Beaver: Practical Partial Snapshots for Distributed Cloud Services

Liangcheng Yu, University of Pennsylvania; Xiao Zhang, Shanghai Jiao Tong University;

Haoran Zhang, University of Pennsylvania; John Sonchack, Princeton University;

Dan Ports, Microsoft / University of Washington; Vincent Liu, University of Pennsylvania

Presented By
Mobaswirul Islam
StudentID: 0424052053

Distributed Snapshots

Amazon:

- Inventory management
- Order processing
- Shipping coordination

States

•How many items are in stock

•which orders are being processed?

•What’s the shipping status for current orders.

Distributed Snapshots

Consistent global state of a distributed system

A global snapshot captures

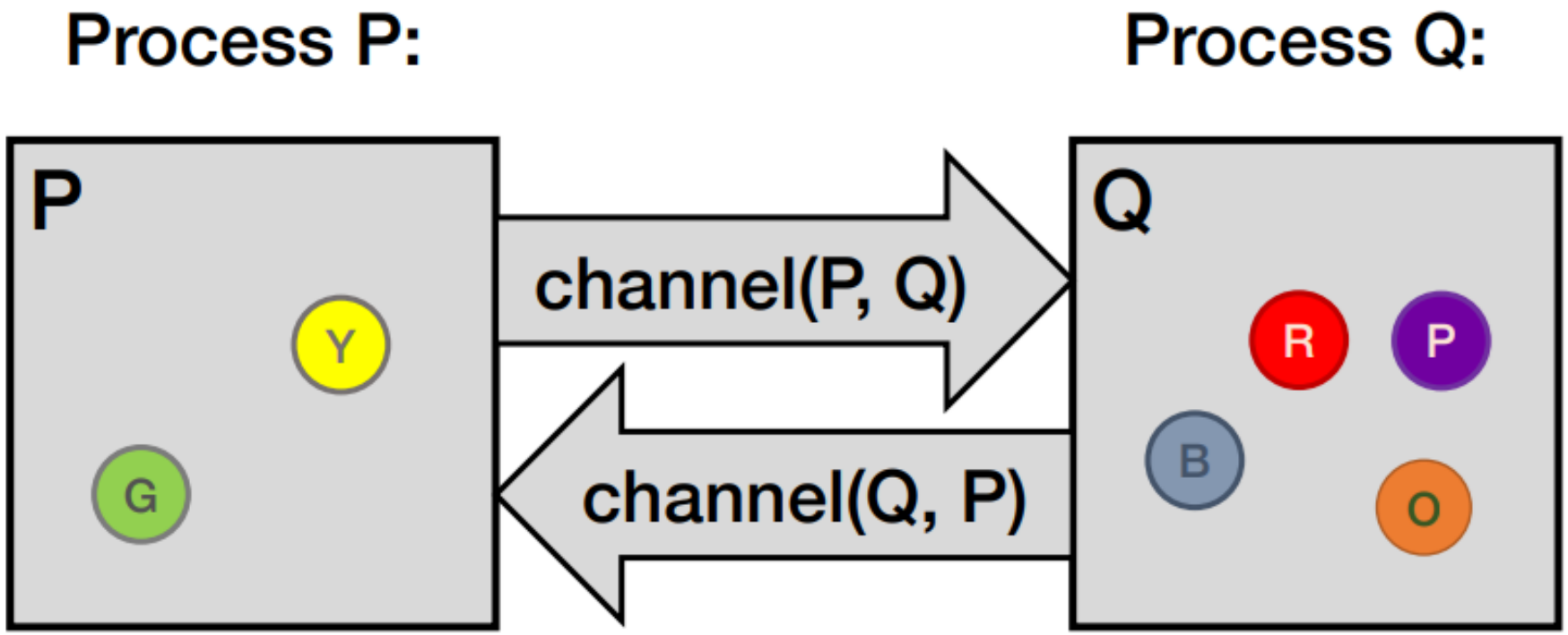
1. The local states of each process (e.g., program variables)
2. The state of each communication channel

Why Snapshots?

- Checkpointing
- Failure Recovery
- Detecting deadlocks
- Collecting garbage
- Other debugging

- System Clocks Skew
- Wouldn’t record messages between processes

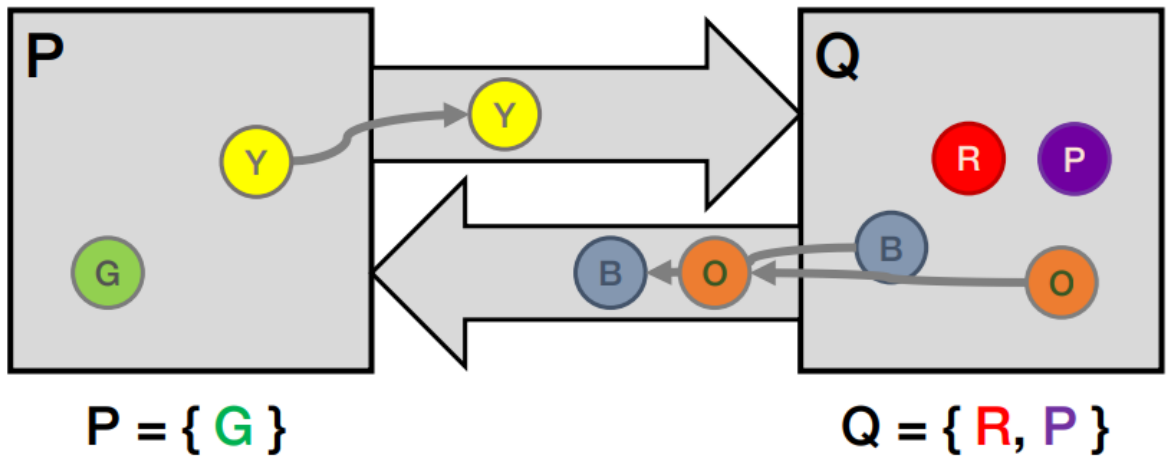
Distributed Snapshots



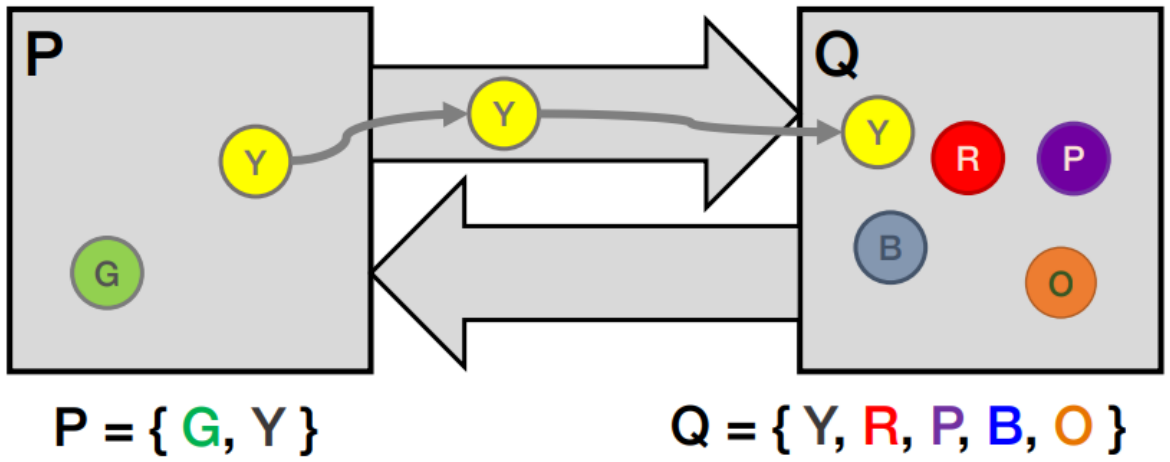
Correct global snapshot = Exactly one of each token

Distributed Snapshots

This snapshot misses Y, B, and O tokens



This snapshot duplicates the Y token



Challenges

- Handling external traffic
- Constantly changing states
- Third party Services
- Causality
- Interdependency

Challenges

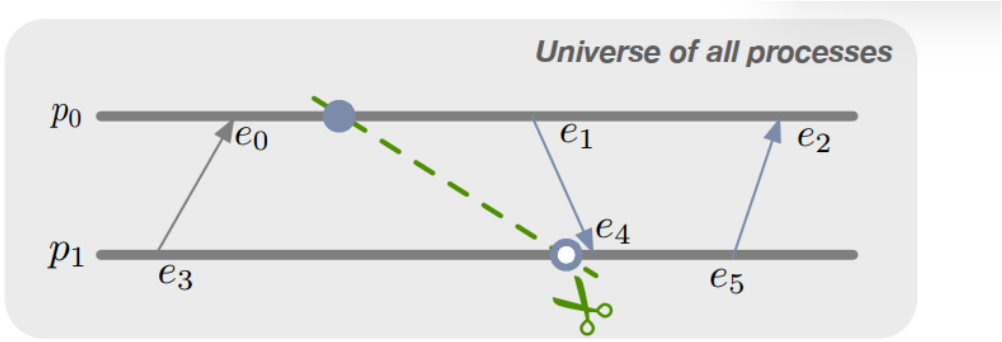


Figure 2: A minimal example of a consistent cut for 2 processes p_0, p_1 and 6 events $e_{0,1,\ldots,5}$. The global snapshot formed from the collection of ○ and ● is a 'causal cut' of the event timelines for all processes, where ● and ○ indicate snapshot initiations triggered out-of-band or by receiving marker messages, respectively.

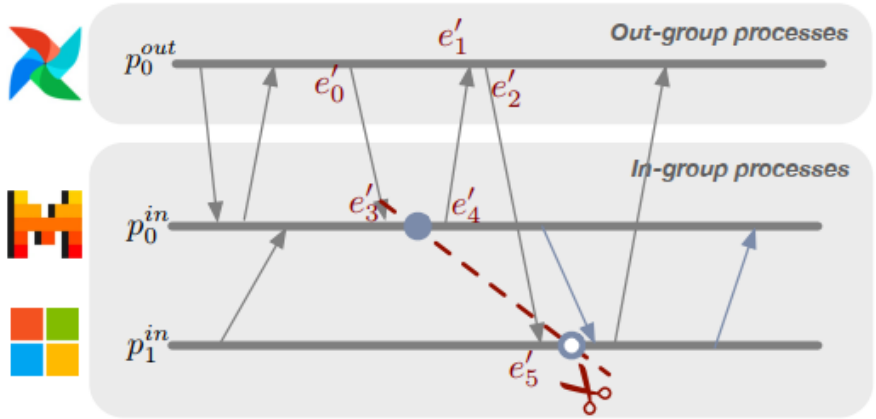


Figure 3: An application where a distributed serving system is accessed by an external user (e.g., an Apache Airflow workflow). The out-group process p_0^{out} imposes a hidden causal relationship $e'_4 \rightarrow e'_5$ between events e'_4 and e'_5, rendering a traditional snapshot of only the serving system inconsistent.

Causal Consistency

- Model to ensure the order of operations reflects their causal relationships.
- If one event influences another, all nodes in the system will agree on the order in which these events occurred.

Challenges (2)

- Don’t stop sending messages
- Don’t stop the application

Previous Solutions/ Algorithms

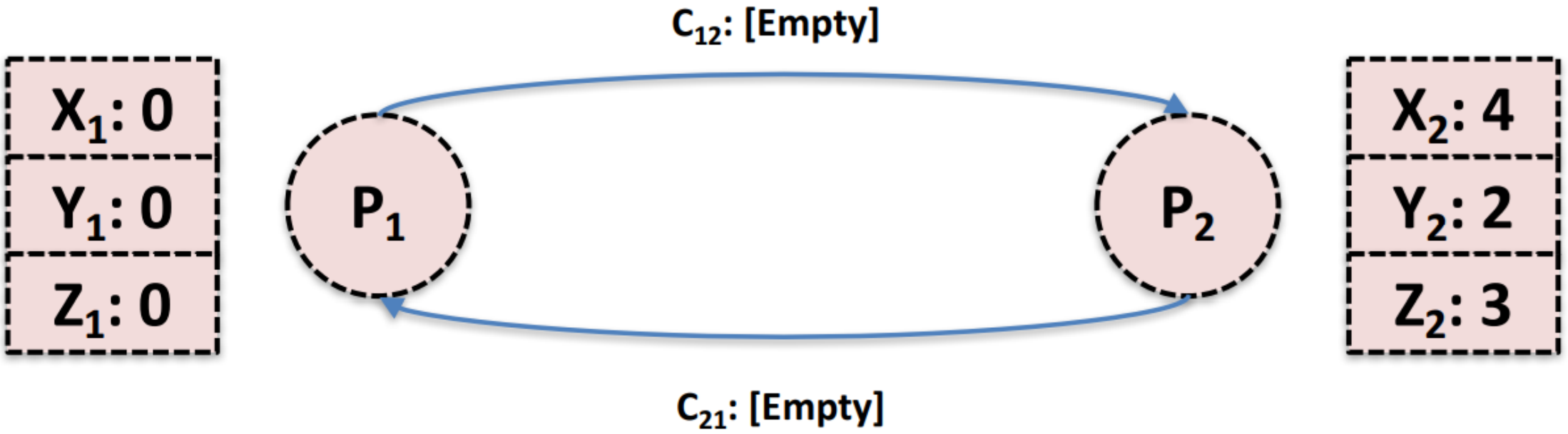
- Chandy-Lamport' Algorithm
- Lai-Yang algorithms

Chandy-Lamport' Algorithm

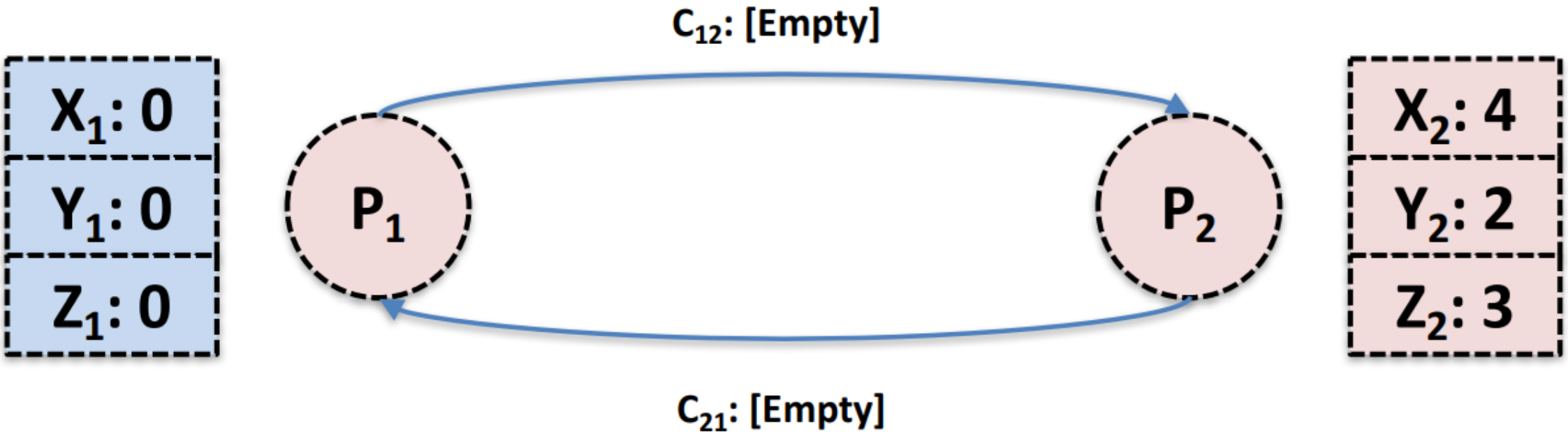
Model

- N processes (P_i … P_n)
- Two FIFO unidirectional channels for each process pair
  - ($P_i \rightarrow P_j$ and $P_j \rightarrow P_i$)
- All messages arrive, intact, not duplicated

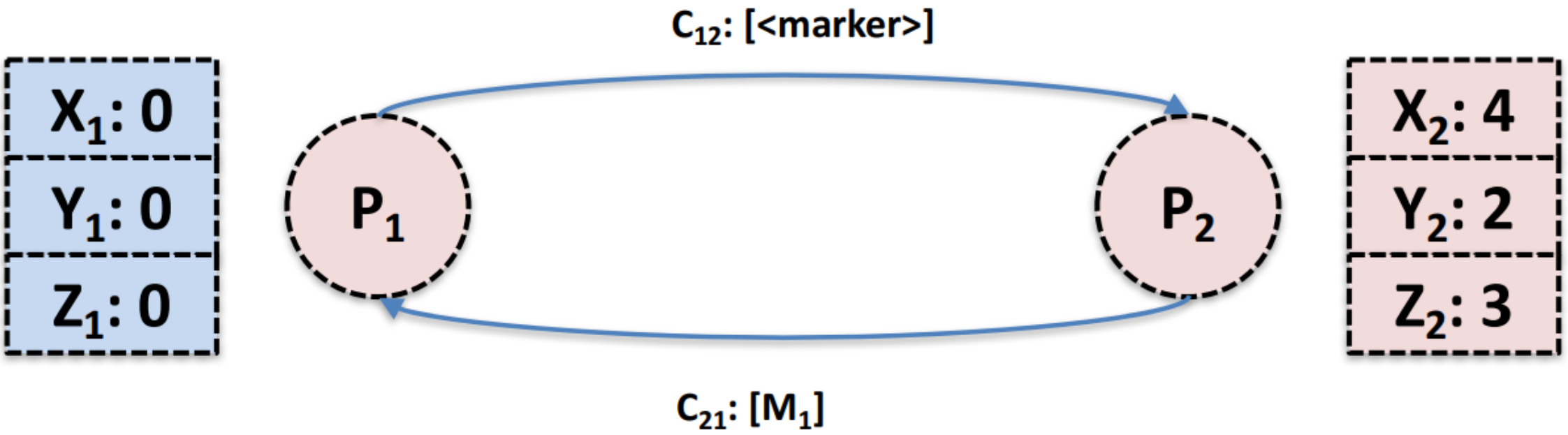
Chandy-Lamport' Algorithm - Simulation



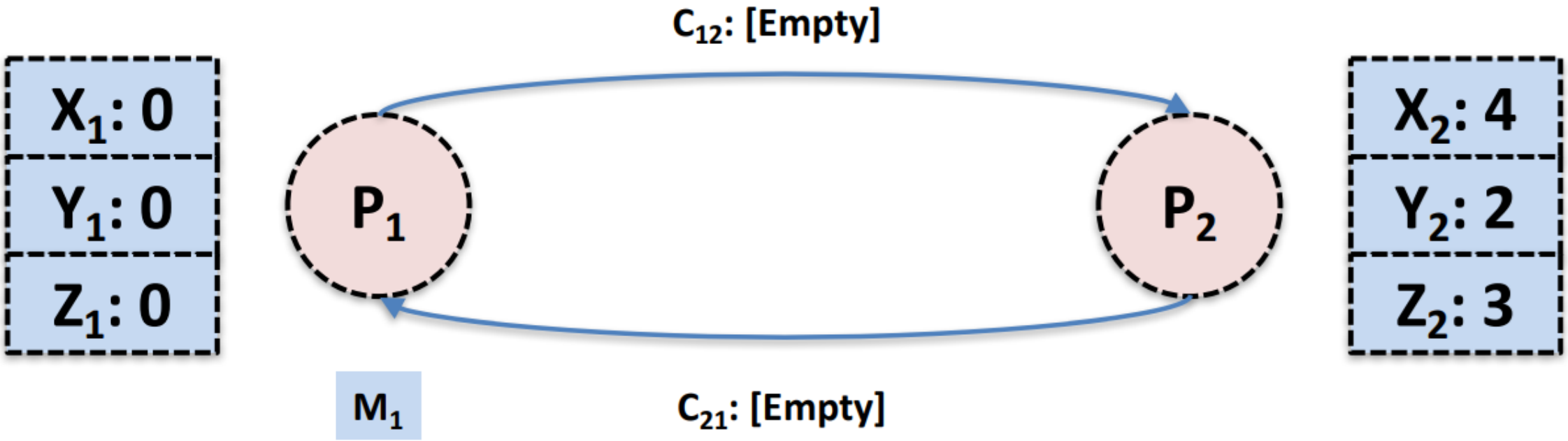
Chandy-Lamport' Algorithm - Simulation



Chandy-Lamport' Algorithm - Simulation



Chandy-Lamport' Algorithm - Simulation



Chandy-Lamport' Algorithm

- Simple
- Effective
- Widely Used

- Modular (Microservice)
- Independent
- Proprietary
- Works as black boxes
- No Control over clients or
- 3rd party services

- Closed System
- Have full control
- Ensures full participation
- No failures
- FIFO Queue

Lai-Yang algorithm (Variant of Chandi-Lamport's Algorithm)

- Non-FIFO
- Lossy Channels
- Piggyback a single marker bit

Why don’t they work?

INTERCONNECTED SERVICES

CLOUD CONSTRAINTS

PRACTICAL CHALLENGES

Why don’t they work?

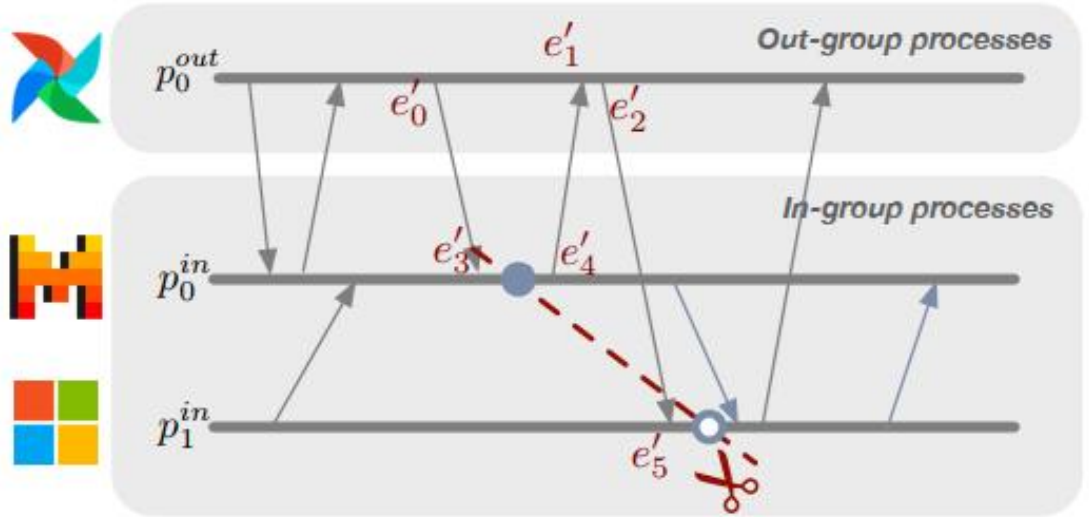


Figure 3: An application where a distributed serving system is accessed by an external user (e.g., an Apache Airflow workflow). The out-group process p_0^{out} imposes a hidden causal relationship $e'_4 \rightarrow e'_5$ between events e'_4 and e'_5, rendering a traditional snapshot of only the serving system inconsistent.

Possible (Obvious) Solution

Converting all cloud services into participants of the snapshot protocol might be possible

- Developer can develop and manage
- Support from the cloud provider
  - To propagate snapshot markers on all packets

- Low cost and agile
- Can lead to overhead

Possible Solution – Partial Snapshot

- Subset participation
- Causal Consistency
- Captures only the relevant interactions

Beaver - Methodology

Beaver

- Consistency
- Scalability
- Efficiency

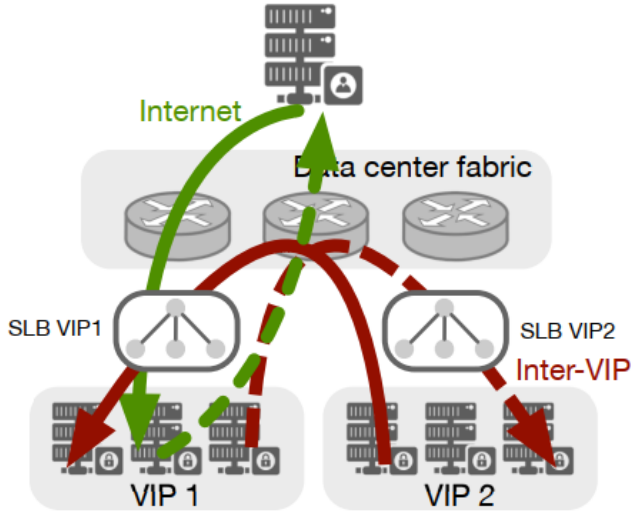


Figure 1: Today's public cloud services place SLBs to handle the external traffic to its VIP in the inbound direction (solid lines to VIP 1). The response to inbound messages (dotted lines from VIP 1) typically bypasses its SLB to minimize the SLB traffic load.

Strawman: Monolithic Gateway Marking

- For all incoming packets
  - Route them through a gateway
  - Tag with snapshot marker
  - Initiate snapshots by tagging all subsequent inbound messages accordingly

Beaver

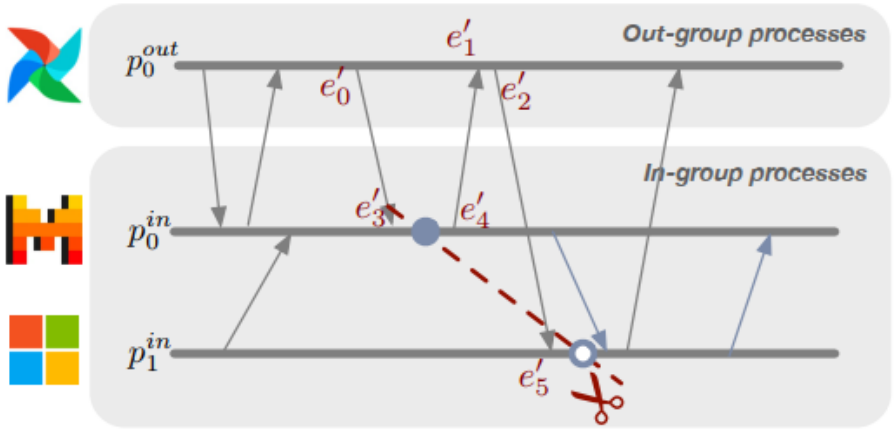


Figure 3: An application where a distributed serving system is accessed by an external user (e.g., an Apache Airflow workflow). The out-group process p_0^{out} imposes a hidden causal relationship $e'_4 \rightarrow e'_5$ between events e'_4 and e'_5, rendering a traditional snapshot of only the serving system inconsistent.

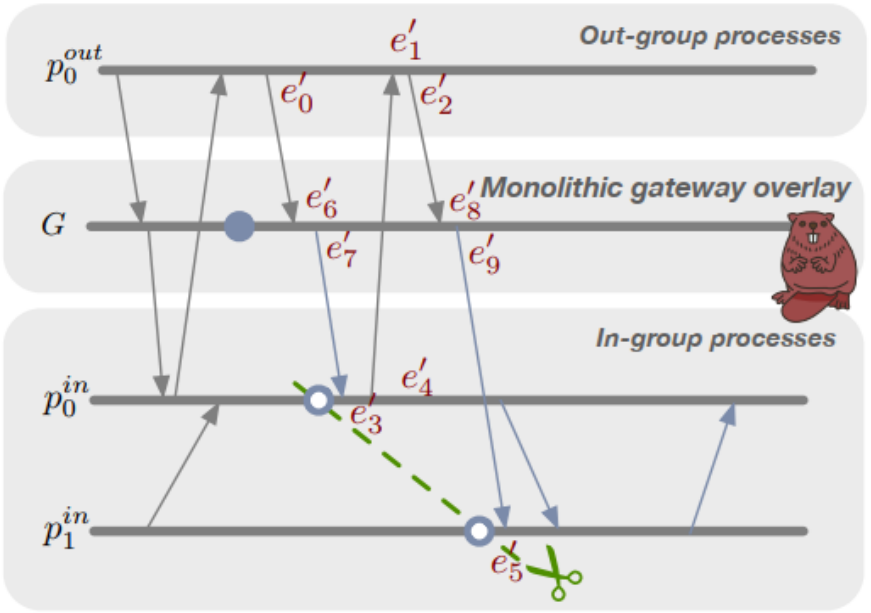


Figure 4: With the gateway indirection, Beaver's MGM results in a new frontier at the in-group process p_1^{in} that precedes rather than succeeds the event e'_5 (as in the scenario of Figure 3), converging to a consistent partial snapshot.

Beaver: SLB as Gateway Marking

- **Initiation**: An initiator process records its local state and sends marker messages to other in-group processes.
- **Marker Propagation**: Markers propagate through all channels.
- **Handling External Dependencies**: When in-group processes interact with out-group processes, the gateway overlay tracks these interactions to maintain causality.
- **Snapshot Completion**: The snapshot is completed once all in-group processes and their causally related out-group interactions have been recorded.

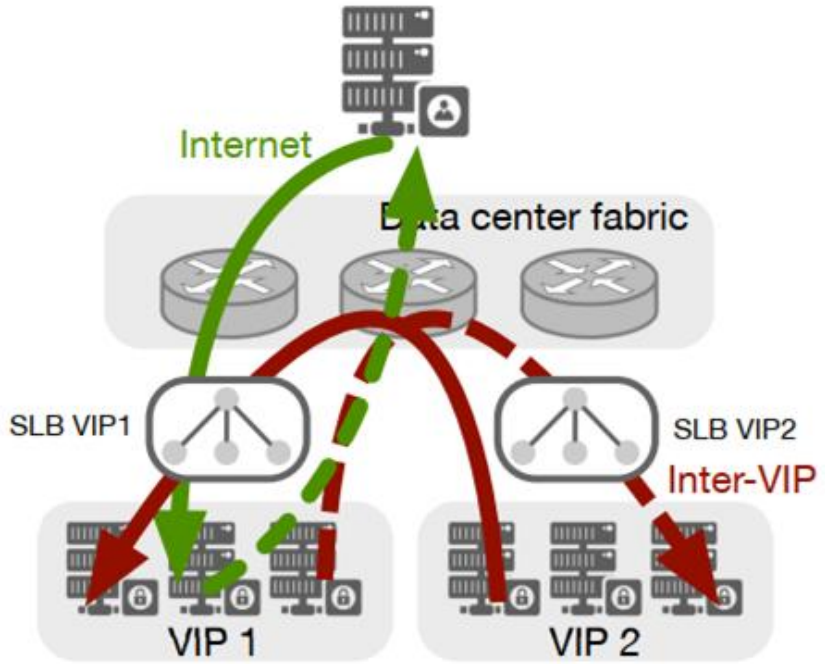


Figure 1: Today's public cloud services place SLBs to har
he external traffic to its VIP in the inbound direction (s
lines to VIP 1). The response to inbound messages (do
lines from VIP 1) typically bypasses its SLB to minimize
SLB traffic load.

Beaver: Optimistic Gateway Marking (OGM)

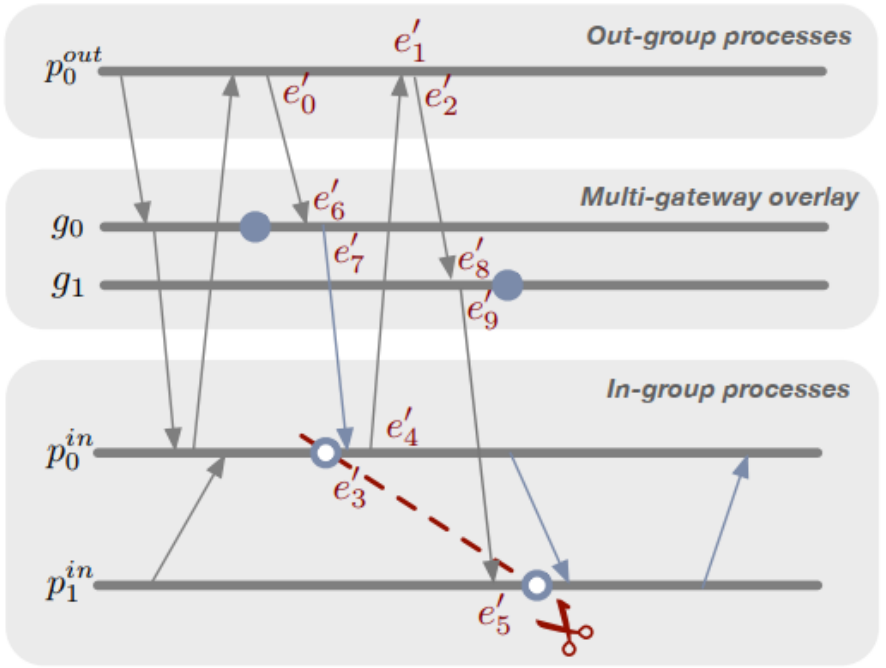
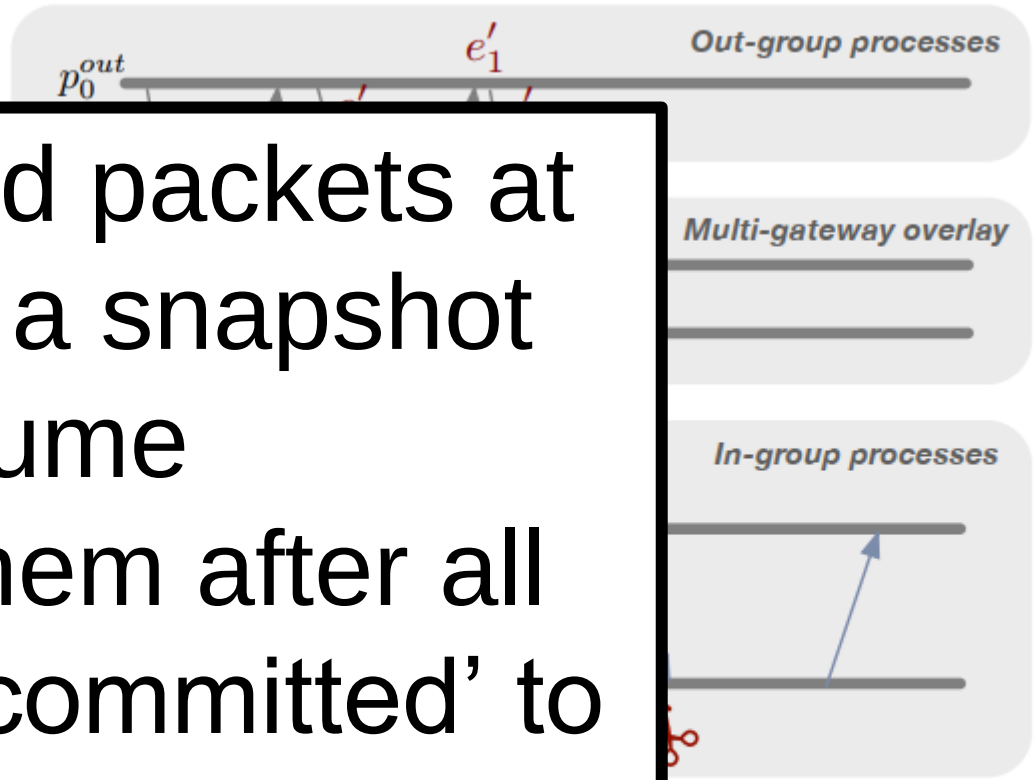


Figure 5: An inconsistent partial snapshot using two asynchronous SLBs g_0, g_1. When $e'_8.m$ arrives at g_1, g_1 has not initiated the new snapshot mode to mark the message, thus triggering the violation.

Beaver: Optimistic Gateway Marking (OGM)

block inbound packets at SLBs during a snapshot and only resume forwarding them after all SLBs have ‘committed’ to the new snapshot.



apshot using two asyn-
rrives at g_1, g_1 has not
hark the message, thus

Beaver: OGM (Solution)

- Detect Inconsistency
- Reject snapshots
- Minimize the rejection rate

Beaver: Causal Relevance and Irrelevance

Incoming message, m, is causally relevant only when

- An initiated SLB (g0) sends a marked message to an in-group node
- That node interacts directly or indirectly with an out-group node
- That out-group node sends m back to a different in-group node via an uninitiated SLB.

Causally relevant messages are rare in the real world

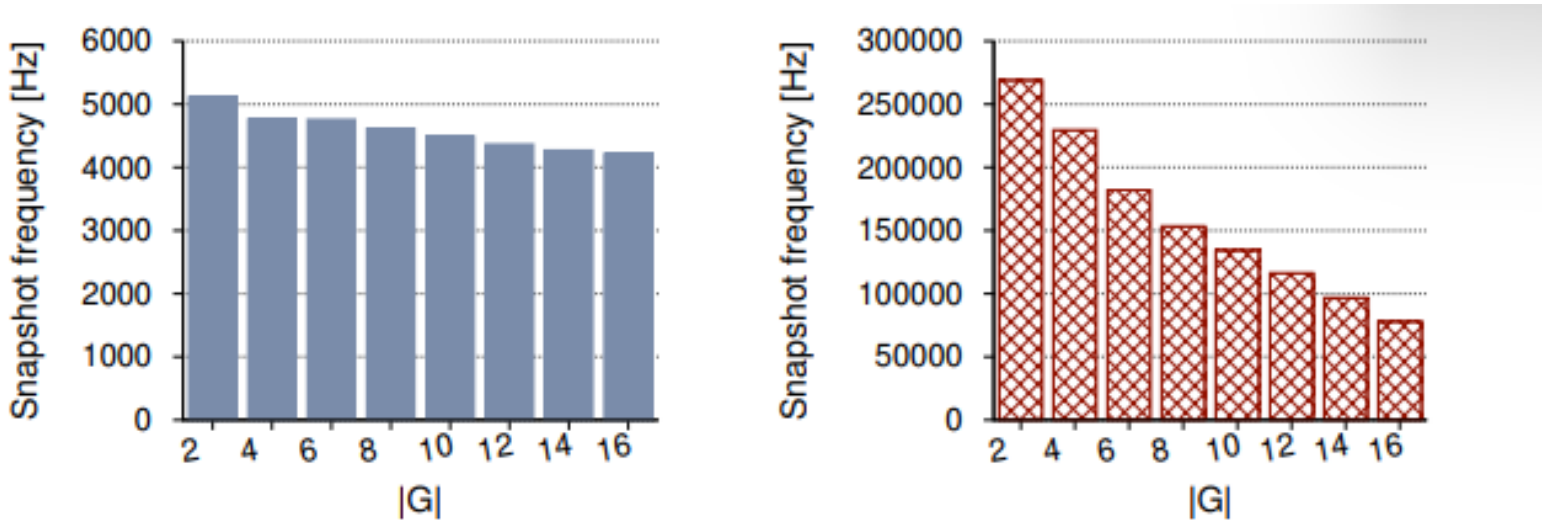


Figure 10: Beaver's sustained snapshot frequency versus a strawman approach with blocking operations at varying scales of SLBs and backend processes.

Evaluation

Supports Fast Snapshot Rates

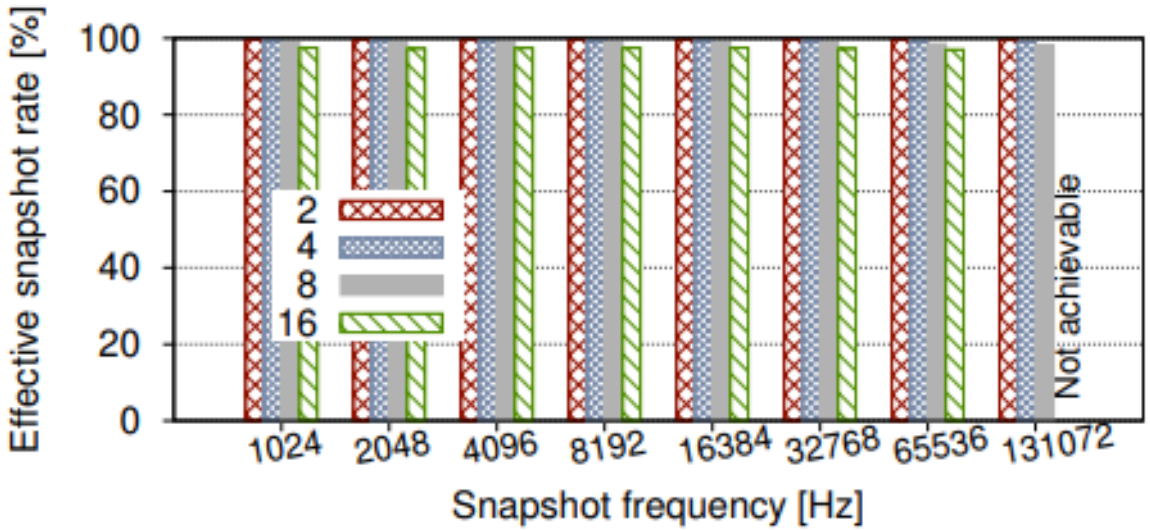


Figure 11: Beaver's effective snapshot rates under varying snapshot frequencies and in-group process scale.

Evaluation

Beaver Invalidates Snapshots Infrequently

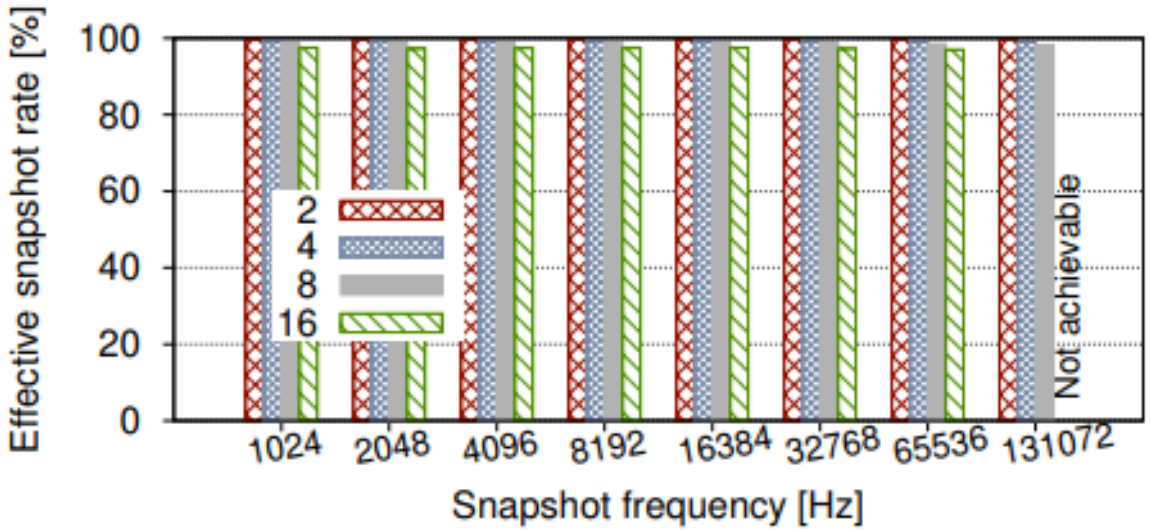
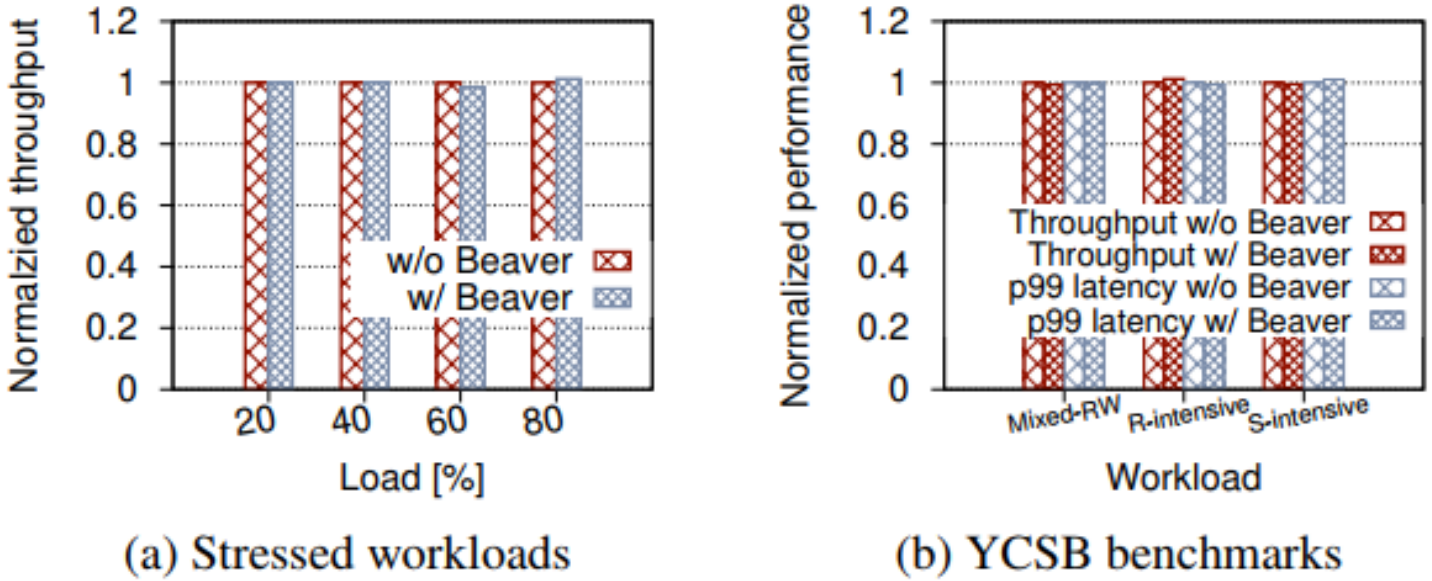


Figure 11: Beaver's effective snapshot rates under varying snapshot frequencies and in-group process scale.

Evaluation

Beaver Invalidates Snapshots Infrequently

Evaluation



(a) Stressed workloads (b) YCSB benchmarks

Figure 14: Performances with and without Beaver's overhead, normalized to the value without Beaver.

Beaver Incurs Near-zero Impact

Beaver: Use cases

DETECTING ANOMALOUS ACCESS

SERVERLESS GARBAGE COLLECTION

INTEGRATION TESTING

IN-FLIGHT MESSAGE TRACKING

DISTRIBUTED DEADLOCK DETECTION

Thank You