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
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

<table>
<thead>
<tr>
<th>AWS Service Charges</th>
<th>$0.95</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Amazon Elastic Compute Cloud</strong></td>
<td></td>
</tr>
<tr>
<td>$0.95</td>
<td></td>
</tr>
<tr>
<td><strong>US East (Northern Virginia) Region</strong></td>
<td></td>
</tr>
<tr>
<td>Amazon EC2 running Linux/UNIX</td>
<td></td>
</tr>
<tr>
<td>$0.020 per Micro Instance (1 micro) instance-hour (or partial hour)</td>
<td>33 Hrs</td>
</tr>
<tr>
<td><strong>Amazon EC2 EBS</strong></td>
<td></td>
</tr>
<tr>
<td>$0.100 per GB-month of provisioned storage (blended price)*</td>
<td>2,367 GB-Mo</td>
</tr>
<tr>
<td>$0.10 per 1 million I/O requests</td>
<td>76,834 IOs</td>
</tr>
<tr>
<td>$0.125 per GB-Month of snapshot data stored (blended price)*</td>
<td>0.344 GB-Mo</td>
</tr>
<tr>
<td><strong>Amazon Simple Notification Service</strong></td>
<td></td>
</tr>
<tr>
<td>$0.00</td>
<td></td>
</tr>
<tr>
<td><strong>US East (Northern Virginia) Region</strong></td>
<td></td>
</tr>
<tr>
<td>First 100,000 Amazon SNS API Requests per month are free</td>
<td>54 Requests</td>
</tr>
<tr>
<td><strong>AWS Data Transfer (excluding Amazon CloudFront)</strong></td>
<td></td>
</tr>
<tr>
<td>$0.00 per GB - data transfer in per month</td>
<td>0.034 GB</td>
</tr>
<tr>
<td>$0.00 per GB - first 1 GB of data transferred out per month</td>
<td>0.001 GB</td>
</tr>
<tr>
<td>$0.00 per GB of regional data transfer in/out (blended price)*</td>
<td>0.000020 GB</td>
</tr>
</tbody>
</table>
Billing Latency

EC2

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

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

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.

Compute Time

• Billing “start” and “stop” could 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

![Graph showing EC2 Compute Time Results]

- t1.micro
- m1.small
- c1.medium

Frac. of instances billed 2 hours vs. $T_{kill} - T_{launch}$
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

<table>
<thead>
<tr>
<th>Δ</th>
<th># launched</th>
<th># ran</th>
<th>Avg. uptime per instance (s)</th>
<th>Hours billed</th>
</tr>
</thead>
<tbody>
<tr>
<td>≤ 16</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>20</td>
<td>1</td>
<td>115</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>20</td>
<td>1</td>
<td>116</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>4</td>
<td>117</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>6</td>
<td>118</td>
<td>3</td>
</tr>
</tbody>
</table>

- 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.

<table>
<thead>
<tr>
<th>Setup</th>
<th>Send % Reported</th>
<th>Receive % Reported</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Univ → EC2</td>
<td>-</td>
<td>95.9%</td>
</tr>
<tr>
<td>(2) EC2 → Univ</td>
<td>94.4%</td>
<td>-</td>
</tr>
<tr>
<td>(3) Zone X → Zone X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(4) Zone X → Zone X (public IP)</td>
<td>97.6%</td>
<td>97.2%</td>
</tr>
<tr>
<td>(5) Zone X → Zone Y</td>
<td>97.1%</td>
<td>97.5%</td>
</tr>
<tr>
<td>(6) Reg X → Reg Y</td>
<td>95.9%</td>
<td>96.8%</td>
</tr>
</tbody>
</table>

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.