request arrival rate, the variance of the request arrival rate, average request processing time, the workload's periodicity and the workload's burstiness.

In this work, we concentrate on *periodicity* and *burstiness* to characterize a workload. Periodicity describes how certain parts of the workload repeat with time, i.e., the similarity between the value of a workload at any point in time to the workload's history. Such similarity, if discovered, can be used by the elasticity controller to predict the future values of the workload. Burstiness measures load spikes in the system, when a running service suddenly experiences huge increases in the number of requests. Bursty workloads where the workload does not follow any pattern for these bursts, are hard to predict. We discuss our choice of these two properties and the methods we use to quantify them in more detail in Section 2.

Classification is a form of supervised learning used to predict the type or class of some input data object based on a group of features [18]. A data object can be anything from a single network request to a data stream [19]. For WAC, the data object is a workload and the classes or types are the different elasticity controllers. Supervised learning algorithms require training using labeled training data, i.e., objects for which we know the classes. The training data should be sufficient to enable the classifier to generalize to future inputs that arrive after training, i.e., new applications and workloads which will start running on the infrastructure in the future. To construct our training set, we collect and analyze a set of 14 real workloads. Since available real workloads are scarce [4], and the accuracy of any classifier depends on the size of the training set, we generate 55 additional synthetic workloads that have more general characteristics compared to the real workloads. Workload analysis and generation are discussed in Section 3.

Each class maps to an available elasticity controller implemented and provided by the cloud provider. This enables a cloud provider or a cloud user to choose the right controller for a certain workload reducing the aggregate prediction errors and allowing an improvement in QoS. Many designs for elasticity controllers are suggested in literature using different approaches such as Control theory [5], Neural networks [20], second order regression [7], histograms [6] and the secant method [21]. A cloud provider has to choose which controllers to implement. We have selected and implemented four controllers to use with WAC as discussed in more details in Section 4.

There are plenty of classifiers suggested in the machine learning literature, out of these Support Vector Machines (SVMs) and K-Nearest-Neighbors (KNN) are very popular [22]. KNN is chosen as the classification algorithm for WAC. We discuss the underlying principles of the two methods and our choice of KNN over SVMs in Section 5. We also present the training of the classifier, the classifier accuracy and discuss our results. Section 6 reviews the related work. We conclude in Section 7.

2 The Analyzer: Workload Characterization Techniques

Characterization of workloads is not an easy task since there are many parameters that can be used for the characterization [23]. With respect to prediction and elasticity, the two main characteristics that affect the performance of workloads are *periodicity* and *burstiness*. As will be discussed later, these two

measures are normalized for different workloads making their measurements depending only on workloads' patterns and trends.

2.1 Measuring Periodicity and Burstiness

The most common method to measure periodicity is to use autocorrelation. Autocorrelation is the cross-correlation of a signal with a time-shifted version of itself [16]. A signal can be any measurement, observation or function that changes with respect to time, space or any other dimension, e.g., a workload, an audio signal or a time series [24]. Autocorrelation takes values between -1 and 1. An autocorrelation of 0 at a time lag of τ means there is no relation between the signal and itself at this time lag. A random signal has an autocorrelation of 0 at all time lags. An autocorrelation of 1 means that the signal strength is exactly the same for this time lag while an autocorrelation near -1 means that the signal has an opposite direction of influence, i.e., if the signal strength is above average, the next value will be likely less than average. The autocorrelation function (ACF) describes the autocorrelation as a function of the time-lag τ. The ACF, R(τ), for a signal X is,

$$R(\tau) = \frac{E[(X_t - \mu)(X_{t+\tau} - \mu)]}{\sigma^2}, \tag{1}$$

where E is the expected value operator, X_t is the value of the signal at time t, μ is the mean of the signal and σ is the variance of the signal. We use the ACF as a measure of workload periodicity. For the rest of the paper we use the terms autocorrelation and correlation coefficient interchangeably.

There are many methods to measure burstiness [25], non of them prevalent though. Gusella [26] suggested the use of the index of dispersion. Minh et. al. [27] suggested the use of normalized entropy. We propose the use of Sample Entropy (SampEn) as a measure of burstiness [28]. Sample entropy is a modification of the Kolmogrov Sinai entropy. It is the negative natural logarithm of the conditional probability that two sequences similar for m points are similar at the next point. It has been used successfully in classifying abnormal (bursty) EEG signals and other physiological signals for over a decade and has been proven robust [29]. An advantage of sample entropy over the index of dispersion and normalized entropy is that SampeEn detects periodic bursts.

Two parameters are needed to calculate SampEn for a workload. First parameter is the pattern length m, which is the size of the window in which the algorithm searches for repetitive bursty patterns. The second parameter is the deviation tolerance r which is the maximum increase in load between two consecutive time units before considering this increase as a burst. When choosing the deviation tolerance, the relative and absolute load variations should be taken in account, For example, a workload increase requiring 25 extra servers for a service having 1000 VMs running can probably be considered withing normal operating limits, while if that increase was for a service having only 3 servers running then this is a significant burst. Thus by carefully choosing an adaptive r, SampEn becomes normalized for all workloads. The deviation tolerance defines what is considered a normal increase and what is considered a burst. If SampEn is equal to 0 then the workload has no bursts. The higher the value for SampEn, the more bursty the workload is.

Data: r, m, W
Result: SampEn

$n \leftarrow Length(W)$;
$B_i \leftarrow 0$;
$A_i \leftarrow 0$;
$X_m = \{X_m(i) | X_m(i) = [x(i), x(i+1), ...x(i+m-1)] \; \forall \; 1 < i < n-m+1\}$;
for $(X_m(i), X_m(j))$ *in* X_m*:* $i < j$ **do**
    Calculate $d[X_m(i), X_m(j)] = \max(|x(i+k) - x(j+k)|) \; \forall \; 0 \leq k < m$;
    if $d[X_m(i), X_m(j) \leq r$ **then**
        $B_i = B_i + 1$;
$B^m(r) = \frac{1}{n-m} \sum_{i=1}^{n-m} \frac{1}{n-m-1} B_i$;
$m = m + 1$
$X_m = \{X_m(i) | X_m(i) = [x(i), x(i+1), ...x(i+m-1)] \; \forall \; 1 < i < n-m+1\}$;
for $(X_m(i), X_m(j))$ *in* X_m*:* $i < j$ **do**
    Calculate $d[X_m(i), X_m(j)] = \max(|x(i+k) - x(j+k)|) \; \forall \; 0 \leq k < m$;
    if $d[X_m(i), X_m(j) \leq r$ **then**
        $A_i = A_i + 1$;
$A^m(r) = \frac{1}{n-m} \sum_{i=1}^{n-m} \frac{1}{n-m-1} A_i$;
SampEn$= \lim\{-\log[\frac{A^m(r)}{B^m(r)}\}$

Algorithm 1: The algorithm for calculating SampEn.

We implemented the sample entropy algorithm as described by Aboy et. al. [30]. The algorithm is shown in Algorithm 1. W is the workload for which SampEn is calculated. The first loop in the algorithm calculates $B^m(r)$, the probability that two sequences (parts) in the workload having m measurements do not have bursts. The second loop in the algorithm calculates $A^m(r)$, the probability that two sequences in the workload having $m+1$ measurements do not have bursts. Then SampEn is calculated as the negative logarithm of the conditional probability that two workloads sequences of length m that do not have bursts, also have no bursts when the sequence length is increased by 1.

3 Workload Analysis

Typically, cloud providers such as Amazon [1] and Rackspace [31] host many services with different workloads. These services typically vary from services that use a handful of machines to services that use a few thousand machines [32]. With the exception of a recently released workload trace from a production cluster at Google [13], no cloud provider provides publicly available cloud workloads [4]. On the other hand there are a few publicly available workloads for various types of Internet applications. We have analyzed some of these workloads in addition to the Google cluster workload in order to understand better the characteristics of the different workloads. We believe that these workloads are representative for many of the workloads running on typical clouds.

Recently, cloud providers started supplying resources for High performance computing and grid applications [33]. Many similar workloads are publicly available of which we use the DAS-2 system workload, the Grid-5000 workload, Nor-

duGrid traces, SHARCNET traces, Large hydrogen collider Computing Grid (LCG) traces and the AuverGrid traces, all available from the grid workload archive [34]. Caching services are another type of application running on cloud infrastructures [1]. We use traces from the IRCache [35] project as representative workloads for this class of applications. The IRCache traces maintain traces from 10 different caching service sites in the USA. We only use traces from 4 sites. In addition, we analyze a one month trace of PubMed server usage [36], traces from the FIFA 1998 world cup [37], the Google cluster data released [13] and traces from Wikipedia [12]. We also deployed a crawler that crawled YouTube during the Olympics and the Paralymics for video view count but this data is not used. Appendix A contains the graphs for the analyzed traces.

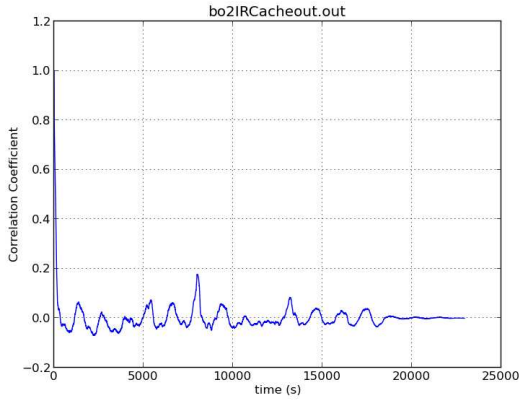
3.1 Periodicity of Real Workloads

A correlogram is a type of plot that shows the ACF of a signal at different lags. The X axis of the plot represents the lag in minutes while the Y axis represents the autocorrelation. Figures 3, 4 and 5 show the correlograms of the analyzed workloads. Figures 3(a) and 3(b) show high ACFs for short lags meaning that for these workloads the value of the workload depends on its value in the near past. Figures 3(c), 3(d), 5(a), 5(c) and 5(d) show high autocorrelation coefficients meaning that their corresponding workloads have strong periodicity. Figures 4 and 5(b) show that their corresponding workloads are almost random. The second column in Table 1 describes the ACFs for the all workloads qualitatively.

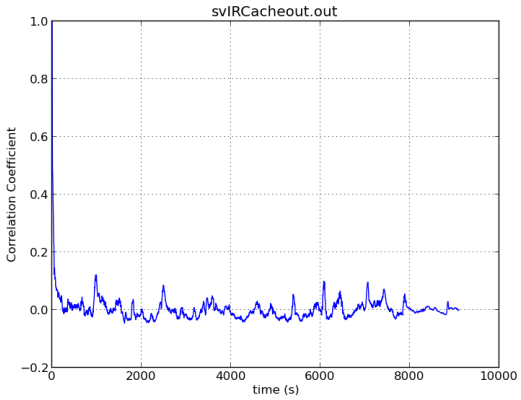
The coefficient of determination is defined as the squared value of the correlation coefficient at a certain lag. It measures the proportion of variance in a dependent variable that can be explained by the independent variable, i.e., the dependence of the future data on the historical data of the workload [38]. Workloads with higher autocorrelation coefficients at certain time lags are easier to predict at these time lags since their coefficient of determination is high.

3.2 Burstiness of Real Workloads

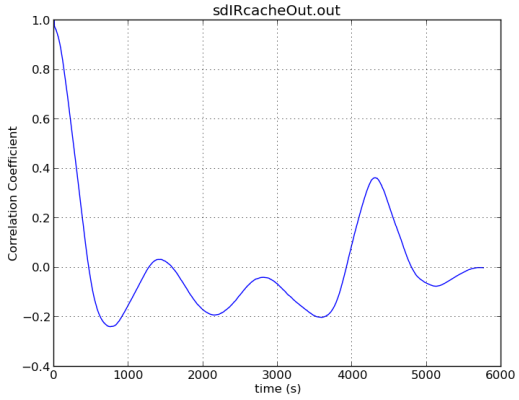
There are two main limitations of SampEn. First, SampEn is expensive to calculate both CPU-wise and memory-wise. The computational complexity (in both time and memory) of SampEn is $O(n^2)$ where n is the number of points in the trace. Second, workload characteristics can change during operation, e.g., when Michael Jackson died 15% of all requests directed to Wikipedia where to the article about him creating spikes in the load [4]. If SampEn is calculated for the whole past, then such recent changes will not be discovered easily. To overcome these two limitations, we divide a trace into smaller equal sub-traces and calculate SampEn for each sub-trace allowing to calculate a weighted average for SampEn giving more weight to more recent SampEn values. This reduces the time required for computing SampEn for a long trace spanning months or years to just a few hours. It is also suitable for online characterization of a workload since SampEn does not have to be recomputed for the whole history but rather for the near past which is much more efficient and fast. Our approach of dividing the signal into smaller sub-traces is similar to the approach used by Costa et. al [39] where they divide a signal to calculate its multi-scale entropy. The main difference between the two approaches is after SampEn is computed



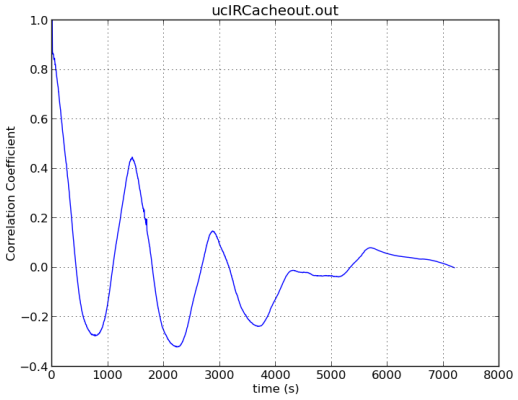
(a) Correlogram for IRCache service runnings at Boulder, Colorado (Bo).



(b) Correlogram for IRCache service running at Silicon Valley, California (SV).

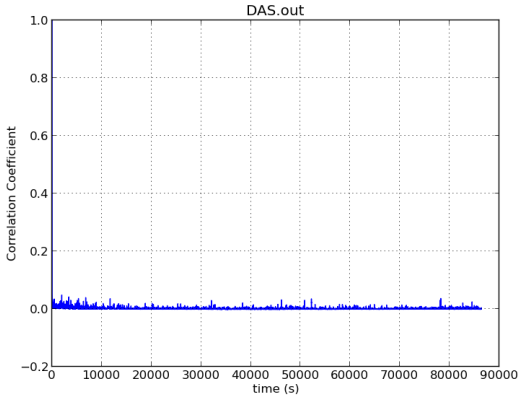


(c) Correlogram for IRCache service running at San Diego, California (SD).

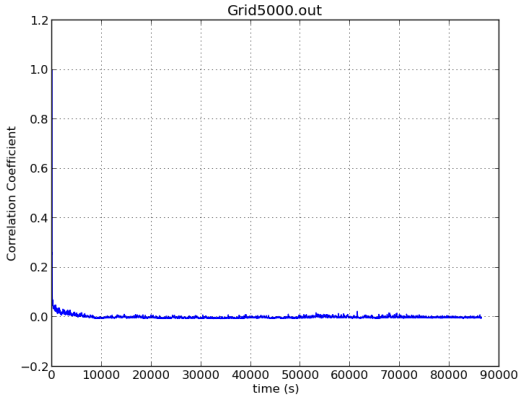


(d) Correlogram for IRCache service running at Urbana-Champaign, Illinois (UC).

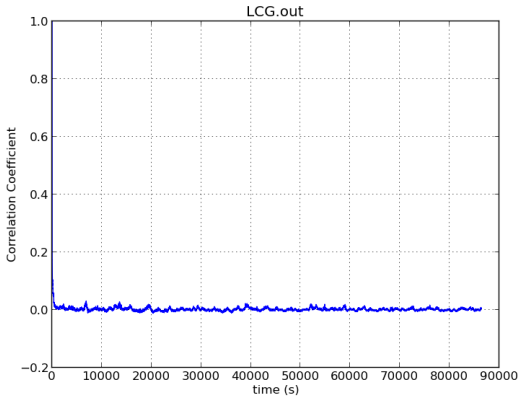
Figure 3: Correlograms of the different Caching Services.



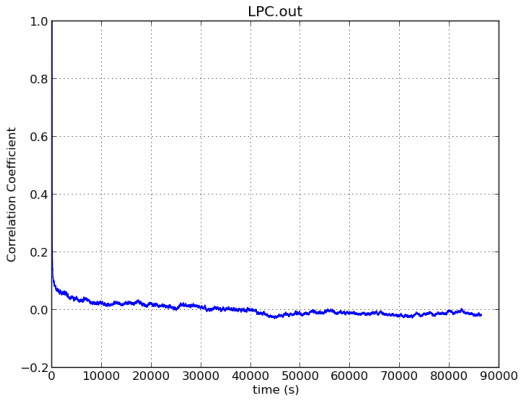
(a) Correlogram for the DAS workload.



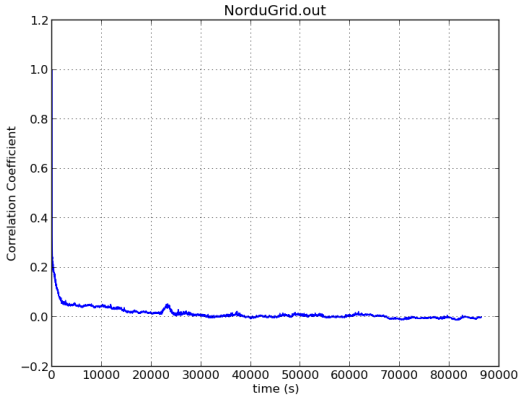
(b) Correlogram for Grid5000 workload.



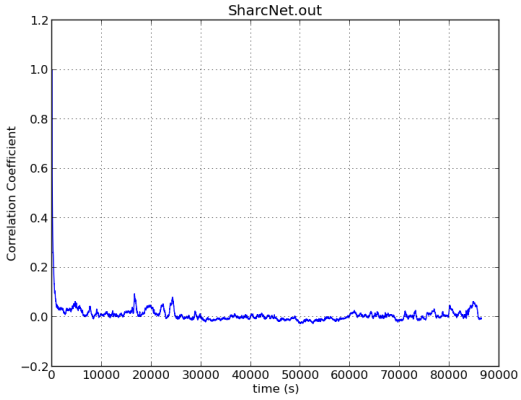
(c) Correlogram for the LCG workload.



(d) Correlogram for LPC workload.

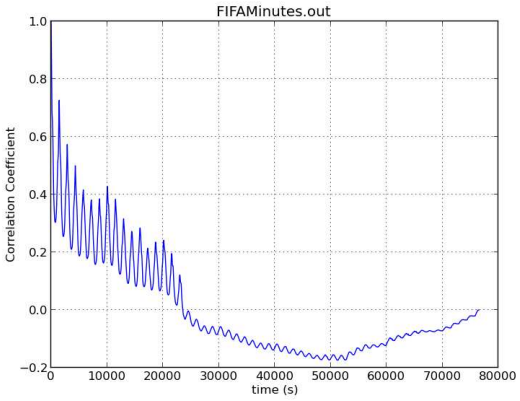


(e) Correlogram for the NorduGrid workload.

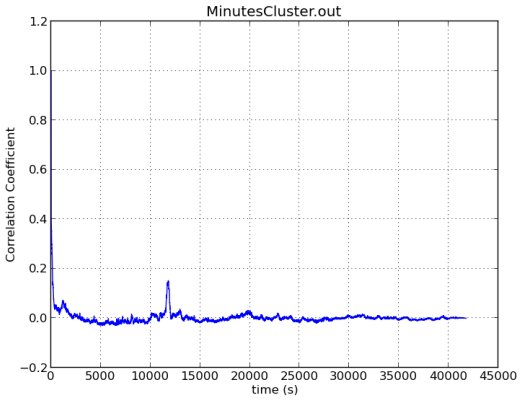


(f) Correlogram for SharcNet workload.

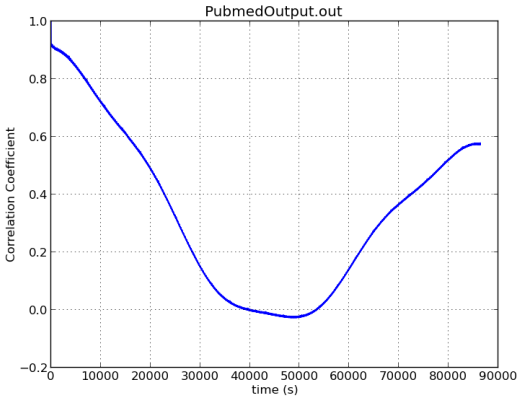
Figure 4: Correlograms of the Grid workloads analyzed.



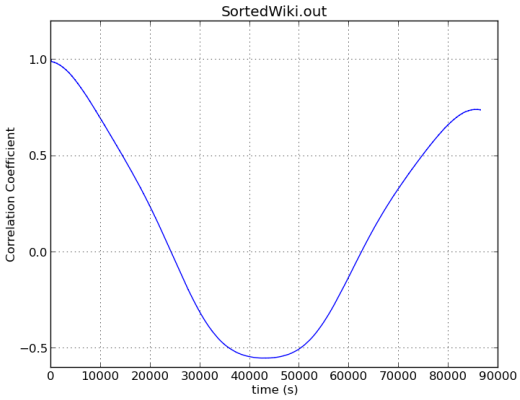
(a) Correlogram for the world cup workload.



(b) Correlogram for the Google Cluster workload.



(c) Correlogram for PubMed access traces.



(d) Correlogram for the Wikipedia workload.

Figure 5: Correlograms of web-hosting workloads and the Google cluster workload analyzed.

Table 1: Workload analysis results.

Workload	ACF	SampEn
Bo-IRCache	Very low. Almost a random signal	0.0
SV-IRCache	Very low. Almost a random signal	0.0017
SD-IRCache	High for small lags. Medium for large lags	1.062
UC-IRCache	High for small lags. Medium for large lags	0.0
DAS	Random signal	0.0
Grid5000	Random signal	236.16
LCG	Random signal	134.0
LPC	Random signal	2.827
NorduGrid	Random signal	8.43
SharcNet	Random signal	2.803
FIFA	High for small lags. Medium for large lags	0.0
Google	Almost a random signal	235.70
PubMed	High	0.0
Wikipedia	High	0.0014

for the smaller sub-trace. Costa et.al plot the results while we take a weighted average.

As already mentioned, there are two main parameters needed to calculate SampEn, the pattern length m and the deviation tolerance r. In addition to these two parameters we add L, the size of each sub-trace when the whole trace is divided. These 3 parameters are set as following:

1. The pattern length m is set to 1 hour.
2. The deviation tolerance r is set to 30% of the seventieth percentile of the workload values, e.g, if the seventieth percentile of the workload values seen is 20000 requests per second, then r is set to 6000 requests. If the number of servers required to handle the load is less than 10 server, i.e., 30% of the maximum workload can be less than the load handled by one server, then the deviation tolerance is set to double the average requests served be a server per unit time.
3. The value of L is set to 1 day.

The third column in Table 1 shows the average SampEn computed for the different workloads. From the table we can see that workloads vary greatly in the amount of burstiness they have. With the exception of the DAS workload, workloads with ACFs around zero have significant burstiness as SampEn is greater than 1 for all of them, while the ones with high or medium ACFs do not show significant burstiness.

3.3 Workload Generation

To obtain accurate classification from a classifier, enough training samples are needed. Available real workloads are scarce and using them only as a training set for a classifier could result in poor ability to generalize and accurately classify newly deployed applications' workloads. We have generated 55 synthetic

Table 2: Average Sample Entropy computed for the sample generated workloads.

Workload	SampEn
WL1	0.0
WL2	236.0
WL3	2.5
WL4	0.002
WL5	0.0014
WL6	0.069

workloads using a mixture of functions and probability distributions. Due to space limitations and since this is not the main focus of this work, we omit most of the details on workload generation. For illustration purpose, Equation 2 shows how WL1 whose ACF is shown in Figure 6(a) is generated.

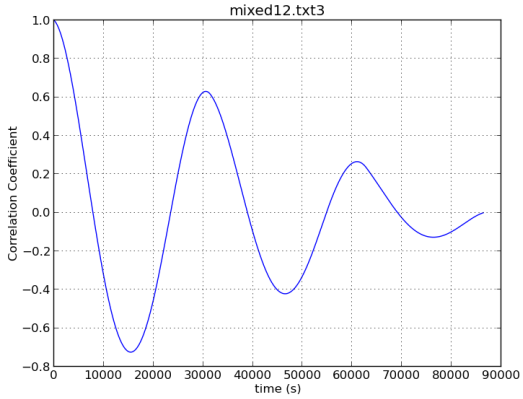
$$x(i) = \log(i+10^3)(|\sin(\frac{3i}{10^4})+100|)(|\sin(\frac{i}{10^4}))+2*10^4|)+W(0.3), \quad \forall\, i \in [0,n], \tag{2}$$

where $x(i)$ is the workload value at time i, $W(0.3)$ is Weibull distribution with a shape parameter equal to 0.3 and n is the length of the synthetic workload required. The log function is used to simulate a workload evolving with time but having similar periodicity. The sinusoids are used to give the workload a pattern similar to diurnal patterns and the Weibull distribution adds some uncertainty in the workload. This is similar to the pattern seen in the Wikipedia workload trace. Figures 6(b) and 6(c) show the correlograms for WL2 and WL3. The two workloads are generated with an equation similar to Equation 2, but with an added component that increases the uncertainty (SampEn) in the workload thus decreasing the ACF. Figures 6(b) and 6(d) show that the ACFs for WL2 and WL4 resemble the ACFs of the DAS, Grid5000 and LCG real workloads whose ACFs are shown in figures 4(a)–4(c) respectively. Figures 6(e) and 6(f) have ACFs different from the ones seen in the 14 investigated workloads. The equations used to generate the 55 synthetic workloads have mostly similar structures, but using different periods for the sinusoids, different amplitudes and different probability distributions such as log normal distributions, gamma distributions and chi-squared distributions as sources of uncertainty.

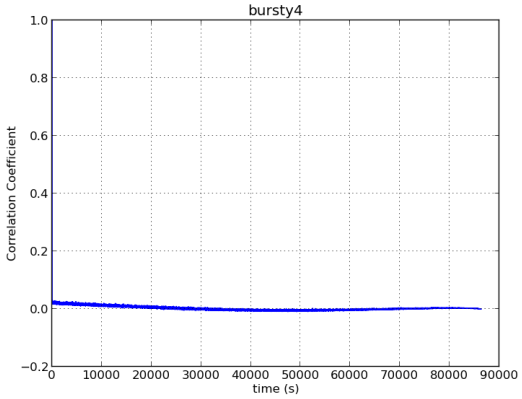
Table 2 shows the SampEn values for the six generated workloads above. Similar to the real workloads, the generated workloads have burstiness levels ranging from no burstiness to very bursty.

4 On the Performance of Different Elasticity Controllers

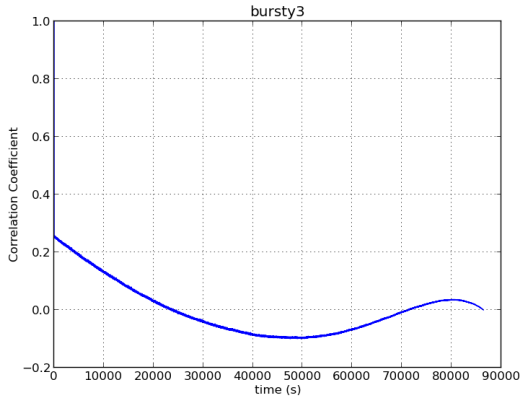
Since different elasticity controllers exhibit different performance with different workloads, a cloud infrastructure provider has to select a number of elasticity controllers to implement. These controllers are the classes to which WAC assigns the different workloads. We implement 3 state-of-the-art elasticity controllers plus a step controller (the vanilla elasticity controller). These controllers repre-



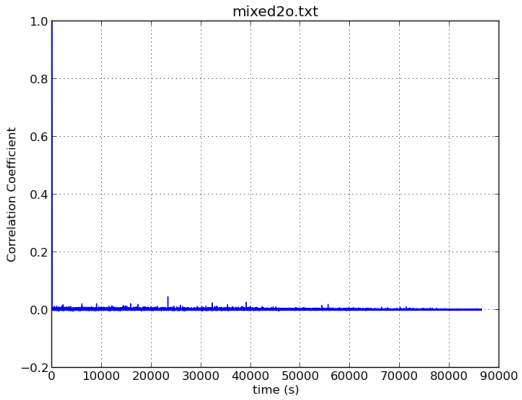
(a) Correlogram for a sample generated workload WL1.



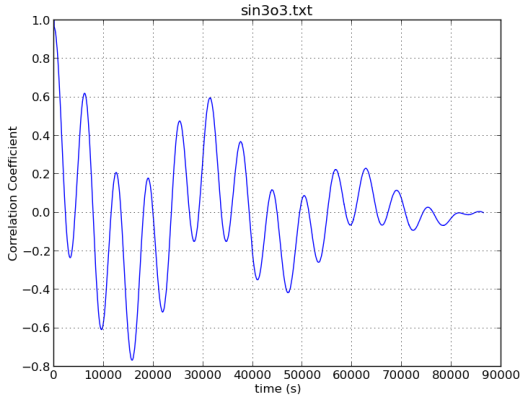
(b) Correlogram for a sample generated workload WL2.



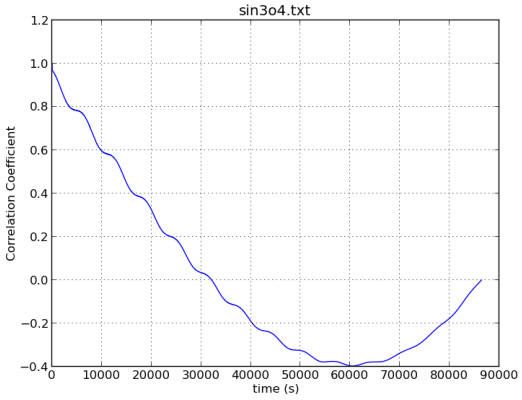
(c) Correlogram for a sample generated workload WL3.



(d) Correlogram for a sample generated workload WL4.



(e) Correlogram for a sample generated workload WL5.



(f) Correlogram for a sample generated workload WL6.

Figure 6: Correlograms of 6 of the generated workloads.

sent the classes to which WAC assigns workloads. The implemented controllers are:

1. The step controller (hereafter called *React*) is a simple reactive controller where the capacity provisioned follows the load changes when they happen and does not predict the required future capacity. If the load is constant, no VMs are added or removed. If the load increases by $\Delta Load$, the controller reacts to the load increase by provisioning S servers that can handle this load increase. Similarly if the load decreases by $\Delta Load$ which is less than a certain threshold, the controller reacts by removing the extra machines. This is similar to the controller described by Chieu et al. [40]. If the controller computes the required capacity every time the load changes, the controller never allocates more resources than required since resources are only allocated when the load increases and are removed once the load decreases below a certain threshold.

2. A Histogram based controller (thereafter called $Hist$). This controller builds histograms for the workload observed within a defined time interval (e.g. between 9:00 and 10:00 on Monday) based on the past values observed. The controller estimates the peak demand of the workload to occur within the defined time interval before the defined time interval begins. The histogram stores the history of the maximum arrival rates observed during the defined period over the past several periods. For each period there is a corresponding histogram which can be used to find the probability distribution of the arrival rate for that period. The controller can predict the capacity required based on a higher percentile of the tail, e.g., the 99^{th} percentile, of the distribution. This design allows the controller to capture seasonal trends such as diurnal or yearly trends depending on the period chosen for building the histograms. The period length chosen for the histogram is 1 hour. The controller has a correction mechanism in case the provisioned capacity is less than the actual required capacity. When the service load becomes more than the provisioned capacity can handle, the correction mechanism provisions more resources to the service. This controller was described by Urgaonkar et. al. [6].

3. A Regression based controller (thereafter called *Reg*) proposed by Iqbal et. al. [7]. The controller is a hybrid controller with a reactive controller for scale-up decisions and a predictive controller for scale-down decisions. When the capacity is less than the load, a scale up decision is taken and new VMs are added to the service in a way similar to *React*. For scale down, the predictive component uses second order regression trained using past workload values to predict future workload. The regression model is recomputed using the complete history of the workload every time a new measurement data is available. If the current load is less than the provisioned capacity, a scale down decision is taken using the regression model. If the regression predicts an increase in the number of servers, the results from the regressor are ignored.

4. An adaptive hybrid controller (thereafter called $AKTE$) that uses the workload slope to predict the future workload. This controller is proposed by Ali-Eldin et. al. [14]. The controller is a hybrid controller having

a reactive component similar to *React* and a predictive component that predicts the future workload from the rate of change of the workload. The slope calculations are adaptive depending on the rate of change of the workload. If the load is increasing or decreasing rapidly and thus the rate of change is closer to ± 1, then the rate at which the controller makes decisions is increased in order to adapt to the rapid workload changes.

4.1 How to Compare Elasticity Controllers' Performance

While most of the body of work on elasticity control uses the request response time or service latency as a measure of a controller's performance, Bodik et. al. [41] argued that this is not the correct reference signal that an elasticity control policy should use since latency have very high variance even for fixed workloads. If this is the reference signal used, the elasticity controller ends up oscillating resources due to noise in the reference signal. Smoothing the noise does not help since it leads to significant control delays.

Bodik et. al. also raise another concern for which they give an example. Suppose there are two workloads running on two different machines and each of them experiences an average service latency of 50 ms for the requests. If the required QoS is that the average service latency is kept below 100 ms, can the two loads be migrated on a single server shutting down one of the servers? This is not true since service latencies do not depend linearly on the amount of resources provisioned.

They also point out to the fact that latency does not show under-provisioning and over-provisioning levels even for non noisy response times that do not vary. This is specially true when we compare the performance of different controllers since two controllers might be achieving the required latency but one of them uses one tenth of the resources used on average by the other controller or one of them can be dropping, on average, less requests than the second one.

We add to their analysis that different applications have different processing requirements and a cloud infrastructure provider should not make any assumptions on how the resources are used, e.g., serving a static web page page hosted on the cloud is different from sorting a Tera-byte of data. In addition, any change in the workload mix changes the achievable latency [9]. Cloud computing promises resources on demand, so the measure of performance should be resource usage, e.g., average CPU and memory consumption or network utilization, or a metric that can be directly mapped to resource usage, e.g., number of requests, which is not the case for latency. These metrics enable the provider to be as generic as possible when specifying the QoS in the Service Level agreements with the customers. We choose to use the request arrival rate as the reference signal in the implementation of the different controllers.

4.1.1 Performance Comparison Metrics

To compare the performance of different controllers, we use the following cost metrics:

1. Average Overprovisioning rate ($\overline{OP}$) is the average number of overprovisioned VMs by the controller per unit time. This is calculated by summing the number of overprovisioned VMs over time and dividing the number

by the total number of time units for which the controller was running. A machine is considered overprovisioned if it is of no use for the next 10 minutes. This reduces the penalty of an algorithm that predicts the future workload well in advance. This is a configurable parameter that can be changed.

2. Average Underprovisioning rate ($\overline{UP}$) is the average number of underprovisioned VMs by the controller per unit time. This is calculated by summing the number of underprovisioned VMs over time and dividing the number by the total number of time units for which the controller was running. Underprovisioning means that the controller failed to provision the resources required to serve all requests on time.

3. Average number of Oscillations ($\overline{O}$) which is the average number of VMs started or shut-down per unit time. The reason to consider ($\overline{O}$) as an important parameter is the cost of starting/stopping a VM. From our experience, starting a machine (physical or virtual) typically takes from 1 minute up to several minutes (almost 20 minutes for an ERP application server) depending on the application running. This time does not include the time required to transfer the images and any data needed but is rather the time for machine boot-up, recontextualization of the images [42] and network setup. Similar time may be required when a machine is shutdown for workload migration and load balancer reconfiguration. In addition, for physical servers, machines consume more energy during bootup and shutdown while not doing any useful work [43].

Since different applications have different provisioning requirements, some applications running on a cloud infrastructure will give more weight to one of the three parameters. For example, a traditional database system might care more about $\overline{O}$ since each change in the number of machines results in a heavy penalty in terms of synchronization to preserve consistency. While an application that uses VM cloning [44] might care more about $\overline{UP}$.

When comparing different controllers, the values of $\overline{OP}$, $\overline{UP}$ and $\overline{O}$ are weighted in order to consider the different scenarios and combinations that an application might have. $\overline{OP}$ is multiplied by a constant α. $\overline{UP}$ is multiplied by a constant β. Similarly, $\overline{O}$ is multiplied by a constant γ. The values for these three weights should be set based on the business level objectives [45] required by the application owner and the quality of service requirements of a service.

We refer to the set of implemented controllers as C_I. To choose the best performing controller for a given workload and a given set of weights, the cost metrics have to be calculated for each implemented controller X. The total cost for using that controller is,

$$Cost_X = \alpha\overline{OP} + \beta\overline{UP} + \gamma\overline{O}. \tag{3}$$

The best controller for the given setup C_Z is thus,

$$C_Z = \{\chi | \chi \in C_I \ \& \ Cost_\chi \leq Cost_Y \ \forall \ Y \in C_I\}. \tag{4}$$

4.2 Performance of the Controllers with Various Workloads

Classification is a supervised learning algorithms, therefore, it requires training using labeled data. For each object in the labeled training set, the object's class in known. For WAC, an object is a workload and a class is one of the implemented elasticity algorithms. In order to label each workload in the training set, we need to know the best performing controller for that workload. We have built a discrete event simulator using Python, numpy and scipy [46] to simulate a cloud provider and test the four controllers and their performance on the workloads. The Regression controller was performing badly on almost all workloads for all three metrics. We have modified the design described in the original paper such that the provisioned capacity is always 1.2 times the suggested capacity by the controller output. This modification improves the performance of the controller considerably.

In our experiments, α varies between 0 to 1 for the real workloads and 0 to 0.5 for the generated workloads, with a step of 0.01, β varies between 1 to 20 for the generated workloads and 1 to 10 for the real workloads with a step of 1 and γ varies between 0 to 20 for the generated workloads and 0 to 10 for the real workloads, with a step of 0.5. The combinations of these values with the ACFs and the SampEn of the different workloads form the different scenarios with which we test WAC. For the real workloads, there are in total 280000 scenarios, while for the synthetic workloads there are in total 2090000 scenarios. Using all combinations of the three weighted metrics for both the real and generated workloads, the controller with the best performance for each combination is labeled. Table 3 shows the percentages of scenarios in which every controller outperforms the other controllers for the different workloads. From the table we see that, *Reg* and $AKTE$ outperform the other controllers considerably for both the real and the synthetic traces. *React* is the least performing controller for both workload types.

We attribute the performance of the controllers to the way they are designed. $AKTE$ is designed with the aim of reducing $\overline{UP}$ and $\overline{O}$ at the cost of higher $\overline{OP}$. It is more suitable for applications where the cost of underprovisioning and the cost of oscillations should be minimized. $Hist$ is designed for workloads that have periodical patterns and low burstiness. $Hist$ does not give any importance to $\overline{OP}$ since scale down mechanisms when the controller provisions extra resources are absent. *React* has no predictive component. It provisions resources after they are needed. This reduces the amount of $\overline{OP}$ at the cost of increased $\overline{UP}$ and $\overline{O}$. The modified *Reg* uses a second order regressor which is suitable for capturing patterns in the workload. *Reg* scales down resources faster $Hist$, thus reducing $\overline{OP}$, but slower than *React*.

5 Workload Classification

We evaluate WAC with both a KNN classifier and an SVM classifier [22]. SVM is a supervised learning method introduced by Boser et. al. [47]. It constructs separating hyperplanes to separate the training data such that the distance between the constructed hyperplanes and any training point is maximized. Training an SVM classifier and tuning its different parameters is complex [48]. We have

Table 3: Number of scenarios in which every controller outperforms the other controllers for the different workloads.

Controller	Real workloads scenarios	Generated workloads scenarios
React	6.55%	0.1%
Reg	33.72%	61.33%
AKTE	47.17%	34.3%
Hist	12.56%	4.27%

tried using an SVM classifier but the training was very time consuming and was giving low classification accuracy. This means that the classifier parameters required tuning which is even more time consuming. We choose to use KNN since it is less complex, fast to train and fast to calculate an assignment once trained.

The KNN algorithm is quite simple. A training set is used to train the classifier. The training set is a set of samples for which the correct classification in known. When a new unknown sample arrives, a majority vote of its K-nearest neighbors is taken. The new sample is then assigned to the class with a majority vote. K is a configurable parameter. The distance to a neighbor is usually the Euclidean distance between the sample and that point. There are two versions of the KNN algorithm. One version gives more weights to votes of closer neighbors while the second version gives equal weights to all K neighbors [22].

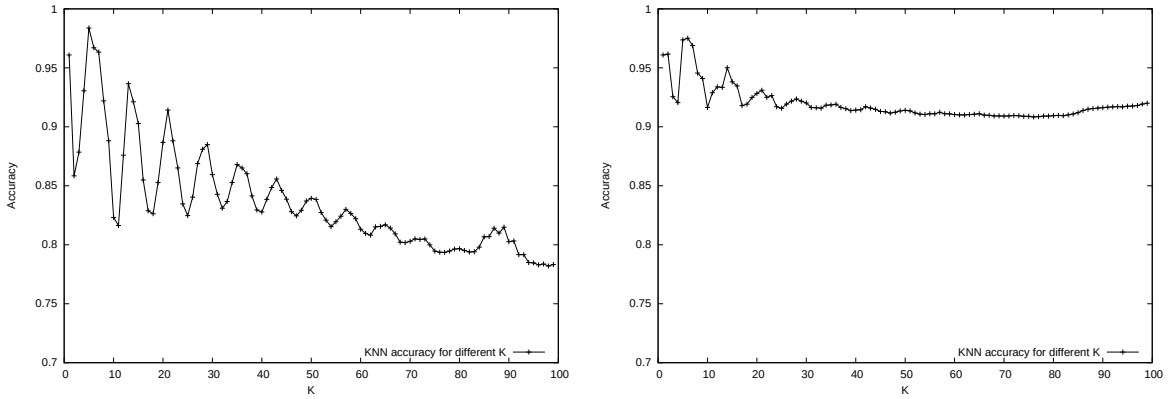
In the experiments, we consider the case when a cloud provider has four controllers deployed and the case when only two controllers, $AKTE$ and Reg, are deployed. $AKTE$ and Reg perform better than $Hist$ and *React* for more than 80% of the workload scenarios considered. Each scenario in the training set is labeled with the top performing controller out of $AKTE$ and Reg for the experiments with only two controllers.

In order to test the accuracy of KNN, the available workload scenarios are divided into a training set and a test set. The training set is used for training the classifier while the test set is used to evaluate the accuracy of the classifier after training. Both sets are labeled with the right answer for each scenario. When the classifier assigns a workload to an elasticity controller, the assignment is compared to the label. If the assignment matches the label, then the classification is correct. The accuracy of the classifier is calculated by calculating the ratio of correct classifications of the test set to the total size of the test set.

5.1 Classification of Real Workloads

In the experiments, the scenarios described in Section 4 are randomly split into a training set and a test set. The training set contains 90% of all the scenarios while the test set has the remaining 10% . Both the training set and the test set are balanced such that no single class dominates the training set thus skewing the classifier.

We test the KNN classifier using both its versions, the one that gives equal weights to all K neighbors and the one that gives more weight to closer neighbors. For each version, we repeat the experiment with K ranging from 1 to 100 and calculate the classification accuracy.



(a) Effect of changing K on the classifier accuracy when all neighbors' votes are given equal weight.

(b) Effect of changing K on the classifier accuracy when closer neighbors' votes are given higher weight.

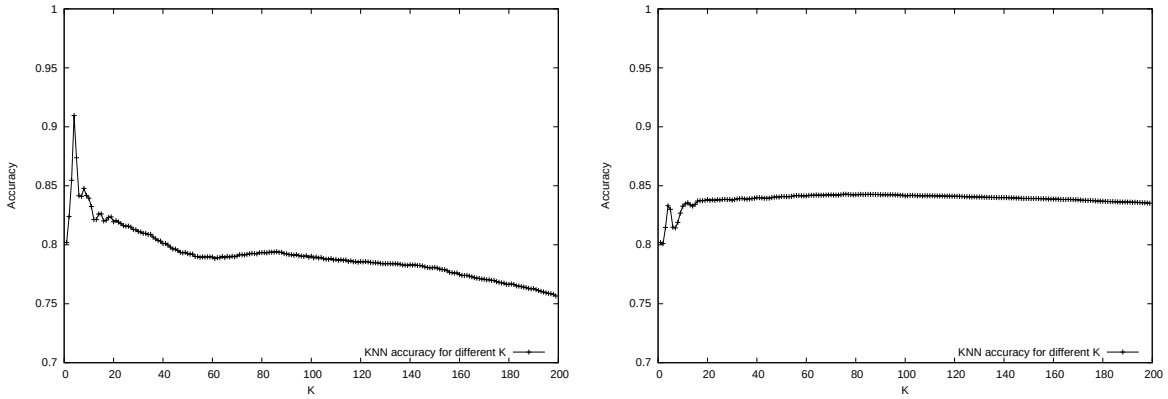
Figure 7: Effect of K on the classifiers' accuracy with only 2 classes considered, $AKTE$ and Reg, for real workloads.

Figure 7 shows the classification accuracy of the classifier when there are only two classes, $AKTE$ and Reg. The X axis of the figure is the value of K. The Y axis of the figure is the accuracy of classification. Figure 7(a) shows the accuracy of the classifier with changing K when all K neighbors' votes are given equal weights. Significant oscillations occur in the accuracy as K increases until K is around 50. When closer neighbors' votes are given higher weight, oscillations are much smaller as shown in Figure 7(b) where the accuracy stabilizes at around 0.92 when K is equal to 25. The maximum accuracy achieved is 0.983 using equal weights for all neighbors for $K = 5$.

Figure 8 shows the classification accuracy of the classifier when there are four classes, $AKTE$, $React$, $Histo$ and Reg, against K when K is varied from 1 to 200. Figures 8(a) and 8(b), show less oscillations compared to Figure 7. The maximum accuracy achieved is 0.91 using equal weights for all neighbors for $K = 4$. The accuracy of classification is reduced by more than 7% when more controllers are available. This might be due to not having enough training data in the training set for the classifier to be able to generalize for four classes. An accuracy of 91% is still considered good. For the service provider it means that instead of having just one controller, the provider can have 4 controllers with only 9% of the workloads assigned in a non optimal way. Since more training data, means more experience, the provider can still decrease the percentage of workloads assigned sup-optimally by constantly adding feedback to the classifier by adding more training points based on new workload behaviors seen. Figures 7(a) and 8(a) show that accuracy decreases with increasing K when all K neighbors' votes are given equal weights. When neighbors' votes are weighted such that closer neighbors have a higher weight, the accuracy stabilizes as K increases as shown in Figures 7(b) and 8(b).

5.2 Classification of Generated Workloads

We repeat the tests described above using only the generated workloads. Figure 9 shows the classification accuracy of the classifier when there are only two



(a) Effect of changing K on the classifier accuracy when all neighbors' votes are given equal weight.

(b) Effect of changing K on the classifier accuracy when closer neighbors' votes are given higher weight.

Figure 8: Effect of K on the classifiers' accuracy with 4 classes considered, mapping to the four implemented controllers, for real workloads.

classes, $AKTE$ and Reg, against K. The maximum accuracy achieved is 0.92 using equal weights for all neighbors for $K = 3$. Figure 10 shows the classification accuracy of the classifier when there are four classes, $AKTE$, $React$, $Histo$ and Reg, against K. The maximum accuracy achieved is 0.94 using equal weights for all neighbors for $K = 3$.

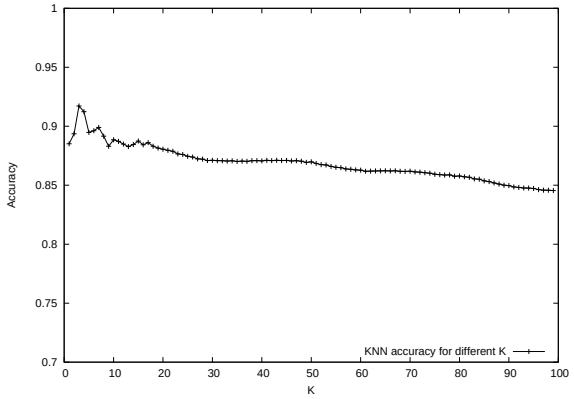
Comparing Figure 7(a) to figures 9 and 10, oscillations in the classifier's accuracy with increasing K is much lower when classifying synthetic workloads compared to when classifying real workloads.

5.3 Classification of Mixed Workloads

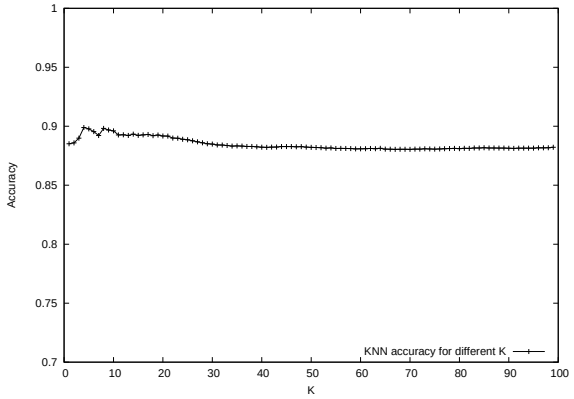
In our last experiment, we combine the scenarios using real and generated workloads in a single set. This set is again randomly shuffled and divided into a training set and a test set. Figure 11 shows the classification accuracy of the classifier when there are only two classes, $AKTE$ and Reg, against K. The maximum accuracy achieved is 0.92 using any of the two versions of the KNN for $K = 5$. Figure 12 shows the classification accuracy of the classifier when there are four classes, $AKTE$, $React$, $Histo$ and Reg, against K. The maximum accuracy achieved is 0.92 using equal weights for all neighbors for $K = 3$. Again, we note that the oscillations in the classifier's accuracy with changing K in figures 11 and 12 are not as severe as in the case of classifying real workloads.

5.4 Discussion of the Results

The oscillations seen in the figures for lower values of K can be attributed to the nature of the KNN algorithm. Let F be the number of features used for classification. KNN constructs an F-dimensional space and places each training point in that space according to the point's features values. It divides the F-dimensional space into neighborhoods where each neighborhood is a group of points neighboring each other and having the same class. These neighborhoods can be scattered in the F-dimensional space. Some of these neighborhoods can

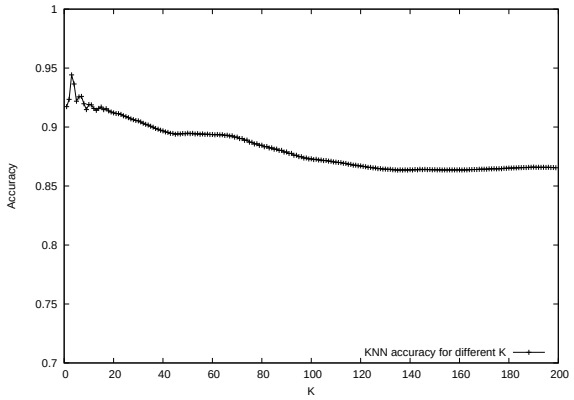


(a) Effect of changing K on the classifier accuracy when all neighbors' votes are given equal weight.

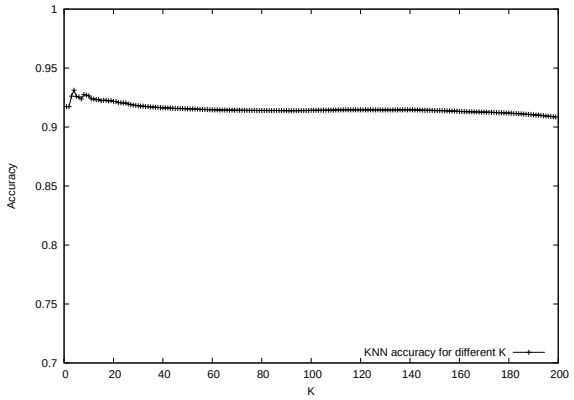


(b) Effect of changing K on the classifier accuracy when closer neighbors' votes are given higher weight.

Figure 9: Effect of K on the classifiers' accuracy with only 2 classes considered, $AKTE$ and Reg, for generated workloads.

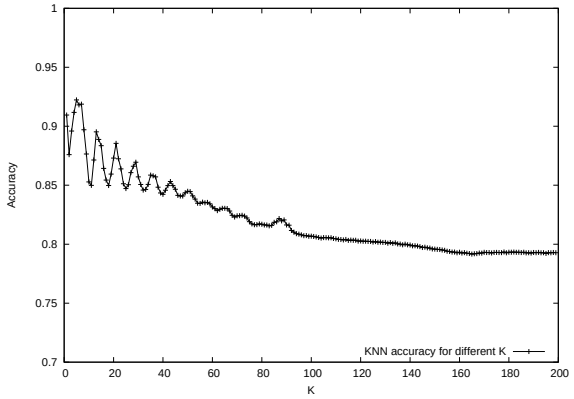


(a) Effect of changing K on the classifier accuracy when all neighbors' votes are given equal weight.

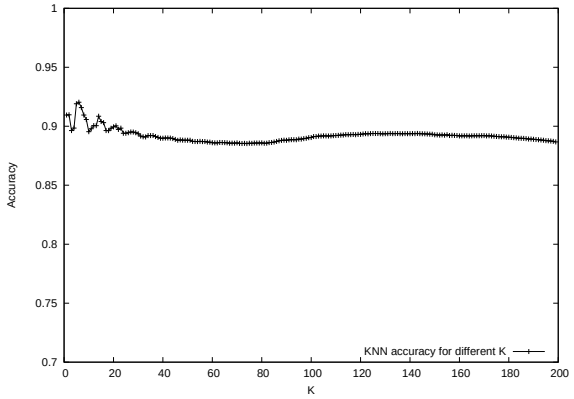


(b) Effect of changing K on the classifier accuracy when closer neighbors' votes are given higher weight.

Figure 10: Effect of K on the classifiers' accuracy with 4 classes considered, mapping to the four implemented controllers, for generated workloads.

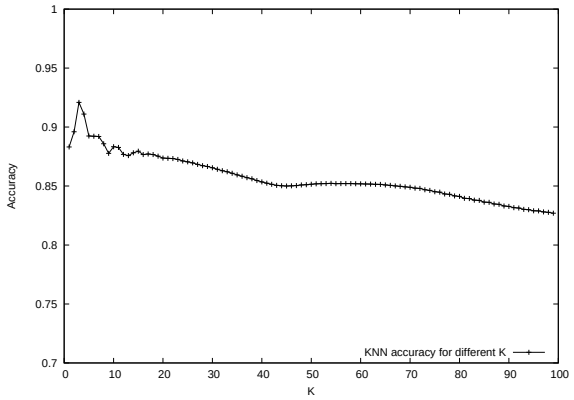


(a) Effect of changing K on the classifier accuracy when all neighbors' votes are given equal weight.

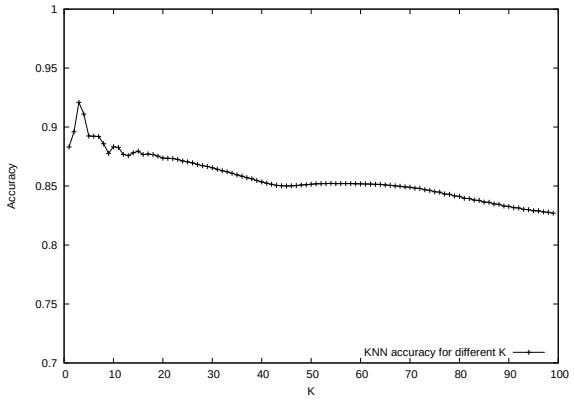


(b) Effect of changing K on the classifier accuracy when closer neighbors' votes are given higher weight.

Figure 11: Effect of K on the classifiers' accuracy with only 2 classes considered, $AKTE$ and Reg, for a mixed set.



(a) Effect of changing K on the classifier accuracy when all neighbors' votes are given equal weight.



(b) Effect of changing K on the classifier accuracy when closer neighbors' votes are given higher weight.

Figure 12: Effect of K on the classifiers' accuracy with 4 classes considered, mapping to the four implemented controllers, for a mixed set.

be small and surrounded completely by a larger neighborhood of a different class. Picking a small value for K results in correctly classifying these smaller neighborhoods for the training data. On the other hand, it can result in overfitting the training data and forming nasty decision boundaries, e.g., a small neighborhood laying completely surrounded within a large neighborhood of a different class. This makes the classifier more sensitive to noise in the training set. On the other hand, a large value of K results in a less noise sensitive classifier and smoother decision boundaries but at the cost of increased training data error, increased computational burden and loss of locality of the estimations since far away points affect the decision. risk of making the classifier more biased towards the class with more training points and having the neighborhood boundaries oversimplified.

Oscillations are more obvious when the distance to all K neighbors have equal weights, When closer neighbors are given more weight, the classification accuracy is more stable with increasing K. By looking at figures 7–11, it can be seen that for all of them, the accuracy of classification decreases with the increasing K for a while and then tends to stabilize for a range of K and then sometimes starts to decrease again. The stabilization for the KNN version with weighted distance is faster and with a stable accuracy higher than the other version.

When setting K in real life deployments, a cloud provider should either choose K with the best accuracy regardless of stability or least K with highest stable accuracy depending on the confidence in the training set. If the training set is comprehensive with most workload characteristics expected, K can be chosen to be the one that gives best accuracy on the training set regardless of the stability. Otherwise, if the training set is not comprehensive K should be chosen to be the least K with the highest stable accuracy.

6 Related Work

Previous work on workload characterization has mainly focused on identifying the different properties of different workload traces collected from running systems. Calzarossa and Serazzi [49] discuss the steps for modeling a workload. As a first step, the characterization level, e.g., the sampling rate of the workload, the basic components, the parameters that describe them and the criteria for evaluating the model accuracy of the workload has to be identified. Then, the system is monitored to log the workload. The measured data is statistically analyzed to know if there are any natural partitions in the workload, e.g., long running requests and short running requests. Statistical analysis involves removal of any outliers having one or more parameters with values greater than a certain percentile, workload sampling to be able to process the recorded trace within reasonable time, static analysis which partitions the workload using a clustering algorithm such as K-means and dynamic analysis using techniques such as time series analysis. The authors discuss a large number of workload modeling studies spanning different systems, namely, centralized batch and interactive systems, centralized database systems, network based systems, and multiprocessor systems such as parallel systems and supercomputers.

Downey and Feitelson [23] discuss the problems in generalizing a model based on data from specific observations to an abstract model of the workload. They

start by giving some examples on the problems and the difficulties faced when creating a multi-level model for a workload. They focus on the problems of single-level models.

The aim of modeling is to find the summary statistics of the different workloads. These fall into three families:

1. Moment-based: e.g., mean, variance and the coefficient of variance, skew (3rd moment) and kurtosis (4th moment).

2. Percentile-based, e.g., median, quartiles and the semi-interquartile range (Measure of Dispersion)..

3. Mode-based: e.g., the most common value observed for distributions having discrete components.

According to the authors, reporting the moments for non-symmetric distributions is misleading and might be meaningless since moments are not robust to outliers. In addition, for some distributions such as the Pareto distribution, the moment of a sample does not converge on the moments of the population. One of the warning signs of non convergence is the failure to achieve sample invariance. To test whether a hypothesis about a distribution is right or not, the Chi-square test or the Kolmogorov-Smirnov test (for continuous distributions) can be used [50]. These tests usually fail for workload characterization since they assume that the testing data should converge to the distribution when a large number of points are used. This is not true for most workloads.

The authors discuss the importance of weighing data points. They give an example with the characterization of parallel jobs when it is assumed that all jobs are of the same weight, when the jobs are weighed by the duration and when the jobs are weighed by the area (duration and the number of cores used). They discuss the correlation between these different attributes and the different ways to include them in the model.

Arlit and Jin presented a detailed workload study of the 1998 FIFA World Cup website [37]. The site logs from May 1st, 1998 until July 23rd, 1998 were analyzed. The logs contain 1.35 Billion requests and almost 5 TB of data were sent to the clients. Data was accessed by more than 2 million unique IP addresses.

Kang et. al. crawled Yahoo! videos website for 46 days [51]. They recorded data for 9986 unique videos in total with video durations ranging from 2 to 7518 seconds. Around 76% of the videos are shorter than 5 minutes and almost 92% are shorter than 10 minutes. They discuss the predictability of the arrival rate with different time granularities. A load spike typically lasted for no more than 1 h and the load spikes are dispersed widely along the time making them hard to predict. They also find that the workload is highly correlated in short term. While the cost could be high if a video site operator over-provisioned the data center based on the over-provisioning factor at a small time scale, these load spikes can be provisioned on a cloud on demand to service the extra load while not over-provisioning the Video site datacenter.

Chen. et. al. analyze a short workload trace that was publicly realized by Google [52]. The trace is 6 hours and 15 minutes long with data collected every 5 minutes, i.e., having just 75 points. The short workload makes it hard to infer the true characteristics of the jobs running on Google's backend cluster.

Mishra et. al. take a first step towards modeling the workload running on Google's backend servers by grouping resources with similar task consumptions in classes (task classification) [53]. They define a multidimensional representation of the resources (task shape) used by the tasks using the average memory in GB and the average number of cores used by a task every five minutes. During scheduling, tasks are assigned such that their shape (time in seconds, memory usage and number of cores) fits the free space on the assigned machine.

They discuss the problems with using a coarse grained classification and explain their approach for task classification. First, they identify the workload dimensions, e.g., task duration, average core usage and average memory usage. Second, they use k-means clustering algorithms to construct preliminary task classes. Third step is to manually identify the break points for the qualitative coordinates of the workload dimensions. Finally, they merge classes to form the final set of task classes which define the workload. They use this approach to identify the load mix of the tasks.

This work is extended by Zhang et. al. [54]. The authors suggest to model task usage shapes by modeling the mean values of run-time tasks resource consumption. They call this model "the mean usage model of tasks usage shapes". This simple model captures the characteristics of the workload running on Google's backend cluster to a high extent. They attribute these results to the low variability of task resource usage in the workload and the characteristics of evaluation metrics.

The authors also analyze six month of Map-Reduce traces from a Facebook cluster and a two weeks Map-Reduce trace from another Internet company [55]. They built a Map-Reduce workload synthesizer that analyzes a Map-Reduce workload and uses the results to produce a synthetic but realistic workload that mimics the original seed workload.

A longer trace provided by Google that spans 29 days and around 650000 jobs [13] was analyzed by Reiss et. al. [56]. While the system is over-booked, the actual CPU and memory utilization is rarely higher than 50% and never exceeds about 60%. The workload is dominated by 'normal production' tasks which have regular patterns and a daily peak to mean ratio of 1.3. Lower priority tasks make up between 10% to 20% of the CPU usage in the cluster. These tasks have no regular patterns and are very bursty. Over 80% of the total utilization of the cluster comes from long running jobs which constitute 2% of the total number of jobs.

Iosup and Epema analyze 15 different grid workloads [57]. They report that the average system utilization of some research grids was as low as 10 to 15% but considerably high on production grids. All 15 grids experienced overloads during some short term periods. A few users dominate the workloads. In most workloads, there is a little intra-job parallelism and even parallel jobs running have low parallelism. They note that loosely coupled jobs are dominating most of the workloads. Loosely coupled jobs are suitable to run on the cloud [58]. Similar findings are reported by Zhang et. al. [54] and the Magellan project [58].

Khan et. al. try to discover from the workload running on different servers which servers frequently have correlated workload patterns. They use a greedy algorithm to solve a computationally intractable problem to find the clusters that constitutes correlated workloads on different servers. Their proposed solution is slow. It takes more than 6 hours to train the algorithm using data collected every 15 minutes for 17 days and doing predictions every 15 minutes.

Viswanatha et al [59], introduce a provisioning system for private clouds that uses an application's resource usage pattern to place applications on servers with appropriate level of configuration capabilities. The authors use the coefficient of variation to identify unstable workloads. Unstable workloads requires placement on clusters that allow dynamic placement. They base their placement decisions on the different elastic capabilities of the clusters and the expected peak workload for each application.

There has been a lot of interest in data stream classification and clustering [60, 61, 62, 19]. Beringer and Hullermeier modify the K-means clustering algorithm to be able to cluster online data streams efficiently. Their work is considered the first work to try to cluster the data streams themselves, compared to for example the work by Guha et. al. [62] who cluster the elements of the stream. Due to their complexity, the computational costs of clustering data streams is high. In addition, the algorithm has to be adaptive in the sense that new clusters may be created at any time when new data points arrive. Similarity between data streams is calculated using the Euclidean distance between their normalization for a given sliding time window.

Keogh and Kasetty discuss the quality of the published work on time-series data mining [63]. They show that much of the published work is of very little utility. They implement the contributions of more than two dozen papers and test them on 50 real workloads. For classification, they implement 11 published similarity measures and compare their performance to the Euclidean distance measure. Non of the 11 proposed similarity measures is able to outperform the Euclidean distance. They show that the error rate for some of the published work was 0.695, that is 69.5% of all classifications was wrong.

Herbst et. al. [64] propose a novel forecasting methodology that self-adaptively correlates workload characteristics classes and existing time series based forecasting approaches. The authors smooth the noise in a times-series before forecasting. They use a decision tree to assign the smoothed time-series to one of several forecasting methods based on time-series analysis. They show that their approach can be used for resource provisioning.

Our work complements the previous studies and uses their finding to build an automated workload analyzer and classifier tool that can identify the classes of the different workloads running on the cloud. Although our main focus is elasticity control, the tool also be used for placement of orthogonal workloads (e.g. computationally intensive workloads and I/O intensive workloads). The tool can also provide admission controllers with insights that can be used when admitting a workload.

7 Conclusion and Future work

In this work, we introduce WAC, a Workload Analysis and Classification tool for assigning workloads to different elasticity controllers in order to improve workload predictions. The tool uses autocorrelation and sample entropy to analyze the workload periodicity and burstiness respectively. This analysis is used for classifying workloads and assigning them to the most suitable elasticity controller for each of them. In addition, the tool allows assignment of an elasticity controller to a workload based on specific quality of service objectives. We have collected a set of 14 real workloads and generated a set of 55 synthetic workloads

to test the tool. We have implemented 4 state-of-the-art elasticity controllers and a step controller. The tool was used to assign the workloads with different quality of service requirements to the implemented controllers in different experiments. The tool shows an accuracy of classification ranging from 91% up to 98% in different experiments.

Acknowledgment

The authors would like to thank the Advanced School for Computing and Imaging at Vrije Universiteit, Amestrdam for providing the DAS workloads, the Grid'5000 team for providing the Grid'5000 traces, the AuverGrid team for providing the AuverGrid traces, John Morton and Clayton Chrusch for providing the SHARCNET traces, the e-Science Group of HEP at Imperial College London for providing the LPC traces and the NorduGrid team for providing the NorduGrid traces. We would also like to thank the Grid Workloads Archive for providing access to these traces. Financial support has been provided in part by the European Community's Seventh Framework Programme under grant agreement #257115, the Lund Center for Control of Complex Engineering Systems, the Swedish Government's strategic effort eSSENCE and the Swedish Research Council (VR) under contract number C0590801 for the project Cloud Control.

References

[1] Amazon Elastic Compute Cloud (amazon ec2). [Online]. Available: https://aws.amazon.com/solutions/case-studies/

[2] A. J. Ferrer, F. Hernndez, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S. K. Nair, G. Kousiouris, K. Konstanteli, T. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, N. Forg, T. Sharif, and C. Sheridan, "Optimis: A holistic approach to cloud service provisioning," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 66 – 77, 2012.

[3] J. Garside. (2013, January) Amazon's record $21bn christmas sales push shares to new high. [Online]. Available: http://www.guardian.co.uk/technology/2013/jan/30/amazon-christmas-record-sales-profits

[4] P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "Characterizing, modeling, and generating workload spikes for stateful services," in *Proceedings of the 1st ACM symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 241–252. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807166

[5] H. C. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage," in *Proceedings of the 7th international conference on Autonomic computing*. ACM, 2010, pp. 1–10.

[6] 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)*, vol. 3, no. 1, p. 1, 2008.

[7] 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*, vol. 27, no. 6, pp. 871–879, 2011.

[8] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma, "Towards autonomic workload provisioning for enterprise grids and clouds," in *10th IEEE/ACM International Conference on Grid Computing, 2009.* IEEE, 2009, pp. 50–57.

[9] R. Singh, U. Sharma, E. Cecchet, and P. Shenoy, "Autonomic mix-aware provisioning for non-stationary data center workloads," in *Proceedings of the 7th international conference on Autonomic computing.* ACM, 2010, pp. 21–30.

[10] M. Arlitt and T. Jin. (1998, August) "1998 world cup web site access logs". [Online]. Available: http://www.acm.org/sigcomm/ITA/

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

[12] G. Urdaneta, G. Pierre, and M. Van Steen, "Wikipedia workload analysis for decentralized hosting," *Computer Networks*, vol. 53, no. 11, pp. 1830–1845, 2009.

[13] J. Wilkes. (2011, November) More google cluster data. [Online]. Available: http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html

[14] A. Ali-Eldin, M. Kihl, J. Tordsson, and E. Elmroth, "Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control," in *Proceedings of the 3rd workshop on Scientific Cloud Computing.* ACM, 2012, pp. 31–40.

[15] M. Morari and E. Zafiriou, *Robust process control.* Morari, 1989.

[16] B. Abraham and J. Ledolter, *Statistical methods for forecasting.* Wiley, 2009, vol. 234.

[17] M. Morari, "Robust stability of systems with integral control," *IEEE Transactions on Automatic Control*, vol. 30, no. 6, pp. 574–577, 1985.

[18] D. Barber, *Bayesian reasoning and machine learning.* Cambridge University Press, 2012.

[19] C. Aggarwal, J. Han, J. Wang, and P. Yu, "On demand classification of data streams," in *Proceedings of the tenth ACM SIGKDD International conference on Knowledge Discovery and Data mining.* ACM, 2004, pp. 503–508.

[20] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 155–162, 2012.

[21] S. Meng, L. Liu, and V. Soundararajan, "Tide: Achieving self-scaling in virtualized datacenter management middleware," in *Proceedings of the 11th International Middleware Conference, Industrial track*. ACM, 2010, pp. 17–22.

[22] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin, "The elements of statistical learning: Data mining, inference and prediction," *The Mathematical Intelligencer*, vol. 27, no. 2, pp. 83–85, 2005.

[23] A. Downey and D. Feitelson, "The elusive goal of workload characterization," *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 4, pp. 14–29, 1999.

[24] B. Girod, R. Rabenstein, and A. Stenger, *Signals and systems*. John Wiley & Sons Inc, 2001.

[25] R. Takano, Y. Kodama, T. Kudoh, M. Matsuda, F. Okazaki, and Y. Ishikawa, "Realtime burstiness measurement," in *4th Intl. Workshop on Protocols for Fast Long-Distance Networks (PFLDnet2006)*, 2006.

[26] R. Gusella, "Characterizing the variability of arrival processes with indexes of dispersion," *Selected Areas in Communications, IEEE Journal on*, vol. 9, no. 2, pp. 203–211, 1991.

[27] T. N. Minh, L. Wolters, and D. Epema, "A realistic integrated model of parallel system workloads," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*. IEEE, 2010, pp. 464–473.

[28] J. S. Richman and J. R. Moorman, "Physiological time-series analysis using approximate entropy and sample entropy," *American Journal of Physiology-Heart and Circulatory Physiology*, vol. 278, no. 6, pp. H2039–H2049, 2000.

[29] J. S. Richman, D. E. Lake, and J. R. Moorman, "Sample entropy," *Methods in enzymology*, vol. 384, pp. 172–184, 2004.

[30] M. Aboy, D. Cuesta-Frau, D. Austin, and P. Micó-Tormos, "Characterization of sample entropy in the context of biomedical signal analysis," in *29th Annual International Conference of the IEEE, Engineering in Medicine and Biology Society, 2007. EMBS 2007.* IEEE, 2007, pp. 5942–5945.

[31] The Rackspace Cloud. [Online]. Available: http://www.rackspace.com/cloud

[32] CycleComputing. (2011, September) New CycleCloud HPC Cluster Is a Triple Threat. [Online]. Available: http://blog.cyclecomputing.com/2011/09/new-cyclecloud-cluster-is-a-triple-threat-30000-cores-massive-spot-instances-grill-chef-monitoring-g.html

[33] Amazon Web Services. (2010, July) High performance computing (HPC) on AWS. [Online]. Available: http://aws.amazon.com/hpc-applications/

[34] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H. Epema, "The grid workloads archive," *Future Generation Computer Systems*, vol. 24, no. 7, pp. 672–686, 2008.

[35] IRCache. Access to trace files. [Online]. Available: http://www.ircache.net/

[36] NCBI. PubMed. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/

[37] M. Arlitt and T. Jin, "A workload characterization study of the 1998 world cup web site," *Network, IEEE*, vol. 14, no. 3, pp. 30–37, 2000.

[38] A. Rubin, *Statistics for evidence-based practice and evaluation*. Brooks/Cole, 2012.

[39] M. D. Costa, C.-K. Peng, and A. L. Goldberger, "Multiscale analysis of heart rate dynamics: Entropy and time irreversibility measures," *Cardiovascular Engineering*, vol. 8, no. 2, pp. 88–93, 2008.

[40] T. Chieu, A. Mohindra, A. Karve, and A. Segal, "Dynamic scaling of web applications in a virtualized cloud computing environment," in *IEEE International Conference on e-Business Engineering, 2009. ICEBE '09.*, oct. 2009, pp. 281 –286.

[41] P. Bodık, "Automating datacenter operations using machine learning," Ph.D. dissertation, University of California, 2010.

[42] D. Armstrong, D. Espling, J. Tordsson, K. Djemame, and E. Elmroth, "Runtime virtual machine recontextualization for clouds," in *Proceedings of the 18th international conference on Parallel processing workshops*, ser. Euro-Par'12. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 567–576.

[43] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1. ACM, 2005, pp. 303–314.

[44] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. De Lara, M. Brudno, and M. Satyanarayanan, "Snowflock: Rapid virtual machine cloning for cloud computing," in *Proceedings of the 4th ACM European conference on Computer systems*. ACM, 2009, pp. 1–12.

[45] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[46] T. E. Oliphant, "Python for scientific computing," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 10–20, 2007.

[47] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 144–152.

[48] G. Bakır, L. Bottou, and J. Weston, "Breaking SVM complexity with cross training," *Advances in neural information processing systems*, vol. 17, pp. 81–88, 2005.

[49] M. Calzarossa and G. Serazzi, "Workload characterization: A survey," *Proceedings of the IEEE*, vol. 81, no. 8, pp. 1136–1150, 1993.

[50] H. W. Lilliefors, "On the Kolmogorov-Smirnov test for normality with mean and variance unknown," *Journal of the American Statistical Association*, vol. 62, no. 318, pp. 399–402, 1967.

[51] X. Kang, H. Zhang, G. Jiang, H. Chen, X. Meng, and K. Yoshihira, "Understanding internet video sharing site workload: A view from data center design," *Journal of Visual Communication and Image Representation*, vol. 21, no. 2, pp. 129–138, 2010.

[52] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "Analysis and lessons from a publicly available Google cluster trace," *University of California, Berkeley, CA, Tech. Rep*, 2010.

[53] A. Mishra, J. Hellerstein, W. Cirne, and C. Das, "Towards characterizing cloud backend workloads: Insights from Google compute clusters," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 4, pp. 34–41, 2010.

[54] Q. Zhang, J. Hellerstein, and R. Boutaba, "Characterizing task usage shapes in Googles compute clusters," 2011.

[55] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "Towards understanding cloud performance tradeoffs using statistical workload analysis and replay," *University of California at Berkeley, Technical Report No. UCB/EECS-2010-81*, 2010.

[56] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the Third ACM Symposium on Cloud Computing*, 2012, pp. 7:1–7:13.

[57] A. Iosup and D. Epema, "Grid computing workloads," *Internet Computing, IEEE*, vol. 15, no. 2, pp. 19–26, 2011.

[58] K. Yelick, S. Coghlan, B. Draney, and R. Canon, "The Magellan report on cloud computing for science," Technical report, US Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR), Tech. Rep., 2011.

[59] B. Viswanathan, A. Verma, and S. Dutta, "Cloudmap: Workload-aware placement in private heterogeneous clouds," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, 2012, pp. 9–16.

[60] J. Beringer and E. Hüllermeier, "Online clustering of parallel data streams," *Data & Knowledge Engineering*, vol. 58, no. 2, pp. 180–204, 2006.

[61] P. Rodrigues, J. Gama, and J. Pedroso, "Hierarchical clustering of time-series data streams," *IEEE Transactions on Knowledge and Data Engineering,*, vol. 20, no. 5, pp. 615–627, 2008.

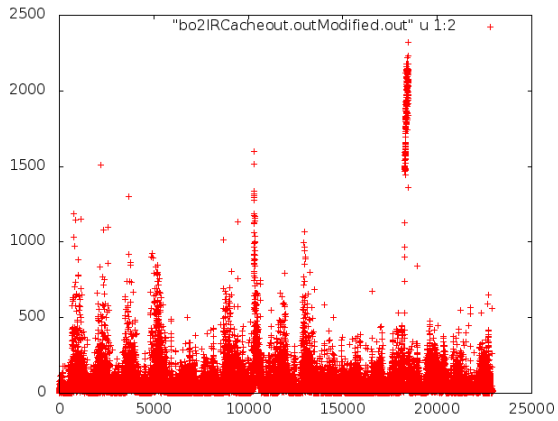
[62] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams: Theory and practice," *IEEE Transactions on Knowledge and Data Engineering,*, vol. 15, no. 3, pp. 515–528, 2003.

[63] E. Keogh and S. Kasetty, "On the need for time series data mining benchmarks: A survey and empirical demonstration," in *Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data mining*. ACM, 2002, pp. 102–111.

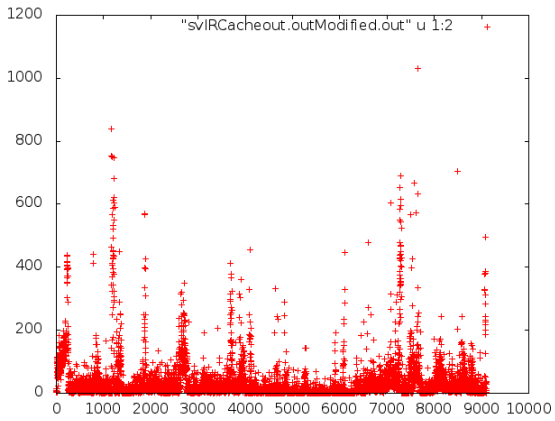
[64] N. R. Herbst, N. Huber, S. Kounev, and E. Amrehn, "Self-adaptive workload classification and forecasting for proactive resource provisioning," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '13. New York, NY, USA: ACM, 2013, pp. 187–198. [Online]. Available: http://doi.acm.org/10.1145/2479871.2479899

A Graphs for the Real Workloads

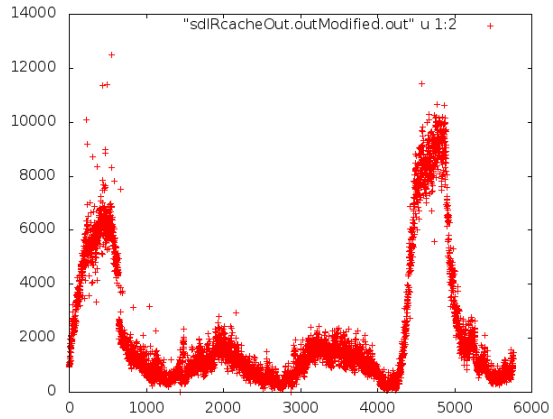
Figures 13, 14 and 15 show the plots of the real workload traces.



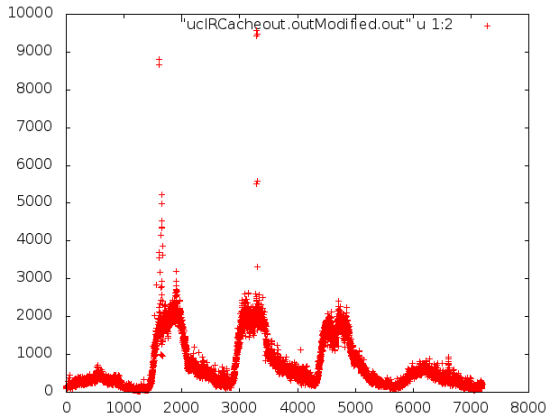
(a) Workload trace for IRCache service runnings ar at Boulder, Colorado (Bo).



(b) Workload trace for IRCache service running at Silicon Valley, California (SV).

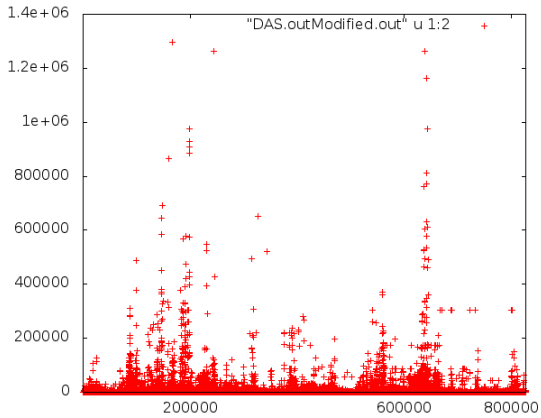


(c) Workload trace for IRCache service running at San Diego, California (SD).

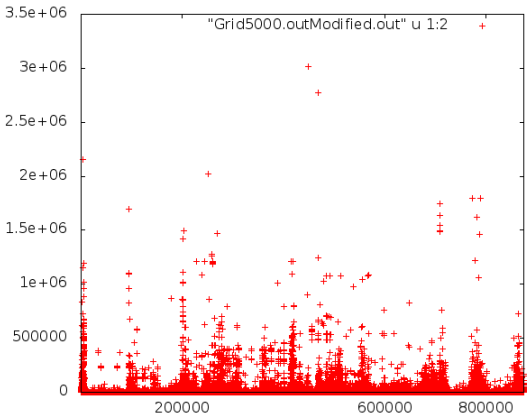


(d) Workload trace for IRCache service running at Urbana-Champaign, Illinois (UC).

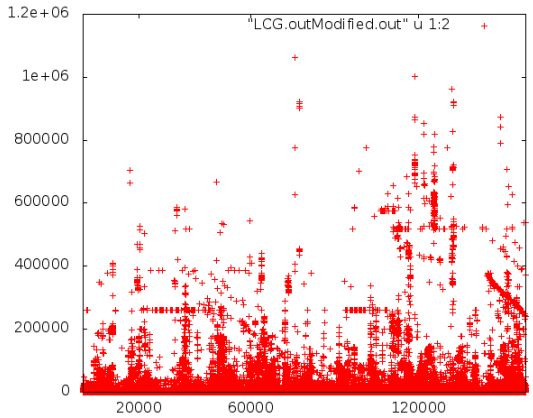
Figure 13: Workload traces of the different Caching Services.



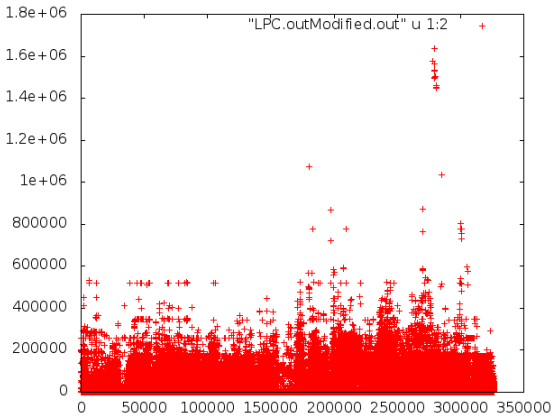
(a) Workload trace for the DAS workload.



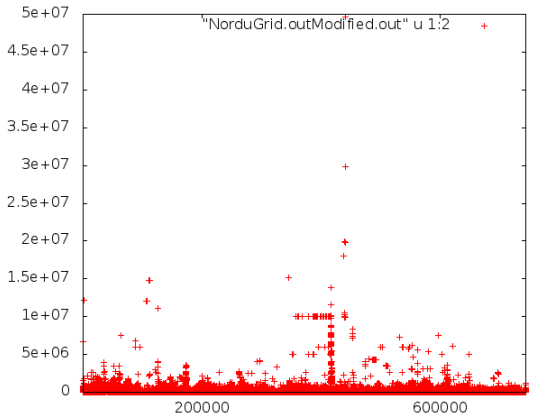
(b) Workload trace for Grid5000 workload.



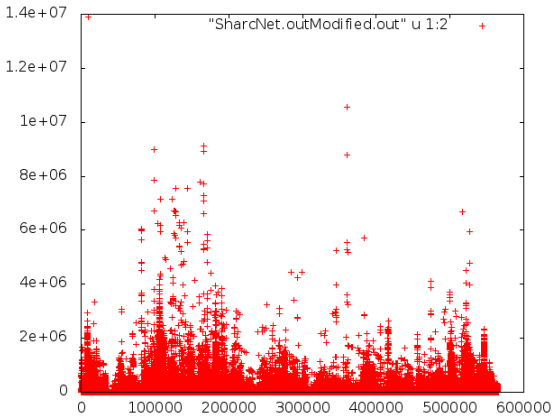
(c) Workload trace for the LCG workload.



(d) Workload trace for LPC workload.

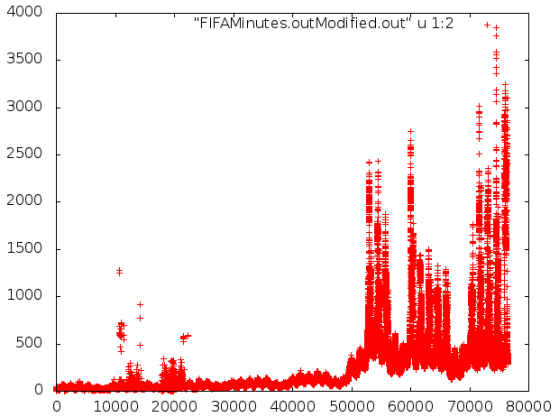


(e) Workload trace for the NorduGrid workload.

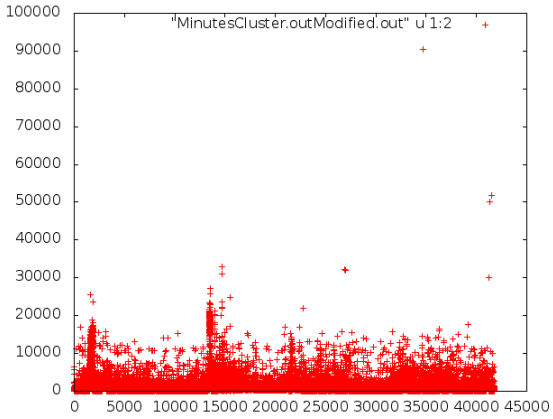


(f) Workload trace for SharcNet workload.

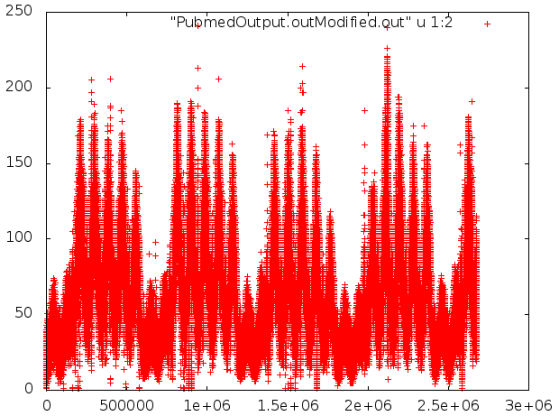
Figure 14: Workload traces of the Grid workloads analyzed.



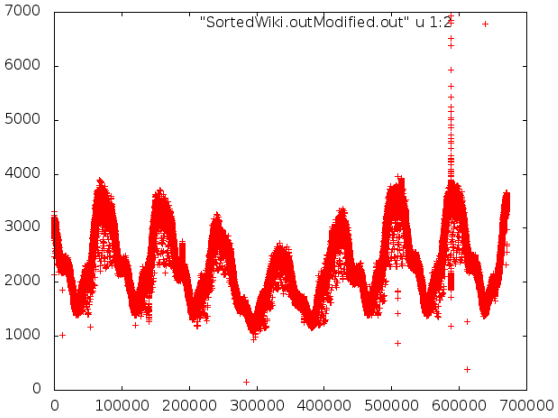
(a) Workload trace for the world cup workload.



(b) Workload trace for the Google Cluster workload.



(c) Workload trace for PubMed access traces.



(d) Workload trace for the Wikipedia workload.

Figure 15: Workload traces of web-hosting workloads and the Google cluster workload analyzed.