Evaluating the Impact of Fine-scale Burstiness on Cloud Elasticity

Sadeka Islam
University of New South Wales
sadeka.islam@student.unsw.edu.au

Srikumar Venugopal
IBM Research - Ireland
srikumarv@ie.ibm.com

Anna Liu
National ICT Australia
anna.liu@nicta.com.au

Abstract

Elasticity is the defining feature of cloud computing. Performance analysts and adaptive system designers rely on representative benchmarks for evaluating elasticity for cloud applications under realistic reproducible workloads. A key feature of web workloads is burstiness or high variability at fine timescales. In this paper, we explore the innate interaction between fine-scale burstiness and elasticity and quantify the impact from the cloud consumer's perspective. We propose a novel methodology to model workloads with fine-scale burstiness so that they can resemble the empirical stylized facts of the arrival process. Through an experimental case study, we extract insights about the implications of fine-scale burstiness for elasticity penalty and adaptive resource scaling. Our findings demonstrate the detrimental effect of fine-scale burstiness on the elasticity of cloud applications.

Categories and Subject Descriptors C.4 [*Computer Systems Organization*]: Performance of Systems; D.2.8 [*Software*]: Software Engineering; I.6.5 [*Model Development*]: Modeling methodologies

General Terms Measurement, Performance, Theory

Keywords Cloud Computing, Elasticity, Burstiness

1. Introduction

Elasticity, in the context of cloud computing, is the ability of a system to increase or decrease resources on-demand in response to changing workload so that there is no disruption and performance degradation in the delivered service. A perfectly elastic system shapes the operational expenses in a "pay-as-you-go" manner and honors the SLA of the delivered service in response to varying workload intensity. Many cloud providers offer developers tools and application programming interfaces (APIs) to control elastic provisioning of resources; examples include Amazon Web Services' Autoscaling API[1], Google Compute Engine Autoscaler[2], and Microsoft Service Management API[3].

As elastic resource provisioning impacts cloud usage costs as well as adherence to SLAs, the preprocessing step for cloud adoption in the enterprise often includes one or more of design, evaluation and benchmarking of provisioning mechanisms using realistic reproductions of actual workloads. Ideally, the workload characteristics play a crucial role in elasticity evaluation and adaptive mechanism design; however, current elasticity evaluation studies have concentrated their effort on the deterministic trend observed at coarse timescales in the order of minutes (e.g., [9, 19]). However, burstiness or high variability in the workload at finer timescales (in the order of seconds) is often overlooked, though its presence in the arrival process induces detrimental effects on the performance and utilization in traditional client-server and virtualized systems [14, 26]. Performance analysts and adaptive system designers have an intuitive feel for its implications on elasticity too, however, those conjectures haven't been validated yet for adaptive cloud applications. An incomplete and vague understanding about the impact of fine-scale burstiness may lead to inaccurate elastic response prediction and inefficient adaptive mechanism design that may in turn impact running cost and profitability.

In this paper, we seek to investigate the impact of fine-scale burstiness on the elasticity of cloud based web applications from the consumer's viewpoint. One specific impediment to achieving this goal relates to the realistic modeling of workloads with fine-scale burstiness; however, workloads reproduced using existing approaches (e.g., [6, 10, 14]), can't resemble the intrinsic stylized facts (e.g., empirical statistical regularities) of fine-scale fluctuations in the original arrival process.

The contribution of this paper is two-fold. First, we propose a novel methodology to model workloads with fine-

SoCC '15, August 27-29, 2015, Kohala Coast, HI, USA.

http://dx.doi.org/10.1145/2806777.2806846

[1] aws.amazon.com/autoscaling/

[2] cloud.google.com/compute/docs/autoscaler/

[3] msdn.microsoft.com/en-us/library/azure/ee460799.aspx

scale burstiness so that the time-dependent regularity structure (a measure of randomness over time) is compatible with the original arrival process. One standard technique to characterize the regularity structure is the estimation of pointwise Holder exponents, a measure of randomness in the workload time series [1]. We advocate an approach to characterize the Holderian regularity from fine-scale fluctuations of the trace. The next step makes use of the regularity information and a user-defined standard deviation to synthesize fine-scale fluctuations from a multifractional Gaussian noise process. The deterministic trend is generated from a normalized shape function by performing rescaling and interpolation. The final step superimposes fine-scale burstiness on the deterministic trend to yield the desired workload. This method is robust and provides the user the flexibility to control the variance of fine-scale fluctuations while preserving the empirical stylized facts of the workload trace.

Second, we conduct a case study in EC2 to explore the elasticity behavior of a web application in response to fine-scale burstiness. Our analysis reveals that fine-scale burstiness has significant implications for elasticity; it overwhelms the under-provisioning scenario with highly correlated peak arrivals near the saturation region and leads to over-provisioning with reduced utilization. Our investigation extracts out additional interesting insights about fine-scale burstiness, for instance, its influence on adaptive resource scaling and the trend in the elasticity penalty rate.

The remainder of the paper is organized as follows. Section 2 defines the research problem and motivation behind this work. Section 3 reviews the state of the art. Section 4 reveals some statistical facts about fine-scale burstiness from the viewpoint of multifractal analysis. Section 5 defines our methodology to model workloads with fine-scale burstiness. Section 6 illustrates an elasticity measurement framework to quantify the impact of fine-scale burstiness. Section 7 describes experimental setup and Section 8 presents empirical observations about the impact of fine-scale burstiness on the elastic response of a cloud based web application. Section 9 offers conclusion and direction to future work.

2. Problem Statement

The evaluation and design of elastic cloud systems is a non-trivial exercise due to the complexity of the workload profiles. Effective elasticity can be achieved if and only if the application's adaptive mechanism is optimized and evaluated under realistic workload conditions. Hence, it is absolutely crucial to benchmark elasticity and design adaptive strategies under realistic reproducible prototypes of the actual workload. Fine-scale burstiness is an inherent characteristic of the web and e-commerce request arrival process [12, 13]. Despite its prevalence, it has not been addressed in the existing elasticity evaluation and benchmarking studies. In this section, we first advocate the need for a novel methodology to model fine-scale burstiness so that the empirical stylized facts of the actual workload are resembled in the reproduced prototype. Next we argue that ignoring fine-scale burstiness in the elasticity evaluation and adaptive scaling design bears the risk of skewing the overall picture and even leading to incorrect conclusions; therefore, it is important to take this factor into account during elasticity benchmarking and optimization.

Fig. 1 demonstrates the presence of fine-scale burstiness in the request arrival process of a Wikipedia workload sample [22]. As we can see, the request arrival rate is not a smooth curve, instead, it consists of many fine-grained fluctuations. The width of the fluctuation band is very large (a rough estimate would be $200-1000$ requests/second) which reflects high volatility in the request arrival process in consecutive intervals. As amplitude increases, the fluctuations tend to become more erratic in nature. A correlation coefficient of -0.659 provides evidence of a strong relationship between amplitude and regularity behavior (i.e., the degree of randomness in the noisy fluctuations) over time for this workload sample (Details on how to quantify regularity is elaborated in Section 5.1). In order to get a reliable estimate on the elasticity behavior, it is necessary to carry out benchmarking and optimization for a realistic reproducible prototype of the actual workload. Fine-scale burstiness, identified as a salient feature of the web and e-commerce workloads, should also be included in the reproduced prototype. As yet, the literature doesn't provide any solution to the realistic modeling of workloads in which the empirical stylized facts about fine-scale burstiness is well-preserved. This is the gap that we address with a novel workload model based on a deterministic time-dependent regularity function.

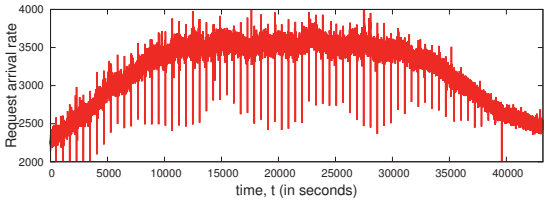


Figure 1: Wikipedia workload snippet (Oct 1 2007)

Now we describe with an example how different assumptions of fine-scale burstiness result in different elasticity behavior. We illustrate this point based on the elastic response of TPC-W application [5] hosted on a dynamic web server farm in AWS EC2 cloud under two different workload assumptions, workload with no fine-scale burstiness versus workload with fine-scale burstiness. The details on how to reproduce a workload with fine-scale burstiness is provided in Section 5. In both experiments, we add an EC2 instance to the server farm when the average CPU utilization goes beyond 70% for 3 consecutive minutes and remove an instance when the average CPU utilization falls below 20% for 10 consecutive minutes using the Autoscaling API[4]. We

[4] aws.amazon.com/autoscaling/

assume each EC2 instance provides 100% of CPU supply per minute.

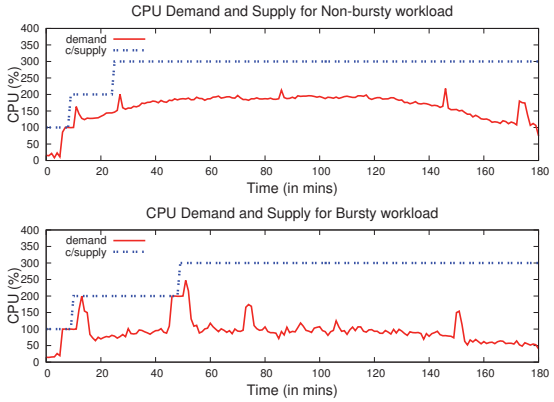


Figure 2: EC2 platform's elasticity behavior under non-bursty and bursty workloads

Fig. 2 shows the elasticity behavior of the CPU resource in response to non-bursty and bursty workload conditions in terms of demand and charged supply (c/supply). Demand means the amount of resource needed by the application to serve the requests with satisfactory performance whereas charged supply indicates the amount of resource for which the cloud platform charges the consumer [9]. The cloud platform starts charging the consumer's application at the very moment it requests for a new resource; however, the resource is available for use after some random delay (typically, several minutes). For this reason, the demand function grows a few minutes later than the charged supply function in the above figure. During the scale-out period, the available resource capacity is scarce; therefore, the server farm may not be able to serve the peaks of the bursty arrivals with graceful QoS. The longer the scale-out period and the spikier the request arrivals, the more the degradation in the perceived QoS of the application.

In the above figure, we observe that these workloads influence the elasticity behavior of the EC2 server farm in different ways (even though the average scale-out delay is similar in both cases, which is around 75 seconds). When the arrival rate exhibits fine-scale burstiness, most of the CPU resource remains under-utilized, thus giving rise to high over-provisioning cost. We also notice that under bursty workload condition the application server had to stay longer in the under-provisioning state. Consequently, the application failed to serve requests with satisfactory QoS due to limited server resources. Table 1 shows the QoS degradation for both workloads because of under-provisioning. Under bursty workload condition, the application couldn't meet the response time constraint of 2 seconds for a significant portion of requests (10.82%), which is 5.49 times higher than the non-bursty workload condition (1.97%). Moreover, a fraction of requests (2.17%) remained unserved due to unavailability or timeout issue under bursty condition. The autoscaling ruleset, although appears to be quite sufficient for the non-bursty workload, couldn't ensure good elastic response for the bursty workload. It caused a significant increase in the over-provisioning and under-provisioning for the fine-grained bursty workload. Therefore, we conclude that optimizing and evaluating elasticity under the assumption of non-bursty workloads reduces the productivity of cloud applications when the workload is bursty in practice. This example also justifies the reason behind our evaluation of elasticity for fine-scale burstiness and quantification of this impact on the monetary penalty from consumers' perspective.

QoS	Non-bursty	Bursty
Requests violating 2 second response time constraint	1.97%	10.82%
Unserved requests	0.79%	2.17%

Table 1: QoS degradation due to under-provisioning

3. Related Work

A solid understanding of workload characteristics is required to ensure effective elasticity of the cloud-hosted applications. Web workloads have been extensively studied and characterized in many publications, such as, [12, 13, 23]. These studies identify burstiness or high variability at the fine timescale and deterministic trend at the coarse timescale in the HTTP request arrival process.

Considerable effort has been spent on burstiness modeling and evaluation for traditional client-server scenarios [6, 10, 14, 25]; some of these studies demonstrate deleterious effect of burstiness on the performance and utilization of the application. Nevertheless, the prototypes generated by these workload models can't resemble the empirical stylized facts (e.g., regularity behavior) of the original arrival process. Xia et al. [25] propose a workload model to generate request arrivals at the fine timescale based on exact self-similarity [3, 24] assumption and fractional Brownian motion (fBm). The roughness or regularity of an fBm process is the same at all timepoints [18]; therefore, the regularity of the reproduced workload using this model remains constant over time. This is not consistent with what we observe in actual workloads where the regularity behavior varies erratically over time. The LIMBO toolkit [6, 10] generates cloud specific workloads based on four aspects: seasonality, trend, burst and noise. The noise component resembles fine-scale burstiness, similar to our notion. However, its modeling using random distribution produces many different realizations for workloads with fine-scale burstiness, most of which lack compatibility with the empirical stylized facts of the actual workload. An incompatible realization may provide a distorted view about the elasticity behavior and in the worst case, may surprise the cloud consumer with unexpected loss of revenue. Mi et al. [14] develop a bursty workload model based on a Markov modulated process; it consists of two states, each generates requests with high and low

think times respectively and the transition probabilities between states are governed by a burstiness parameter "Index of dispersion". This approach, too, generates many incompatible realizations of the actual workload and can't guarantee the preservation of empirical stylized facts in the reproduced prototype.

Several studies [20, 26, 27] suggest the importance of burstiness for performance evaluation and adaptive resource provisioning in the cloud. Tai et al. [20] propose an adaptive load-balancing algorithm to balance bursty arrival of jobs to a fixed number of EC2 processing servers. Yin et al. [26] analyze and predict the performance and utilization of a Xen virtualization environment under bursty loads. Youssef et al. [27] provide adaptive capacity planning techniques for service providers under bursty workload. Our work differs from those in that we evaluate the impact of fine-scale burstiness on elasticity from the consumer's viewpoint, whereas these works propose adaptive solutions for cloud service providers. The results of our analysis can be used as complementary data to help them better understand the consumer's detriment and design improved adaptability techniques under bursty loads.

4. Multifractal Analysis

The concept of "scaling" refers to the absence of any notable change in a finite sequence when observed at different timescales [18]. As a consequence of this phenomenon, the whole and its parts appear statistically identical to each other; processes with this property are known as exactly self-similar or scale-invariant. Fractional Brownian motion (fBm) is a widely accepted model to represent exactly self-similar processes. Most of the web and e-commerce workloads, however, are approximately self-similar [12, 13]; scaling holds only within a finite range of timescales and the scaling behavior at the fine timescales does not exactly resemble those at the coarse timescales. This type of processes exhibits strongly correlated trend at the coarse timescales and noisy oscillations at the fine timescales. A global scaling exponent on asymptotic self-similarity is typically used to explain the strongly correlated trend feature at the limit of the coarsest scales. The sudden dips or spikes are described with respect to the local scaling exponent which is related to the degree of randomness in the timeseries. If the local scaling exponent varies with time, the process is referred to as multifractal. Wavelets are perfect tools for analyzing sudden discontinuities or sharp changes in the timeseries; hence, wavelet based analysis is often used to study the complex scaling phenomena in the timeseries.

To analyze the scaling behavior of the request arrival rate process $R(t)$, we adopt the Wavelet Transform Modulus Maxima (WTMM) method. The wavelet transform (WT) decomposes the function $R(t)$ into elementary space-scale contributions with an analyzing wavelet $\psi(t)$ by means of translations and dilations. Wavelet transform acts as a microscope; it reveals more details on the local characteristics of a function while moving towards smaller timescales a. The partition function $Z(q,a)$ is then computed by summing up the q^{th} moment of the local maxima of the wavelet coefficients at each timescale a.

The basic scaling hypothesis states that the partition function, $Z(q,a)$, should behave as [15]:

$$Z(q,a) \sim a^{\tau(q)}, a \to 0^{+} \tag{1}$$

where, $q \in \mathbb{R}$ is the order of the moment, a is the timescale and $\tau(q)$ is the scaling exponent. Positive values of q accentuate the strong inhomogeneities in the timeseries whereas the negative values of q accentuate the smoothest ones. The slope of the log-log plot between the partition function $Z(q,a)$ and timescale a yields the scaling exponent $\tau(q)$. If the $\tau(q)$ exponents are linear in q, the timeseries is monofractal or exactly self-similar; the slope of the $\tau(q)$ spectrum provides the constant global measure of self-similarity in the timeseries (also known as the Hurst or Holder exponent h). Otherwise, the timeseries is multifractal, which means that the scaling properties of the timeseries are inhomogeneous and varying with time; the local inhomogeneities are described in terms of local Holder exponents $h(t)$. The Holder exponent h can be interpreted as a measure of randomness or regularity of the timeseries at a specific timepoint [1]. A formal definition of this exponent will be provided in Section 5.1. This exponent varies within the range $(0,1)$; the higher the value of h, the less volatile and smoother the timeseries is. Depending on the value of h, the timeseries could be positively correlated ($h > 0.5$), uncorrelated ($h = 0.5$), or negatively correlated ($h < 0.5$).

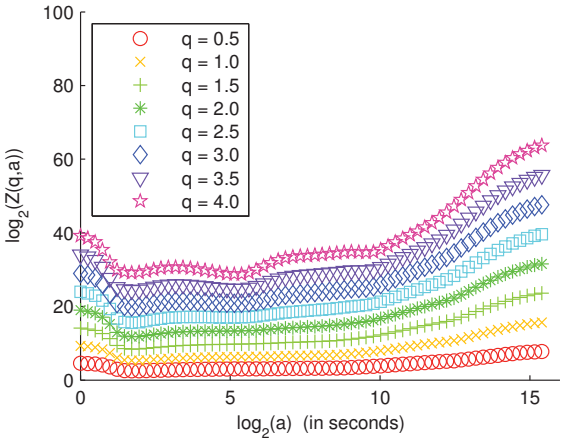


Figure 3: Partition function, Z(q,a) vs. Timescale, a

Fig. 3 shows the log-log plot of $Z(q,a)$ as a function of different timescales a for the Wikipedia workload shown in Section 2. In this plot, the slope of the partition function $Z(q,a)$ changes with the timescale a, indicating the presence of multiple scaling exponents in the timeseries. Moreover, the slopes at the fine timescale are relatively lower than

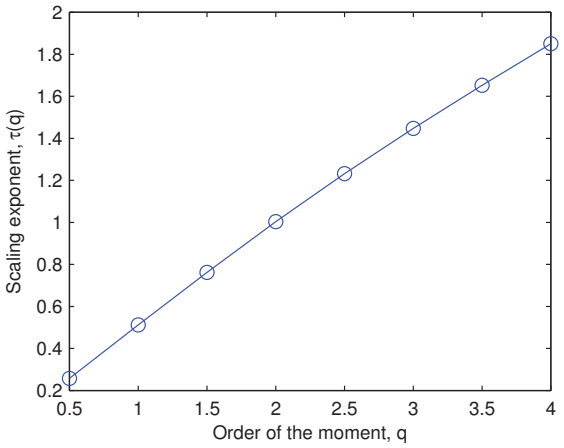


Figure 4: Scaling exponent, $\tau(q)$ vs. q

those at the coarse timescale; it implies that the scaling exponents (and Holder exponents h) are lower at fine-scale region, thereby suggesting high variability in the request arrival rate. The non-linear scaling spectrum $\tau(q)$ versus q in Fig. 4 shows evidence for the multifractal behavior of the request arrival process at the fine timescale, which is also congruent with previous analyses [12, 13]. For the computations performed in this section, we acknowledge the use of Multifractal Toolbox [7].

Our modeling technique relies on the local scaling property of the multifractal process; fine-scale burstiness is synthesized by exploiting the regularity structure of the arrival process, as directed by time-varying Holder exponents.

5. Modeling Methodology

This section presents our approach to model workloads with fine-scale fluctuations. We assume that a workload consists of two components [12]: deterministic trend governed by the asymptotic self-similarity parameter H and highly variable noisy oscillations governed by the Holder function $h(t)$ and standard deviation σ. We characterize how the deterministic trend and the noisy oscillations evolve with time with a shape function and a regularity function respectively. Later workload prototypes are reproduced using these template functions. The standard deviation σ refers to the magnitude of fine-scale burstiness; the larger the σ, the more bursty is the workload prototype at the fine-scale.

The following steps illustrate our methodology in detail.

5.1 Step 1: Characterization of Pointwise Regularity

The notion "regularity" is used to measure the degree of randomness in a finite sequence [17]. A quantitative understanding about the regularity of a process can be obtained by estimating the Holder exponent h at each point. A function $f(t)$ is Holderian with an exponent $h \in (0,1)$ at point t, if there exists a constant c such that the following condition holds for all t' in the neighborhood of t [1]:

$$|f(t) - f(t')| \leq c|t - t'|^h \tag{2}$$

One way to estimate regularity is the application of the Oscillation method, which determines the degree of abrupt fluctuations at each point with respect to its neighborhood [21]. This method restates the above condition as follows:

$$\exists c, \forall \tau \; osc_\tau(t) \leq c\tau^h \tag{3}$$

where, $osc_\tau(t)$ is defined as,

$$\begin{aligned} osc_\tau(t) &= \sup_{|t-t'|\leq\tau} f(t') - \inf_{|t-t'|\leq\tau} f(t') \\ &= \sup_{t',t''\in|t-\tau,t+\tau|} |f(t') - f(t'')| \end{aligned} \tag{4}$$

The exponent h is then estimated as the slope of the regression between the logarithm of the oscillations $osc_\tau(t)$ and the logarithm of the size τ of the neighborhood in which the oscillations are computed. The higher the value of the Holder exponent h, the more regular or smoother is the behavior at that point t. In practice, the self-similar or monofractal functions have the same Holder exponent for all t. In contrast, multifractal functions have a range of Holder exponents which vary over the values of t.

We characterize regularity as follows; first, the relative noise $\Delta(t)$ is extracted out from the request arrival rate $R(t)$:

$$\Delta(t) = R(t) - R(t-1) \tag{5}$$

Next the Holder exponents $h(t)$ are estimated from $\Delta(t)$ using an implementation of the Oscillation method discussed in [2]. The implementation makes use of several parameters: r_{min}, r_{max} and $base$, which define neighborhoods of different dimensions around t. The maximum absolute difference in the oscillations in the neighborhood $[t - base^r, t + base^r]$ is computed for all integer values of $r \in [r_{min}, r_{max}]$. Afterwards, the Holder exponent h at that point is estimated from the slope of the linear regression on the logarithm of the maximum absolute differences in the oscillations with respect to the logarithm of the neighborhood sizes. This is repeated for all timepoints t to characterize the underlying regularity structure $h(t)$ of the noisy oscillations.

5.2 Step 2: Synthesis of Fine-scale Burstiness

In this step, we reproduce the fine-scale burstiness of the original timeseries based on multifractional Gaussian noise (mGn), specified completely in terms of the Holder function $h(t)$ and standard deviation σ. Before diving into the detail, let's rehearse some basic concepts on mGn.

A fractional Gaussian noise $G = (G_t : t = 1, 2, \ldots)$ is a zero-mean stationary Gaussian process with two parameters: a constant Holder exponent $h \in (0,1)$ and variance $\sigma^2 = var(G_t)$. The distribution of the fGn is characterized by its auto-covariances at intervals $\tau \in \mathbb{Z}$ between discrete timepoints G_t [3]:

$$cov(\tau) = \frac{\sigma^2}{2}(|\tau+1|^{2h} - 2|\tau|^{2h} + |\tau-1|^{2h}) \quad (6)$$

As we can see from Eq. 6, the distribution of the fGn exhibits long term correlation between discrete timepoints G_t, uniquely determined by the Holder exponent h. fGn is considered as the increment process of the self-similar timeseries, fractional Brownian motion (fBm). However, the constant h in its autocovariance structure implies the uniform nature of the oscillations, which pose a disadvantage in modeling the erratically changing oscillatory behavior of real-world phenomena. To address this limitation, multifractional Gaussian noises (mGn) have been introduced, where the constant Holder exponent h is generalized by a time-varying Holder function $h(t)$ [16]. The distribution of the mGn is, therefore, non-stationary and varying with time.

Next we turn to the synthesis of fine-scale burstiness. Suppose, our goal is to generate a workload with N request rates. We estimate the Holder exponents for these N points by carrying out cubic spline interpolation on the given Holder function. Later we sample each timepoint specific burstiness from an mGn process parameterized with $h(n)$ and σ.

Algorithm 1: Synthesis of workload with fine-scale burstiness

Input : Sample size, N
Holder function from trace, h_t
Standard deviation, σ
Shape function, f
Lower bound, b
Upper bound, u

Output: Workload with fine-scale burstiness, λ

1 $seed \leftarrow getRandomSeed()$;
2 $j \leftarrow linspace(0,1,N)$;
3 $h_n \leftarrow interpolateHolderExponents(h_t, j)$;
4 $mGn(0) \leftarrow fBm(N,0,h_n(0),\sigma,seed)$;
5 **for** $i \leftarrow 1$ **to** $N-1$ **do**
6 $\quad mGn(i) \leftarrow fBm(N,i,h_n(i),\sigma,seed) - fBm(N,i-1,h_n(i),\sigma,seed)$;
7 **endfor**
8 $X \leftarrow rescaleAndInterpolate(f,j,b,u)$;
9 $\lambda \leftarrow X \bigoplus mGn$;

5.3 Step 3: Trend Construction and Superposition

The next step is to construct the deterministic trend from a normalized shape function using rescaling and spline interpolation; however, it can be substituted by the existing curve-fitting and statistical modeling techniques for coarse-scale workloads, e.g., [4, 6, 9]. Note that we derive the shape function by exploiting the min-max scaling method on the average arrival rates at some coarse timescale a. The following step combines fine-scale burstiness with the deterministic trend using superposition (i.e., the point-by-point addition of two timeseries).

Algorithm 1 elaborates Step 2 and Step 3. It generates a workload of sample size N for a given standard deviation σ that resembles the underlying regularity structure of the trace at the fine timescale. At line 2, a row vector j is created with N linearly spaced points in the interval $[0,1]$. Line 3 performs cubic spline interpolation on h_t to generate Holder exponents corresponding to the N points in vector j. Next a multifractional Gaussian noise process is synthesized based on interpolated Holder exponents h_n. At line 6, the noise $mGn(i)$ for the i^{th} sample is produced by taking the sampled increment of the fBm parameterized with $h_n(i)$ and σ.

One of the most popular approaches for synthesizing fBm is the Random Midpoint Displacement algorithm [11]. To generate an fBm process $Z(t)$ in an interval $[0,T]$, we start out by setting the values $Z(0) = 0$ and sampling $Z(T)$ from a Gaussian distribution with mean 0 and variance σ^2. In the next step, $Z(T/2)$ is constructed as the average of the values of the two endpoints plus an offset which is a Gaussian random variable with variance σ_1^2:

$$Z(\frac{T}{2}) = \frac{Z(0)+Z(T)}{2} + Gauss(0,\sigma_1^2) \quad (7)$$

where σ_1 is computed as follows:

$$\sigma_1 = \sigma 2^{-H}\sqrt{1-2^{2H-2}} \quad (8)$$

In the following step, the two intervals $[0,T/2]$ and $[T/2,T]$ are further sub-divided and the midpoints are computed as the average of the values of the two endpoints plus a Gaussian random offset whose standard deviation is 2^{-H} times the previous level's standard deviation. This process continues recursively until the maximum number of levels is reached.

The next step of our algorithm constructs the trend part by rescaling the normalized shape function f bounded by the range $[b,u]$ and then interpolating the datapoints from the rescaled function, as shown in Line 8. And finally, the fine-scale fluctuations and the trend are combined using superposition in Line 9. We acknowledge the use of Fraclab [8] routines for the computations performed in this section.

5.4 Working Example

We illustrate our methodology with a 12 hour trace segment of Wikipedia workload starting from 11:20:00 PM Oct 1, 2007 and viewed at per-second resolution. We use the Wikipedia traces because these are the most recent publicly available real workload traces for web-scale applications. At first, the regularity exponents are estimated from the noisy oscillations using the method outlined in Section 5.1. We keep the neighborhood size small so that the abrupt irregularity in the noisy oscillations does not get smoothed out. The parameters for estimating the neighborhood are: $base = 2.1$, $r_{min} = 1$ and $r_{max} = 2$.

Next, Algorithm 1 is executed to generate a sample of $N = 10800$ points whose fine-scale burstiness has a standard deviation σ and obeys the regularity structure $h(t)$. It rescales the shape function f within the range $[100, 650]$ and performs spline interpolation on the rescaled function to produce the trend part. It should be noted that the shape function is derived from the min-max scaling or normalization of the average arrival rates at $a = 300$ seconds. The final step of the algorithm combines both the trend and the fine-scale burstiness to yield the desired workload.

The Holder exponents $h(n)$, trend $X(n)$ and the resulting workload $\lambda(n)$ are shown in Fig. 5. The Holder exponents vary within the range $[0.065451, 0.23649]$. The fluctuation band gets thick around the trend curve because of negatively correlated noisy oscillations $[h(n) < 0.5]$. The dips of $h(n)$ are assumed to bear high risk as they induce abrupt irregularity in the arrival rates. Also, the declining tendency of the dips with regard to amplitude suggests increased degree of fluctuations at higher amplitudes (so the events of very high arrivals will be followed by events of very low arrivals). The correlation coefficient between amplitude and regularity for this workload prototype is -0.617 which is close to that of the original workload (-0.659). It also preserves the time-dependent stylized facts of the original arrival process (as the fine-scale noises are generated using the regularity function of the actual workload). It is important to ascertain the cloud application's elastic response to the abrupt change in fluctuation band. It is also necessary to understand the effect of fine-scale burstiness on adaptive resource scaling; when its presence can be abstracted out and when it would matter.

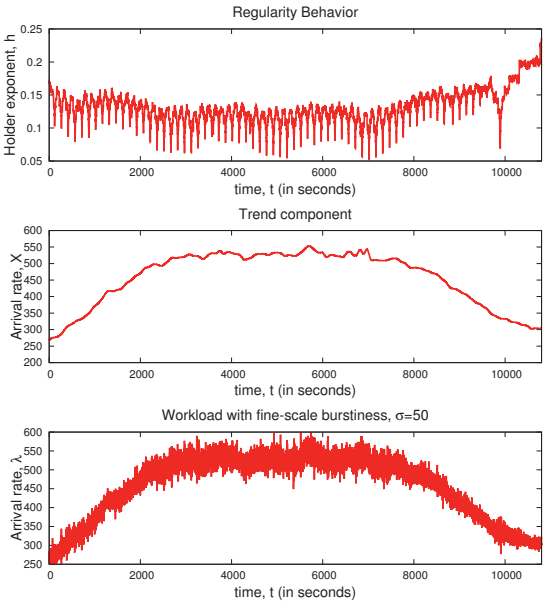


Figure 5: Holder function, deterministic trend and generated workload with fine-scale burstiness

5.5 Comparison with Other Methods

In this section, we demonstrate with an example how existing approaches fail to reproduce empirical stylized facts in the generated workload. Fig. 6 shows a realization of the Wikipedia workload generated using the LIMBO toolkit [6, 10]. Looking at this figure, we observe that this prototype workload can mimic the coarse-scale structure of the actual workload. However, the Holder function estimated from this prototype doesn't resemble the regularity behavior of the original arrival process (compare with Fig. 5). The correlation between amplitude and regularity for this prototype workload is -0.52 which implies significantly weaker correlation than what is observed in the actual workload (-0.659). The prototype generated by our workload model, in contrast, closely follows the time-dependent regularity structure and approximately preserves the correlation between amplitude and regularity of the original arrival process (-0.617). We have already discussed in Section 2 how different assumptions of fine-scale burstiness result in different elasticity behavior. Therefore, it is vitally important to evaluate adaptive cloud systems under realistic bursty workload to ensure effective elasticity in production.

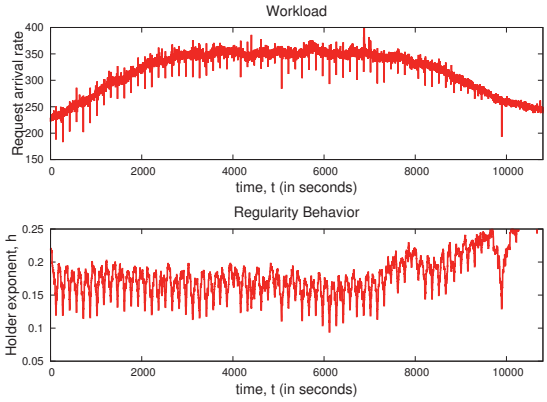


Figure 6: Workload reproduced using LIMBO toolkit

6. Elasticity Measurement Framework

We intend to measure the application's elastic response to the workloads with fine-scale burstiness in terms of its impact on the consumer's finances. A previous work by some of the authors [9] proposed an elasticity measurement framework that quantifies the financial detriment of imperfect elasticity from the consumer's perspective.

According to this framework, the over-provisioning penalty for a given workload is measured as the excess payment for idle resource capacity over the duration of the workload execution. The steps are as follows: compute the unit cost for each resource type in the given virtual machine, aggregate the excess capacity (i.e., the difference between charged supply and demand) for each resource type over the execution interval; the total over-provisioning penalty is then com-

puted by multiplying the excess capacity with the unit cost for each resource type, and then summing up the financial detriments for excess capacity across all types.

The under-provisioning penalty for a given workload is measured as the opportunity cost of requests violating the SLA during the scale-out period. It can be computed by quantifying the fraction of requests that violates the predefined QoS threshold and then measuring the monetary cost of the unsatisfactory QoS behavior with a mapping function.

The total penalty rate is then computed by aggregating the over-provisioning and under-provisioning penalties incurred over the interval of workload execution and normalizing it with the total hours of the execution. A small penalty rate indicates better elastic response to the given workload.

7. Experimental Setup

In this paper, we present a case study based on the TPC-W benchmark and report on the elasticity implications of fine-scale burstiness in a multi-tiered system. TPC-W is a widely used e-commerce benchmark that emulates the complex user interactions of a Business-to-Consumer (B2C) website [5]. We host the online bookshop implementation of the TPC-W application on a server farm of EC2 m1.small instances with 32-bit architecture. This instance type enables the consumer's application to scale in small, low-cost increments with the varying workload intensity; for this reason, it is frequently used as the basic building block for autoscaling the server farm[5]. An EC2 m1.small instance provides the equivalent CPU capacity of a $1.0 - 1.2$ GHz 2007 Opteron or 2007 Xeon processor, 1.7 GB of memory and 160 GB of local instance storage[6]. We host both the web server and application server on the same EC2 instance because the web server simply forwards the dynamic requests to the application server, thus service rate of this front-end instance is dominated by the work of the application server. Instead of keeping the images in the same EC2 instance, we deploy them in S3[7]. In this paper, our prime concern is the elasticity implications of the application server in response to fine-scale burstiness; hence, we host the back-end database into an m1.xlarge RDS[8] MySQL instance so that it doesn't become a bottleneck tier and its influence on the end-to-end performance is negligible.

The web server farm resides behind an internet-facing load-balancer[9]. The number of EC2 instances of the server farm is dynamically adapted in response to the workload demand using Autoscaling API. All experiments start with only one EC2 instance in the server farm; the capacity of the server farm is adapted in response to the workload intensity based on the following ruleset: increase the server farm capacity by one instance when the average CPU utilization goes beyond 70% for 3 consecutive minutes and decrease the capacity by one instance when the average CPU utilization goes below 20% for 10 consecutive minutes. The maximum number of EC2 instances used at the peak varied between 3 and 4 across different runs of the experiment because of the performance variability and heterogeneity issues in the cloud. We host the server farm on a single availability zone for these experiments.

Instead of using the TPC-W workload generator, we use Jmeter[10] to specify our predefined workload pattern. Jmeter provides a convenient interface to generate an open workload and vary the number of concurrent requests per second (RPS) over time using XML script. A user session is defined in terms of browsing-specific requests; each user lands on a homepage, then searches for new books in a random genre and then browses the detail of a randomly selected item.

To compute the penalties, we adhere to the concrete SLA choices made by the executable benchmark in [9]. We expect that at least 95% of all requests will see a response within 2 seconds and there won't be any service disruption. Otherwise, a financial penalty of 12.5¢ will apply for each additional 1% of requests for which the 2 second threshold is breached. Furthermore, for each 1% of dropped requests, a financial penalty of 10¢ is charged. The SLA evaluation period is considered to be an hour. The pricing of m1.small instance is 4.4¢ per instance-hour, which is used to compute the over-provisioning penalty for the CPU resource.

8. Case Study

In this secton, we report our findings on the impact of fine-scale burstiness on the elasticity of cloud applications. The purpose of this analysis is two-fold. First, we seek to investigate the impact of fine-scale burstiness on elasticity penalty by measuring its effect on over-provisioning and under-provisioning with respect to a reference smooth workload (i.e., the workload with no fine-scale burstiness). We also aim to determine how fine-scale burstiness influences adaptive resource scaling. Next we conduct sensitivity analysis to explore the relationship between fine-scale burstiness and elasticity penalty. Table 2 shows the results of our case study. The execution time of each workload is 190 minutes where the first 5 minutes and last 5 minutes are warmup and cooldown periods respectively, and have been omitted in the analysis. The penalty rates, reported in this table, represent the average of three runs of a workload.

8.1 Effect of Fine-scale Burstiness on Elasticity

Fig. 7 depicts the elasticity behavior of a workload with fine-scale burstiness (sigma150) as compared to the smooth workload. The topmost plot shows the CPU demand and charged supply over time. Note that the scale of the time

[5] www.rightscale.com/blog/cloud-cost-analysis/will-aws-t2-replace-30-percent-instances-not-so-fast

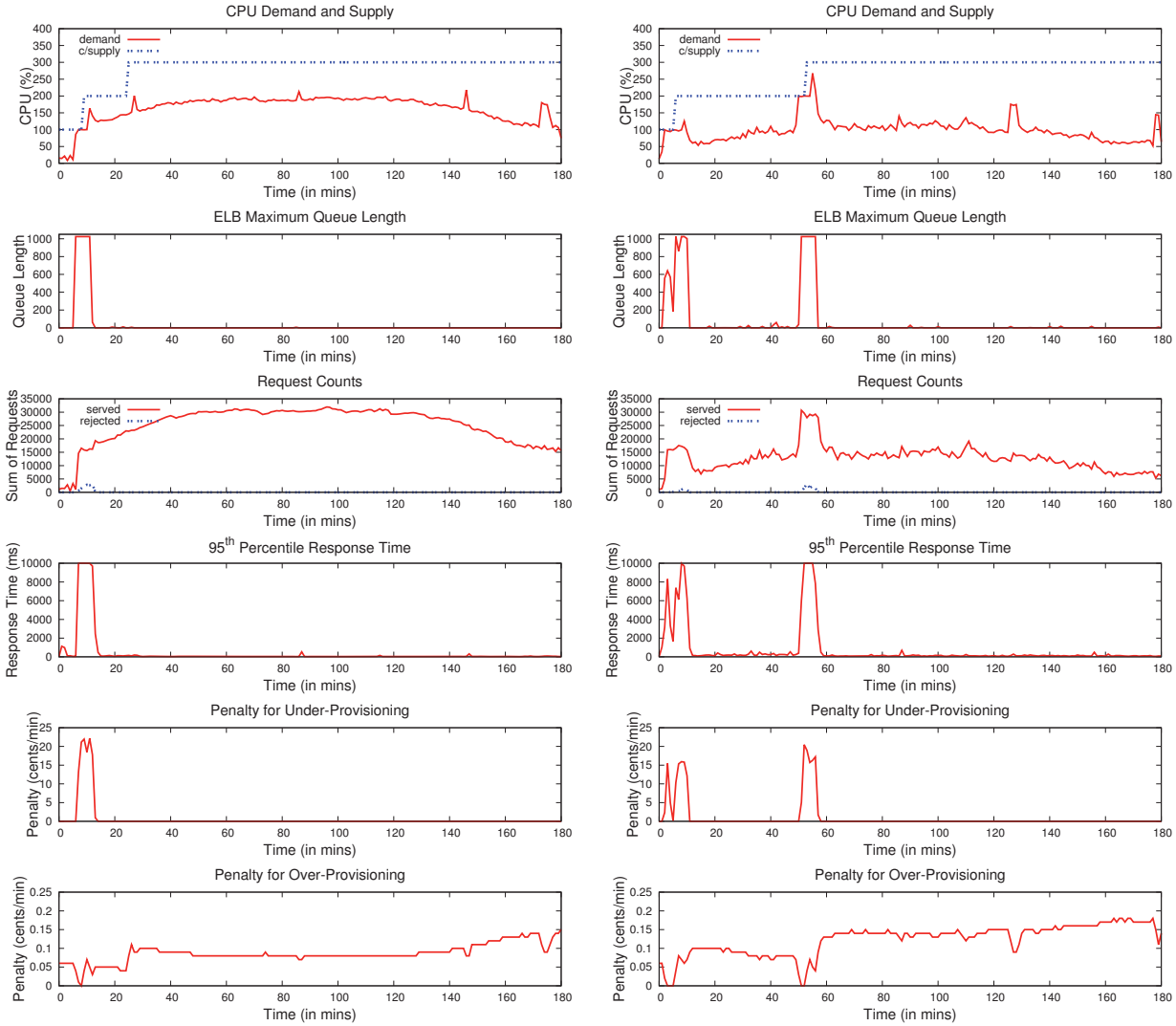
[6] aws.amazon.com/ec2/previous-generation/

[7] aws.amazon.com/s3/

[8] aws.amazon.com/rds/

[9] aws.amazon.com/elasticloadbalancing/

[10] jmeter.apache.org/



(a) case study: smooth workload

(b) case study: sigma150 workload

Figure 7: EC2 platform's elasticity behavior for non-bursty and bursty workloads (smooth vs. sigma150)

	P_{over}/hr	P_{under}/hr	P_{total}/hr
smooth ($\sigma = 0$)	8.70¢	2.04¢	10.74¢
sigma50 ($\sigma = 50$)	10.24¢	8.87¢	19.11¢
sigma100 ($\sigma = 100$)	10.20¢	10.21¢	20.41¢
sigma150 ($\sigma = 150$)	10.08¢	17.08¢	27.16¢
sigma200 ($\sigma = 200$)	9.77¢	14.04¢	23.81¢
sigma250 ($\sigma = 250$)	10.62¢	23.59¢	34.21¢

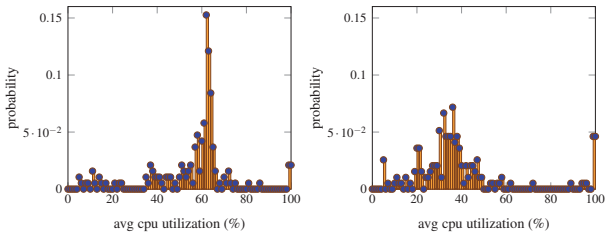
Table 2: Elasticity penalty for fine-scale burstiness

axis is in minutes; we couldn't sample data at the fine granularity of seconds due to the limitation of the CloudWatch[11] API. The sigma150 workload's CPU demand exhibits high variability; it is lightly loaded most of the time, except at the events of abrupt surges. In contrast, the smooth workload's CPU usage varies steadily; its average CPU consumption is also higher. For this reason, we observe lower over-provisioning penalty for the smooth workload.

The empirical probability density of the average CPU utilization for both workloads is shown in Fig. 8. If the request arrival process doesn't exhibit fine-scale burstiness, then there is a large mass clustered around $55\% - 70\%$ in the distribution of CPU utilizations. In contrast, the distribution of CPU utilizations appears to be extreme for the sigma150; the distribution is bimodal, one centering around 30% and the other one at 100%. Therefore, it is apparent that the EC2 server farm spent an increased fraction of time in the over-provisioned state while serving the sigma150 workload and incurred excess payment for idle resource capacity. The other mode at 100% indicates the fraction of time in the under-provisioned state (waiting for the scale-out rule

[11] aws.amazon.com/cloudwatch/

to trigger as the CPU demand had an abrupt surge) which is almost 2 times higher than the smooth workload; during this interval, an increased percentage of requests were served with very high response time and some requests were even dropped or timed out, thus accruing financial penalty because of SLA violation. Therefore, we see that fine-scale burstiness significantly deteriorates the elasticity of cloud-hosted applications. An adaptive scaling strategy, which is optimized and evaluated under non-bursty assumption, fails to guarantee effective elasticity when the actual workload has fine-scale burstiness.



(a) smooth workload (b) sigma150 workload

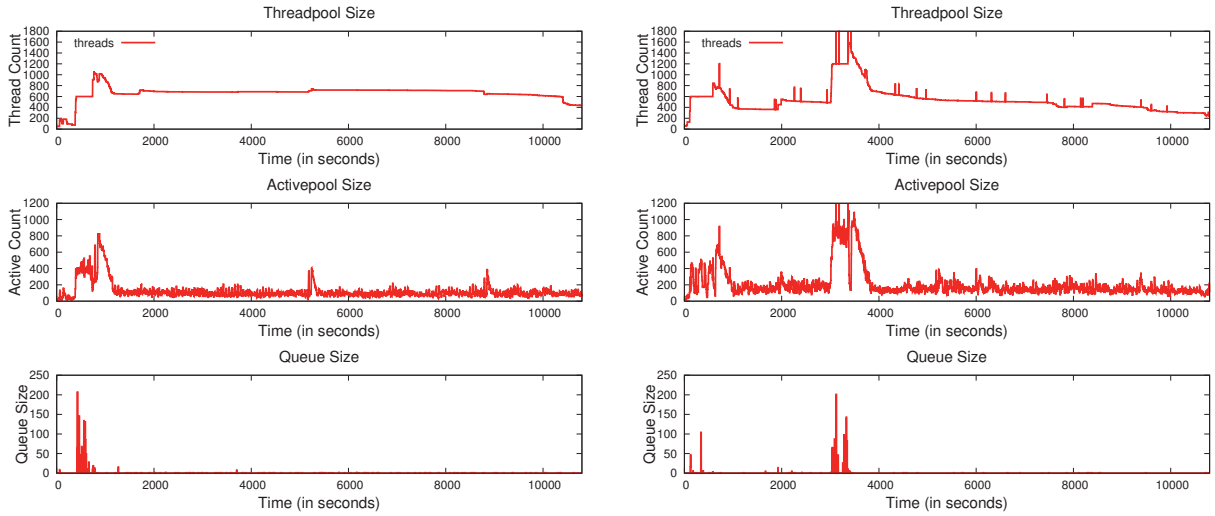
Figure 8: Estimated probability of average cpu utilization

Next we focus on the positioning of the scale-out points in the CPU graph (see Fig. 7). The first scale-out points for both workloads are relatively close (only 3 minutes apart); however, the second scale-out points appear to be quite apart (28 minutes apart). After closely inspecting the Holder exponents and the deterministic trend in Fig. 5, we infer that fine-scale burstiness influences adaptive resource scaling. Initially, the deterministic trend had steep growth as compared to the fine-scale fluctuations (up to 1640 second) and became the dominant factor in adaptive scaling. This is why the first scale-out for both workloads occurred almost at the same time. Afterwards the Holder exponents started to decline and caused increased variability in the request arrival rate. The application server served the fluctuating arrivals with a smaller thread pool (363 threads), as compared to that of the smooth workload (644 threads). Also, the concurrency level of the active pool got balanced by alternating episodes of high and low arrival rates (see Fig. 9 - note that the time-axis is in seconds[12]). This balancing-out effect reduced the CPU demand to some extent and delayed the provisioning of the third EC2 instance for the sigma150 workload.

We now turn our attention to the under-provisioning scenario. Workloads with fine-scale burstiness lead to higher under-provisioning penalty. The reason can be explained with respect to the abrupt surge of correlated request arrivals and its prolonged stay in the saturation regions for the sigma150 workload. This is quite surprising; because the input workload doesn't have any surge of several minutes, only transient fluctuations around the deterministic trend. This phenomenon can be explained with respect to the saturation region near the second scale-out point. A close inspection of the traces and the Holder graph reveals that at around 2800 second (47^{th} minute in the CPU graph of Fig. 7), the dips of the Holder exponents hit very low (around 0.065); this caused high variability in the request arrival rate. This variability provoked the application server to work at its maximum capacity with abrupt sharp increase in the threadpool, active pool and queue (see Fig. 9). At some point, the application server couldn't increase its service capacity enough to handle the episodes of very high arrival rate; so only a small fraction of the requests got processed immediately and the rest were kept waiting in the queue. When the queue got filled up, the application started rejecting requests with HTTP 503 error indicating service unavailability. Users whose requests got rejected and timed out in the queue, retried at a later timepoint; this is how a huge amount of requests got piled up during that interval and magnified the under-provisioning scenario. The RequestCount graph in Fig. 7(b) supports our conjecture, indicating the presence of abrupt surge in the application's served throughput. Most of the requests during this interval experienced very high response time because of queuing effect at the application server and the ELB; the long tails in the 95^{th} percentile response time graph is a direct reflection of this overloaded situation. The fraction of the requests, which found the service unavailable, contributed to the under-provisioning penalty even more. Fig. 10 depicts the overall response time percentiles for both bursty and non-bursty workloads; we see that the higher percentiles of the sigma150 pass through slower response time values (only 90.7% requests were served within 2 second threshold). It indicates that the presence of fine-scale burstiness induces deleterious effect on the response time of a significant portion of overall requests. Moreover, the autoscaling ruleset, configured at coarse-scale granularity (in the order of minutes), couldn't ensure effective elasticity in response to fine-scale bursty arrivals. This phenomenon suggests the need for fine-grained monitoring metrics and more agile adaptive policies to guarantee effective elasticity under fine-scale burstiness. Therefore we conclude that fine-scale burstiness has significant implications for under-provisioning; it may give rise to sudden large correlated surges that in turn cause long response time tails and high request defection rate, if not properly handled.

A few remarks on the performance variability effect. We observed the elasticity behavior for each workload in three different randomized experimental conditions by varying parameters, such as, time-of-the-day, availability zone etc. We noticed extreme CPU demand pattern for fine-scale bursty workloads in all experimental conditions. The under-provisioning penalty deteriorated for the fine-scale bursty workloads in all runs, however, the extent of degradation varied with the scale-out delay of the EC2 platform. Be-

[12] We collected these data using Command-line JMX Client (crawler.archive.org/cmdline-jmxclient/)



(a) case study: smooth workload

(b) case study: sigma150 workload

Figure 9: Behavior of the Tomcat application server under non-bursty and bursty workloads (smooth vs. sigma150)

cause of their highly volatile demand pattern and abruptly correlated request arrivals at the resource saturation region, the elasticity of the fine-scale bursty workloads exhibits increased susceptibility to the random scaling delay of the cloud platform.

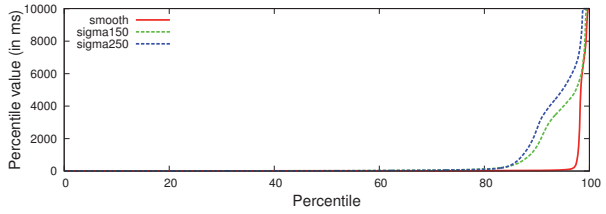


Figure 10: Response time percentiles

8.2 Trends in the Elasticity Penalty Rate

Fig. 11 illustrates the trends in the elasticity penalty rate as a function of the magnitude of the fine-scale burstiness σ. The over-provisioning penalty rate remains almost stable, perhaps because of the cheap price and resource heterogeneity of the cloud VMs. However, the under-provisioning penalty rate follows an upward trend. We also observe that the under-provisioning penalty rate stays lower than that of over-provisioning up until $\sigma = 100$. After that point, the under-provisioning penalty grows beyond its over-provisioning counterpart and starts dominating the total penalty. Since the over-provisioning penalty rate stays almost constant, the total penalty rate mimics the trend of the under-provisioning penalty curve. Therefore, we conclude that adaptive scaling strategies designed under non-bursty assumption performs worse in bursty condition; the degradation in elasticity penalty is proportional to the magnitude of the fine-scale burstiness in the arrival stream.

Our observations are summarized as follows:

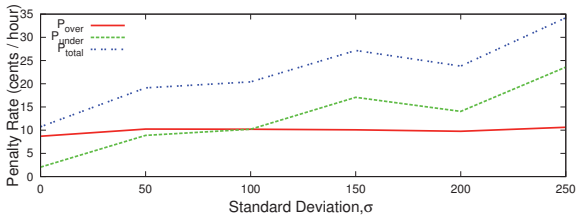


Figure 11: Trends in elasticity penalty rates

- *Fine-scale burstiness causes increased queuing effect and request defection rate, thus degrades the under-provisioning scenario with increased SLA violations. It also causes reduced resource utilization at the application server, thus results in high over-provisioning penalty.*
- *Both fine-scale burstiness and deterministic trend have influence on adaptive resource scaling.*
- *Elasticity penalty rate tends to get higher with the increase in fine-scale burstiness. There exists a cut-off threshold beyond which under-provisioning penalty dominates the total penalty.*

9. Conclusion

The analysis of the elasticity behavior under reproducible realistic workloads is an essential step in cloud adoption. Fine-scale burstiness is an inherent feature of the web request arrival process. In this paper, we provide a novel methodology to reproduce the burstiness so that it can resemble the stylized facts of the original arrival process. We exemplify the effectiveness of our methodology through an experimental case study in AWS cloud and extract valuable insights about its elasticity behavior in response to workloads with fine-scale burstiness. Our findings demonstrate that fine-scale

burstiness has significant implications for elasticity; it deteriorates the under-provisioning and over-provisioning scenarios with increased SLA violations and reduced resource utilization respectively. Therefore, ignoring the fine-scale burstiness in the design and analysis of adaptive systems may provide an over-optimistic view about the elasticity behavior which may turn out to be disastrous in practical scenarios. In future, we intend to repeat this case study for other cloud platforms and other web access traces to better understand the elasticity behavior under fine-scale burstiness. We also plan to come up with new resource management techniques that adapt well under bursty conditions. We consider this paper as an important step towards that direction.

Acknowledgments

We thank our shepherd, Dilma Da Silva and anonymous reviewers for their valuable feedback. We also thank Amazon Web Services for a generous research grant.

References

[1] P. Abry, P. Goncalves, and J. L. Véhel. *Scaling, fractals and wavelets*, volume 74. John Wiley & Sons, 2010.

[2] O. Barrière. *Synthèse et estimation de mouvements browniens multifractionnaires et autres processus à régularité prescrite: définition du processus auto-régulé multifractionnaire et applications*. PhD thesis, Nantes, 2007.

[3] J. Beran. *Statistics for long-memory processes*, volume 61. CRC Press, 1994.

[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*, pages 241–252. ACM, 2010.

[5] D. F. García and J. García. Tpc-w e-commerce benchmark evaluation. *Computer*, 36(2):42–48, 2003.

[6] N. Herbst and S. Kounev. Limbo: a tool for modeling variable load intensities. In *Proceedings of the 5th ACM/SPEC international conference on Performance engineering*, pages 225–226. ACM, 2014.

[7] E. A. Ihlen. Introduction to multifractal detrended fluctuation analysis in matlab. *Frontiers in physiology*, 3, 2012.

[8] INRIA. Fraclab: A fractal analysis toolbox for signal and image processing. URL `https://fraclab.saclay.inria.fr`.

[9] S. Islam, K. Lee, A. Fekete, and A. Liu. How a consumer can measure elasticity for cloud platforms. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, pages 85–96. ACM, 2012.

[10] J. G. v. Kistowski. Modeling variations in load intensity profiles. 2014.

[11] W.-C. Lau, A. Erramilli, J. L. Wang, and W. Willinger. Self-similar traffic generation: The random midpoint displacement algorithm and its properties. In *Communications, 1995. ICC'95 Seattle,'Gateway to Globalization', 1995 IEEE International Conference on*, volume 1, pages 466–472. IEEE, 1995.

[12] D. Menascé, V. Almeida, R. Riedi, F. Ribeiro, R. Fonseca, and W. Meira Jr. In search of invariants for e-business workloads. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 56–65. ACM, 2000.

[13] D. A. Menascé, V. A. Almeida, R. Riedi, F. Ribeiro, R. Fonseca, and W. Meira Jr. A hierarchical and multiscale approach to analyze e-business workloads. *Performance Evaluation*, 54 (1):33–57, 2003.

[14] N. Mi, G. Casale, L. Cherkasova, and E. Smirni. Injecting realistic burstiness to a traditional client-server benchmark. In *Proceedings of the 6th international conference on Autonomic computing*, pages 149–158. ACM, 2009.

[15] J.-F. Muzy, E. Bacry, and A. Arneodo. Multifractal formalism for fractal signals: The structure-function approach versus the wavelet-transform modulus-maxima method. *Physical review E*, 47(2):875, 1993.

[16] R.-F. Peltier, J. L. Véhel, et al. Multifractional brownian motion: definition and preliminary results. 1995.

[17] S. Pincus and B. H. Singer. Randomness and degrees of irregularity. *Proceedings of the National Academy of Sciences*, 93 (5):2083–2088, 1996.

[18] R. H. Riedi. Multifractal processes. Technical report, DTIC Document, 1999.

[19] B. Suleiman, S. Sakr, S. Venugopal, and W. Sadiq. Trade-off analysis of elasticity approaches for cloud-based business applications. In *Web Information Systems Engineering-WISE 2012*, pages 468–482. Springer, 2012.

[20] J. Tai, J. Zhang, J. Li, W. Meleis, and N. Mi. Ara: Adaptive resource allocation for cloud computing environments under bursty workloads. In *Performance Computing and Communications Conference (IPCCC), 2011 IEEE 30th International*, pages 1–8. IEEE, 2011.

[21] C. Tricot. *Curves and fractal dimension*. Springer, 1995.

[22] G. Urdaneta, G. Pierre, and M. van Steen. Wikipedia workload analysis for decentralized hosting. *Elsevier Computer Networks*, 53(11):1830–1845, July 2009.

[23] U. Vallamsetty, K. Kant, and P. Mohapatra. Characterization of e-commerce traffic. *Electronic Commerce Research*, 3(1-2):167–192, 2003.

[24] W. Willinger, M. S. Taqqu, W. E. Leland, and D. V. Wilson. Self-similarity in high-speed packet traffic: analysis and modeling of ethernet traffic measurements. *Statistical science*, pages 67–85, 1995.

[25] C. H. Xia, Z. Liu, M. S. Squillante, L. Zhang, and N. Malouch. Web traffic modeling at finer time scales and performance implications. *Performance Evaluation*, 61(2):181–201, 2005.

[26] J. Yin, X. Lu, H. Chen, X. Zhao, and N. N. Xiong. System resource utilization analysis and prediction for cloud based applications under bursty workloads. *Information Sciences*, 279:338–357, 2014.

[27] A. Youssef and D. Krishnamurthy. Cloud service level planning under burstiness. In *Performance Evaluation of Computer and Telecommunication Systems (SPECTS), 2013 International Symposium on*, pages 107–114. IEEE, 2013.