A Day Late and a Dollar Short

The Case for Research on Cloud Billing Systems

Robert Jellinek **Yan Zhai** **Thomas Ristenpart** **Michael Swift**

Outline

- Motivation: Why study cloud billing?
- Our contributions
- Our study
- Results
- Conclusions and future work

Motivation: Cloud Billing

- Many **performance**, **reliability**, and **cost efficiency** studies of the cloud.
- Little attention has been paid to their **billing systems**.
- Pay-as-you-go pricing model relies upon complex, large-scale billing systems

amazon web services

Google Cloud Platform

Motivation: Cloud Billing

- Resource accounting is an interesting challenge.

- How to track all compute resource usage in
  - **real time,**
  - at **fine granularity**
  - maintaining **accuracy,**
  - and not hurting **performance**?

Study of cloud billing mechanisms.

- We were able to:
  - **Disambiguate billing** by reverse engineering
  - **Uncover bugs**:
    - Race conditions in EC2
    - Inconsistencies across billing interfaces in EC2
    - Rackspace bug causing overcharges
  - **Detect systematic undercharging** from caching/ aggregation
  - **Characterize performance** of *billing latency*.

Study Overview

- Guiding question:

  How **accurate**, **timely**, and **predictable**
  are customer-facing billing interfaces?

- Measured billing for:
  - **compute time**
  - **storage** (IOPS and capacity)
  - **network** usage

- Experimented on AWS, GCE, and Rackspace
- Calculate **billing latency** of billing interfaces.

Methodology

- **Instrument providers' API calls** to launch/terminate instances and create/delete storage volumes, in order to capture fine-grained timing data about their usage.
- Launch an instance and **execute one of several workloads** (network tests; I/O tests; or timed idle to test instance-hour thresholds) to measure resource usage.
- **Fetch OS-based resource-usage** data from *procfs* and *Netfilter / iptables* in order to compare with the amount ultimately billed.
- **Terminate instance** after workload completion. In cases where we measure instance-hour thresholds, terminate at some fixed number of seconds after various instance-lifetime events, in order to isolate the interval that the provider uses to calculate billing.
- **Poll for billing updates** over all measured resources.

Billing Interfaces

- EC2:
  - Web-based GUI **management console**
  - Programatically accessible CSVs:
    - **Hourly**
    - **Monthly** (to date)
    - **Cost-allocation** (allows user to tag resources and filter costs by tag)
- GCE:
  - Web-based GUI interface
- Rackspace:
  - Web-based GUI interface

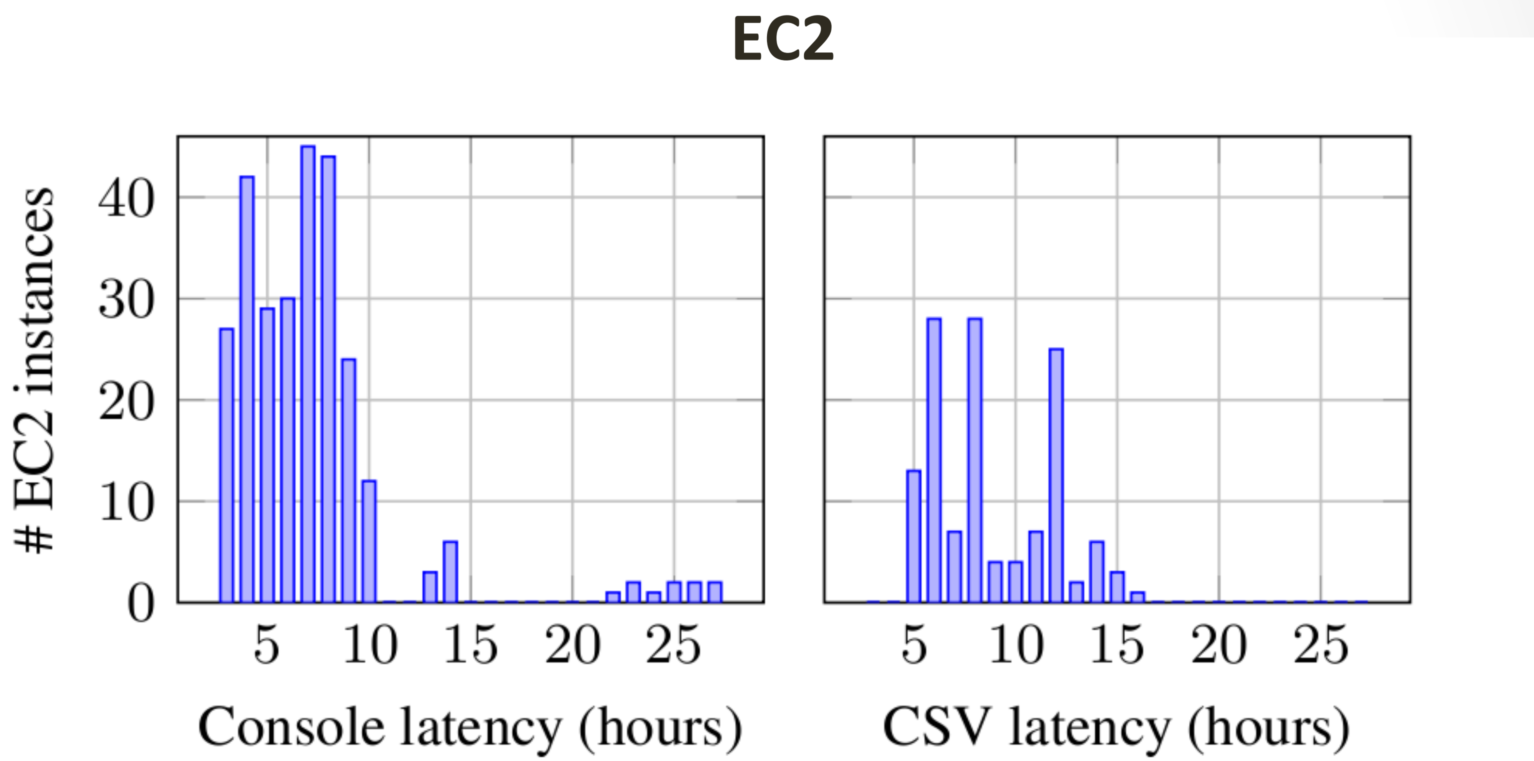
Billing Interfaces

Details

Expand All Services | Collapse All Services — Printer Friendly Version

AWS Service Charges		**$0.95**
Amazon Elastic Compute Cloud Download Usage Report »		**$0.95**
US East (Northern Virginia) Region		
Amazon EC2 running Linux/UNIX		
$0.020 per Micro Instance (t1.micro) instance-hour (or partial hour)	33 Hrs	0.66
Amazon EC2 EBS		
$0.100 per GB-month of provisioned storage (blended price)*	2.367 GB-Mo	0.24
$0.10 per 1 million I/O requests	76,834 IOs	0.01
$0.125 per GB-Month of snapshot data stored (blended price)*	0.344 GB-Mo	0.04
Amazon Simple Notification Service Download Usage Report »		**$0.00**
US East (Northern Virginia) Region		
First 100,000 Amazon SNS API Requests per month are free	54 Requests	0.00
AWS Data Transfer (excluding Amazon CloudFront)		**$0.00**
$0.000 per GB - data transfer in per month	0.034 GB	0.00
$0.000 per GB - first 1 GB of data transferred out per month	0.001 GB	0.00
$0.000 per GB of regional data transfer in/out (blended price)*	0.000020 GB	0.00

Billing Latency

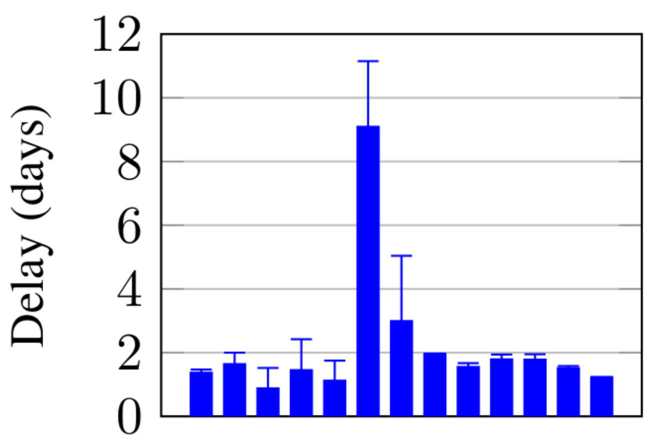


Web Console
Avg latency: 6:41 hours
Std dev: 4:10 hours

CSV
Avg latency: 8:15 hours
Std dev: 3 hours

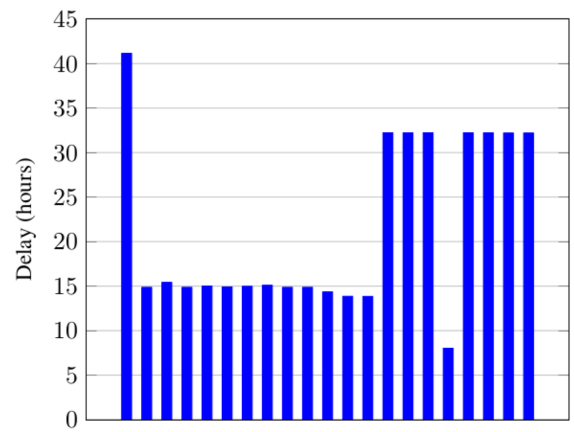
GCE/Rackspace Billing Latency

GCE



Lower bounds on GCE billing latency for 13 instances, in DAYS. Error bars indicate upper bounds.

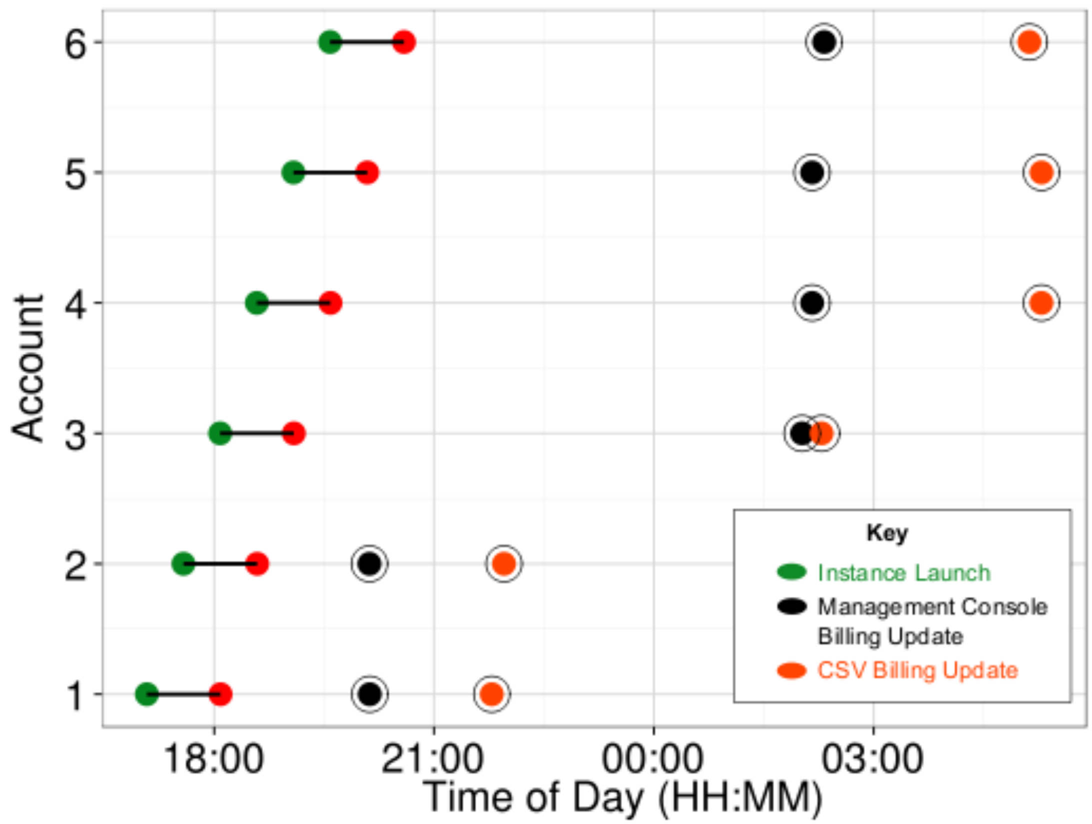
Rackspace



Rackspace billing latency for 21 instances in HOURS, +/- 10 minutes. All billing updates occurred between 9-10am UTC.

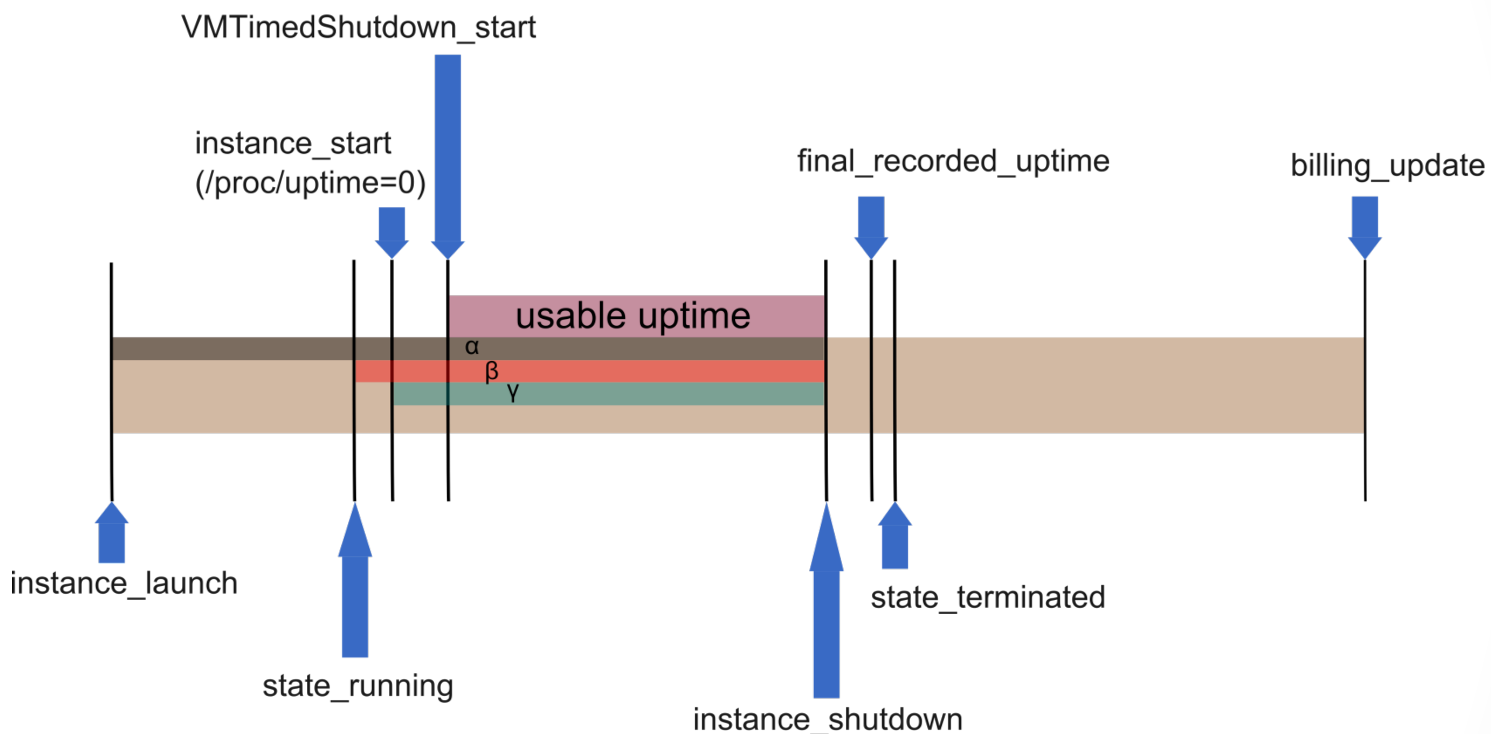
EC2: Why such latency?

EC2: Staggered Launch Times



We deliberately staggered the start times of instances. The billing update schedule suggests **periodic batch processing.**

What is “Compute Time”?



Major events in an instance lifetime

Compute Time

- Many events in an instance lifetime.
- Providers: “Pricing is per instance-hour consumed for each instance, from the time an instance is launched until it is terminated or stopped.”[1]
- This is ambiguous. We tried to reverse engineer exactly when the “start” and “stop” timestamps occur.

[1] https://aws.amazon.com/ec2/pricing/

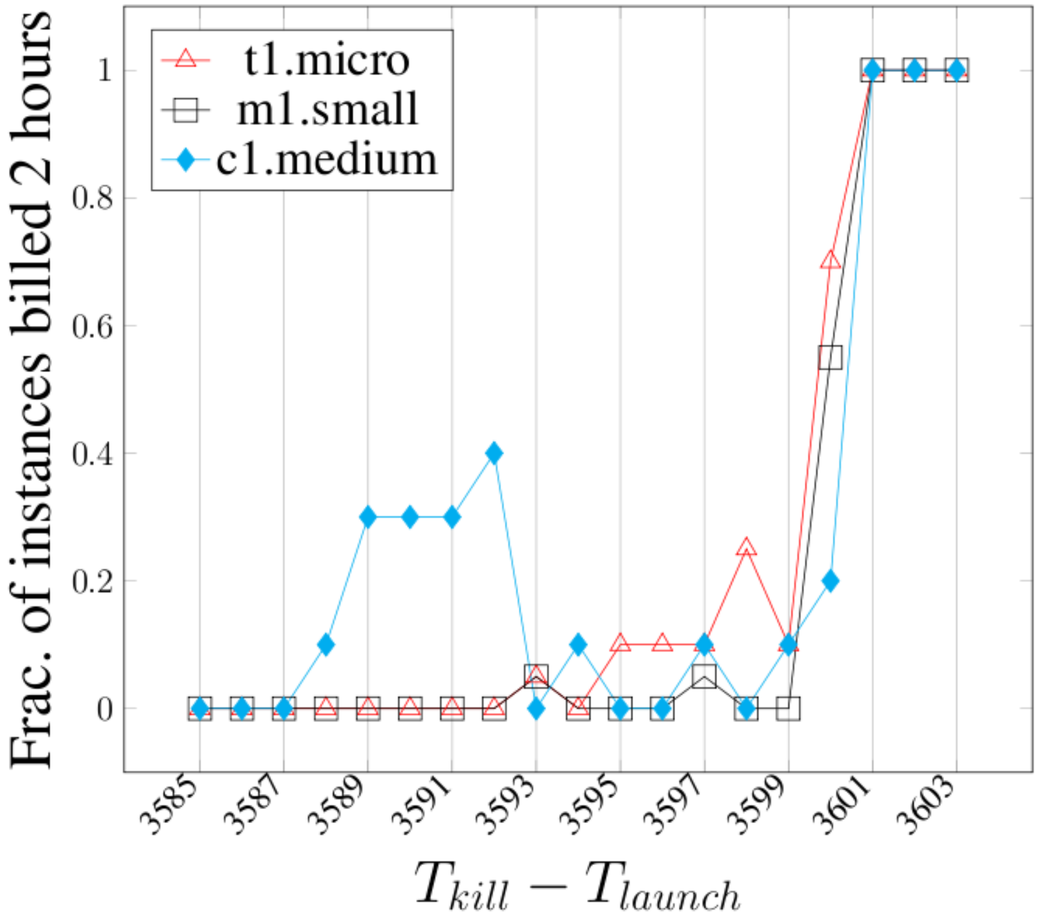
Compute Time

- Billing “start” and “stop” could to correspond to various events in an instance lifetime, for example:
- Start:
  - When user launches instance.
  - When launch request is serviced (could be queued).
  - When instance boot is complete.
  - When /proc/uptime is zero.
- Stop:
  - When user initiates shutdown from within instance.
  - When user initiates terminate from management console.
  - When termination is complete (opaque to user).

Compute Time

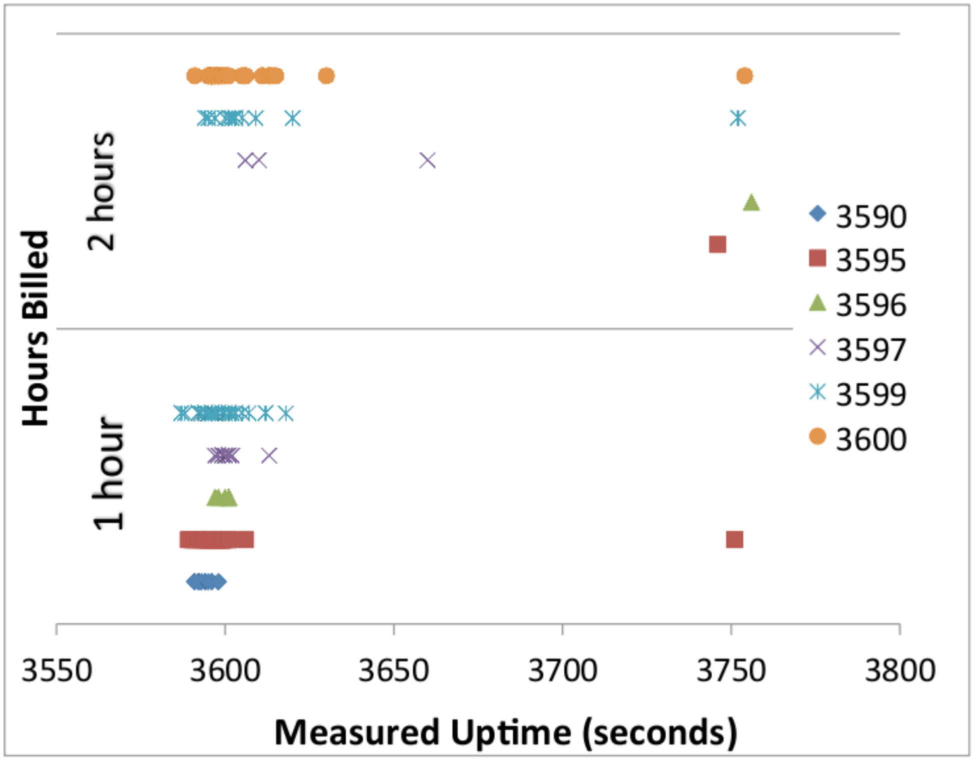
- We determined the most probable timestamps for each service, but there was still jitter.
- Suggests variance outside what we are able to measure by polling the providers' API.

EC2 Compute Time Results



Compute Time Anomalies

Other anomalies in EC2 compute billing



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.

Compute Time

Δ	# launched	# ran	Avg. uptime per instance (s)	Hours billed
≤ 16	20	0	0	0
17	20	1	115	0
18	20	1	116	0
19	20	4	117	0
20	20	6	118	3

- We created a special "fast boot" kernel that booted and immediately sent heartbeat messages to our control server.
- Terminated instances Δ seconds after launch.
- Race condition causes some instances to not get billed, but yield roughly 2 minutes of free uptime.

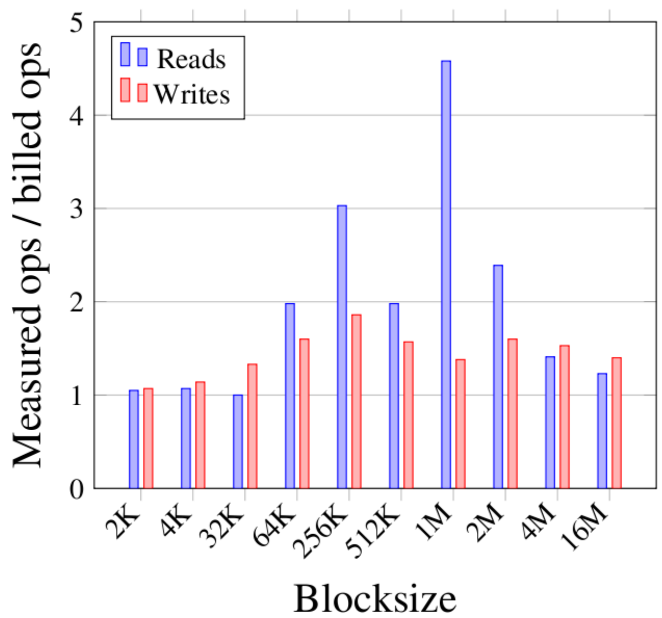
Storage Billing

- Provider charges based on its view of storage ops

- Storage:
  - In Rackspace deleting a volume before detaching from an instance caused it to hang and accrue charges.
  - I/O charges in EC2 lower than /proc/diskstats would suggest; caching or aggregation?

- Storage example:
  - Write(4kb); write(4kb); -> 1 storage op
  - Write (4kb); seek(1 million); write(4kb) -> 2 storage ops

IOPS Aggregation/Caching?



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

Network Billing

- In EC2 for Internet-outbound traffic, underbilled by 5.6% of Netfilter measurements.
- Rackspace underbilled 1 GB of Internet-outbound traffic for 2 of 11 instances by 35 MB and 125 MB.

Setup	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%

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.

Conclusions

- Future research should investigate the tradeoffs between performance on the one hand, and accurate, timely, transparent resource accounting and billing on the other.
- This will likely necessitate collaboration with industry.
- It seems that today, it should be feasible for providers to expose a billing API, to enable programmatic queries of billing information.

Thank you.