Consistency-Based SLA for Cloud Storage

Patryk Hes
Distributed Systems course, 2016

Based on:
Consistency-Based Service Level Agreements for Cloud Storage
Douglas B. Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos K. Aguilera, Hussam Abu-Libdeh
Cloud Storage

- We consider **NoSQL** data stores (key-value)
- Many different solutions available
Trade-offs

- Read operations: **strong** vs **eventual** consistency
- All mentioned solutions support choosing between those two consistencies...
Trade-offs

- Read operations: **strong** vs **eventual** consistency
- All mentioned solutions support choosing between those two consistencies... but do developers know what's best for their apps?
Wasted resources

data = WeakRead(key);
display(data);
latestdata = StrongRead(key);
if (data != latestdata) {
    display(latestdata);
}
SLI, SLO, SLA
definition by examples

- Service level indicator
  website load time

- Service level objective
  Website load time is under 300ms in 99.9% cases.

- Service level agreement
  For every minute that website load time is not under 300ms in 99.9% cases, we pay $100.
SLA components in cloud storage

A handful of facts (as of 2013) should tell us something about the possible components of SLA:

- DynamoDB charges 2× more for consistent reads
- Amazon loses 1% of sales for each 100ms of latency
SLA components in cloud storage

• Let's make an agreement on **read** operations
• Indicators:
  – Consistency (strong > ... > eventual)
  – Latency (lower is better)
• Utility measure: how much user is willing to pay for consistency + latency
Pileus
(/ˈpaɪliəs/; Latin for cap)

source: youtube.com
Pileus SLAs

- Don't pick one solution – let SLA be a list of alternative preferences (*sub*-SLAs)
- Preferences should be ordered by utility

<table>
<thead>
<tr>
<th>Rank</th>
<th>Consistency</th>
<th>Latency</th>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>strong</td>
<td>150ms</td>
<td>1.0</td>
</tr>
<tr>
<td>2.</td>
<td>eventual</td>
<td>150ms</td>
<td>0.5</td>
</tr>
<tr>
<td>3.</td>
<td>strong</td>
<td>1s</td>
<td>0.25</td>
</tr>
</tbody>
</table>
Pileus architecture
from a bird's eye view

cloud storage

secondary site

replication

primary site

GET

PUT

client
Pileus
cloud (storage) upon a cloud (storage)
Key-value operations

CreateTable(name)
DeleteTable(name)

table = OpenTable(name)
session = BeginSession(table, sla)
EndSession(session)
Put(session, key, value)
value, conditionCode = Get(session, key, sla)

Get, Put – single-operation transactions
Data sharding

- Big tables can be sharded into smaller *tablets*.
- Tablets are the **granularity** of replication and are **independently replicated** on multiple storage nodes.
Data sharding

- Every tablet has its primary site
- Site = set of storage nodes within a datacenter

E.g. in Azure: site = three strongly-consistent nodes
Consistency guarantees

strong

A **Get**(key) returns the value of the last preceding **Put**(key) performed by any client.
Consistency guarantees

eventual

A Get(key) returns the value written by any Put(key).
Consistency guarantees

read-my-writes

A \textbf{Get(key)} returns the value written by the last preceding \textbf{Put(key)} in the same session or returns a later version.
Consistency guarantees

monotonic

A `Get(key)` returns the same or a later version as a previous `Get(key)` in this session.
Consistency guarantees bounded\(t\)

A **Get**(key) returns a value that is stale by at most \(t\) seconds.
Consistency guarantees

causal

A Get(key) returns the value of a latest Put(key) that causally precedes it or returns some later version. The causal precedence relation < is defined such that $op1 < op2$ if either:

(a) $op1$ occurs before $op2$ in the same session
Consistency guarantees

causal

A \textbf{Get}(key) returns the value of a latest \textbf{Put}(key) that causally
precedes it or returns some later version. The causal
precedence relation \(<\) is defined such that \(op1 < op2\) if either:

\begin{itemize}
  \item \(op1\) is a \textbf{Put}(key) and \(op2\) is a \textbf{Get}(key)
  that returns the version put in \(op1\)
\end{itemize}
Consistency guarantees

causal

A `Get(key)` returns the value of a latest `Put(key)` that causally precedes it or returns some later version. The causal precedence relation `<` is defined such that `op1 < op2` if either:

(c) for some `op3`, `op1 < op3 < op2`
Wasted resources, revisited

- Simplest way to be sure we get most of SLA: broadcast **Get**s to all sites
- Problem: money
- Pileus approach:

  maximize expected utility while sending a **Get** to one site at a time
Storage metadata

- State managed by each node:
  - tablet store = set of (key, value, timestamp) tuples
  - high timestamp = update timestamp of the latest data object version received by node

- All Puts assigned increasing timestamp on primary site

- Assuming we can combat network splits and data loss, we have a prefix of the overall Put sequence operations on every node.
Client's knowledge

- Client collects the information on highest timestamps of sites when performing **Get, Put** – or sometimes, on demand.
- **PNodeCons**(node, consistency, key) – estimate of the probability that the given storage node is sufficiently up-to-date to provide the given consistency guarantee for the given key.
- **PNodeLat**(node, latency) – estimate of the probability that the node can respond to **Get**s within the given response time.
- \( \text{PNodeSla} = \text{PNodeCons} \times \text{PNodeLat} \)
SelectTarget(SLA, key) =
maxutil = -1;
bestnodes = {};
bestlatency = ∞;
targetSLA = null;

foreach subSLA in SLA
  foreach node in key.replicas
    util = PNodeSla(node, subSLA.consistency,
      subSLA.latency, key) * subSLA.utility;
    if (util > maxutil)
      targetSLA = subSLA;
      maxutil = util;
      bestnodes = node;
    else if (util = maxutil)
      bestnodes = bestnodes + node;

foreach node in bestnodes
  if (node.latency < bestlatency)
    bestnodes = node;
    bestlatency = node.latency;
return targetSLA, bestnodes;
Experimental set-up

- Workload = Yahoo! Cloud Serving Benchmark
- Clients perform equal numbers of **Puts** and **Gets** to collection of 10k keys
- Session = 400 **Gets** and **Puts**
Experimental set-up: shopping cart

• Picked SLA

<table>
<thead>
<tr>
<th>Rank</th>
<th>Consistency</th>
<th>Latency</th>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>read-my-writes</td>
<td>300ms</td>
<td>1.0</td>
</tr>
<tr>
<td>2.</td>
<td>eventual</td>
<td>300ms</td>
<td>0.5</td>
</tr>
</tbody>
</table>

• Network topology

- **Primary**
  - England
- **Secondary**
  - US West
  - India
  - China

Edge labels:
- England to US West: 149ms
- England to India: 287ms
- England to China: 308ms
- US West to India: 161ms
- US West to China: 436ms
- India to China: 181ms
Experimental set-up: shopping cart

- U.S. West
- England
- India
- China
# Experimental set-up: shopping cart

<table>
<thead>
<tr>
<th>Client</th>
<th>Target SubSLA</th>
<th>Get from U.S.</th>
<th>Get from England</th>
<th>Get from India</th>
<th>SubSLA Met</th>
<th>Avg. Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>U.S.</td>
<td>1</td>
<td>90.9%</td>
<td>9.1%</td>
<td>0%</td>
<td>100%</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td>England</td>
<td>1</td>
<td>0%</td>
<td>100%</td>
<td>0%</td>
<td>100%</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td>India</td>
<td>1</td>
<td>0.2%</td>
<td>0%</td>
<td>95.9%</td>
<td>96.1%</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0%</td>
<td>0%</td>
<td>3.9%</td>
<td>3.9%</td>
<td></td>
</tr>
<tr>
<td>China</td>
<td>1</td>
<td>95.1%</td>
<td>0%</td>
<td>0.4%</td>
<td>95.5%</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4.5%</td>
<td>0%</td>
<td>0%</td>
<td>4.5%</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Consistency</th>
<th>Latency</th>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>read-my-writes</td>
<td>300ms</td>
<td>1.0</td>
</tr>
<tr>
<td>eventual</td>
<td>300ms</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Diagram:
- **Primary** nodes: US West, England, India, China
- **Secondary** nodes: US West, India, China
- Latencies: 149ms, 287ms, 308ms, 436ms, 161ms, 181ms
Experimental set-up: password

- Picked SLA

<table>
<thead>
<tr>
<th>Rank</th>
<th>Consistency</th>
<th>Latency</th>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>strong</td>
<td>150ms</td>
<td>1.0</td>
</tr>
<tr>
<td>2.</td>
<td>eventual</td>
<td>150ms</td>
<td>0.5</td>
</tr>
<tr>
<td>3.</td>
<td>strong</td>
<td>1s</td>
<td>0.25</td>
</tr>
</tbody>
</table>

- Network topology – the same as before
Experimental set-up: password

- **Primary**
- **Random**
- **Closest**
- **Pileus**

### Average Utility Per Read

- **U.S. West**
- **England**
- **India**
- **China**
## Experimental set-up: password

<table>
<thead>
<tr>
<th>Client</th>
<th>Target SubSLA</th>
<th>Get from U.S.</th>
<th>Get from England</th>
<th>Get from India</th>
<th>SubSLA Met</th>
<th>Avg. Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>U.S.</td>
<td>1</td>
<td>0%</td>
<td>100%</td>
<td>0%</td>
<td>99.4%</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0.6%</td>
<td></td>
</tr>
<tr>
<td>England</td>
<td>1</td>
<td>0%</td>
<td>100%</td>
<td>0%</td>
<td>100%</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td>India</td>
<td>1</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0%</td>
<td>0%</td>
<td>100%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td>China</td>
<td>1</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0%</td>
<td>100%</td>
<td>0%</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>

**Consistency**

<table>
<thead>
<tr>
<th>Consistency</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>strong</td>
<td>150ms</td>
</tr>
<tr>
<td>eventual</td>
<td>150ms</td>
</tr>
<tr>
<td>strong</td>
<td>1s</td>
</tr>
</tbody>
</table>
Varying latency

<table>
<thead>
<tr>
<th>Consistency</th>
<th>Latency</th>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>strong</td>
<td>150ms</td>
<td>1.0</td>
</tr>
<tr>
<td>eventual</td>
<td>150ms</td>
<td>0.5</td>
</tr>
<tr>
<td>strong</td>
<td>1s</td>
<td>0.25</td>
</tr>
</tbody>
</table>
### Varying utility

#### Consistency
- **strong**
- **eventual**

#### Latency
- **150ms**
- **1s**

#### Utility
- **1.0**
- **0.5 × multiplier**
- **0.25 × multiplier**

#### SLA Multipliers
- **2x**
- **1.5x**
- **1x**
- **0.5x**
- **0.1x**