Improving the Efficiency of Cloud Systems through Enhanced Service Flexibility

Mohammad Shahrad
“You got to change your research area! Cloud was for the past decade.”

Public Cloud Revenue Projected To Reach $411B By 2020

$411B is not trivial!

<table>
<thead>
<tr>
<th>Rank</th>
<th>Country</th>
<th>Nominal GDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>Iran</td>
<td>$ 439.5 B</td>
</tr>
<tr>
<td>27</td>
<td>Austria</td>
<td>$ 416.6 B</td>
</tr>
<tr>
<td>28</td>
<td>Norway</td>
<td>$ 398.8 B</td>
</tr>
<tr>
<td>29</td>
<td>UAE</td>
<td>$ 382.6 B</td>
</tr>
</tbody>
</table>

Growing Services
AWS Services

Usage Patterns

Service Valuation

Heterogeneous Infrastructure

Flexibility is a must in the cloud.
Various VM Types to Suit Everyone

Amazon Elastic Compute Cloud (EC2) offers 106 instance types. (on 9/18)

<table>
<thead>
<tr>
<th>Instance Family</th>
<th>Current Generation Instance Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>General purpose</td>
<td>t2.nano</td>
</tr>
<tr>
<td>Compute optimized</td>
<td>c4.large</td>
</tr>
<tr>
<td>Memory optimized</td>
<td>r4.large</td>
</tr>
<tr>
<td>Storage optimized</td>
<td>d2.xlarge</td>
</tr>
<tr>
<td>Accelerated computing</td>
<td>f1.2xlarge</td>
</tr>
</tbody>
</table>

Current cloud offerings are far from fully flexible!

All-or-nothing Service-level Agreements (SLAs):
- Uptime guarantees [SoCC ’16]
- Isolation mechanisms

Lack of provider/client information sharing:
- Demand uncertainty [SoCC ’17]
- Application demand
1. Incentivizing Self-Capping to Increase Cloud Utilization

Cristian Klein  Liang Zheng  Mung Chiang  Erik Elmroth  David Wentzlaff
A Quick Survey

How much is the installment cost of a typical cloud datacenter?

(A) $O(1M)$  (B) $O(10M)$  (C) $O(100M)$

What is the typical CPU utilization of cloud servers?

(A) ~10% ✓  (B) ~30% ✓  (C) ~75% ✓

Latency-sensitive interactive services  Mix of workloads including online services  Large continues batch workloads
Managing Uncertainty is Fundamentally Challenging

Demand Fluctuations

QoS guarantees

Spare Capacity
Can we motivate tenants to fluctuate less?

Mechanisms to control capacity demand fluctuations.

Economic incentives to change behavior.

Graceful Degradation Techniques
Graceful Degradation (GD) Methodology

Adaptive Bitrate Streaming

Brownout Self-Adaptation
Deactivating non-essential content to maintain response time.

Shaping the Capacity Demand

Capacity demand variation of a real service provider (Bitbrains) using 1,250 VMs.

- **Capacity pair** \((C_b, C_d)\)
- **Price pair** \((p_b, p_d)\)

- **Not delivered** (too expensive)
- **Charged based on usage** \((p_d)\)
- **Always Charged** \((p_b)\)
System Overview

- **capacity controller**
- **price controller**
- **hypervisor**
- **service provider**
- **infrastructure provider**
- **clients**

- **GD-compliant application**
- **capacity demand**
- **capacity request**
- **dynamic price**
- **capacities**
- **queries**

Diagram:
- Clients send queries to the **GD-compliant application**.
- The application requests capacity from the **capacity controller**.
- The capacity controller determines the capacity demand and sends it to the **price controller**.
- The price controller sets the price based on demand and sends it to the **hypervisor**.
- The hypervisor provides capacities to the service provider.
Demand Distributions Generally Consistent

![Graphs showing CPU Utilization for Bitbrains and Materna from different months.]

- **7/2013**
- **8/2013**
- **9/2013**
- **11/2015**
- **12/2015**
- **1/2016**
Tenant’s Profit Maximization

Given a price pair, tenants can select the best capacity pair:

<table>
<thead>
<tr>
<th>Optimal Reserved Capacity</th>
<th>Optimal Capacity Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c^<em><em>b = \int</em>{c^</em><em>b}^{c</em>{\text{max}}} f(c) \text{dc} = \frac{p_b}{p_d}$</td>
<td>$c^<em><em>d = \frac{\int</em>{c^</em><em>d}^{c</em>{\text{max}}} k R(c, \frac{c^<em><em>d}{c}) f(c) \text{dc}}{\int</em>{c^</em><em>d}^{c</em>{\text{max}}} p_d f(c) \text{dc}}$</td>
</tr>
</tbody>
</table>

$k$: Degree of homogeneity for revenue function

$R(c, \lambda \theta) = \lambda^k R(c, \theta)$
Dynamic Pricing to Control Utilization

Regardless of revenue function:

\[ \text{Reserved Capacity} \propto \frac{p_d}{p_b} \]
\[ \text{Total Capacity} \propto \frac{1}{p_d} \]

Monolithic incentives
Evaluation

- Simulations using real-world traces
  - Fast design space exploration
- Implemented and tested a prototype
  - Validation of simulations
- Scalability tests

<table>
<thead>
<tr>
<th></th>
<th>Cutting Peaks</th>
<th>Filling Valleys</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD-compliant</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>GD-noncompliant</td>
<td>✗</td>
<td>✔️</td>
</tr>
</tbody>
</table>
Effective Utilization ($u_e$)

More degradation

More sensitive to GD

Less sensitive to GD

- Capacity getting more expensive compared to revenue
- Capacity getting cheaper compared to tenant’s revenue

GD-compliant ($k=k_0=0.7$)
GD-compliant ($k=0.9k_0$)
GD-compliant ($k=1.1k_0$)
GD-noncompliant
A Multi-tenant Scenario

More degradation

Increased on-demand price

Dynamic $P_d$ is updated every day.
Prototype Evaluation

- Used **Xen** hypervisor (best CPU scaling)
- GD-enabled RUBiS (eBay-like benchmark)
- Scaled down traces in two dimensions:
  - Time-wise (60X faster)
  - Magnitude-wise
    - (50,000X less, to fit in 32 cores)

https://github.com/cristiklein/gdinc-experiment
Demand uncertainty decreases utilization.

A flexible service model can incentivize tenants to fluctuate less.

Graceful degradation resilience methodology can be used.

A well-defined pricing model allows:
- Profit maximization for tenants;
- Control utilization for infrastructure providers.
2.

Availability Knob

Flexible User-defined Availability in the Cloud

David Wentzlaff
State of Uptime Service Level Objectives (SLO)

Google Compute Engine Service Level Agreement (SLA)

Downtime: 4min 20sec per month

Downtime: 21min 40sec per month
In case the uptime SLO not met, ...

<table>
<thead>
<tr>
<th>Monthly Uptime Percentage</th>
<th>Service Credit Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less than 99.99% but equal to or greater than 99.0%</td>
<td>10%</td>
</tr>
<tr>
<td>Less than 99.0%</td>
<td>30%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Monthly Uptime Percentage</th>
<th>Percentage of monthly bill for the respective Covered Service in the Region affected which did not meet SLO that will be credited to future monthly bills of Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>99.00% - &lt; 99.99%</td>
<td>10%</td>
</tr>
<tr>
<td>95.00% - &lt; 99.00%</td>
<td>25%</td>
</tr>
<tr>
<td>&lt; 95.00%</td>
<td>50%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MONTHLY UPTIME PERCENTAGE</th>
<th>SERVICE CREDIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 99.99%</td>
<td>10%</td>
</tr>
<tr>
<td>&lt; 99%</td>
<td>25%</td>
</tr>
<tr>
<td>&lt; 95%</td>
<td>100%</td>
</tr>
</tbody>
</table>
Is there also a dark side to cloud industry’s achievement culture?
What’s wrong with fixed high availability?

Cloud customers:
• Various downtime demands
• Different service valuations

Heterogeneous cloud infrastructure
• Network redundancy
• Server type (CPU, storage, etc.)

Redundancy is not free!
The Availability Knob (AK)
Changes Required to Support AK

API

Cloud Scheduler

Management

Failure stats

Scheduling
Flexible Availability SLAs

1. Desired Availability & Calculation Period
   (e.g., 99.8% / 7 days)

2. Availability Price Scale
   e.g., (99.99%, 1), (99.9%, 0.95)

3. Variable Service Credit

![Graph showing service credit and availability for different providers and availability levels.](image)
The AK Scheduler

Scheduling policy:
For candidate servers to host a VM, find the cheapest resource so that expected time to next failure meets requested downtime.

Extra run-time policies:
• Benign VM Migration
• Deliberate Downtime
**Benign VM Migration**

Periodic migration of over-served VMs to cheaper resources.

---

**Deliberate Downtime**

Deliberately fail VMs near the end of their period to:
- Build market incentives
- Bid redeemed resources

![Graph showing relationship between Delivered Uptime and Requested Uptime](graph_image)
Incentive Compatibility

Providers can:
- Neglect meeting SLOs

Customers can:
- Run buggy applications
- Cause deliberate downtime
- Ask for unreal uptime

AK uses game theory to build a pricing scheme:
1. Providers maximize profit by not violating SLOs
2. Clients pay less by asking for their true demands
Evaluation of AK

- Infrequency of Failures
  - Accelerated testing
  - Large scale simulations

1. Stochastic simulations in MATLAB
2. Prototype implementation with OpenStack
A Peek into the OpenStack Prototype
Availability-aware Scheduler

1000 machines, 12000 users, Normal demand distribution in [99.9, 99.999], 6 months long, BVM every 1hr for top 10% of over-served clients
Benign Migrations Offload VMs to Cheaper Machines
Supply chain flexibility $\rightarrow$ market efficiency

Knowing user demand can enable better management.

Game theory to ensure mutual economic incentives.

Leveraging reliability/cost trade-offs
There are numerous inefficiency corner cases in the cloud, but …

"Big guys can’t scratch their back!"