The Straw that Broke the Camel’s Back: Safe Cloud Overbooking with Application Brownout

Luis Tomas, Cristian Klein, Johan Tordsson, Francisco Hernandez-Rodriguez

Umeå University, Sweden

Cloud and Autonomic Computing (CAC)
2014-09-10 London, UK
Problem: Low Utilization

• Causes
  – Pre-defined VM sizes: S, M, L, XL
  – Uncertainty about application requirements
  – VM sprawl
    • Private data-center: idle VM not stopped by owner
Overbooking

- Accept VMs based on **predicted (real)** instead of **requested (nominal)** size
- Best if there is a “stream” of short-lived VMs:
  - Analytics, continuous build server, etc.
Previous Work: Overbooking

System Overview

Try to avoid overpassing the total real capacity:
- Performance degradation and/or failures
- Avoid overpassing total capacity at each server

Acceptance decisions
Risks evaluation

Monitoring and Profiling Tools

L. Tomás, J. Tordsson,
Improving cloud infrastructure utilization through overbooking, CAC, 2013
Fuzzy Overbooking Controller

- Proposed several strategies
  - Trade-off between risk and utilization

<table>
<thead>
<tr>
<th></th>
<th>Average utilization</th>
<th>% problems</th>
<th>problem size</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Risk</td>
<td>38.56 %</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pessimistic</td>
<td>61.94 % (1.61)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Realistic</td>
<td>78.08 % (2.02)</td>
<td>1.07</td>
<td>0.26%</td>
</tr>
<tr>
<td>Optimistic</td>
<td>74.39 % (1.93)</td>
<td>8.80</td>
<td>0.33%</td>
</tr>
<tr>
<td>ACSOS</td>
<td>75.39% (1.96)</td>
<td>6.58</td>
<td>0.33%</td>
</tr>
<tr>
<td>CapacityAware</td>
<td>77.03 % (2.00)</td>
<td>0.88</td>
<td>0.30%</td>
</tr>
<tr>
<td>ACSOS-Opt</td>
<td>74.30 % (1.93)</td>
<td>6.40</td>
<td>0.32%</td>
</tr>
<tr>
<td>ACSOS-Real</td>
<td>77.87 % (2.02)</td>
<td>0.25</td>
<td>0.17%</td>
</tr>
<tr>
<td>ACSOS-Cap</td>
<td>78.11 % (2.03)</td>
<td>0.87</td>
<td>0.22%</td>
</tr>
</tbody>
</table>

- Average utilization increased
- Performance was sometimes degraded
  - Due to unexpected events

Previous Work: Brownout

• Disable non-essential content
  – Minimally intrusive
• E.g. recommendations
  – 50% increase in sales *
• Challenge
  – Maximize non-essential content
  – Avoid high response times

Brownout: Inside a Replica

Client → Logic → Database

$\tau$ = target response time

$\theta$ = probability of serving non-essential content (dimmer)

$t$ = response times

C. Klein, M. Maggio, F. Hernández-Rodriguez, K-E Årzén,
Brownout: building more robust cloud applications, ICSE, 2014
Brownout: Results
## Recap of Previous Work

<table>
<thead>
<tr>
<th>Overbooking</th>
<th>Brownout</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔ Increase utilization</td>
<td>✔ Maintains application responsiveness</td>
</tr>
<tr>
<td>✔ Knowledge about data center utilization</td>
<td>✔ Knowledge about applications performance</td>
</tr>
<tr>
<td>✔</td>
<td>✔ Quick reaction</td>
</tr>
<tr>
<td>✗ Increase (risk of) performance degradation</td>
<td>✗ Reduce users’ experience</td>
</tr>
<tr>
<td></td>
<td>✗ Tend to keep resources slightly underutilized</td>
</tr>
</tbody>
</table>

### Goal

- ✔ Increase utilization
- ✔ Maintain responsiveness
- ✔ Quickly react to unexpected situations
BOB: Brownout + OverBooking

Two approaches:
• naïve: brownout and overbooking used “as-is”
• smart: explicit communication is added
Smart BOB

- Brownout controller sends matching value
  \[ f_i^k = 1 - \frac{t_i^k}{\bar{t}_i} \]

- New target utilization is computed

```
Algorithm 1 Target Utilization Controller

Configuration parameters: duration, update interval of output
TU_{min} and TU_{max}, minimum and maximum acceptable target
utilization
incrementStep, how much to increase utilization

1: TU \leftarrow 0.80 \{ default target utilization \}
2: while true do
3:    n_i \leftarrow \text{number of negative matching values sent by application}
4:    \quad i \text{ since the last update}
5:     if num > 0 then
6:        totalImpact = \sum n_i
7:        \quad \text{decrementStep} = \sqrt{\frac{\text{totalImpact}}{\text{num} \times \text{duration}}}
8:        TU < \quad \text{TU}_{\max} - \text{TU}_{\min} \times \text{decrementStep}
9:     else
10:        \quad TU \leftarrow TU + (\text{TU}_{\max} - \text{TU}_{\min}) \times \text{incrementStep}
11:    saturate TU between TU_{\min} and TU_{\max}
12:    send TU to overbooking controller
13:    sleep for duration
```
Evaluation

**RQ1:** Does BOB work?

**RQ2:** How important is communication?

- **Infrastructure:**
  - Single machine 32 cores, QEMU hypervisor

- **Workload:**
  - Services (interactive applications)
  - Jobs (non-interactive applications)

![Wikipedia-like load](image1)

![FIFA-like load](image2)

![Extreme varying](image3)
Results: RUBiS, Wikipedia (1)

Utilization is increased from 53% to 78%

No Overbooking

Overbooking w/o brownout

95th percentile response time increases from 171ms to 752ms
Results: RUBiS, Wikipedia (2)

Utilization is maintained

95th percentile response time decreases from 752ms to 408ms

Overbooking w/o brownout

Non-essential content drops from 100% to 71.4%

Naïve BOB
Results: RUBiS, Wikipedia (3)

Utilization is only slightly decreased from 78% to 72%

Naïve BOB

Non-essential content increases from 71.4% to 98.1%

Smart BOB
Results: RUBiS, Varying load

Naïve BOB
Target utilization 70%

Naïve BOB
Target utilization 80%

Either allocated capacity or user experience suffers

Smart BOB
Auto-tunes target utilization
Conclusions and Perspectives

• Overbooking relies on predictions
  – Unexpected events => incorrect overbooking decisions
    => degraded application performance
• Brownout: building more robust applications
  – Discard non-essential content
• Brownout + OverBooking
  – Improve utilization
  – Minimize application degradation
  – *Explicit communication proved useful*

• Perspectives
  – CPU pinning to reduce overbooking impact
  – Pricing models for brownout applications
Thank you for your attention!

Cristian Klein

The Straw that Broke the Camel’s Back: Safe Cloud Overbooking with Application Brownout

https://github.com/cloud-control
References

Appendix
New Problem: Unexpected Events

- 82% of end-users give up on a lost payment transaction*
- 25% of end-users leave if load time > 4s**
- 1% reduced sale per 100ms load time**
- 20% reduced income if 0.5s longer load time***

* JupiterResearch  ** Amazon  ***Google
Robustness to Model Uncertainties

\[ \alpha = \tilde{\alpha} \cdot \Delta \alpha \]
Replica Controller (1)

- Need to adapt to changes
  - Number of users
  - Available capacity
- Not all requests take the same time
  - E.g., cached in memory, disk
- Need to reject disturbances
  - E.g., NTP daemon firing up, cron jobs
Replica Controller (2)

• Start from a simple model

\[ t^{k+1} = \alpha^k \cdot \Theta^k + \delta t^k \]

• Adaptive PI controller

\[ \Theta^{k+1} = \Theta^k + \frac{1 - p_1}{\alpha} \cdot e^{k+1} \]

• \( \alpha \) estimated using RLS