Bursting With Possibilities – an Empirical Study of Credit-Based Bursting Cloud Instance Types

Philipp Leitner and Joel Scheuner

software evolution & architecture lab
University of Zurich
Zurich, Switzerland
leitner@ifi.uzh.ch, joel.scheuner@uzh.ch

Abstract—**We study the performance and cost efficiency as perceived by the end user of a specific class of Infrastructure-as-a-Service (IaaS) cloud instances, namely credit-based bursting instances. This class of instance types has been introduced by Amazon EC2 in summer 2014, and behaves on a fundamental level differently than any other existing instance type, either from EC2 or other vendors. We introduce a basic formal model for fostering the understanding and analysis of these types, and empirically study their performance in practice. Further, we compare the performance of credit-based bursting cloud instance types to existing general-purpose types, and derive potential use cases for practitioners. Our results indicate that bursting instance types are cost-efficient for CPU-bound applications with an average utilization of less than 40%, as well as for non-critical IO-bound applications. Finally, we also discuss a simple boosting scheme that enables practitioners to improve the cost efficiency of their bursting instance usage under given constraints.**

I. Introduction

Cloud computing [1] is gaining more and more traction as a way to easily and cost-efficiently deploy services over the Web. Within the cloud computing paradigm, two distinct models have emerged. In Platform-as-a-Service (PaaS), cloud customers rent a complete managed runtime environment from the provider. While appealing, this model is known to come with various restrictions, including limited access to backend instances, vendor lock-in, and restrictions in terms of supported application models [2]. Hence, consumers often fall back to Infrastructure-as-a-Service (IaaS). In IaaS, consumers directly rent virtualized computing resources from the cloud, typically in the form of virtual machines (VMs) and virtualized hard disks. Instance types are the typical abstraction that IaaS providers use to define the amount of resources available to any given VM. Most providers make available a large number of different instance types, categorized into various "families" of types for different use cases (e.g., general-purpose, IO-optimized, CPU-optimized). A particularly interesting instance type family are "bursting" instance types, e.g., `t1.micro` in Amazon Web Services (AWS) and `f1.micro` in Google Compute Engine (GCE). These instances share all computing resources, including their CPU, with other tenants. Hence, they are typically the cheapest available option in a cloud. Unsurprisingly, recent studies have found that bursting instance types are particularly prone to performance unreliability due to noisy neighbors and unpredictability of the scheduler [3], [4].

However, in summer 2014, AWS has made the second generation of bursting instance types publicly available, in the following referred to as the `t2` family. Unlike previous types and the offerings of competitors (which are typically best-effort oriented and consequently highly unpredictable), `t2` types now operate on two distinct performance levels, a peak and a baseline performance level. Each instance has a replenishable account with credits for running on peak performance and drops to baseline performance when its credits run out. While this specific model is currently only available in AWS EC2, we assume that other providers will soon follow with similar offerings. Hence, we provide a first empirical and analytical study of the implications of this new instance type family for practitioners. Concretely, we address the following research questions:

RQ1: How do `t2` instance types perform in terms of CPU and IO speed in comparison to other instances?
RQ2: When are `t2` bursting instance types more cost-efficient than other instance types?
RQ3: How do `t2` instance types perform in comparison to the previous generation (`t1`) types?

Note that in this research we focus on the `t2.micro`, `t2.small`, and `t2.medium` types. At the time of this writing, AWS has additionally introduced another type to this family, `t2.large`, which was not yet available during the experimental phase of our study.

We introduce a basic model that formally captures the performance behavior of these instance types for analysis. Following, we sketch a number of practical use cases and discuss the characteristics of applications for which `t2` instances are the cheapest option. We empirically show that general-purpose instances are more cost-efficient for highly-loaded services. However, for services with an average utilization of 40% or less, `t2` instances provide vastly better performance per US dollar spent. We also show that `t2` instances are attractive for smaller, non-critical IO-bound services, such as small databases. Finally, we discuss the basic idea of credit boosting, a simple scheme that allows cloud customers to improve the performance-cost ratio of `t2` instances.

II. Credit-Based Bursting Instance Types

As foundation for the remainder of this study, we now formally define the underlying model of credit-based bursting instance types, and explain how this model is currently implemented in Amazon EC2.

A. Basic Model

Consider a cloud consumer who is renting a set I of bursting cloud instances. Each instance $i \in I$ has a defined instance type $i^t \in T$, where T is the set of available credit-based bursting instance types (e.g., `t2.micro`). Each instance $i \in I$ can operate on two defined CPU performance levels, a *peak performance level* $s_p(t)$ and a *baseline performance level* $s_b(t)$, with $s_p(t), s_b(t) \in \mathbb{R}^+$. Both, the peak and baseline performance level are dependent on the concrete instance type. Further, we assume performance levels to be defined by a positive real number ($\mathbb{R}^+$), where lower numbers represent better performance. That is, the performance level is assumed to represent the time it takes an instance to execute a defined benchmark task (e.g., find all prime numbers between 0 and 10.000 using the Sieve of Eratosthenes). Generally, for all instances, the peak performance level is substantially preferable to the baseline performance level to the cloud consumer ($\forall i \in I : s_p(i^t) << s_b(i^t)$). At this point we ignore the fact that different CPUs in practice perform differently for different types of CPU operations (e.g., floating point arithmetics versus integer arithmetics). Due to how credit-based bursting instance types are technically implemented by cloud operators (see Section II-B), this difference is of little concern for this study.

The applicable performance level at any point in time depends on the amount of *credits* an instance currently has available, defined as $i_c \in \mathbb{R}^+$. An instance can operate at peak performance level as long as it has a positive amount of credits ($i_c > 0$). Whenever an instance operates at peak performance (the CPU is non-idle), its credits will *deplete* with a rate of $t_d \in \mathbb{N}^+$ per hour. Simultaneously, its credits *replenish* with a constant rate of $t_r \in \mathbb{N}^+$ per hour as long as the instance is running (i.e., not in the "stopped" or "terminated" state), independently of instance usage. For practical reasons, the rate of replenishment is typically substantially lower than the rate of depletion ($\forall t \in T : t_r << t_d$). An instance is throttled to baseline performance level if it has completely depleted its credits.

Instances typically do not start with empty credit. Rather, instances receive an initial amount of credits on startup $t_s \in \mathbb{N}^+$, which allows new instances to immediately operate at peak performance for a predictable amount of time after startup. Conversely, idle instances cannot accrue credits forever, as the maximum amount of credits per instance is capped at $t_m \in \mathbb{N}^+$. For all instance types, this upper limit t_m is designed so that an instance can build up credits for up to 24 hours at a time. As by definition the credit balance can never deplete below 0 (instances cannot operate at peak performance after fully depleting their credits), this means that $\forall i \in I : 0 \leq i^t_c \leq i^t_m$ at any time.

Note that this process of credit depletion and replenishment is continuously executed in the background by the cloud provider. Credit depletion operates on a millisecond basis and with fractional credits. That is, in practice, assuming a freshly started instance i has an average CPU load of 50% over its first hour of operation, the instance's credit balance will be $i_c = i^t_s - \frac{i^t_d}{2} + i^t_r$. Assuming that $i^t_s > \frac{i^t_d}{2}$, the instance is guaranteed to be able to able to operate at a CPU performance of $s_p(i^t)$ through the hour. However, given that typically $\frac{i^t_d}{2} > i^t_r$, the instance will use parts of its credits in the process. In order to actually accumulate credits, the average CPU utilization of the instance will need to be below $\frac{i^t_r}{i^t_d}$. We denote this utilization rate, which keeps the instance credit account constant, as the *standard instance utilization* $t_{\bar{u}}$. Finally, an instance has defined hourly costs $c(t) \in \mathbb{R}^+$, which depend on its type and are represented as US dollars per started billing time unit (BTU), e.g., one hour.

B. Implementation in Amazon EC2

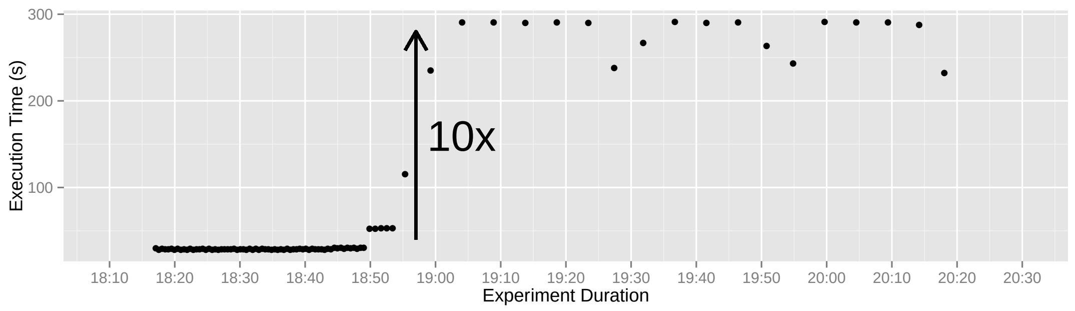
This model is already implemented and publicly available in the (at the time of writing) current generation of bursting instance types (`t2`) in AWS EC2. Four different concrete instance types that follow this model are available: `t2.micro`, `t2.small`, `t2.medium`, and `t2.large`. In the following, we focus on the former three instance types.

Name (`t2.*`)	$s_p(t)$	$s_b(t)$	t_s	t_m	t_d	t_r	$t_{\bar{u}}$
`micro`	2	20	30	144	60	6	10%
`small`	2	10	30	288	60	12	20%
`medium`	1	2.5	60	576	120	24	20%

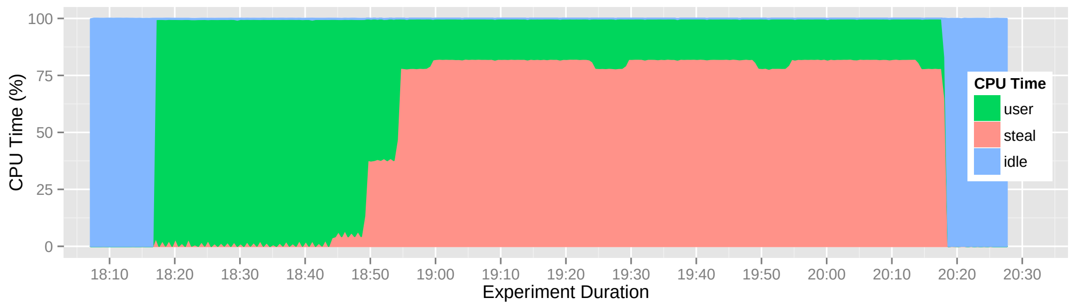
Table I: Basic model parameters of the `t2` instance type family in AWS EC2.

All types are backed by the same Intel Xeon processors with 2.5GHz standard CPU frequency and the possibility to (for shorter terms) go up to a frequency of 3.3GHz. All of them differ in the provided baseline performance, as well as in most other model parameters. Table I summarizes the specifications in the notation of our basic model. The information in this table is largely based on the official EC2 documentation[1]. The values given for $s_p(t)$ and $s_b(t)$ should be seen as relative metrics, which only make sense in comparison to the other performance values in this table. The values for t_s and t_m represent credits, where each credit allows for using a single CPU core for one minute at peak performance. That is, given that the `t2.medium` instance type has two cores, fully utilizing this instance for 1 minute uses twice as much credits as the other instance types, hence the twice as large depletion rate t_d. Both t_d and t_r are given as hourly rates. Thus, a

[1]http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/t2-instances.html



(a) Repeated execution of `sysbench` on a `t2.micro` instance. At 18:50, i_c approaches depletion and the CPU performance of the instance is throttled to its baseline level.



(b) To the cloud user, this throttling is visible as large amounts of CPU time being "stolen" by the hypervisor.

Figure 1: Repeated execution of sysbench on a `t2.micro` instance

`t2.micro` instance being idle for an entire hour accrues enough credits allowing it to operate at peak performance for 6 minutes. The `t2.micro` type has a normal instance utilization $t_{\bar{u}}$ of 10% whereas the other `t2` types are designed to be used 20% of the time.

Once a bursting instance depletes its entire credit, the hypervisor managing this instance will schedule less CPU time to this instance, hence slowing it down to the baseline performance level. To the user, this appears as substantial levels of *CPU steal time* in the instance. Note that, unlike in best-effort based bursting instance types, such as the older `t1` generation in EC2 or the `f1` types in GCE, this throttling is entirely independent from whether another tenant is actually using the "stolen" CPU time. Users can at any time query AWS CloudWatch for the current credit balance, or the credit usage during a specific time period.

Figure 1 illustrates these principles for a `t2.micro` instance in the `eu-west-1` region. We have started an instance of this type using a standard Ubuntu 14.10 base image[2] and then, in an infinite loop, executed the standard Linux `sysbench`[3] CPU benchmark utility to produce CPU load and measure the instance's performance.

In Figure 1a, we observe remarkably stable performance (substantially more so than what we and others have experienced for other instance types with shared CPUs [3], [4]) up to around 18:50. At this point, performance drops to about 10% of its peak performance level (i.e., benchmark execution time increases by factor 10) because the instance is running low on credits. From then on, the observed performance was more variable, as the instance performs most of its time at baseline level, with short bursts to peak performance whenever it received a small amount of credits from the hypervisor. Note that, in difference to the basic model described in Section II-A, AWS does not abruptly throttle instances all the way to their baseline performance once they run out of credits. Rather, and as we can see between 18:50 and 19:05, instances are smoothly throttled once the are running low on credits. Figure 1b illustrates how throttling appears to the instance owner. Until 18:50, (user) CPU utilization is close to 100% and steal time is close to 0%. When the instance is being throttled, steal time rises to a quasi-constant 80%, and the actual CPU time available to the user drops to 20%. As before, even at baseline performance, there are short bursts of slightly increased available CPU time whenever the instance receives a small amount of credits. The reported idle time provides additional confidence that our CPU benchmark actually stressed the CPU with 100% load (i.e., 0% idle) between 18:17 and 20:18.

III. Analysis and Comparison

We now benchmark what actual performance the values from Section II-B map to, and how this performance compares to other commonly used and previously studied instance types.

[2]https://cloud-images.ubuntu.com/locator/ec2/
[3]http://manpages.ubuntu.com/manpages/raring/man1/sysbench.1.html

A. Study Setup

We have used Cloud Workbench [5], [6] (CWB) and the `sysbench` benchmark to collect empirical performance data in two dimensions: CPU performance (integer arithmetics) and disk IO (combined hard disk read/write speed). We have evaluated all studied `t2` instance types on their peak and baseline performance levels, along with the general-purpose instance types `m3.medium` and `m3.large` and the smallest compute-optimized type `c4.large`. To measure peak performance of bursting instances, we started an instance and immediately ran the respective benchmark. As all benchmark executions lasted less than 30 minutes, t_s on all instance types was sufficient to guarantee that instances always ran at peak performance level in this setup (cp. also Table I). To measure the baseline performance, we again acquired an instance of the specific type, fully depleted its startup credits by repeatedly executing a CPU benchmark for 70 minutes, and then executed the targeted benchmark. For each configuration, we collected 50 data points. All data was collected between May, 1st and May, 15th 2015 in the `eu-west-1` region of AWS EC2. All data, as well as the CWB files used to define the benchmarks and analysis code for the R statistical computing program, are available on GitHub[4].

B. Results

We now discuss our benchmarking results in terms of CPU and IO performance and cost efficiency. Further, we compare `t2` bursting instance types to the previously-studied `t1` generation.

1) CPU Performance Comparison: For benchmarking CPU performance, we used the `sysbench` CPU benchmark, which is a focused micro-benchmark measuring the performance of integer arithmetics. The result of the benchmark is the duration it took to execute a defined set of arithmetic operations. We follow a similar procedure as in [7] and normalize our results using the mean performance of a single `m3.medium` instance as a baseline for comparison (59 seconds). We refer to this unit as *medium-instance equivalents*. `sysbench` CPU results s are converted to medium-instance equivalents using the transformation $\theta : \mathbb{R} \rightarrow \mathbb{R}$, defined as $\theta(s) = \frac{59}{s}$. Intuitively, a performance of *x medium-instance equivalents* for an instance type means that a user would need to acquire *x* `m3.medium` instances to achieve the same mean integer arithmetics performance.

Figure 2 depicts the benchmarking results for all instance types in our study. For `t2` types, we benchmarked both the performance on peak and baseline performance in Boxplot notation. The dashed horizontal line represents the baseline performance of a single `m3.medium` instance. This data is also summarized in Table II and Table III. We use $\bar{m}$ to denote the arithmetic mean of all 50 benchmark observations, and m_σ for the relative standard deviation in percent.

We have observed empirical results close to what we expected based on Table I. All bursting instance types are substantially faster than `m3.medium` on peak performance level s_p, and slower on baseline level s_b. For `t2.micro`, we achieved a performance of 0.21 medium-instance equivalents on baseline versus 2.06 on peak performance level. This is close to the expected 10-fold speedup between peak and baseline. Peak performance of `t2.micro` and `t2.small` are comparable and close to twice the performance achieved by a `m3.medium` instance, while the peak performance of `t2.medium` is about 2 times faster compared to the other `t2` types. The performance of all `t2` instance types is rather predictable on both performance levels, with relative standard deviations between 3% and 8% of the mean. However, we experienced a small number of outliers in our experiments, which suggests that even in `t2` bursting instance types, cloud users still occasionally need to deal with slow instances potentially due to noisy neighbours and shared CPUs.

	t2.micro (0.014 $ / h)		t2.small (0.028 $ / h)		t2.medium (0.056 $ / h)	
	s_p	s_b	s_p	s_b	s_p	s_b
$\bar{m}$	2.06	0.21	1.98	0.41	3.99	0.87
m_σ	3%	8%	4%	6%	5%	6%

Table II: CPU benchmarking results for `t2` instance type as medium-instance equivalents. Prices are for Linux instances in the `eu-west-1` region, and as of May, 21st, 2015.

We also put these results in relation to other common current-generation instance types. `m3.medium` and `m3.large` are the two cheapest general-purpose instance types, and `c4.large` is the cheapest CPU-optimized instance type. `t1.micro` is the cheapest previous-generation bursting instance type. Data for this instance type is taken from our previous study and also publicly available [3].

	m3.medium 0.077 $ / h	m3.large 0.154 $ / h	c4.large 0.132 $ / h	t1.micro 0.02 $ / h
$\bar{m}$	1	3.51	4.19	1.41
m_σ	≈0%	≈0%	≈0%	28%

Table III: CPU benchmarking results for other current-generation instance types as medium-instance equivalents. Prices are for Linux instances in the `eu-west-1` region, and as of May, 21st, 2015.

These results show that, on peak performance, all `t2` instances outperform `m3.medium` instances by at least a factor of 1.98. However, on baseline performance, all general-purpose instance types outperform bursting instance types in terms of performance and performance predictability. In fact, we have not experienced any relevant variability (<0.15) in our CPU benchmarking results for `m3.medium`, `m3.large`, and `c4.large`. Our comparison with previous-generation bursting instances of the `t1.micro` type has shown that the peak performance of such instance types is substantially slower than of current-generation bursting instances (1.41 medium-instance equivalents versus 2.06 for the smallest bursting instance types). Further, and more interestingly, these instances are substantially less predictable in terms of performance, with a relative standard deviation of 28% of the mean even on peak

[4]https://github.com/sealuzh/bursting-cloud-instances

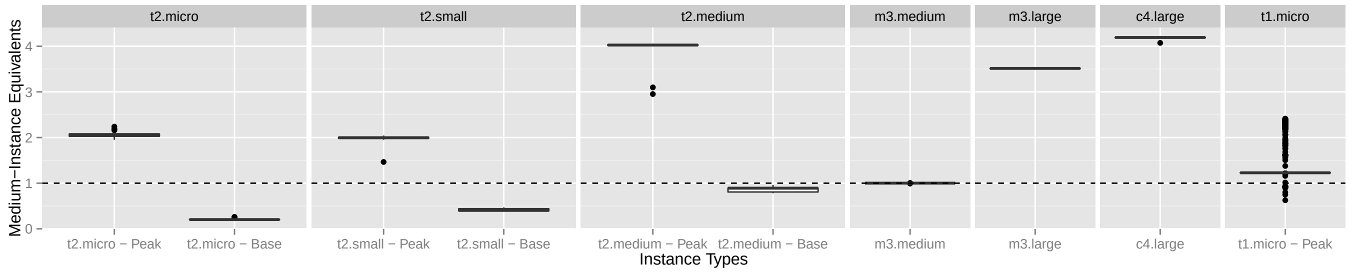


Figure 2: CPU performance of all analyzed instance types. Performance of types of the `t2` family is reported at peak and baseline performance.

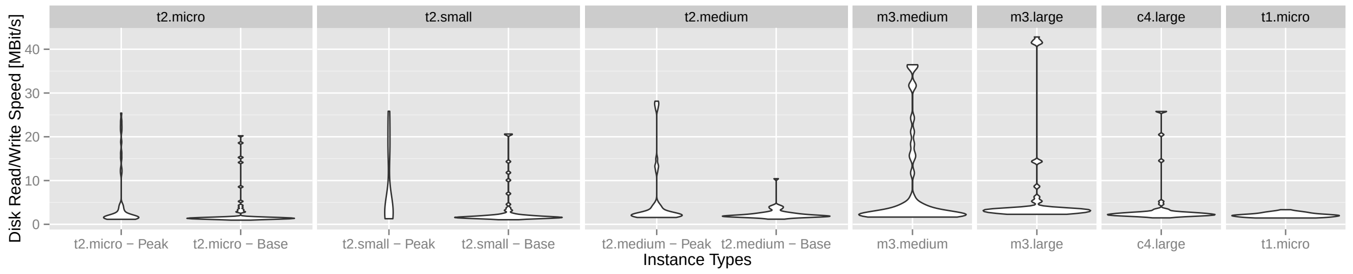


Figure 3: IO performance of all analyzed instance types. Performance of types of the `t2` family is reported at peak and baseline performance.

performance. This is due to the fact that these instance types are not always served with the same hardware model, unlike all current-generation AWS instance types [3].

2) IO Performance Comparison: In addition to CPU performance, we also evaluated the different instance types in terms of their combined disk read/write speed. We have again used the `sysbench` implementation of an IO micro-benchmark, which repeatedly reads and writes large files (4 GByte) to the hard disk, and measures the combined read/write speed in MBit per second.

Consistently with earlier research [3], [5], [8], we have seen IO performance vary much more between benchmarking runs than CPU speed. Arguably, this is due to IO performance being much more susceptible to noisy neighbors and the detrimental effects of cloud multi-tenancy. However, in addition, we have also seen that for all analyzed instance types, IO benchmarking results were not normally distributed. The measured IO performance of all instance types typically was between 2.5 MBit/s and 7 MBit/s, but all instance types had occassional outlier instances that performed an order of magnitude better (between 20 MBit/s and 40 MBit/s). These outliers performed roughly on the same IO performance level that we have seen in an earlier study [3], while the bulk of instances nowadays performs substantially worse than a year ago. We assume that this is due to AWS currently changing their overall approach to IO management, including a stronger focus on Provisioned IOPS[5].

[5]https://aws.amazon.com/about-aws/whats-new/2014/06/16/introducing-the-amazon-ebs-general-purpose-ssd-volume-type/

	`t2.micro`		`t2.small`		`t2.medium`	
	s_p (MB/s)	s_b (MB/s)	s_p (MB/s)	s_b (MB/s)	s_p (MB/s)	s_b (MB/s)
$\bar{m}$	4.9	3.1	7.8	3.3	5.3	2.4
m_σ	138.4%	142.5%	108.5%	108.5%	137.3%	57.6%

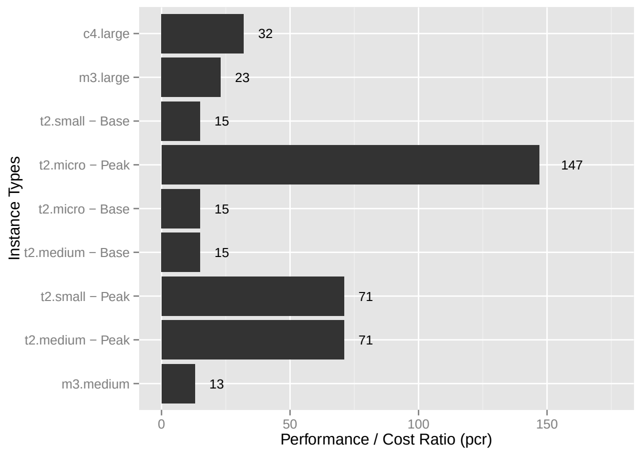
Table IV: IO benchmarking results for `t2` instance types in MBit per second. Prices are as in Table II.

Due to the non-normality of IO data, we plot our results in Beanplot notation [9] rather than as Boxplots (Figure 3). Generally, while we have seen statistically significant differences between instance types, as well as between performance on peak and baseline level for `t2` instances, these differences are very small in comparison to the substantial deviation of all results (see also Table IV for concrete values for `t2` instances, and Table V for all comparison instance types). Note that we have not experienced any positive outliers for previous-generation bursting instances (`t1.micro`),

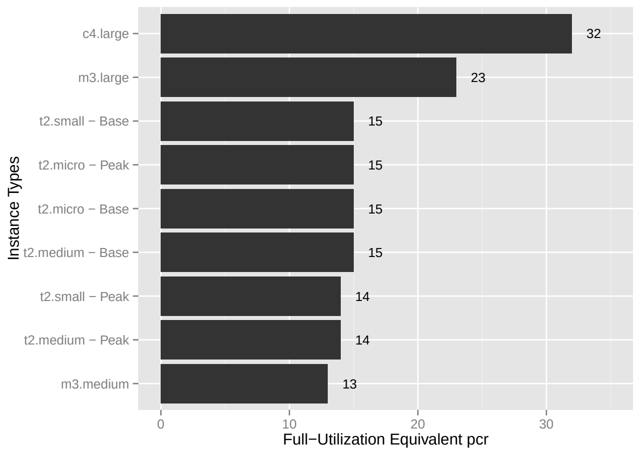
	`m3.medium` (MB/s)	`m3.large` (MB/s)	`c4.large` (MB/s)	`t1.micro` (MB/s)
$\bar{m}$	7.75	6.94	3.99	2.11
m_σ	135.7%	156.1%	137%	22.7%

Table V: IO benchmarking results for other current-generation instance types in MBit per second. Prices are as in Table II.

Summarizing, our detailed IO benchmarking results remain largely inconclusive, arguably because we appear to have analyzed EC2 during a transitional period. However, it is interesting to note that, by and large, differences between different instance types, as well as between peak and baseline



(a) Comparison of performance-cost ratios *pcr(t)* of different instance types.



(b) Comparison of full-utilization equivalent performance-cost ratios *unpcr(t,100)*.

Figure 4: Comparison of cost efficiency

performance for `t2` instances, are not as relevant anymore as what we and others have experienced in previous studies [3]. This indicates a longer trend towards more homogeneous IO performance across instance types, which has implications for practitioners and researchers.

3) Comparison of CPU Cost Efficiency: So far, we have discussed and contrasted the performance of instance types independently of their hourly costs. As indicated in [10], such an isolated view is often of limited usefulness in a cloud computing context. Rather, practitioners are typically interested in the performance per US Dollar spent of an instance type, which we now discuss.

We define the *performance-cost ratio* as $pcr(t) = \frac{\bar{m}}{c(t)}$. The unit of *pcr* are medium-instance equivalents per US dollar and hour. We visualize pcr for all configurations and CPU performance in Figure 4a. Evidently, on peak performance level, all bursting instance types provide tremendous per-cost values. For instance, the cheapest type `t2.micro` surpasses the compute-optimized `c4.large` type in CPU performance per USD almost by a factor of 5 as long as the instance's credits are not depleted. Conversely, on baseline performance level, bursting instance types are less cost-efficient than all other types in our study with the interesting exception of `m3.medium`.

However, looking at the pcr alone is misleading, as this metric does not consider the fact that, in order to get the performance-cost ratio indicated above, users can only utilize a bursting instance a fraction of the time (corresponding to $t_{\bar{u}}$ in Table I, e.g., 10% for `t2.micro`), and need to let the instance idle to replenish credits the rest of the time. Hence, it makes sense to consider a second, related metric, the *utilization-normalized performance-cost ratio* ($unpcr$). Intuitively, $unpcr$ can be interpreted as the costs of operating a cluster of bursting instances, so that one instance can always be operated at peak performance under the assumed utilization level. For an utilization of 100% and the `t2.micro` type, this can be achieved by acquiring 10 instances and alternating requests between them, so that each instance is idle 90% of the time and the credit balance of each instance remains stable indefinitely. We formally define $unpcr(t, u)$ as $unpcr(t, u) = \frac{pcr(t)}{\lceil \frac{u}{t_{\bar{u}}} \rceil}$ for a given instance type $t \in T$ and an utilization level $u \in [0; 100]$. In this definition, the term $\lceil \frac{u}{t_{\bar{u}}} \rceil$ represents the number of bursting instances that are required to indefinitely operate at peak performance under the assumed utilization level. $\lceil\ \rceil$ denotes rounding up to the next full natural number, as it makes little sense to consider fractions of virtual machine instances. Figure 4b visualizes $unpcr$ for full utilization ($u = 100$) and all configurations. For all non-bursting types, as well as for bursting types at baseline performance, $pcr(t) = unpcr(t, 100)$ by definition. This visualization shows that currently, all `t2` types are designed with a similar $unpcr(t, 100)$ target of 14 to 15. `m3.medium` instances are slightly less cost-efficient with an $unpcr(t, 100)$ of 13, while we see economies of scale become relevant for the larger `m3.large` and `c4.large` instance types. These results reinforce the rather intuitive notion that bursting instance types are very cost-efficient if used only sporadically, but quickly become inefficient in sustained usage, i.e., when used with high utilization. Our results also show that there is currently no clear advantage to using `m3.medium` instances, as they are less cost-efficient than bursting instance types even in sustained use.

4) Comparison to Previous-Generation Bursting Instances: Another interesting question is how current-generation `t2` instance types compare to the previously available `t1` types, most importantly `t1.micro`. Both instance type families share similar basic ideas, but the actual implementation varies considerably. Ultimately, `t1.micro` is a high-variance, best-effort based instance type, while current-generation bursting instance types follow a largely predictable performance trajectory, as discussed in Section II-A. This is illustrated in

Figure 5, which visualizes the repeated execution of two example `sysbench` CPU benchmarks on a `t1.micro` and a `t2.micro` instance. The trace for `t1.micro` has been taken from [3]. Both setups used the same benchmark in the same configuration, as well as the same benchmarking tool chain.

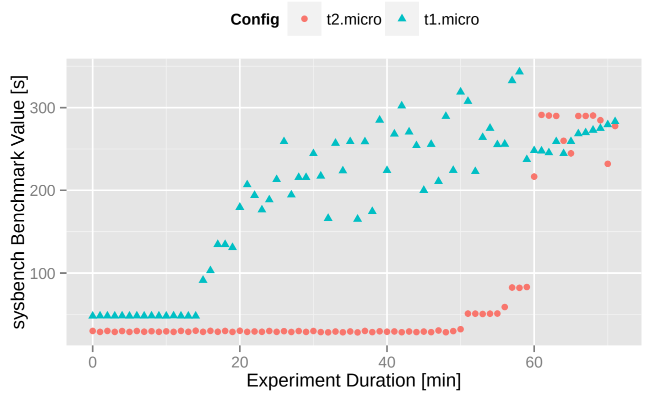


Figure 5: Comparison of performance development of stressed `t1` and `t2` instances.

Both instances start on a constant peak performance level. After a period of sustained usage, the performance of both instances deteriorates. For the `t1.micro` instance, this means that the instance largely behaves erratically, getting CPU time whenever other co-located tenants are not using their share. The `t2.micro` instance exhibits predictable performance levels at peak and baseline performance, with a brief 15-minute period of graceful performance degradation. This performance is largely independently of the usage patterns of the instance's neighbors. Another important observation is that `t2` instances are always served with the same hardware model, unlike `t1.micro` instances, as well as unlike previous-generation general-purpose instance types [7], [11]. This naturally increases the predictability of performance for the cloud consumer.

IV. Usage Scenarios

Based on the empirical data presented in Section III, we now discuss three practical usage scenarios for `t2` instances.

A. *Hosting Services with Low or Irregular Load.*

Given our empirical result that bursting instance types offer superior performance per US dollar spent as long as instances are given time to replenish their credit, an obvious usage scenario is to use them for services or applications with low or irregular overall utilization. These include new services, products, or Web servers of small start-up companies, which simply do not yet have a large, established customer base. Alternatively, bursting instance types are also attractive for services whose usage is subject to substantial variation over the time of a day. This can include, for instance, regional commercial services, which are primarily used during working hours. In such scenarios, bursting instances can replenish credits during off-times in order to operate at peak performance during peak hours.

Conversely, our results have also shown that for highly-loaded CPU-bound applications, larger general-purpose or compute-optimized instance types provide better performance-cost ratio pcr. Hence, a relevant question is where the cutoff point is. This is visualized in Figure 6. We have depicted the utilization-normalized performance-cost ratio for increasing average utilization ($unpcr$). Up to 40% average utilization, bursting instance types offer the best per-cost CPU performance. Starting with 40% utilization, `c4.large` offers better pcr, while bursting instances still outperform `m3.large` on this utilization level. This changes at 60% utilization, from which on `m3.large` is also more cost-efficient than any bursting instance type. `m3.medium` is generally less cost-efficient than any bursting or non-bursting alternative, even under 100% usage. It should be noted that these cutoff points are valid only under the assumptions that (1) the service is primarily CPU-bound, and (2) the user requires peak performance whenever the service is actually used. Specifically, if a user is willing to operate at baseline performance for some percentage of the time, she may be able to operate with less instances to satisfactory performance. Hence, in this case, the cutoff will shift to the right.

B. *Hosting Non-Critical IO or Network-Bound Services.*

Another interesting empirical result of our study was that there currently is a trend towards more homogeneous IO performance across instance types. This suggests that bursting instance types are an attractive alternative for some services that are IO-bound (e.g., small databases or file servers). As the IO performance of bursting instances does not degrade substantially even at baseline performance, this remains true even if the average utilization is close to 100%. Due to how IO is typically implemented in public clouds, network and IO performance is usually strongly correlated. Hence, we speculate that the same results also hold for services that are primarily network-bound, (e.g., small Web servers).

However, users need to keep in mind that IO performance is, in absolute terms, very low for all smaller current-generation instance types without Provisioned IOPS. Further, as discussed in Section III, the variability in terms of IO performance is tremendous for all studied instance types, indicating that all studied instance types should not be used for IO or network-bound applications where stability and predictability of performance is critical, such as many customer-facing or business-critical applications.

C. *Boosting PCR via Systematic Instance Restarting.*

Finally, due to how bursting instance types are currently implemented, users are able to *boost* the pcr of their instances to some extent. The basic feature that enables boosting is that bursting instances receive an initial amount of credits on startup or reboot $t_s > 0$. As discussed in Section II-B, t_s is currently designed to allow instances to operate on peak

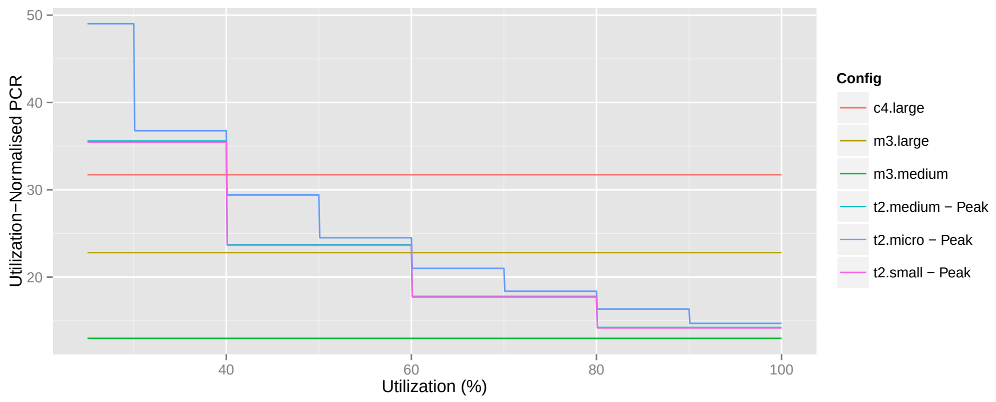


Figure 6: *unpcr(t,u)* of different instance types with increasing utilization *u*.

performance level for 30 minutes after startup. Hence, one simple strategy to maximize performance per US dollar is to start a bursting instance, use it until all its credits are depleted, and then reprovision or reboot the instance to refresh its startup credits. Given that (re-)starting an instance takes less than 5 minutes [12], this allows users to operate a bursting instance at peak performance level for one hour per at most 10 minutes off-time, or 85% of the time. Depending on the concrete instance type, this leads to a 2- to 4-fold increase of $unpcr(t, 100)$, and, consequently, substantial cost savings for the user.

However, there are limits to the practical usefulness of this scheme. Firstly, for practical reasons, this scheme is primarily useful for redundantly hosted services, such as Web servers in an auto-scaling group (ASG). Secondly, there is a hard limit to the number of bursting instances any AWS account can create within 24 hours in a region (at the time of writing set to 100 instance starts or reboots), arguably to prevent excessive abuse of this boosting scheme. After this limit is exceeded, new bursting instances start with an empty credit balance ($t_s = 0$). Ultimately, this means that not more than 2 `t2.medium` instances can be operated full-time using this scheme per region and AWS account.

V. Threats to Validity

As with any empirical research, there are threats and limitations to our work, which we discuss in the following.

Construct Validity. During study design, a number of design decisions had to be made. While there was no choice to select a different cloud provider aside from EC2 (as credit-based bursting instance types were not yet available elsewhere at the time of writing), we had to select regions, benchmarks, and comparison instance types. Our primary approach here was to select regions and types that have already been extensively studied in related work, so that we are able to put our work in context to the larger body of research.

Internal Validity. While the 50 repetitions of each benchmark configuration were sufficient to accurately represent CPU performance, it can be argued that our data set for IO was not extensive enough. Although we acknowledge this threat, we are confident that our fundamental conclusions are well supported by our data. While monitoring CPU steal time during benchmark executions (vgl. Figure 1b), we found no evidence that our experiment could suffer from noisy neighbors as pointed out in [13] for previous generation instances. Additionally, we have chosen to apply all benchmarks in their out-of-the-box standard configuration, without any instance-specific parameter tuning, to avoid biasing our study towards specific instance types via unbalanced optimization. This means that the interested reader will likely be able to improve the concrete numerical results we have achieved for each instance type by manually optimizing the benchmark configuration.

External Validity. The primary threat to the generalizability of this work is that we have only looked at performance using micro-benchmarking. While this is a common methodology that also other seminal papers in the field have employed (e.g., [14], [15]), this leaves the question open to what extent our results are also valid for real-life applications that make use of a wide range of computational resources at the same time. We plan to conduct additional research using application benchmarks (e.g., RUBiS [16] or Cloudstone [10]) to mitigate this threat in the future.

VI. Related Research

Our work follows an established subfield of cloud computing research, the benchmarking and empirical evaluation of public IaaS cloud services. Ostermann *et al.* [17] presented one of the first comprehensive performance evaluation study in a cloud computing environment. Existing studies at this time, such as Walker [18], were limited in size and scope. At about the same time, Jackson *et al.* [15] published a closely related study also aimed towards assessing the suitability of cloud services for scientific computing. Iosup *et al.* [19] followed up with an even larger-scale study across four different cloud providers with different types of scientific workloads. In summary, all these studies concluded that cloud computing services need performance improvements in order to be competitive for the scientific community.

Such empirical cloud benchmarking evaluations are discussed and generalized on a meta level by literature about designing benchmarks and performance analysis studies for cloud environments. Binnig *et al.* [20] initiated a fundamental discussion about suitable workloads for the cloud, arguing that traditional benchmarks are insufficient for analyzing novel cloud services. One aspect they have identified as particularly important in the cloud is the ability to adapt to peak loads which our work considers on a per VM basis with the focus on CPU performance. Folkerts *et al.* [21] discuss cloud benchmarking challenges regarding meaningful metric choice, workload design, workload implementation, and trustful performance analysis studies. They also give advices how to address them based on sample use cases. Based on existing work and their own experiences, Iosup *et al.* [22] proposed a generic architecture for benchmarking IaaS cloud providers and listed 10 challenges in conducting IaaS cloud benchmarks.

Constant evolution of existing cloud services and the introduction of new service types requires continuous reevaluation that is addressed by frameworks and tools aiming towards scalable cloud evaluations. Expertus, introduced in [23] and extended in [24], proposes a code generation based approach whereas Cloud Crawler [25] favors a declarative approach for defining benchmarks. CloudBench [26] has the ability to run complex and dynamic scale-out workloads and Cloud WorkBench [5], [6] leverages Infrastructure-as-Code DevOps technology to define cross-platform and cross-cloud compatible benchmarks. Additional work that persuades similar goals include Smart CloudBench [27], CloudCmp [28], CloudGauge [29], OLTP-Bench [30], [31], and the Yahoo Cloud Serving Benchmark (YCSB) framework [32].

The majority of existing work in benchmarking IaaS cloud services focuses on micro-benchmarks. Gillam *et al.* [33] and Li *et al.* [34] executed several benchmarks across multiple cloud providers to measure CPU, memory, disk, and network performance characteristics. Salah *et al.* [35] conducted a study to compare the isolated VM performance (i.e., without considering network performance) of popular cloud service providers. Ghoshal *et al.* [36] and Wang *et al.* [37] specifically compared IO performance in virtualized cloud environments. The latency how fast newly acquired cloud services are provisioned is a cloud-specific aspect that is covered in a study by Mao *et al.* [12].

However, a fewer number of works are also available using application benchmarks, including Lenk *et al.* [38], who used OpenSSL (in addition to various micro-benchmarks), Borhani *et al.* [39], who used Wordpress to benchmark various IaaS instance types, and Ferdman [40], who used various different scale-out application workloads.

By and large, these works have focused primarily on general-purpose instance types and, occasionally, compute- or IO-optimized types. The behavior of bursting instances has only recently started to receive attention from research. Wen *et al.* [4] analyzed the CPU performance characteristics of the (previous generation) bursting instance type `t1.micro` in AWS. They proposed injecting short periods of idleness into longer running workloads to diminish host-level throttling and thus achieve overall performance improvements. Leitner *et al.* [3] have benchmarked various smaller instance types, including `t1.micro` in AWS EC2 and `f1.micro` in GCE. However, the focus of this work was clearly on benchmarking general-purpose types. Though not analyzing bursting instances in particular, Xu *et al.* [41] thoroughly investigated the behavior of a credit-based CPU scheduler which plays an important role in causing long tails (i.e., extraordinary worse performance for in high percentiles) in cloud environments. This paper is the first to specifically study the performance behavior of the `t2` instance type family from an end-user perspective, showing that this family behaves differently from all other currently available instance types and hence deserves specific research attention.

VII. Conclusions

In this paper, we analyzed the `t2` generation of credit-based, bursting cloud instance types. We have introduced a basic model that allows to formally discuss and analyze such instances, and empirically compared their performance to current-generation general-purpose and compute-optimized instances, as well as to previous-generation bursting instances. We observed that, unlike previous bursting instances, the performance of `t2` instances is highly predictable even after sustained usage. Further, we have shown that bursting instance types provide a superior performance-cost ratio in terms of CPU performance as long as the average utilization of instances is below 40%. This makes bursting instance types an attractive option for practitioners hosting services with a lower number of users, or whose usage varies over the course of a day. Finally, we have discussed how a systematic instance restarting scheme can be used to further boost the performance-cost ratio of bursting instances.

The primary threat to the validity of this work is that we have only looked at performance using micro-benchmarking. Hence, as part of our future work, we plan to validate the results presented here using application benchmarks or actual business applications, such as on-line transaction processing (OLTP) or scientific computing applications. This will show to what extent the more fundamental analyses presented in this paper are also applicable to real use cases practitioners have. Further, we need to extend our research to also cover the newly-released `t2.large` type.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 610802 (CloudWave).

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing As the 5th Utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[2] J. Cito, P. Leitner, T. Fritz, and H. C. Gall, "The Making of Cloud Applications – An Empirical Study on Software Development for the Cloud," in *Proceedings of the 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE)*, 2015.

[3] P. Leitner and J. Cito, “Patterns in the Chaos - a Study of Performance Variation and Predictability in Public IaaS Clouds,” *ArXiv e-prints*, 2014.

[4] J. Wen, L. Lu, G. Casale, and E. Smirni, “Less can be More: micro-Managing VMs in Amazon EC2,” in *Proceedings of the 2015 IEEE International Conference on Cloud Computing (CLOUD’15)*, 2015.

[5] J. Scheuner, P. Leitner, J. Cito, and H. Gall, “Cloud WorkBench - Infrastructure-as-Code Based Cloud Benchmarking,” in *Proceedings of the 6th IEEE International Conference on Cloud Computing Technology and Science (CloudCom’14)*, 2014.

[6] J. Scheuner, J. Cito, P. Leitner, and H. Gall, “Cloud WorkBench: Benchmarking IaaS Providers based on Infrastructure-as-Code,” in *Proceedings of the 24th International World Wide Web Conference (WWW’15) - Demo Track*, 2015.

[7] B. Farley, A. Juels, V. Varadarajan, T. Ristenpart, K. D. Bowers, and M. M. Swift, “More for Your Money: Exploiting Performance Heterogeneity in Public Clouds,” in *Proceedings of the Third ACM Symposium on Cloud Computing (SoCC’12)*, 2012, pp. 20:1–20:14.

[8] D. Kossmann, T. Kraska, and S. Loesing, “An Evaluation of Alternative Architectures for Transaction Processing in the Cloud,” in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD’10)*. ACM, 2010, pp. 579–590.

[9] P. Kampstra, “Beanplot: A boxplot alternative for visual comparison of distributions,” *Journal of Statistical Software*, vol. 28, no. c01, 2008. [Online]. Available: http://EconPapers.repec.org/RePEc:jss:jstsof:28:c01

[10] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, A. Fox, and D. Patterson, “Cloudstone: Multi-Platform, Multi-Language Benchmark and Measurement Tools for Web 2.0,” in *Cloud Computing and Its Applications*, 2008.

[11] Z. Ou, H. Zhuang, J. K. Nurminen, A. Ylä-Jääski, and P. Hui, “Exploiting Hardware Heterogeneity Within the Same Instance Type of Amazon EC2,” in *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Computing (HotCloud’12)*, 2012, pp. 4–4.

[12] M. Mao and M. Humphrey, “A Performance Study on the VM Startup Time in the Cloud,” in *Proceedings of the Fifth IEEE International Conference on Cloud Computing (CLOUD ’12)*, 2012, pp. 423–430.

[13] J. O’Loughlin and L. Gillam, “Addressing issues of cloud resilience, security and performance through simple detection of co-locating sibling virtual machine instances,” in *Proceedings of the 5th International Conference on Cloud Computing and Services Science*, 2015, pp. 60–67.

[14] Z. Hill and M. Humphrey, “A Quantitative Analysis of High Performance Computing with Amazon’s EC2 Infrastructure: the Death of the Local Cluster?” in *GRID*. IEEE, 2009, pp. 26–33.

[15] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright, “Performance analysis of high performance computing applications on the amazon web services cloud,” in *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, ser. CLOUDCOM ’10, 2010, pp. 159–168.

[16] C. Amza, A. Chanda, A. Cox, S. Elnikety, R. Gil, K. Rajamani, W. Zwaenepoel, E. Cecchet, and J. Marguerite, “Specification and implementation of dynamic web site benchmarks,” in *Proceedings of the IEEE International Workshop on Workload Characterization (WWC-5)*, Nov 2002, pp. 3–13.

[17] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, “A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing,” in *Cloud Computing*, ser. Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering. Springer, 2010, vol. 34, pp. 115–131.

[18] E. Walker, “Benchmarking Amazon EC2 for High-Performance Scientific Computing,” *USENIX Login*, vol. 33, no. 5, pp. 18–23, October 2008.

[19] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, “Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 931–945, Jun. 2011.

[20] C. Binnig, D. Kossmann, T. Kraska, and S. Loesing, “How is the Weather Tomorrow?: Towards a Benchmark for the Cloud,” in *Proceedings of the 2nd International Workshop on Testing Database Systems (DBTest’09)*, 2009.

[21] E. Folkerts, A. Alexandrov, K. Sachs, A. Iosup, V. Markl, and C. Tosun, “Benchmarking in the cloud: What it should, can, and cannot be,” in *Selected Topics in Performance Evaluation and Benchmarking*, ser. Lecture Notes in Computer Science. Springer, 2013, vol. 7755, pp. 173–188.

[22] A. Iosup, R. Prodan, and D. Epema, “Iaas cloud benchmarking: approaches, challenges, and experience,” in *Proceedings of the 2013 international workshop on HotTopics in cloud services*. ACM, 2013, pp. 1–2.

[23] D. Jayasinghe, G. Swint, S. Malkowski, J. Li, Q. Wang, J. Park, and C. Pu, “Expertus: A Generator Approach to Automate Performance Testing in IaaS Clouds,” in *5th IEEE International Conference on Cloud Computing (CLOUD)*, 2012.

[24] D. Jayasinghe, J. Kimball, S. Choudhary, T. Zhu, and C. Pu, “An Automated Approach to Create, Store, and Analyze large-scale Experimental Data in Clouds,” in *14th IEEE International Conference on Information Reuse and Integration (IRI)*, 2013.

[25] M. Cunha, N. Mendonça, and A. Sampaio, “A Declarative Environment for Automatic Performance Evaluation in IaaS Clouds,” in *6th IEEE International Conference on Cloud Computing (CLOUD)*, 2013.

[26] M. Silva, M. Hines, D. Gallo, Q. Liu, K. D. Ryu, and D. Da Silva, “CloudBench: Experiment Automation for Cloud Environments,” in *IEEE International Conference on Cloud Engineering (IC2E)*, 2013.

[27] M. B. Chhetri, S. Chichin, Q. B. Vo, and R. Kowalczyk, “Smart CloudBench – Automated Performance Benchmarking of the Cloud,” in *6th IEEE International Conference on Cloud Computing (CLOUD)*, June 2013, pp. 414–421.

[28] A. Li, X. Yang, S. Kandula, and M. Zhang, “Cloudcmp: Comparing public cloud providers,” in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC ’10)*, 2010, pp. 1–14.

[29] M. A. El-Refaey and M. A. Rizkaa, “Cloudgauge: A dynamic cloud and virtualization benchmarking suite,” in *19th IEEE International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE)*, June 2010, pp. 66–75.

[30] C. Curino, D. E. Difallah, A. Pavlo, and P. Cudré-Mauroux, “Benchmarking oltp/web databases in the cloud: The oltp-bench framework,” in *Proceedings of the Fourth International Workshop on Cloud Data Management (CloudDB ’12)*, 2012, pp. 17–20.

[31] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudré-Mauroux, “Oltp-bench: An extensible testbed for benchmarking relational databases,” *Proceedings of the VLDB Endowment*, vol. 7, no. 4, 2014.

[32] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, “Benchmarking Cloud Serving Systems with YCSB,” in *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC’10)*, 2010.

[33] L. Gillam, B. Li, J. O’Loughlin, and A. Tomar, “Fair Benchmarking for Cloud Computing Systems,” *Journal of Cloud Computing: Advances, Systems and Applications*, 2013.

[34] A. Li, X. Yang, S. Kandula, and M. Zhang, “Cloudcmp: Shopping for a cloud made easy,” in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud’10)*, 2010. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=136451

[35] K. Salah, M. Al-Saba, M. Akhdhor, O. Shaaban, and M. Buhari, “Performance evaluation of popular Cloud IaaS providers,” in *Proceedings of the 6th International Conference on Internet Technology and Secured Transactions (ICITST)*, 2011, pp. 345–349.

[36] D. Ghoshal, R. S. Canon, and L. Ramakrishnan, “I/O Performance of Virtualized Cloud Environments,” in *Proceedings of the Second International Workshop on Data Intensive Computing in the Clouds (DataCloud-SC ’11)*, New York, NY, USA, 2011, pp. 71–80.

[37] G. Wang and T. S. E. Ng, “The Impact of Virtualization on Network Performance of Amazon EC2 Data Center,” in *Proceedings of the 29th Conference on Information Communications (INFOCOM’10)*, 2010.

[38] A. Lenk, M. Menzel, J. Lipsky, S. Tai, and P. Offermann, “What Are You Paying For? Performance Benchmarking for Infrastructure-as-a-Service Offerings.” in *Proceedings of the 2011 IEEE International Conference on Cloud Computing (CLOUD’11)*, 2011, pp. 484–491.

[39] A. H. Borhani, P. Leitner, B. Lee, X. Li, and T. Hung, “WPress: An Application-Driven Performance Benchmark for Cloud-Based Virtual Machines,” in *Proceedings of the 18th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, 2014, pp. 101–109. [Online]. Available: http://dx.doi.org/10.1109/EDOC.2014.23

[40] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, “Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware,” *SIGARCH Computer Architecture News*, 2012.

[41] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey, “Bobtail: Avoiding Long Tails in the Cloud,” in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (NSDI’13)*, 2013, pp. 329–342.