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)

Rank	Consistency	Latency(ms)	Utility
1	Strong	75	1
2	RMW	150	0.8
3	Eventual	750	0.05

Table 1: Example of an SLA

- Monitor replicas of the underlying storage system
- Route read operations to servers that can best meet a given consistency-based 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:

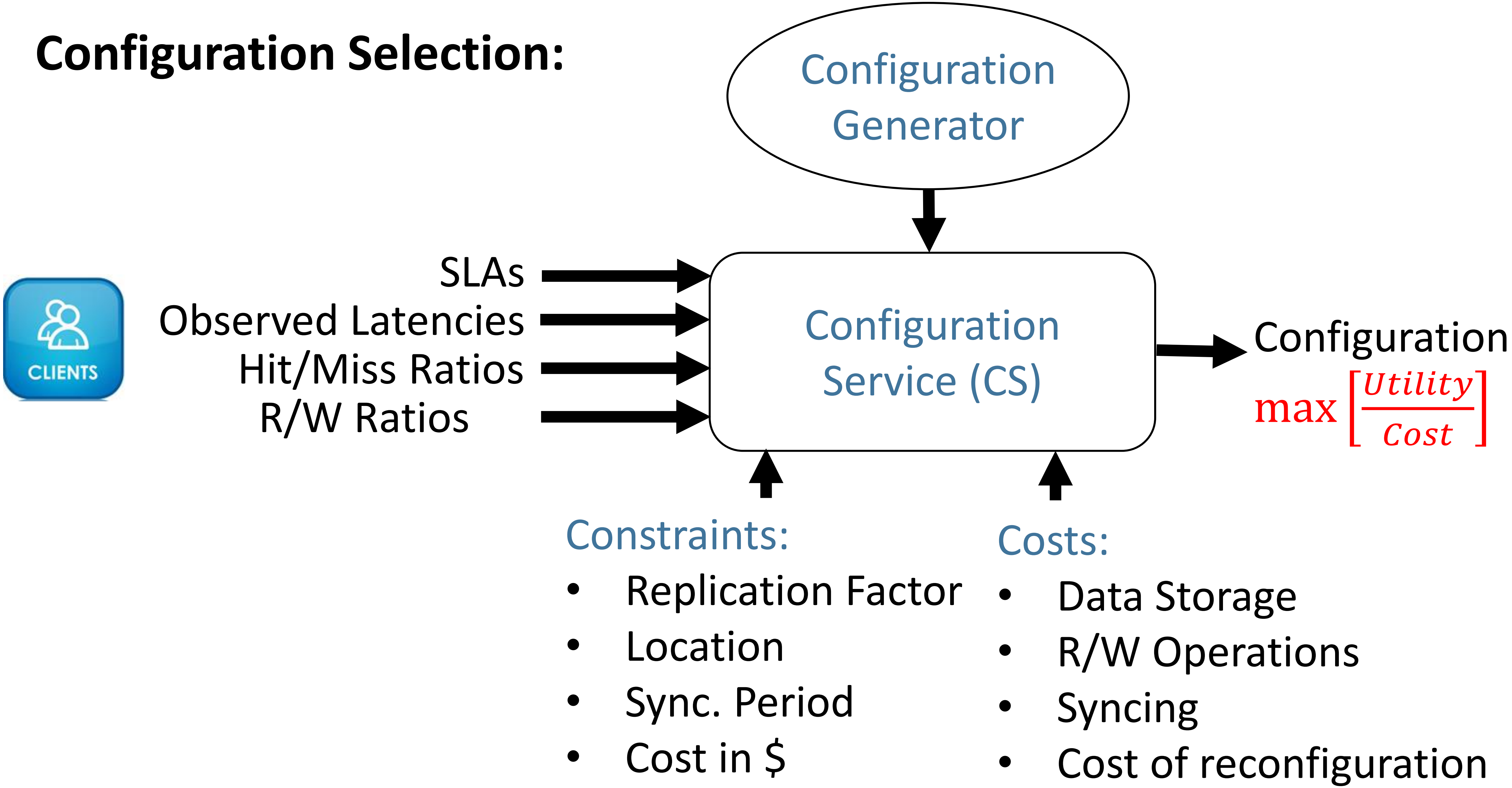


Fig.1 Configuration Selection

- 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:

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

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

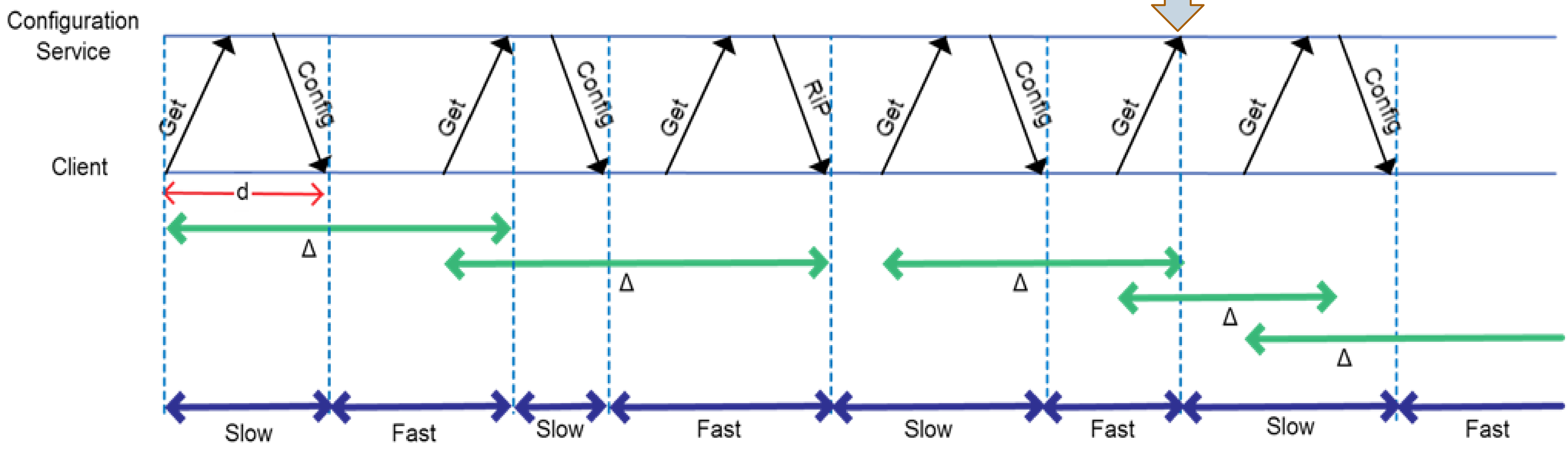


Fig.2 Client Execution Modes

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*

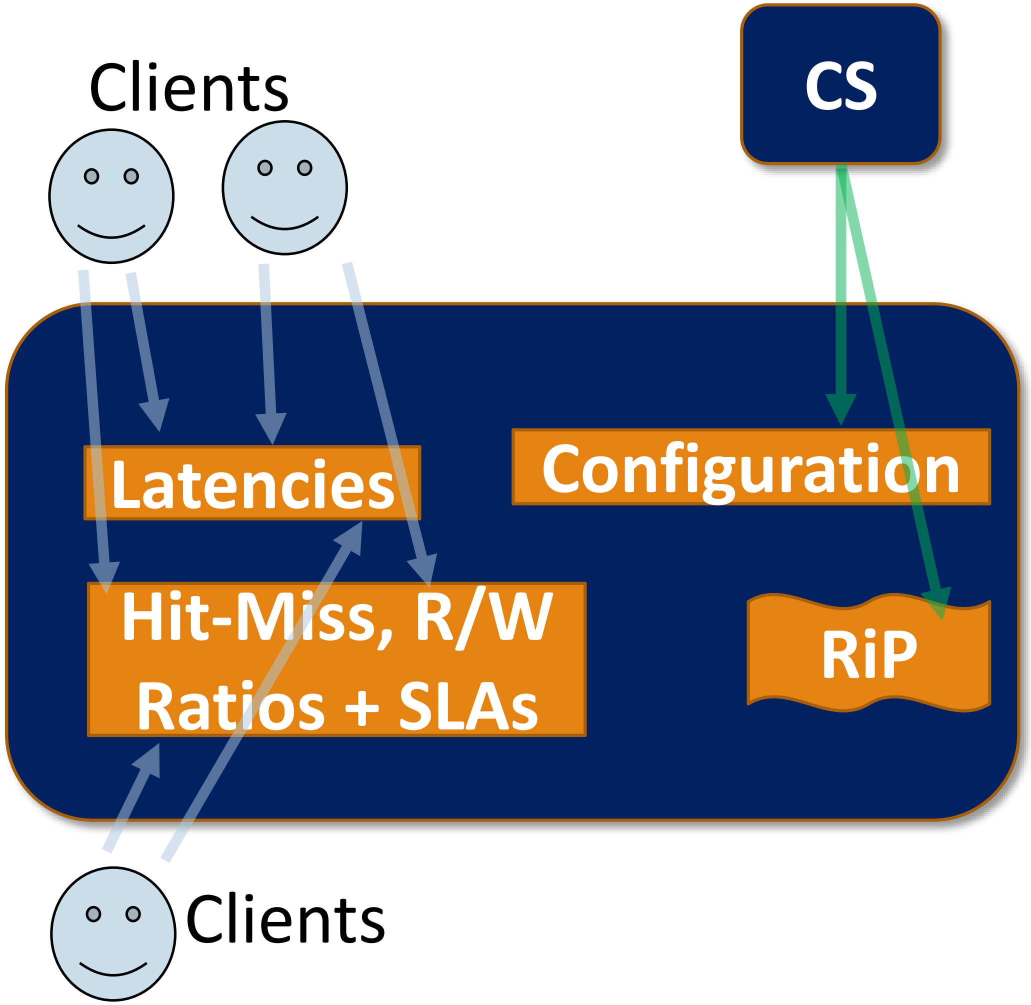


Fig.3 Writes to Shared Container

Client Read Operations:

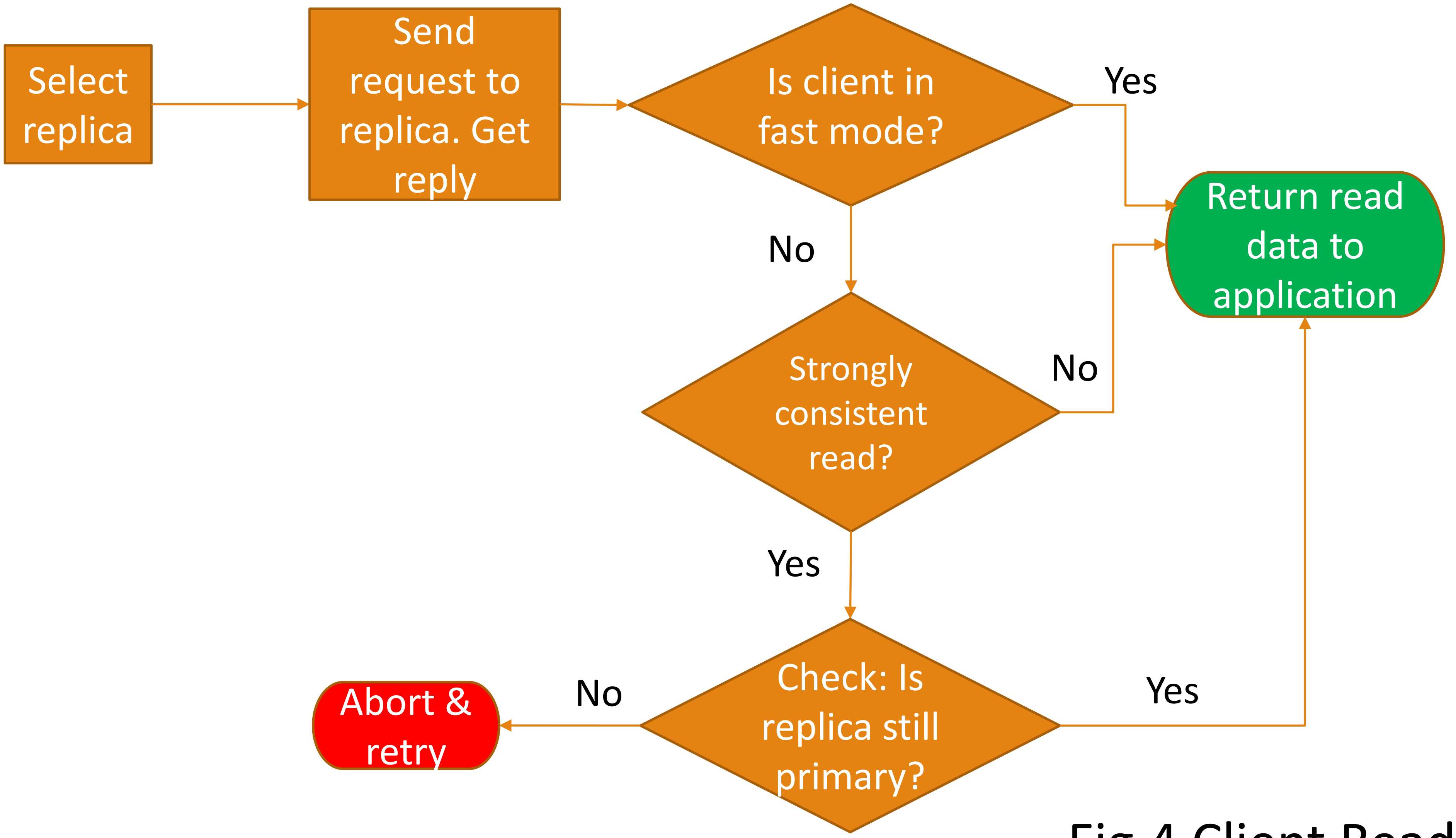


Fig.4 Client Read Operation

Client Write Operations (Single-Primary Write):

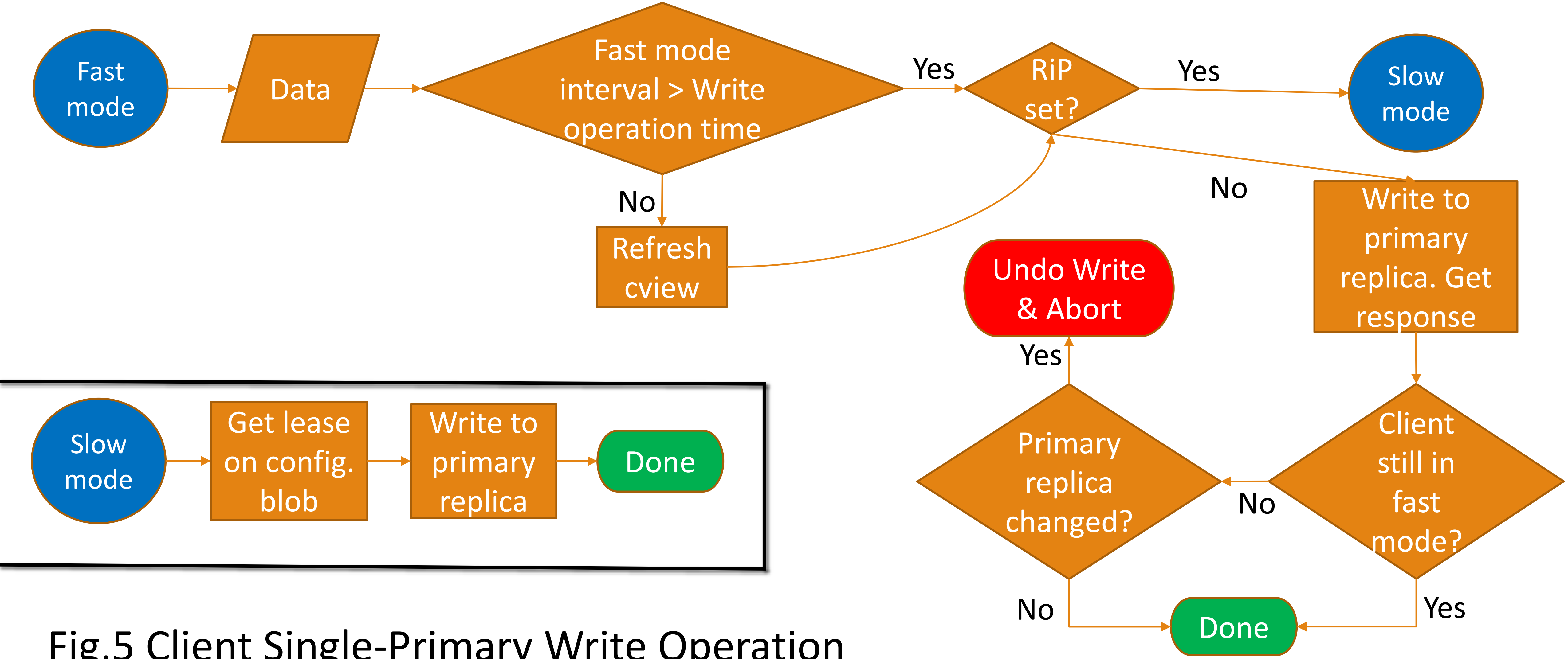
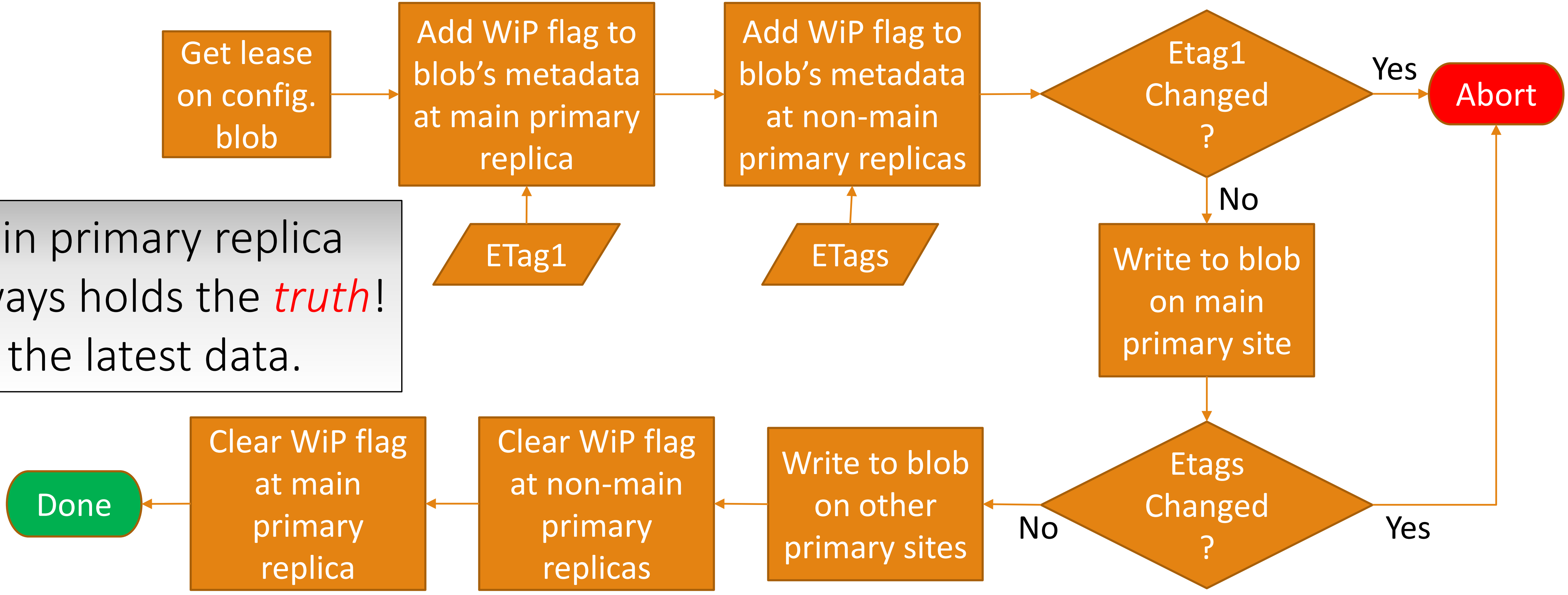


Fig.5 Client Single-Primary Write Operation

Client Write Operations (Multi-Primary Write):



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

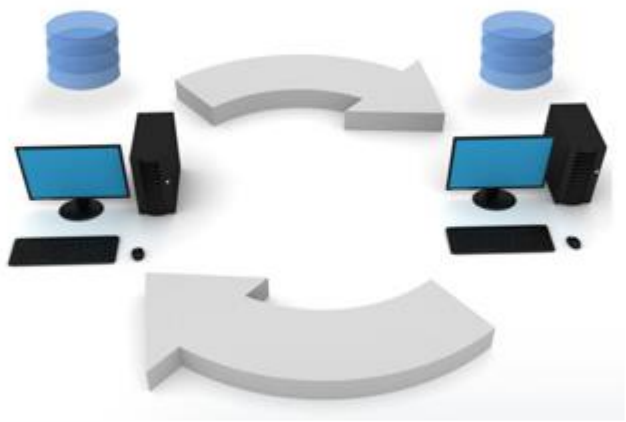
Adjust Synchronization Period *(adjust_sync_period)*:

- 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 *(add/remove_secondary($site_i$))*:

- E.g.: Consider an online multiplayer game

- Add secondary replica near users (at $site_i$) during peak times

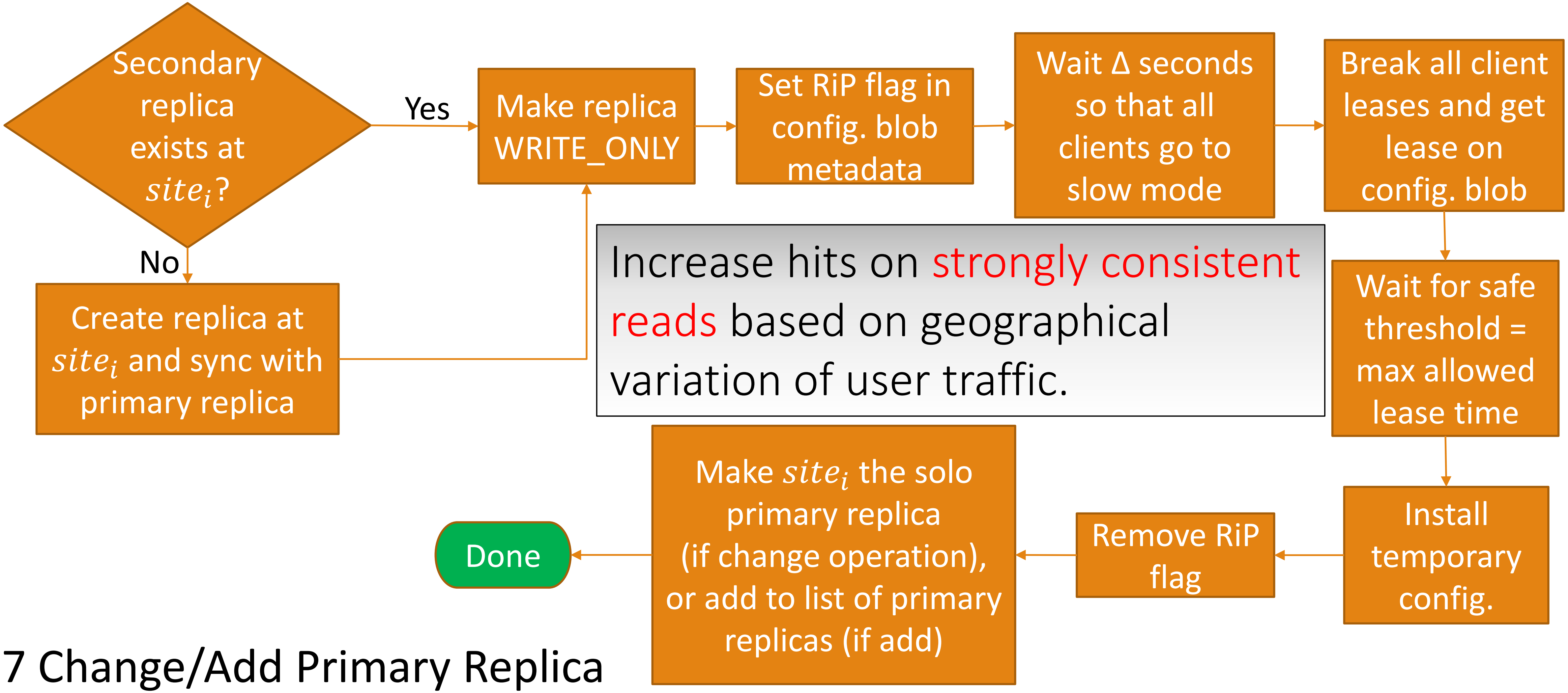
- Will provide better utility in case of this SLA

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

Rank	Consistency	Latency(ms)	Utility
1	RMW	40	1
2	Monotonic	90	0.6
3	Eventual	450	0.01

Table 2: SLA of an online multiplayer game

Change/Add Primary Replica *(change/add_primary($site_i$))*:



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

Rank	Consistency	Latency(ms)	Utility
1	Strong	100	1
2	RMW	100	0.7
3	Eventual	250	0.5

Table 3: SLA Used for Experimentation

	6h	4h	2h
AOU	0.76	0.81	0.85
AOU Improvement % over No reconfiguration	5	12	18
AOU Improvement % over Max Achievable AOU	20	45	65

Table 4: AOU Observations for Different Reconfiguration Periods

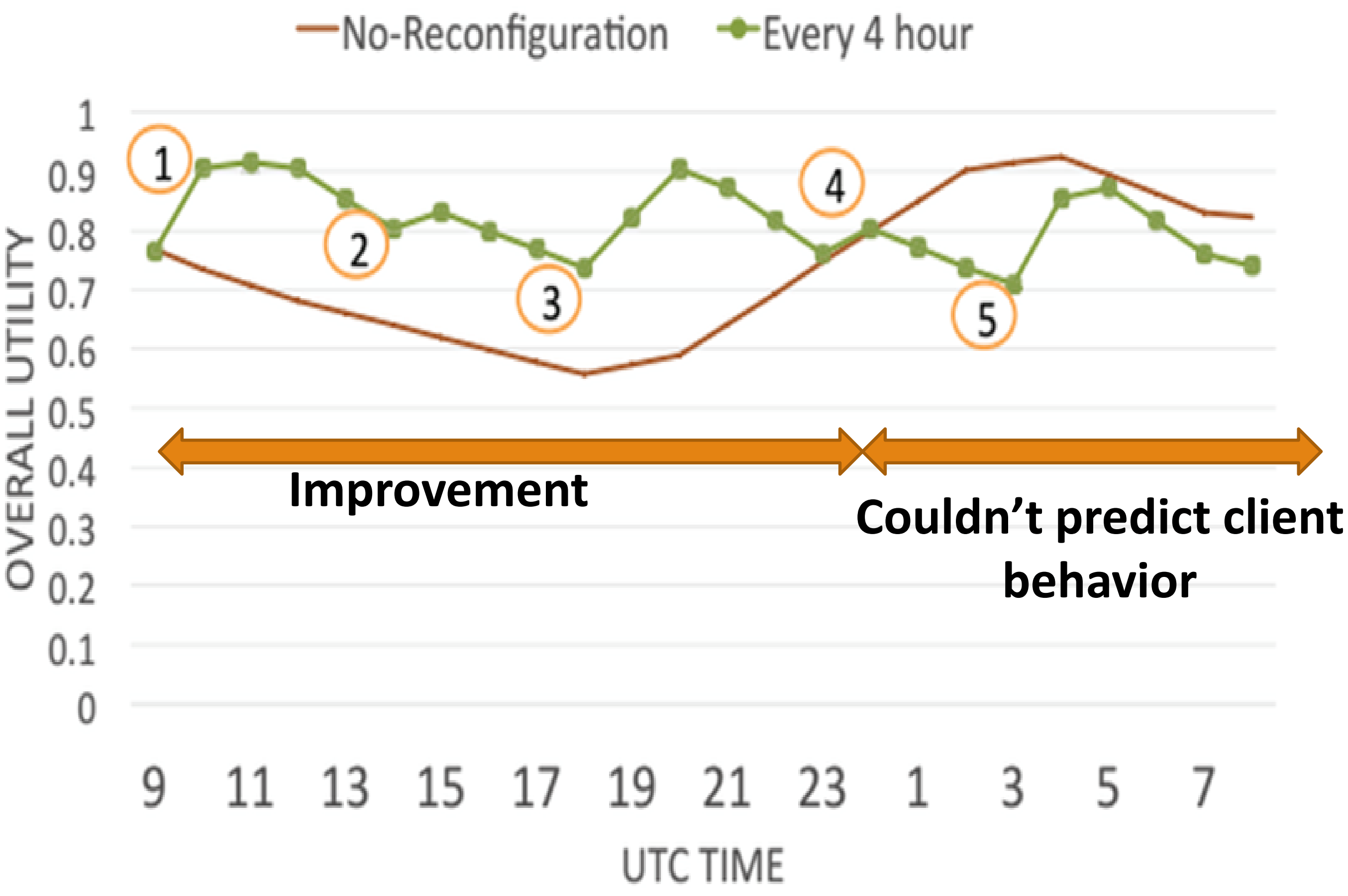


Fig.8 Tuba With a 4-Hour Reconfiguration Period

Action	Configuration Pri.	Sec.	CS Reconfiguration Operation
1	SEA	WEU	*change_primary(WEU)*
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	…	…	……………………………………

Table 5: Tuba Reconfigurations done

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

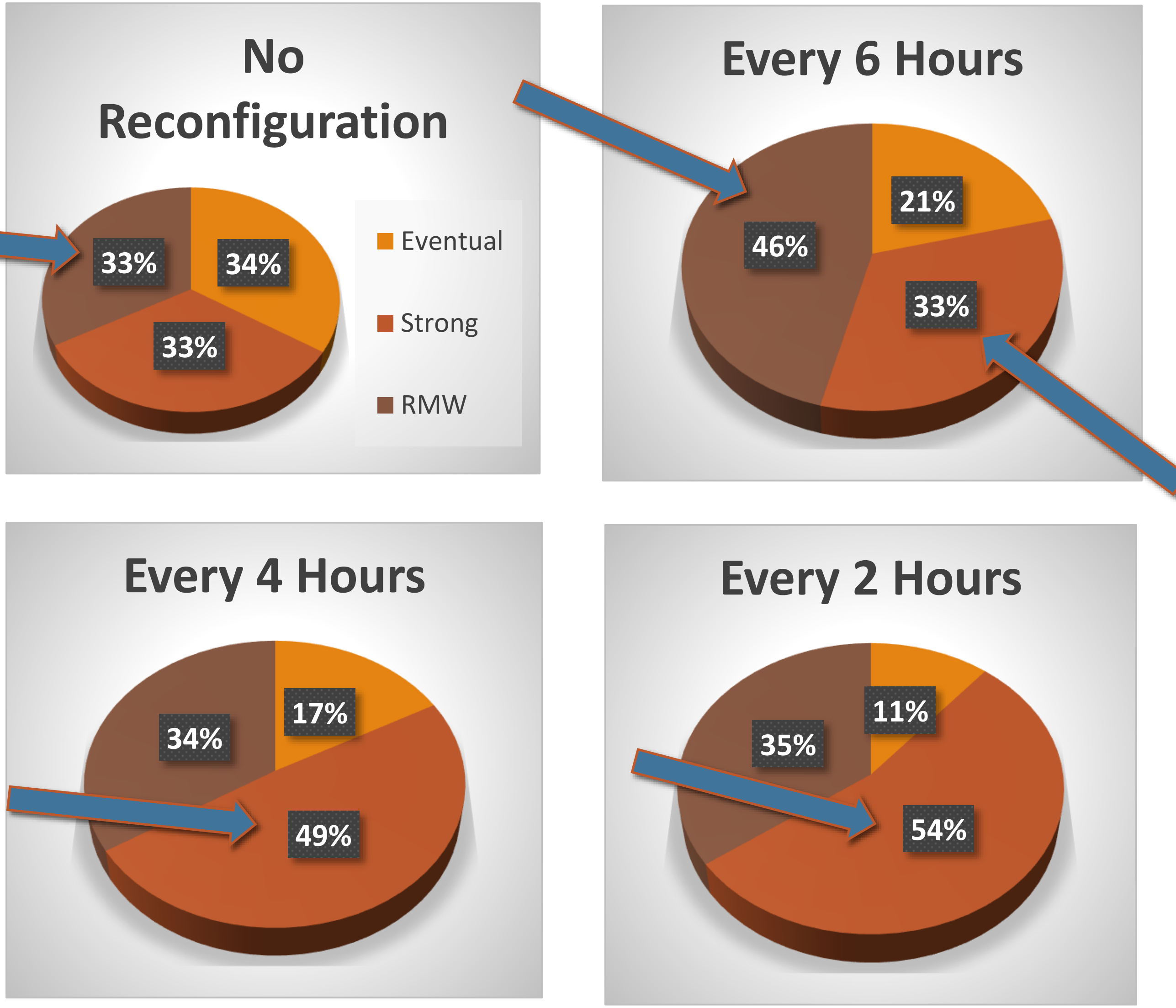


Fig.9 Hit Percentage of SubSLAs

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 → 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 → 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?