A Day Late and a Dollar Short: The Case for Research on Cloud Billing Systems

Robert Jellinek Yan Zhai Thomas Ristenpart Michael Swift
University of Wisconsin

Abstract

Cloud computing platforms such as Amazon Web Services, Google Compute Engine, and Rackspace Public Cloud have been the subject of numerous measurement studies considering performance, reliability, and cost efficiency. However, little attention has been paid to *billing*. Cloud providers rely upon complex, large-scale billing systems that track customer resource usage at fine granularity and generate bills reflecting measured usage. However, it is not known how visible such usage is to customers, and how closely provider charges correspond to customers' view of their resource usage.

We initiate a study of cloud billing systems, focusing on Amazon EC2, Google Compute Engine, and Rackspace, and uncover a variety of issues, including: inherent difficulties in predicting charges; bugs that lead to free CPU time on EC2 and over-charging for storage in Rackspace; and long and unpredictable billing-update latency. Our measurements motivate further study on billing systems, and so we conclude with a brief discussion of open questions for future work.

1 Introduction

Public cloud computing systems are revolutionizing how computing power is purchased. Rather than capital expenditures on equipment up-front, cloud customers can dynamically request resources such as storage, bandwidth, and computing power, and pay only for their usage. Powering all this is a spectrum of technologies for resource management and delivery to tenants. Measuring and improving the reliability and performance of cloud networks [4, 15, 21], storage [6, 8–10, 20], and compute resources [14, 17, 22, 23] has seen significant attention from the academic community.

While some studies have looked at the cost implications of varying workloads on pay-as-you-go cloud platforms [11, 12, 19], the efficacy and performance of the *billing systems* that enable fine-grained, dynamic resource purchasing have, by comparison, has received a paucity of research attention. A notable exception is the work on verifiable resource accounting [5,18] which suggests the use of trustworthy computing mechanisms to prove resource usage at a very fine granularity, e.g., CPU cycles used. The motivation stems from the perception that cloud providers may maliciously overcharge their customers (c.f., [13]). Resource-as-a-service clouds provide resources at fine granularities [1], but like these other works are forward-looking and do not address billing concerns in existing clouds. As it stands, the research community has not investigated the problems faced in billing in practice: whether benign (let alone malicious) errors arise, what kind of performance billing systems offer, and whether the trajectory of billing-system design and implementation is set to meet the needs of potential future applications.

We provide the first step in enacting a research agenda towards answering these questions. In particular, we perform the first measurement study of cloud billing systems, focusing in particular on three public infrastructure-as-a-service (IaaS) clouds: Amazon's Elastic Compute Cloud (EC2) [3], Google's Compute Engine (GCE) [7], and Rackspace's Public Cloud [16]. For each cloud, we measure its billing latency, as well as the transparency and predictability of its billing for compute time, network usage, and storage.

While we a priori expected this to be straightforward, in practice it is surprisingly difficult due to the coarse granularity of bills, the lack of API support for programmatic access to bills, and the fact that providers base bills on resource-usage events that are opaque to customers. Despite these challenges, we are able to (a) *reverse-engineer* ambiguous aspects of billing; (b) *characterize performance*, namely in measuring and documenting substantial billing latency between when a resource is consumed and charges are visible to the customer; (c) *uncover bugs*, such as a possible race condition in EC2 CPU billing that provides two minutes of free compute time, inconsistencies across EC2 billing reports, and a bug in Rackspace storage billing that leads

to overcharges; and (d) *detect systematic undercharging*, largely due to opaque caching/batching mechanisms.

Our measurements imply the existence of significant challenges in this space, and highlight the need for more research on the question of how to build scalable, effective billing systems. We therefore conclude with a set of open questions to be answered in order to make progress towards this goal.

2 Billing Update Latency

We start by assessing the *performance* of the three clouds' billing systems. In later sections we look at how individual resources are billed. We refer to the delay between the use of a resource and the bill for the corresponding charge as *billing latency*. Ideally, the bill would be updated with low and predictable latency. In reality, we find that bills are often *delayed* by hours to days after a workload runs, and that latency to receive a bill can *vary widely*. For brevity we omit our experimental details for Rackspace and GCE.

EC2 console and CSV billing latency. We measure the latency of billing information both through the EC2 web-based console and through the billing reports it publishes as comma-separated-value (CSV) files. In total, we ran 122 m1.small instances and 150 t1.micro instances on 19 EC2 accounts, using Ubuntu 12.04 LTS unless otherwise noted. Each account ran a single instance at a time for 3590–3600 seconds and was then idle until all billing updates had been observed. EC2's web-based management console updated the most quickly, at an average of 6:23 hours after instance start (std. dev. 4:13 hours), while the downloadable CSVs were available only 7:36 hours after instance start (std. dev. 2:59 hours). This delay is both long, and highly unpredictable.

To better understand the high variability in the timing of billing-update schedules during prolonged instance runs, we launched a second experiment in which we ran 25 instances each for 10 hours. We performed two separate runs and started instances staggered by 30 seconds in order to avoid sending too many API calls at once. The first bills arrived an average of 4.2 hours after launching the instances, while one or two additional updates each arrived at 6-hour intervals after the first. These results indicate that the variability could be due to a regular 6-hour billing cycle, and that instances launched near the end of the cycle experience lower latency than those launched near the beginning. While updates across accounts occurred at roughly the same times, one account was billed for 7 hours in the first update and 3 in the second while another was billed for 2 hours in the first update, 5 in the next, and 4 in the third (similar variability occurred across all instances). This suggests that instance-hour usage data is sampled at different times even for instances started at approximately the same time, making it impossible for a customer to predict when all usage is registered in a given billing update.

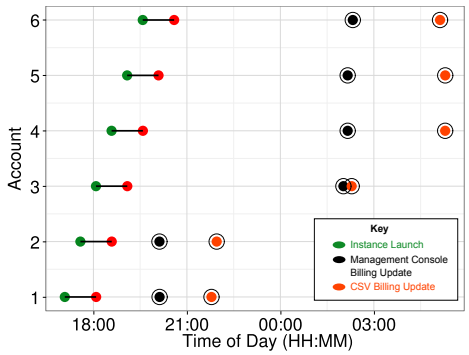


Figure 1: Time series of one-hour runs using 6 accounts. Shown is instance start and stop time (connected green and red dots), console billing time (circled black dot), and S3 CSV billing time (circled red dot).

Based on these timing results, we investigated a set of runs shown in Figure 1 where we deliberately staggered the start time of instances by 30 minutes (shown as green dots followed by lines). One can see a clear time when bills seem to be updated, suggesting that the billing system periodically updates bills from accounting data as a batch job. However, there was no indication that bills are collected on a fixed daily schedule, as we observe no universally fixed time of day when updates occurred.

Other EC2 billing APIs. The CloudWatch resource-usage monitoring service has even longer update latency. We ran 10 c1.medium instances on accounts with no other activity, and waited for the corresponding bill by polling the CloudWatch API [2]. CloudWatch took an average of 11:04 hours (std. dev. 8 minutes) to register the usage. This is much more predictable than the management console, but almost double the latency.

The cost-allocation ("tagged") billing CSV files, which allow a customer to preemptively tag resources with a key-value pair and later differentiate their billing based on these user-defined categories, experienced even more severe update delays. We ran two batches of tests. In the first batch we ran 3 instances each for 120, 3610, and 7210 seconds, expecting to see 1, 2 and 3 instance-hours billed, respectively. The first billing update took 13 hours, while the second took 33. In the second batch of tests (instances run for 120, 3620, 7220 seconds), the first update arrived after 24 hours, and the second after 56 hours. Two 120-second instances never appeared on the cost-allocation CSV bill, but were billed for 1 hour on all other billing interfaces, indicating inconsistency across interfaces. Interestingly, some cost-allocation CSV billing entries reported instance-hour usage as decimals to 8 digits (though the decimals did not correctly reflect the partial hour used), while others were

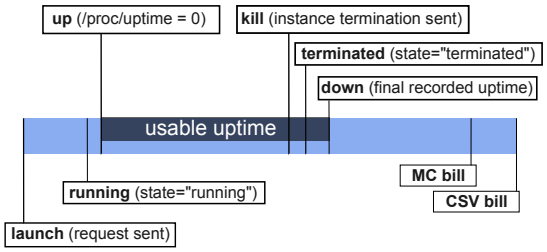


Figure 2: Events in an EC2 VM instance lifetime. Time stamps are provided only for *launch* and *terminated*.

rounded, as usual for instance-hour billing.

Discussion. All three providers have long delays in providing bills. We found that for GCE, over 13 tests we observed an average update latency of 2.2 days, minimum of 0.9 days, and maximum of 9.1 days, while for Rackspace, we observed an average latency of 21.5 hours with a minimum of 8 hours and maximum of 41.1 hours. Of the providers, only Rackspace seemed to update their bills at a consistent time of day: between 9–10am UTC Amazon provides the most timely usage data (about a 6 hour delay for the web, longer for other interfaces), but its update time is still not predictable and multiple update periods are sometimes required before all usage is reported. Rackspace only updates bills once per day, while GCE takes even longer and also exhibited extreme outliers in latency. As a result, it is difficult to use billing information programmatically, as there is no guarantee of when it is available or whether a given update reflects all usage up to that moment.

3 Compute Time Billing

The first question about CPU billing is *when* a provider is charging. Providers typically advertise that they begin and end charging when an instance "starts" and "stops," but these terms are ambiguous and could correspond to any of the many distinct timestamps in an instance lifetime. Figure 2 depicts some of the major events in an instance lifetime. EC2, Rackspace, and GCE all provide T_{launch} timestamps, while only GCE offers additional timestamps for the times that instance `CREATE` and `DELETE` API calls were issued and completed.

In our experiments, all other timestamps were collected by polling the providers' APIs (10 times per second for EC2; once per second for Rackspace and GCE, which rate-limited requests) to register when an instance's status changed. This method is inexact, however, due to the inherent jitter introduced by variable network latency, server response time, and polling granularity. This semantic gap between a provider's and a customer's view of instance-lifetime events means that a customer cannot be certain that their calculations (and corresponding deployment decisions) are accurate.

For experiments on EC2, we determined that the interval $T_{kill} - T_{running}$ best represented billable time after testing many other timestamp intervals and determining that this had the strongest correlation to billed hours. We used the 272 combined m1.small and t1.micro instance runs from Section 2. In these executions, we ensured that $\Delta = T_{kill} - T_{running}$ took on a value in the range of 3590–3600 seconds by controlling T_{kill}. Despite this narrow window, these runs demonstrated variability in the relative timing of the various instance lifetime events on EC2, as shown in Table 3. For example, the difference $T_{up} - T_{running}$ is often negative, meaning an instance is up and running before EC2 marks it as such.

Value	Min	Median	Avg	Max	SD
$T_{running} - T_{launch}$	17	24	30	264	26
$T_{up} - T_{running}$	-159	0	-2	12	21
$T_{down} - T_{term}$	-1	0	1	26	3

Table 3: Measured variability in EC2 instance lifetime events, in seconds.

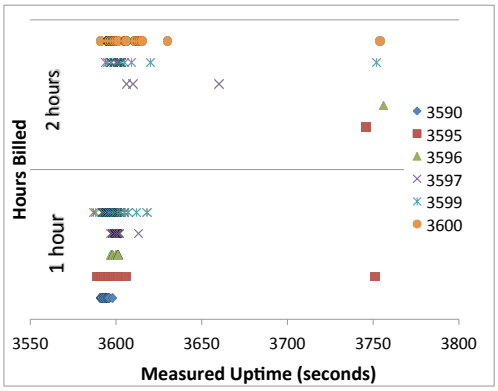


Figure 4: Measured uptime $T_{down} - T_{up}$ for 272 EC2 instances run with $3590 \leq T_{kill} - T_{launch} \leq 3600$ versus the number of hours billed.

Figure 4 shows a scatter plot relating $T_{down} - T_{up}$ (the measured uptime) to whether the instance was billed for two hours (plotted above the line) or one hour (below the line), for the 272 instances. Within each half, instances corresponding to each Δ value are on the same row, starting with 3590 and going up to 3600. We find several trends. First, there exists a set of outliers around uptime of 3750. These instances ran 2.5 minutes longer than requested, and in one case such an instance was billed for only an hour, meaning Amazon *underbilled* for the actual uptime. Second, for the bulk of the data points, we see that while the average uptime is close to Δ, there is a horizontal spread of points indicating a variability range of approximately 10 seconds. Finally, we see that as Δ increases, the percentage of instances charged two hours increases.

We performed additional tests to more finely de-

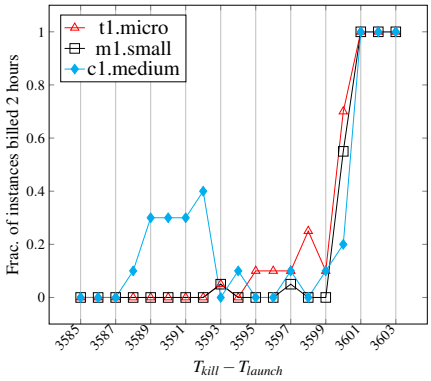


Figure 5: The fraction of EC2 instances of distinct types billed for two hours versus $T_{kill} - T_{launch}$.

termine whether $T_{kill} - T_{running}$ provides good prediction of billing (if not uptime), and if this holds for a broader range of instance types. We launched 20 m1.small instances simultaneously using 20 accounts and terminated Δ seconds after $T_{running}$, for each of $\Delta \in \{3585..3603\}$. We repeat this procedure with t1.micro and with c1.medium instances using 20 and 10 instances for each instance type per value of Δ, respectively. Figure 5 shows the fraction of instances billed for one hour versus two across these tests. Surprisingly, the c1.medium instances had the highest variance in the number of hours billed. This data suggests that customers should terminate their instances 13 seconds before $T_{running} + 3600i$ to ensure being billed for only i hours, but only if they go out of their way to get an accurate timestamp $T_{running}$. The amount of uptime received, however, will vary on the order of several minutes.

A bug in EC2 compute billing. We were intrigued by the cases in Figure 5 with uptime greater than one hour that were charged for only one hour. We also found that if we terminated an instance early in its lifetime we were occasionally not getting billed. To explore further, we created a VM image with a minimal kernel with network support that boots quickly, connects to our local controller and frequently sends short messages with the current timestamp. The first and last timestamps of successful connections to our controller give uptime for the instance.

We performed a series of runs where we launched this image and sent a `terminate` API call exactly Δ seconds after T_{launch} for $\Delta \in \{2..20\}$, with 20 instances per value of Δ. For $\Delta \leq 16$ we got no usable uptime, but for each of $\Delta \in \{17, 18\}$, 1 out of 20 instances received approximately 116 seconds of free uptime, while for $\Delta = 19$, 4 of 20 instances received 117 seconds of unbilled, usable uptime. For $\Delta = 20$, 6 of 20 instances received an average of 118 seconds of uptime, but 3 of these instances were billed for an hour of usage each, indicating a race condition in EC2's infrastructure between the billing system and the instance management system. This free time is difficult to use economically, though, given the chance of receiving a bill for a whole hour of usage.

Setup Case	Send % Reported	Receive % Reported
(1) Univ → EC2	-	95.9%
(2) EC2 → Univ	94.4%	-
(3) Zone X → Zone X	-	-
(4) Zone X → Zone X (public IP)	97.6%	97.2%
(5) Zone X → Zone Y	97.1%	97.5%
(6) Reg X → Reg Y	95.9%	96.8%

Table 6: Average ratios (in percent) of billed traffic volume to measured traffic volume for the sender (second column) and receiver (third column). A "-" indicates tests for which no billing occurred, which was correct relative to the EC2 billing model.

4 Network Billing

A key challenge in predicting the bill for networking is knowing how much a packet will cost. While GCE and Rackspace have relatively simple network billing models, Amazon charges different rates for inbound and outbound traffic between instances and the Internet, to a public IP in the same zone, to another zone, and to another region. We perform six experiments on EC2 that reflect the six possibilities. The results are shown in Table 6: for "Univ → EC2" the sender was hosted in our local university and the recipient in EC2; for "Zone X → Zone X" the sender and receiver are in the same zone using private IP addresses (unless otherwise indicated), etc. Cases (1) and (3) should be free according to EC2's pricing model, while the remaining four cases should be billed. We perform three runs for each of the six configurations, and report the average percentage of network traffic billed out of that measured by NetFilter in the instance's kernel. Billing data was measured from the detailed CSV.

Compared to measuring filtered traffic in the kernel, EC2 consistently underbills. In the case of Internet-outbound traffic, which is the most expensive category of network traffic, (case (2) in Table 6), EC2 undercharges by an average of 5.6%. Across all experiments, EC2 undercharges traffic by 3.4%.

We found that GCE and Rackspace network-traffic billing was more predictable perhaps partly due to their simpler network billing models, with two exceptions in Rackspace. In 2 of 11 Rackspace instances we sent 1GB from the instance to our local controller, and Rackspace charged for 35MB and 125MB less than was successfully sent and received. This may indicate a bug in the Rackspace network billing mechanism.

5 Storage Billing

Providers charge for block storage, both by the bytes stored over time and per I/O operation (only EC2).

Similarly to billing for compute time, charges for volume allocation are sensitive to opaque volume creation and deletion timestamps, while I/O billing is subject to caching and aggregation that can happen between the guest OS and the provider, making it impossible for a customer to measure.

Storage charges. We tested GB-month storage on EC2, GCE, and Rackspace by allocating volumes of a set size and for a set period, and then comparing the expected charge versus that reported on bills. In EC2 and GCE, the bills were consistent with expected charges, and for the sake of brevity we omit details. For Rackspace, however, creating a 100 GB drive and then deleting it resulted in charges that were 36.3% *higher* than expected. The reason was a bug that we uncovered: delete operations can hang if a volume has not first successfully "detached," leaving the volume unavailable yet still unfairly accruing charges. We have notified Rackspace of this issue.

I/O charges in EC2. We perform reads and writes totaling 1 GB to a 100 GB EBS volume using `fio` with direct I/O (in order to bypass caching by the guest OS) and with varying block sizes. Figure 7 shows the ratio between the number of reads (writes) made to the volume and the number of reads (writes) charged for across a representative subset of block sizes. Across all experiments, the number of measured operations is either the same or higher than the number of charged operations by up to a factor of 4.6, indicating we are billed for fewer operations than our instance performs.

These discrepancies could be due to aggregation and caching within EC2's infrastructure, or simply undercharging. To tease these apart, we performed additional experiments in which, fixing the block size at 4096 bytes, we read from and wrote to random locations within the 100 GB volume. For these tests, the EC2-reported I/Os differed from instance-reported I/Os by only 0.8% and 0.2%, respectively, which suggests that EC2 is not undercharging but instead aggregating requests (for writes) or both aggregating and caching (for reads), which lowers costs but increases opacity of billing.

6 Conclusions

Our measurements surface a number of undesirable features of today's cloud billing ecosystem that arise at the intersection of the billing model and implementation. First, billing events mostly occur within the cloud infrastructure, making them largely *unobservable* to customers. Second, billing systems are *asynchronous* and make bills available long after the resource consumption occurs. Finally, billing systems tend to *aggregate* data across many events or instances into a single line item. This suits the provider: it alleviates the need to retain fine-grained information about usage and to invest in making bills available more quickly. We note that although our study is a snapshot of these billing systems at one moment in time and the systems are continuously evolving, these characteristics reflect fundamental design choices that we believe should be reconsidered.

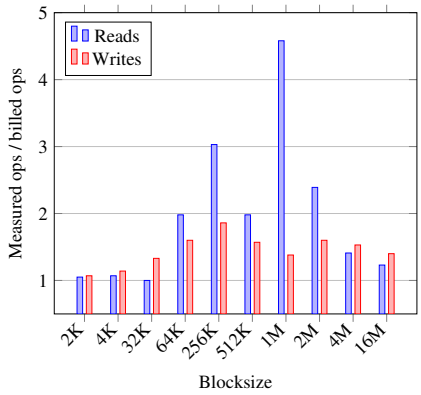


Figure 7: Ratio of number of storage operations measured by `/proc/diskstats` to number of operations billed by EC2.

We propose that providers adopt a *transparent billing* model in which billing information is *accessible*, *understandable*, *timely*, and *predictable*. Concretely, we believe cloud providers should offer a billing API that exposes the provider's current view of key resources: compute time (including provider-captured start and stop timestamps), network (total usage, broken down by traffic category), and storage (both volume use start and stop timestamps, and number of I/Os billed). Furthermore, the API should include a *valid-as-of* timestamp if the information is not real-time.

Such an interface would enable a new paradigm of *cost-based computing*, where customers could optimize their deployments for cost in real time and accurately audit their usage, gaining assurance that they are being billed accurately. Similarly, providers can vary their prices in real time to control congestion. It would also open access to an abundance of real-time resource-usage data, which could be used for anomaly detection, accountability, etc.

This sets a goal for cloud billing systems, and leaves open several questions. In particular, given the enormous scale and fine granularity of universal resource accounting and billing, what tradeoffs are inherent in exposing a *transparent billing* API? How close to real-time can a provider perform reporting, and at what expense? And, if such an API can be exposed, what new applications would that introduce? We believe these are fundamentally important questions, the answers to which will be of great value to both cloud providers and customers.

References

[1] AGMON BEN-YEHUDA, O., BEN-YEHUDA, M., SCHUSTER, A., AND TSAFRIR, D. The resource-as-a-service (raas) cloud.

In *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing* (2012), USENIX Association, pp. 12–12.

[2] AMAZON. Amazon Cloudwatch. `http://aws.amazon.com/cloudwatch/`.

[3] AMAZON. Amazon EC2. `aws.amazon.com/ec2/`.

[4] BALLANI, H., JANG, K., KARAGIANNIS, T., KIM, C., GUNAWARDENA, D., AND OSHEA, G. Chatty tenants and the cloud network sharing problem. *Proceedings of Usenix NSDI, Lombard, IL* (2013).

[5] CHEN, C., MANIATIS, P., PERRIG, A., VASUDEVAN, A., AND SEKAR, V. Towards verifiable resource accounting for outsourced computation. In *Proceedings of Virtual Execution Environments – VEE 2013* (2013), ACM.

[6] CIDON, A., RUMBLE, S., STUTSMAN, R., KATTI, S., OUSTERHOUT, J., AND ROSENBLUM, M. Copysets: reducing the frequency of data loss in cloud storage. In *Presented as part of the 2013 USENIX Annual Technical Conference* (2013), USENIX, pp. 37–48.

[7] GOOGLE. Google compute engine. `https://cloud.google.com/products/compute-engine`.

[8] GORDON, A., AMIT, N., HAR'EL, N., BEN-YEHUDA, M., LANDAU, A., SCHUSTER, A., AND TSAFRIR, D. Eli: bare-metal performance for I/O virtualization. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (London, UK, March 2012), pp. 411–422.

[9] IOSUP, A., YIGITBASI, N., AND EPEMA, D. H. J. On the performance variability of production cloud services. In *CCGRID* (2011), pp. 104–113.

[10] JU, X., SOARES, L., SHIN, K. G., RYU, K. D., AND DA SILVA, D. On fault resilience of openstack. In *Proceedings of the 4th annual Symposium on Cloud Computing* (2013), ACM, p. 2.

[11] KOSSMANN, D., KRASKA, T., AND LOESING, S. An evaluation of alternative architectures for transaction processing in the cloud. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* (2010), ACM, pp. 579–590.

[12] KRASKA, T., HENTSCHEL, M., ALONSO, G., AND KOSSMANN, D. Consistency rationing in the cloud: Pay only when it matters. *Proceedings of the VLDB Endowment 2*, 1 (2009), 253–264.

[13] LIU, M., AND DING, X. On trustworthiness of cpu usage metering and accounting. In *Distributed Computing Systems Workshops (ICDCSW), 2010 IEEE 30th International Conference on* (2010), IEEE, pp. 82–91.

[14] OU, Z., ZHUANG, H., NURMINEN, J. K., YLÄ-JÄÄSKI, A., AND HUI, P. Exploiting hardware heterogeneity within the same instance type of amazon EC2. In *HotCloud* (2012).

[15] POTHARAJU, R., AND JAIN, N. When the network crumbles: an empirical study of cloud network failures and their impact on services. In *Proceedings of the 4th annual Symposium on Cloud Computing* (2013), ACM, p. 15.

[16] RACKSPACE. Rackspace open cloud - open source cloud backed by fanatical support. `www.rackspace.com/open-cloud/`.

[17] SCHAD, J., DITTRICH, J., AND QUIANÉ-RUIZ, J.-A. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. In *VLDB* (Sept. 2010).

[18] SEKAR, V., AND MANIATIS, P. Verifiable resource accounting for cloud computing services. In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop* (2011), ACM, pp. 21–26.

[19] SOBEL, W., SUBRAMANYAM, S., SUCHARITAKUL, A., NGUYEN, J., WONG, H., KLEPCHUKOV, A., PATIL, S., FOX, A., AND PATTERSON, D. Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0. In *Proc. of CCA* (2008), vol. 8.

[20] TERRY, D. B., PRABHAKARAN, V., KOTLA, R., BALAKRISHNAN, M., AGUILERA, M. K., AND ABU-LIBDEH, H. Consistency-based service level agreements for cloud storage. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (2013), ACM, pp. 309–324.

[21] WANG, G., AND NG, T. S. E. The impact of virtualization on network performance of amazon EC2 data center. In *IEEE INFOCOM* (2010).

[22] WILLIAMS, D., JAMJOOM, H., LIU, Y.-H., AND WEATHERSPOON, H. Overdriver: Handling memory overload in an oversubscribed cloud. In *International Conference on. Virtual Execution Environments* (2011), pp. 205–216.

[23] XU, Y., MUSGRAVE, Z., NOBLE, B., AND BAILEY, M. Bobtail: avoiding long tails in the cloud. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation* (2013), pp. 329–342.