Measuring Cloud Workload Burstiness

Ahmed Ali-Eldin*, Oleg Seleznjev†, Sara Sjöstedt-de Luna†, Johan Tordsson*, Erik Elmroth*
*Department of Computing Science, Umeå University
{ahmeda, tordsson, elmroth}@cs.umu.se
† Department of Mathematics and Mathematical Statistics, Umeå University
{oleg.seleznjev, sara}@math.umu.se

Abstract

Workload burstiness and spikes are among the main reasons for service disruptions and decrease in the Quality-of-Service (QoS) of online services. They are hurdles that complicate autonomic resource management of datacenters. In this paper, we review the state-of-the-art in online identification of workload spikes and quantifying burstiness. The applicability of some of the proposed techniques is examined for Cloud systems where various workloads are co-hosted on the same platform. We discuss Sample Entropy (*SampEn*), a measure used in biomedical signal analysis, as a potential measure for burstiness. A modification to the original measure is introduced to make it more suitable for Cloud workloads.

I. Introduction

Workload spikes and burstiness decrease online applications' performance and lead to reduced QoS and service disruptions [6], [34]. We define a workload spike (sometimes referred to as a burst or a flash crowd [5]) to be a sudden increase in the demand on an object(s) hosted on an online server(s) due to an increase in the number of requests and/or a change in the request-type mix [29].

Some spikes occur due to a non-predictable event in time with non-predictable load volumes while others occur due to a planned event but with non-predictable load volumes. Figure 1 shows the load on Michael Jackson's English Wikipedia page during the period around his death. Two significant spikes can be seen in the figure. The first spike occurred right after his death with the load on the page increasing by three orders of magnitude. The second visible spike occurred 12 days after his death during his memorial service. Before the memorial service, the load on the page was still higher than normal by one order of magnitude. The memorial service resulted in another increase, one order of magnitude larger than the load before the service. While it is impossible to predict the first spike, it is important to detect and mitigate against the spike with minimal reduction in the QoS. The second spike on the other hand can be mitigated against before its occurrence since it is a planned event.

A *bursty* workload is a workload having a significant number of spikes. The spikes make it harder to predict the future value of the load. An example of a workload that exhibits little burstiness is shown in Figure 2(a) for 10% of all user requests issued to Wikipedia [30]. Most of the variations in the load are due to the diurnal variations in the usage of Wikipedia. Figure 2(c) shows the load on IR-Cache servers [18] deployed in San Diego (SD-IRCache) and Figure 2(d) shows the load on the servers of the FIFA 1998 world cup during the last two

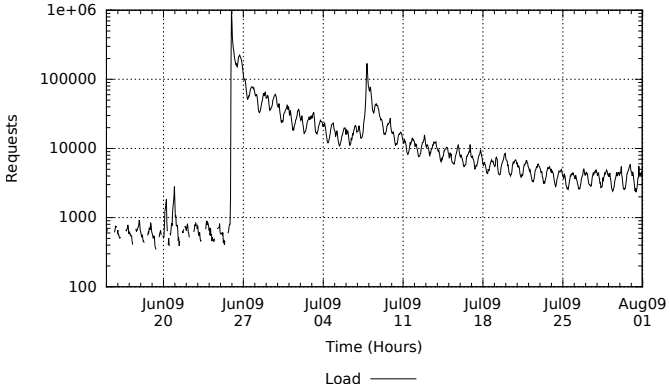


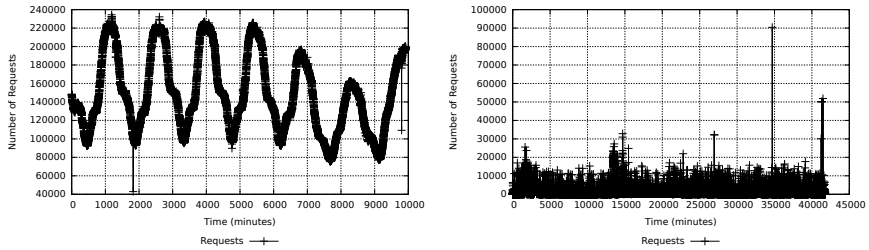
Fig. 1. Spike on the Michael Jackson Wikipedia entry after his death.

weeks before the games ended, with both workloads having moderate burstiness. These bursts are either due to planned events with hard to predict magnitude or just increased service usage. Figure 2(b) shows strong burstiness in the number of tasks submitted per minute to a Google cluster [36].

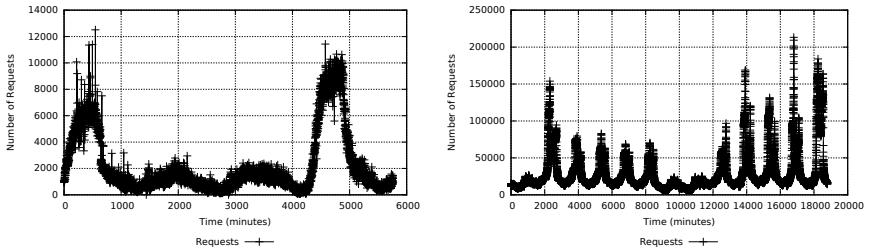
Bursty workloads complicate cloud resource management since cloud providers host a multitude of applications with different workloads in their datacenters. Problems such as service admission control, Virtual Machine (VM) placement, VM migration and elasticity [14] are examples of resource management problems that are complicated due to workload spikes and burstiness. Our interest in quantifying burstiness is driven by our work on cloud elasticity [3]. Workload burstiness has a profound effect on the performance of any autoscaling algorithm. Some elasticity algorithms are better suited for periodic workloads, where the repetitive workload patterns can be used for estimating the future load [2]. Other algorithms are better performing on bursty workloads without repetitive patterns [3]. Since there is no one-size-fits-all solution for workload predictions for elasticity control, we needed a measure for burstiness that is able to compare and classify workloads [4].

We identified some requirements for a burstiness metric to be robust and work on a wide range of scenarios.

1) The metric should be able to capture changes in a wide set of workload types. For example, using the exponential increase will result in missing out bursts where the required number of servers increases from 1000 servers to 1500 servers.
2) The parameters used for calculating the metric should be intuitive, and therefore easy to set.



(a) Workload with little burstiness (Wikipedia). (b) Bursty workload (Tasks in a Google cluster).



(c) Workload with spikes that have a hard to predict magnitude (IRCache servers). (d) Workload with spikes that have a hard to predict magnitude (FIFA world cup servers).

Fig. 2. Different workloads have different burstiness.

3) The metric should be able to operate on short data sequences and to be fast to compute.
4) The metric should differentiate between a gradual workload increase and a sudden one. For example, techniques using entropy are not able to do that.

This work describes our search for a measure of burstiness that is able to differentiate between different workloads while fulfilling the above requirements. Section II discusses the state-of-the-art in both burst detection and quantifying burstiness. While burst detection can be considered as a different problem from quantifying workload burstiness, we include some of the techniques for burst detection and discuss how we considered modifying these to be able to quantify bursts. The limitations of the various reviewed measures are discussed. We propose a measure, *AvgSampEn*, for quantifying burstiness based on *SampEn*, a measure used for quantifying abnormalities in physiological signals. Section III explains the limitations of *SampEn* and our modifications to make it suitable for cloud workloads. We compare the proposed measure with the state-of-the-art in quantifying burstiness.

II. State-of-the-art

The problem of burst detection has been studied in many contexts including bursts in human communications, neuron spike trains, and seismic signals [15], [19], [13]. We have surveyed a large body of work on signal analysis, time-series analysis, workload analysis, spike detection and modeling, workload generation and autonomic management. Unfortunately, all the surveyed papers do not agree on a single definition of what a true burst is. We believe that the techniques discussed in this section are a representative set of the state-of-the-art. While visual inspection has been used for workload spike detection [6], [32], we focus on automated techniques for detecting spikes and quantifying burstiness.

A. Detecting bursts

1) Index of dispersion: The index of dispersion (I) is a measure of burstiness used in communication networks [12]. It was used by Caniff et al. [7] to detect bursts in server workloads. Given a time-series with inter-arrival times (weakly stationary), I is defined as,

$$I = SCV(1+2\sum_{k=1}^{\infty}\rho_k), \tag{1}$$

where SCV is the squared-coefficient of variation (the ratio between the variance and the mean squared) and ρ_k is the lag-k autocorrelation coefficient. Since it is not practical to calculate the autocorrelation coefficient for the inter-arrival time series at ∞, the authors use a batch of $K = arrival\_rate \times C$ requests, where $arrival\_rate$ is the request arrival rate and C is a constant which they set to 2000 in their design. The authors substitute the infinite sum in the exact equation by the first 10% of the batch size. To detect the start and end of a spike, they use the algorithm shown in Algorithm 1. The algorithm detects a burst by comparing the changes in I through time. While choosing the different parameters of the algorithm is not intuitive, e.g., C, they can be set by the application owners.

2) Exponential increase: Wendel and Freeman [35] define a *flash crowd* as a period over which request rates increase exponentially. If r_{t_i} is the average request rate per unit time over a period t_i, then a spike should satisfy the following conditions,

$$r_{t_i} > 2^i \times r_{t_0}, \forall i \in [0,k], \tag{2}$$

and,

$$max_i(r_{t_i}) > m \quad and \quad max_i(r_{t_i}) > n \times r_{avg}, \tag{3}$$

where m is the minimum request rate required to consider the workload increase as a spike, and k is the least sustained modest period of continuous workload increase required to consider a change in the request rate to be a spike. The constant n specifies how much increase in the maximum sustained rate

Algorithm 1: Using the index of dispersion for burst detection.

```
for current batch of K requests do
    current_I← calculate I for current batch
    batch_rate← arrival rate for current batch
    total_rate← update average arrival rate (all requests)
    SCV ← update SCV (all requests)
    if (|current_I-old_I| > 2 × SCV) AND (old_I/current_I >2
    OR old_I/current_I <0.5) then
        if batch_rate >total_rate then
            burst starts
        else
            burst ends
        old_I← current_I
```

must be achieved compared to the service's average rate, r_{avg}, in order to consider the increase a spike. A spike is over when

$$2^{-1} \times r_{t_j} < r_{t_{j+1}} < 2 \times r_{t_j} \tag{4}$$

The choice of the three parameters, k, m and n, is arbitrary and in most cases application dependent. These three parameters can be chosen by the application owner or after profiling an application before its deployment. The main limitation of this method is the assumption that a spike should have exponential growth. One example where spikes were not increasing exponentially is the caching servers' workload in Figure 2(c) where some spikes occurred during the first day where the load increased by less than a factor of two. Another example is large systems serving millions of requests. For a system serving more than 0.5 million requests per minute like Wikipedia, an increase of as little as 10% in the total load can be considered as a spike [2].

Algorithm 2: Using the standard deviation from the moving average for burst detection.

```
Calculate Moving Average MA_w over a period of w
time units
Calculate Threshold ← (MA_w) + x × std(MA_w)
bursts← {|MA_w(i)| > Threshold}
```

3) Standard deviation from the moving average: Vlachos et al. [31] use the simple algorithm shown in Algorithm 2 to detect bursts in online search queries. The algorithm calculates the moving average and the standard deviation of a workload over a window of w time units. A burst is any increase above a threshold equals to the summation of the average and x standard deviations. By setting x large enough, only real spikes are captured by the algorithm.

4) Limitations: With the exception of the index of dispersion based method, the measures and techniques described above are only suitable for online burst detection but not for quantifying workload burstiness. While the index of dispersion method can be used as a measure of burstiness, the choice of C is not intuitive. One possible modification to quantify burstiness using the above methods is to identify and count spikes in each workload over a certain period of time. The more bursty workloads will have larger number of spikes over the same period. The drawback of this technique is its inability to discover periodic bursts, e.g., more people search for the word "Cinema" on search engines during the weekends [31].

B. Quantifying burstiness

Many of the work on quantifying burstiness is based on the information theoretic entropy (H) [11] which is a measure of uncertainty in a workload X. Entropy is calculated using,

$$H(X) = -\sum_{i=1} p_i \times \log p_i, \tag{5}$$

where p_i is the probability of the workload having the value X_i to occur. The main limitations with using entropy are the need for a long sequence to be able to calculate the probabilities, its inability to differentiate between slowly increasing workloads and sudden spikes, and its inability to take into account burst periodicity.

1) Slope of entropy plots: Wang et al. [33] introduce entropy plots, plots showing the entropy of a workload at different aggregation levels for the workload. If entropy is calculated at a fine resolution, no correlation in the load is observed. The higher the resolution of aggregation is, the more correlations that can be captured by H. If the plot shows a linear relationship then the correlation and burstiness is stable across different workload resolutions. The slope can then be used as an indicator for workload burstiness. If there is a change in the workload burstiness and the plot is non-linear, then this method does not work [21].

2) Normalized entropy: Minh et al. [21] introduce the use of normalized entropy as a measure of workload burstiness. It is known that the maximum value achievable for H is $\log N$ [11]. Given a time interval T , Minh et al. divide the interval into N equal sized intervals. The normalized entropy is then defined as,

$$H_{NE}(X) = -\frac{\sum_{i=1}^{N} p_i \times \log p_i}{\log N}, \tag{6}$$

where p_i denotes the probability that a job arrives in interval i. H_{NE} is bounded between 0 and 1. According to Minh et al., H_{NE} values close to zero indicates stronger burstiness.

3) Burst density: Shen et. al. [28] propose to use signal processing techniques to measure the burst density in a workload. The authors use the Fast Fourier Transform (FFT) to calculate the coefficients that represent the amplitude of each frequency component for recent resource usage, e.g., $L = \{l_{t-W_a}, ..., l_{t-1}\}$ where L is the time series of recent resource usage and W_a is the number of measurements considered. They consider the top k (e.g., 80%) frequencies in the frequency spectrum as high frequencies and apply inverse FFT over the high frequency components to synthesize the burst pattern. They calculate a burst density metric β as the percentage (or number) of positive values in the extracted burst pattern compared to the total signal. Higher percentages of β indicate strong burstiness and vice versa. One main disadvantage of this technique is the complexity in choosing k, the number of top frequencies that are considered as burst components. We discuss the limitations for both the normalized entropy and burst density techniques in Section III-C in more details.

4) The Hurst parameter: The Hurst parameter (exponent or coefficient) has been used as a measure for burstiness in the literature for numerous workloads [27], [17], [20]. The Hurst parameter, H, is a measure of the level of self-similarity of a time-series. Self-similar processes have heavy-tailed distributions, and thus, there burstiness can be observed at all time scales.

Various practical issues with estimating the Hurst exponent have been discussed in the literature [9]. In the literature, there are many suggested techniques to estimate the Hurst exponent differ for the same time-series to an extent making the estimation unreliable [22]. Just picking one algorithm from the literature to estimate the Hurst parameter and applying it to different workloads "is likely to end up with a misleading answer or possibly several different misleading answers" [9]. Using the Hurst parameter to quantify cloud workload burstiness can thus lead to wrong conclusions about the workloads running resulting in "bad" management decisions.

III. A NEW BURSTINESS MEASURE

Sample Entropy (*SampEn*) is a robust burstiness measure that was developed by Richman et al. over a decade ago [26], [25]. It is used to classify abnormal (bursty) physiological signals. It was developed as an improvement to another burstiness measure, Approximate Entropy, widely used previously to characterize physiological signals [24]. Sample Entropy is defined as *"the negative natural logarithm of the (empirical) conditional probability that sequences of length m similar point-wise within a tolerance r are also similar at the next point"*.

It has two advantages over Shannon's entropy: i) being able to operate on short data sequences and, ii) it takes into account gradual workload increases and periodic bursts. These advantages make it an interesting potential measure for workload burstiness as a workload having periodic bursts, e.g., every weekend, is easier to manage compared to workloads with no repetitive bursts.

Three parameters are needed to calculate *SampEn* for a workload. The 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. The last parameter is the length of the workload which can easily be computed. We therefore focus on m and r and their choice.

The deviation tolerance defines what is a normal increase and what is 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. If *SampEn* is equal to 0 then the workload has no bursts. The higher the value for *SampEn*, the more bursty the workload is.

A. Implementation of the algorithm

There is one main limitation of *SampEn*, it 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 [1]. [1] In addition, workload characteristics might change during operation, e.g., when Michael Jackson died, 15% of all requests directed to Wikipedia were to the article about him creating spikes in the load [6]. If *SampEn* is calculated for a long history, then recent changes are hidden by the history.

Algorithm 3: The algorithm for calculating *AvgSampEn*.

Data: r, m, T, L
Result: *AvgSampEn*

$N \leftarrow Length(L)$;
$P_{divided} \leftarrow \{T(L.k).....T(L.(k+1)\}\ \forall k \in \{0,N\}$;
$Tot_{SampEn} \leftarrow 0$;
for W *in* $P_{divided}$ **do**
  $n \leftarrow Length(W)$;
  $B_i \leftarrow 0$;
  $A_i \leftarrow 0$;
  $X_m \leftarrow \{X_m(i)|X_m(i) = [x(i),...x(i+m-1)]\ \forall\ 1 < i < n-m+1\}$;
  for $(X_m(i), X_m(j))$ *in* X_m*:* $i \neq j$ **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**
      $B_i \leftarrow B_i + 1$;
  $B^m(r) \leftarrow \frac{1}{n-m} \sum_{i=1}^{n-m} \frac{1}{n-m-1} B_i$;
  $m = m+1$
  $X_m \leftarrow \{X_m(i)|X_m(i) = [x(i),...x(i+m-1)]\ \forall\ 1 < i < n-m+1\}$;
  for $(X_m(i), X_m(j))$ *in* X_m*:* $i \neq j$ **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**
      $A_i \leftarrow A_i + 1$;
  $A^m(r) \leftarrow \frac{1}{n-m} \sum_{i=1}^{n-m} \frac{1}{n-m-1} A_i$;
  $Tot_{SampEn} \leftarrow |-\log[\frac{A^m(r)}{B^m(r)}]| + a \times Tot_{SampEn}$;
$AvgSampEn \leftarrow Tot_{SampEn}/N$;

To address these two points, we modified the sample entropy algorithm [1] by dividing the trace into smaller equal sub-traces. *SampEn* is calculated for each sub-trace. A weighted average, *AvgSampEn*, is then calculated for all *SampEn* values for the sub-traces. More weight can be given to more recent *SampEn* values. This way the time required for computing *SampEn* is reduced since n is reduced significantly. Our modification also enables online characterization of workloads since *SampEn* is not recomputed for the whole workload history but rather for the near past. This approach of dividing the workload into smaller sub-traces is similar to the approach used by Costa et al. [10] where they divide a signal to calculate its multi-scale entropy. The main difference

[1] There are some suggested implementations that significantly improve the complexity to be between O(n) and O($n^{3/2}$) [23] making *SampEn* an even more attractive measure.

between the two approaches occurs after *SampEn* is computed for the smaller sub-trace, Costa et al. plot the results while we take a weighted average.

Our algorithm is shown in Algorithm 3. [2] T is the workload for which *SampEn* is calculated. The trace is divided into N sub-traces of length L (lines 1 to 3). For each sub-trace, W, *SampEn* is calculated. The first loop in the algorithm (lines 9 to 14) calculates $B^m(r)$, the estimate of the probability that two sequences in the workload having m measurements do not have bursts. The second loop in the algorithm (lines 15 to 19) calculates $A^m(r)$, the estimate of the probability that two sequences in the workload having $m+1$ measurements do not have bursts. Then *SampEn* for the sub-trace is calculated and is added to the sum of the *SampEn* values of all previous sub-traces multiplied by a weighting factor a (line 20). The average *SampEn* for the whole trace is then calculated.

B. Choosing the parameters

SampEn has been shown relatively robust towards the choice of the two main parameters, the deviation tolerance, r, and the pattern length, m [25], [8]. Ideally, r should be chosen such as to capture all true bursts while m should be long enough to capture interesting periodic patterns. While choosing m, another contributing factor is the granularity of data, studying patterns occurring yearly is different from studying patterns occurring daily or hourly. Some studies suggest different techniques to set the two parameters in the context of biomedical signals [8], [16].

Choosing r, some studies suggest the use of some percentage of the standard deviation of the signal [25], [24]. Lake et. al. [16] show that this can lead to a reduction in the calculated *SampEn* due to the inflation of the standard deviation by the spikes. We have thus chosen a different approach to calculate r. Instead of using the standard deviation of the trace, r is set as a percentage of some high percentile of the workload, e.g., 50% of the 75^{th} percentile of the load values. Any increase above that value will be considered as a spike. Since percentiles in general are a *robust measure of scale*, the errors described by Lake et. al. are no longer an issue.

For choosing m, using Auto-Regressive models [16] or using the combination of the first minimum value of the nonlinear correlation function called average Mutual Information (MI) and the calculation of False Nearest Neighbor (FNN) [8] has been proposed. Cloud workloads are different from biomedical signals since most cloud workloads will have some inherent pattern, e.g., hourly, daily and weekly patterns. Therefore, m should be picked up to capture these patterns instead of using more complicated models or methods suitable for biomedical signals. The value of L should be chosen to be longer than m. Since L controls the rate at which *AvgSampEn*, it should be chosen to suit the frequency with which decisions are taken.

C. Comparison to the state-of-the-art

We calculated *AvgSampEn* for a set of more than 70 workloads (more than 10 real workloads and 60 synthetic traces). Due to space limitations, we only show *AvgSampEn* values for a subset of these loads, the workloads in Figure 2. The traces shown all have minutes time granularity. We set the pattern length m to one hour and the value of L to one day. The deviation tolerance r is set to 30% of the 70^{th} percentile of the load values during a period L. The weighting factor a is set to one.

[2]While there are implementations of *SampEn* in Matlab, we chose to implement our algorithm in Python.

TABLE I. WORKLOAD ANALYSIS RESULTS.

Workload	*AvgSampEn*	H_{NE}	β_{80}	$\beta_{0.0001}$
Wikipedia	0.22	0.998	0.45	0.5
FIFA	0.48	0.988	0.27	0.6
SD-IRCache	3.3	0.975	0.25	0.23
Google	236.25	0.9	0.28	0.4

Table I shows the *AvgSampEn* values for the 4 workloads Figure 2 compared to the use of Normalized entropy described in Section II-B2 and the burst density metric described in section II-B3. The burst density metric, β, was computed when different percentage of frequencies constitute the burst pattern. We choose to show only two values representing two extremes, when the top 80% and the top 0.0001% frequencies constitute the burst pattern.

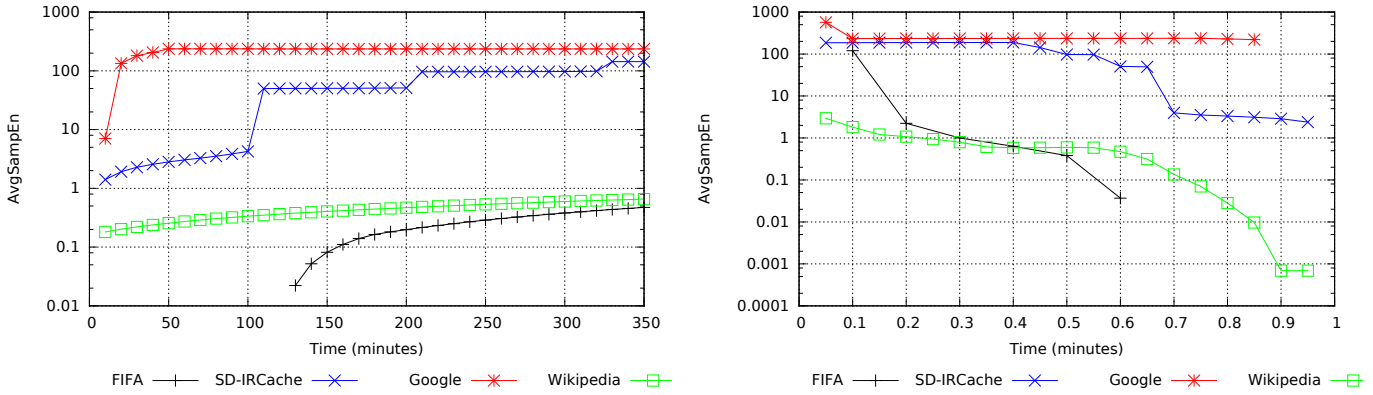
Since the Wikipedia workload shows very little burstiness and is repetitive, the value of *AvgSampEn* is very low, almost zero, indicating no burstiness. The SD-IRCache has more bursts where the load almost doubles in a very short period and thus the value of *AvgSampEn* increases and is almost one. The number of tasks submitted to the Google cluster is very bursty in nature, thus *AvgSampEn* is very high. Our experiments on the other workloads showed similar results for the values of *AvgSampEn* with different workloads. We therefore used *AvgSampEn* in our workload analyzer and classifier [4].

The values of H_{NE} given in Table I show that H_{NE} changes marginally with the different workloads. A workload with clear and strong burstiness such as the Google workload has an H_{NE} value almost equal to the Wikipedia workload which is very stable. On the other hand, *AvgSampEn* varies significantly with the workload burstiness. The increase in *AvgSampEn* with increasing burstiness is non-linear due to the logarithmic function. The burst density metric is clearly unable to differentiate the workloads in terms of burstiness. It is also not robust with changing parameters, i.e., the order of the workloads changes with changing the percentage of frequencies considered.

D. Sensitivity of AvgSampEn

To check the sensitivity of *AvgSampEn* while varying the two main parameters, r and m, we calculate the value of *AvgSampEn* when the value of m varies between ten minutes and six hours with steps of ten minutes, i.e., when m takes the value of 10, 20, 30, ..., 360 minutes. For each step for m we vary r between the 5^{th} percentile of all workload values multiplied by 0.05 to the 95^{th} percentile multiplied by 0.95 with an increase of 5 percent in the percentiles and 0.05 in the multiplicative factor.

To give an example, Figure 3(a) shows the ranking of the values of *AvgSampEn* when r is set to the 50^{th} percentile of the workload values, i.e., the median workload value, multiplied by 0.5. We multiply the percentile by a factor to make the



(a) Sensitivity of the Workload when r is set to be the 50^{th} percentile of the workload and m varies between ten minutes to six hours.

(b) Sensitivity of the Workload when m is set to five hours and r varies between the 5^{th} percentile to the 95^{th} percentile of the load.

Fig. 3. The ranking of the workloads is relatively stable for varying m but shows less stability when r varies and the two workloads have *AvgSampEn* values close to each other.

differences between the tolerances chosen large and to be consistent with the way we defined r in III-C. The pattern length, m, varies in the figure between ten minutes to 6 hours. It can be seen that the ranking of the workloads is robust with respect to the pattern length. On the other hand, this ranking differs with the ranking in Table I. The difference in ranking is for the FIFA and Wikipedia workloads, the least two bursty workloads.

Figure 3(b) shows the ranking when m is set to 5 hours while r is left to vary as described above. While the ranking of the two most bursty workloads is stable, the least two bursty workloads ranking switch when r is set to the 40^{th} percentile of the workload values multiplied by 0.4. A low value for r, the deviation tolerance, results in less stable workload ranking since r defines the sensitivity of the measure to what a "true burst" is. It is thus advised to use a large enough r to make sure that only "true bursts" are taken in to account when estimating the burstiness of a workload.

IV. Discussion

While *AvgSampEn* has been able to quantify burstiness in all the workloads we have tested it with, it is still far from perfect. The computational complexity of the algorithm is $O(n^2)$, even for the modified algorithm [23]. The time required to update *AvgSampEn* depends highly on L and m. In addition, the choice of the parameters, while intuitive, is arbitrary. Despite all the limitations, we were able to use *AvgSampEn* for workload classification and assignment to the most suitable elasticity algorithm with very high accuracy [4]. The sensitivity analysis showed that the measure is sensitive to the deviation tolerance chosen. To choose a "correct" deviation tolerance, one has to have a definition of a "true" burst. Defining bursts is still a problem that is far from being solved in the literature and it requires more focus within the community. We leave this for future work.

V. Acknowledgment

We thank Xiaohui Gu and Hiep Nguyen for providing us with their implementation of the burst density calculation algorithm. Financial support has been provided in part by the Swedish Government's strategic effort eSSENCE and the Swedish Research Council (VR) under contract number C0590801 for the project Cloud Control.

References

[1] Aboy, M., Cuesta-Frau, D., Austin, D., and Micó-Tormos, P. Characterization of sample entropy in the context of biomedical signal analysis. In *IEEE EMBS* (2007), IEEE, pp. 5942–5945.

[2] Ali-Eldin, A., Rezaie, A., Mehta, A., Razroev, S., Sjöstedt-de Luna, S., Seleznjev, O., Tordsson, J., and Elmroth, E. How will your workload look like in 6 years? Analyzing Wikimedia's workload. In *IC2E* (2014), IEEE Computer Society.

[3] Ali-Eldin, A., Tordsson, J., and Elmroth, E. An adaptive hybrid elasticity controller for cloud infrastructures. In *NOMS* (2012), IEEE, pp. 204–212.

[4] Ali-Eldin, A., Tordsson, J., Elmroth, E., and Kihl, M. Workload classification for efficient auto-scaling of cloud resources. Tech. Rep. UMINF 13.13, Umeå University, Department of Computing Science, 2013. Available: www8.cs.umu.se/research/uminf/index.cgi?year=2013&number=13.

[5] Ari, I., Hong, B., Miller, E. L., Brandt, S. A., and Long, D. D. Managing flash crowds on the internet. In *MASCOTS* (2003), IEEE, pp. 246–249.

[6] Bodik, P., Fox, A., Franklin, M. J., Jordan, M. I., and Patterson, D. A. Characterizing, modeling, and generating workload spikes for stateful services. In *SoCC* (2010), ACM, pp. 241–252.

[7] Caniff, A., Lu, L., Mi, N., Cherkasova, L., and Smirni, E. Fastrack for taming burstiness and saving power in multi-tiered systems. In *ITC* (Sept 2010), pp. 1–8.

[8] Chen, X., Solomon, I., and Chon, K. Comparison of the use of approximate entropy and sample entropy: Applications to neural respiratory signal. In *IEEE-EMBS* (Jan 2005), pp. 4212–4215.

[9] Clegg, R. A practical guide to measuring the hurst parameter. In *Proceedings of 21st UK Performance Engineering Workshop, School of Computing Science, Technical Repo* (2005), N. Thomas, N. Thomas.

[10] Costa, M. D., Peng, C.-K., and Goldberger, A. L. Multiscale analysis of heart rate dynamics: Entropy and time irreversibility measures. *Cardiovascular Engineering 8*, 2 (2008), 88–93.

[11] Cover, T. M., and Thomas, J. A. *Elements of information theory*. John Wiley & Sons, 2012.

[12] Gusella, R. Characterizing the variability of arrival processes with indexes of dispersion. *IEEE Journal on Selected Areas in Communications 9*, 2 (1991), 203–211.

[13] Karsai, M., Kaski, K., Barabási, A.-L., and Kertész, J. Universal features of correlated bursty behaviour. *Scientific reports 2* (2012).

[14] KIHL, M., ELMROTH, E., TORDSSON, J., ÅRZÉN, K.-E., AND ROBERTSSON, A. The challenge of cloud control. In *8th International Workshop on Feedback Computing* (2013).

[15] KLEINBERG, J. Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery 7*, 4 (2003), 373–397.

[16] LAKE, D. E., RICHMAN, J. S., GRIFFIN, M. P., AND MOORMAN, J. R. Sample entropy analysis of neonatal heart rate variability. *American Journal of Physiology-Regulatory, Integrative and Comparative Physiology 283*, 3 (2002), R789–R797.

[17] LELAND, W. E., TAQQU, M. S., WILLINGER, W., AND WILSON, D. V. On the self-similar nature of ethernet traffic. In *SIGCOMM* (1993), vol. 23, ACM, pp. 183–193.

[18] LOGS, N. S. C. A. Url: ftp://ircache. nlanr. net, 2000.

[19] MATHIOUDAKIS, M., AND KOUDAS, N. TwitterMonitor: Trend detection over the twitter stream. In *SIGMOD* (2010), ACM, pp. 1155–1158.

[20] MENASCÉ, D., ALMEIDA, V., RIEDI, R., RIBEIRO, F., FONSECA, R., AND MEIRA JR, W. In search of invariants for e-business workloads. In *Proceedings of the 2nd ACM conference on Electronic commerce* (2000), ACM, pp. 56–65.

[21] MINH, T. N., WOLTERS, L., AND EPEMA, D. A realistic integrated model of parallel system workloads. In *CCGrid* (2010), IEEE, pp. 464–473.

[22] MONTANARI, A., TAQQU, M., AND TEVEROVSKY, V. Estimating long-range dependence in the presence of periodicity: an empirical study. *Mathematical and computer modelling 29*, 10 (1999), 217–228.

[23] PAN, Y.-H., WANG, Y.-H., LIANG, S.-F., AND LEE, K.-T. Fast computation of sample entropy and approximate entropy in biomedicine. *Computer methods and programs in biomedicine 104*, 3 (2011), 382–396.

[24] PINCUS, S. M. Approximate entropy as a measure of system complexity. *Proceedings of the National Academy of Sciences 88*, 6 (1991), 2297–2301.

[25] RICHMAN, J. S., LAKE, D. E., AND MOORMAN, J. R. Sample entropy. *Methods in enzymology 384* (2004), 172–184.

[26] RICHMAN, J. S., AND MOORMAN, J. R. Physiological time-series analysis using approximate entropy and sample entropy. *AJP-Heart and Circulatory Physiology 278*, 6 (2000), H2039–H2049.

[27] RISKA, A., AND RIEDEL, E. Disk drive level workload characterization. In *ATC* (2006), pp. 97–102.

[28] SHEN, Z., SUBBIAH, S., GU, X., AND WILKES, J. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *SoCC* (2011), ACM, p. 5.

[29] SINGH, R., SHARMA, U., CECCHET, E., AND SHENOY, P. Autonomic mix-aware provisioning for non-stationary data center workloads. In *ICAC* (2010), ACM, pp. 21–30.

[30] URDANETA, G., PIERRE, G., AND VAN STEEN, M. Wikipedia workload analysis for decentralized hosting. *Computer Networks 53*, 11 (2009), 1830–1845.

[31] VLACHOS, M., MEEK, C., VAGENA, Z., AND GUNOPULOS, D. Identifying similarities, periodicities and bursts for online search queries. In *SIGMOD* (2004), ACM, pp. 131–142.

[32] WANG, C., TALWAR, V., SCHWAN, K., AND RANGANATHAN, P. Online detection of utility cloud anomalies using metric distributions. In *NOMS* (2010), IEEE, pp. 96–103.

[33] WANG, M., AILAMAKI, A., AND FALOUTSOS, C. Capturing the spatio-temporal behavior of real traffic data. *Performance Evaluation 49*, 1 (2002), 147–163.

[34] WANG, Q., KANEMASA, Y., KAWABA, M., AND PU, C. When average is not average: large response time fluctuations in n-tier systems. In *ICAC* (2012), ACM, pp. 33–42.

[35] WENDELL, P., AND FREEDMAN, M. J. Going viral: Flash crowds in an open cdn. In *SIGCOMM* (New York, NY, USA, 2011), IMC '11, ACM, pp. 549–558.

[36] WILKES, J. More Google cluster data. Accessed: May, 2013.