A Self-Configurable Geo-Replicated Cloud Storage System

BY: MASOUD SAEIDA ARDEKANI AND DOUGLAS B. TERRY

PRESENTATION BY: CHINMAY KULKARNI
Background:

- Geo-Replication: Replicas on servers at multiple locations
- Consistency: Strong, Eventual, RMW, Monotonic, etc.
- Latency-Consistency Tradeoff
- Primary Replicas: Writes and Strongly Consistent Reads. Secondary Replicas: Intermediary Consistency Reads
- Pileus is a replicated key-value store that allows users to define their CAP requirements in terms of SLAs
Brief Overview of Pileus (A “CAP” Cloud):

- **SLA**: Interface between client and cloud service. Wish list. “I want the strongest consistency possible, as long as read operations return in under x ms.”

- Clients specify consistency-based SLAs which contain acceptable latencies and a utility (preference/weight)

- Monitor replicas of the underlying storage system

- Route read operations to servers that can best meet a given consistency-based SLA

<table>
<thead>
<tr>
<th>Rank</th>
<th>Consistency</th>
<th>Latency(ms)</th>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Strong</td>
<td>75</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>RMW</td>
<td>150</td>
<td>0.8</td>
</tr>
<tr>
<td>3</td>
<td>Eventual</td>
<td>750</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Table 1: Example of an SLA
• Pileus – Shortcomings:
  ➔ Pre-defined configuration
  ➔ Static

• Key issues:
  ➔ Where to place primary and secondary replicas?
  ➔ How many to deploy?
  ➔ Synchronization Period?

• Why not dynamically reconfigure replicas?
  ➔ Tuba
Main Contributions of Tuba:

• Dynamically, automatically and periodically reconfigure replicas to deliver maximum overall utility to clients

• Does this while respecting SLAs, costs and replication constraints

• Client can continue to read and write data while reconfiguration is carried out in parallel

• Leverage geo-replication for increased locality and availability
Configuration Selection:

- Configuration Service (CS)
- Configuration Generator
- SLAs
- Observed Latencies
- Hit/Miss Ratios
- R/W Ratios

Constraints:
- Replication Factor
- Location
- Sync. Period
- Cost in $

Costs:
- Data Storage
- R/W Operations
- Syncing
- Cost of reconfiguration

\[
\text{max} \left[ \frac{\text{Utility}}{\text{Cost}} \right]
\]

Fig.1 Configuration Selection
• Greedy Choice: Replicate data in ALL datacenters. BUT, there are constraints and cost considerations

• Ratios Aggregation for clients in the same locations with the same SLAs → Reduced computation

• New configuration is computed based on missed subSLAs and consistency requirements → E.g.: missed subSLA for strong consistency – Add Primary replica near client

• Constraint satisfaction

• Execute reconfiguration operations
Client Execution in Tuba – 2 Modes:

1. **Fast Mode**: Client has the latest configuration and holds a lease on the configuration for \((\Delta - d)\) seconds.

2. **Slow Mode**: Client suspects that the configuration has changed.

---

Client can’t read config. Because CS has exclusive lock

---

Fig. 2 Client Execution Modes
Tuba Implementation Details:

- Implemented on top of Microsoft Azure Storage (MAS)
- Extension of Pileus (Consistency–based SLAs taken from Pileus)
- Tuba = MAS + multi-site geo-replication + automatic reconfiguration

1. How do clients and the CS communicate?

2. How are client operations (Read/Write Operations) carried out?

3. How are CS reconfiguration operations carried out?
Client-CS Communication:

• Clients use a designated MAS shared container to communicate with the CS

• Clients periodically write their observed latencies, Hit-Miss Ratios, SLAs and Read-Write Ratios which the CS reads

• CS stores latest configuration and the RiP (Reconfiguration-in-Progress) flag

• Tuba allows clients to cache the current configuration of a tablet called a *cview*

---

**Fig.3 Writes to Shared Container**
Client Read Operations:

1. Select replica
2. Send request to replica. Get reply
3. Is client in fast mode?
   - Yes: Return read data to application
   - No: Strongly consistent read?
     - Yes: Check: Is replica still primary?
       - Yes: 
       - No: Abort & retry
     - No: 

Fig.4 Client Read Operation
Client Write Operations (Single-Primary Write):

1. Data
2. Fast mode
3. Fast mode interval > Write operation time
4. RiP set?
5. Yes: RiP set?
   - Yes: Write to primary replica. Get response
   - No: Refresh cview
6. No: Undo Write & Abort
7. Slow mode

Slow mode
1. Get lease on config. blob
2. Write to primary replica
3. Done

Primary replica changed?
1. Yes: Client still in fast mode?
   - Yes: Done
   - No: Slow mode
2. No: Done

Fig. 5 Client Single-Primary Write Operation
Client Write Operations (Multi-Primary Write):

Get lease on config. blob → Add WiP flag to blob’s metadata at main primary replica → Add WiP flag to blob’s metadata at non-main primary replicas:

- ETag1 Changed?
  - Yes → Abort
  - No → Write to blob on main primary site

- ETags Changed?
  - Yes → Write to blob on other primary sites
  - No → Clear WiP flag at non-main primary replicas → Clear WiP flag at main primary replica → Done

Main primary replica always holds the *truth!* i.e. the latest data.

Fig. 6 Client Multi-Primary Write Operation
CS Reconfiguration Operations:

- Adjust synchronization period
- Add Secondary Replica
- Remove Secondary Replica
- Change Primary Replica
- Add Primary Replica
Adjust Synchronization Period (*adjust_sync_period*):

- Defines how often secondary replicas sync with primary replicas

- ↓ sync period, ↑ freq of sync, ↑ up-to-date secondary replicas, ↑ chance of hitting intermediary consistency read subSLAs

- Less costly as compared to adding/moving replicas

- No directly observable change for clients
Add/Remove Secondary Replica \((\text{add/remove\_secondary}(\text{site}_i))\):

- E.g.: Consider an online multiplayer game

- Add secondary replica near users (at \(\text{site}_i\)) during peak times

- Will provide better utility in case of this SLA

- Can remove the secondary replica once user traffic goes down to reduce cost

<table>
<thead>
<tr>
<th>Rank</th>
<th>Consistency</th>
<th>Latency (ms)</th>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RMW</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Monotonic</td>
<td>90</td>
<td>0.6</td>
</tr>
<tr>
<td>3</td>
<td>Eventual</td>
<td>450</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Table 2: SLA of an online multiplayer game
Change/Add Primary Replica \((\text{change/add\_primary}(\text{site}_i))\):

- **Secondary replica exists at \(\text{site}_i\)?**
  - **Yes**: Make replica WRITE\_ONLY
    - Set RiP flag in config. blob metadata
    - Wait \(\Delta\) seconds so that all clients go to slow mode
    - Break all client leases and get lease on config. blob
    - Wait for safe threshold = max allowed lease time
    - Install temporary config.
    - Remove RiP flag
    - Make \(\text{site}_i\) the solo primary replica (if change operation), or add to list of primary replicas (if add)
    - **Done**
  - **No**: Create replica at \(\text{site}_i\) and sync with primary replica

Increase hits on **strongly consistent reads** based on geographical variation of user traffic.

**Fig. 7 Change/Add Primary Replica**
Fault-Tolerance in Tuba:

- Replica Failure:
  - Rare. Each site is a collection on 3 Azure servers
  - Failed replicas can be removed via reconfiguration operations
    - `add_primary(site_i)`,
    - `change_primary(site_i)`,
    - `remove_secondary(site_i)`,
    - `add_secondary(site_i)`

- Client Failure:
  - What if client fails mid-way through a multi-primary write?
  - Recovery process used to complete the writes. Reads from the main primary replica (the *truth*).
• CS Failure:
  ➔ No direct communication between clients and CS
  ➔ If CS fails, clients can still remain in fast mode (provided RiP flag is not set)
  ➔ Even if RiP flag is on, clients can do R/W in slow mode
  ➔ If the RiP flag is on for too long, impatient clients waiting too long in slow mode can clear it
  ➔ RiP off, so CS aborts reconfigurations (incase it was alive and just slow)
  ➔ Changes made to RiP flag are conditional on ETags
Experiments:

- **Setup:**
  - 3 storage accounts (SUS, WEU and SEA)
  - Active clients are normally distributed along US West Coast, WEU and Hong Kong
  - Simulate the workload of users in different areas at different times
  - 150 clients at each site (over a 24-hour period)
  - Each tablet accessed by 450 distinct clients everyday
  - Primary replica in SEA and secondary replica in WEU
  - Global replication factor = 2
  - No multi-primary schemes allowed
  - YCSB Workload B (95% Reads and 5% Writes)
Average Overall Utility (AOU):
→ Average utility delivered for all read operations from all clients

Experiments done with no reconfiguration, reconfigurations every 2 hours, every 4 hours and every 6 hours

Tuba with no reconfigurations = Pileus and AOU for 24-hour period is 0.72

With constraints max AOU = 0.92

Table 3: SLA Used for Experimentation

<table>
<thead>
<tr>
<th>Rank</th>
<th>Consistency</th>
<th>Latency (ms)</th>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Strong</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>RMW</td>
<td>100</td>
<td>0.7</td>
</tr>
<tr>
<td>3</td>
<td>Eventual</td>
<td>250</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Table 4: AOU Observations for Different Reconfiguration Periods

<table>
<thead>
<tr>
<th></th>
<th>6h</th>
<th>4h</th>
<th>2h</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOU</td>
<td>0.76</td>
<td>0.81</td>
<td>0.85</td>
</tr>
<tr>
<td>AOU Improvement % over No reconfiguration</td>
<td>5</td>
<td>12</td>
<td>18</td>
</tr>
<tr>
<td>AOU Improvement % over Max Achievable AOU</td>
<td>20</td>
<td>45</td>
<td>65</td>
</tr>
</tbody>
</table>
Fig. 8 Tuba With a 4-Hour Reconfiguration Period

Table 5: Tuba Reconfigurations done

<table>
<thead>
<tr>
<th>Action</th>
<th>Configuration Pri.</th>
<th>Configuration Sec.</th>
<th>CS Reconfiguration Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SEA</td>
<td>WEU</td>
<td><em>change_primary</em>(WEU)</td>
</tr>
</tbody>
</table>
| 2      | WEU                 | SEA                | *add_secondary*(SUS)  
|        |                     |                    | *remove_secondary*(SEA)                        |
| 3      | WEU                 | SUS                | *change_primary*(SUS)                         |
| 4      | SUS                 | WEU                | *add_secondary*(SEA)  
|        |                     |                    | *remove_secondary*(WEU)                        |
| 5      | SUS                 | SEA                | *change_primary*(SEA)                         |
| 6      | ...                 | ...                | ************************************************** |

Could’t predict client behavior
Results:

- Improvements in hit percentages for strongly consistent reads due to reconfiguration

- Reconfiguration done automatically → No manual intervention – Faster
  → No need to stop the system
  → Client R/W operations occur in parallel to the reconfiguration operations

Fig. 9 Hit Percentage of SubSLAs
Pros/Advantages of Using Tuba:

1. Dynamically change configurations to handle change in client requests

2. Change configurations on a per-tablet basis

3. Client R/W operations can be executed in parallel with reconfiguration

4. Easily extensible to existing systems that are already using MAS/Pileus

5. Provides default constraints to avoid aggressive replication

6. Reduced computation using hit-miss ratio aggregation

7. Good fault-tolerance (recovery processes, client RiP flag over rides, etc.)
Cons/Future Work:

1. Scalability Issues since configuration generator generates all possible configurations. At 10,000 clients and 7 storage sites \(\rightarrow\) 170 seconds

2. Pre-pruning instead of post-pruning based on constraint satisfaction

3. Make CS proactive instead of reactive. Make reconfigurations by predicting future poor utility \(\rightarrow\) Machine learning methods

4. For multi-primary operations, the first primary node is the main primary. Choose one so as to reduce overall latency?

5. Clients keep polling for new configuration. Use Async. messages instead?
Conclusion:

• Tuba is a geo-replicated key-value store that can dynamically select optimal configurations of replicas based on consistency-based SLAs, constraints, costs and changing client demands

• Successfully uses utility/cost to decide the optimal configuration

• Carries out automatic reconfiguration in parallel with client R/W operations

• Tuba is extensible: built on top of Microsoft Azure Storage and extends Pileus

• Provides increase in consistency. E.g.: With 2-hour reconfigurations, reads that returned strongly consistent data increased by 63%. Overall utility went up by 18%. 
Piazza Questions/Discussion Points:

• Are there times when system blocks?
  → While adding/changing primary replica, no writes from when CS takes lease on configuration till new configuration is set up
  → But this duration is short (1 RTT from CS to config blob + safe threshold)

• No experiments to measure reconfiguration load & failure cases

• No SLA validation mechanisms. No constraints → default constraints

• Security issues

• Client failure → Multiple recovery processes are wasteful
Thanks for listening!

Questions?