TOPSI: Totally-Ordered Prefix Parallel Snapshot Isolation

Nuno Faria & José Pereira
INESCTEC and University of Minho
Portugal

8th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC'21), April 26, 2021

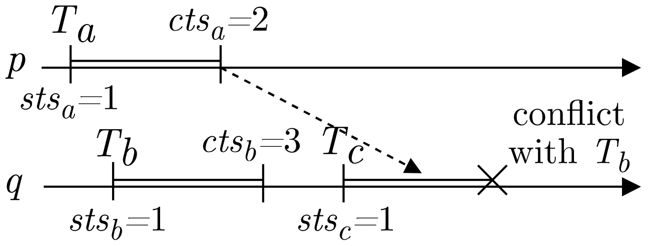
- Applications developers seek transactional and strong consistency guarantees to ease developments;

- However, those guarantees have a greater negative impact on the performance of distributed database systems:
  - Network latencies;
  - Distributed coordination;
  - Fault tolerance;
  - Metadata overhead.

- Relying on weaker consistencies (e.g., eventual) is not viable for a large range of use cases;

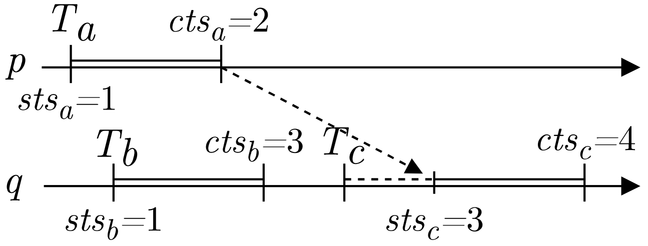
- Snapshot Isolation, widely used in centralized databases, proves to be a performance challenge in a distributed setting:
  - Globally consistent snapshots;
  - Monotonic snapshot evolution.

- ***Generalized Snapshot Isolation* (GSI)** allows transactions to execute over an older snapshot:
  - (+) Reduces blocking;
  - (-) Increases abort probability;
  - (-) A transaction might not be able to read its immediate writes.



T_a : $w(x)$
T_b : $w(y)$
T_c : $w(y), w(z)$

- ***Prefix-Constant Snapshot Isolation* (PCSI)** extends GSI to ensure that a snapshot contains all locally committed transactions:
  - (+) Reduces abort probability;
  - (+) A transaction can read its immediate writes;
  - (-) Increased response time due to blocking.



T_a : $w(x)$
T_b : $w(y)$
T_c : $w(y), w(z)$

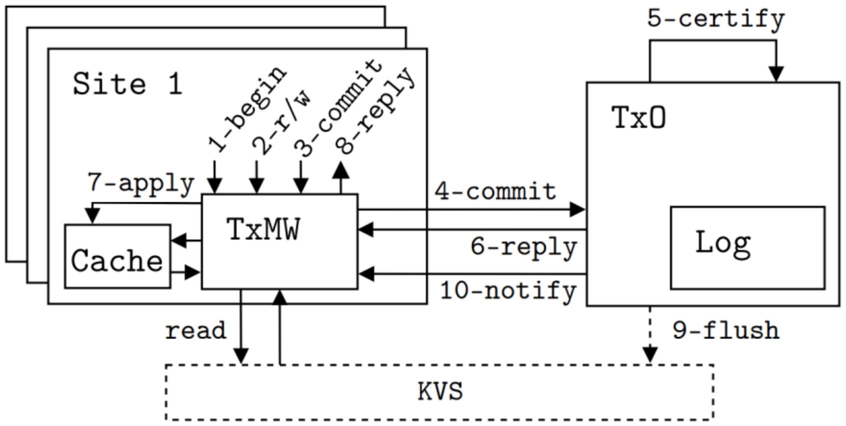
- ***Parallel Snapshot Isolation* (PSI)** allows different sites to apply transactions in different orders, as long as causal dependencies are respected:
  - (+) Reduces blocking;
  - (+) A transaction can read its immediate writes;
  - (+) Reduces abort probability;
  - (-) Independently evolving snapshots might not be viable for strict use cases

- Common implementations of PSI are built under the assumption of data partitioning, by sharding or by restricting an object's writes to a single site;

- Different sites end up with **different views of the transaction history**, meaning they cannot be applied to systems that disaggregate execution from storage;

- Different histories points to a more complex time management, e.g, vector clocks (more expensive snapshot materialization, conflict computation, site's connection and disconnection).

- Ensures PSI;
- Guarantees that the history at different sites **converge to the same totally ordered sequence of transactions**;
- No restrictions on data storage. It can be partitioned, completely replicated, or even shared among sites;
- Optimal timestamp management with low storage and computing overheads. No special coordination is necessary on site connection and disconnection.



- **Key-value store (KVS):**
  - Central component that stores all data;
  - Delivers data to the TxMWs based on the order it was flushed.

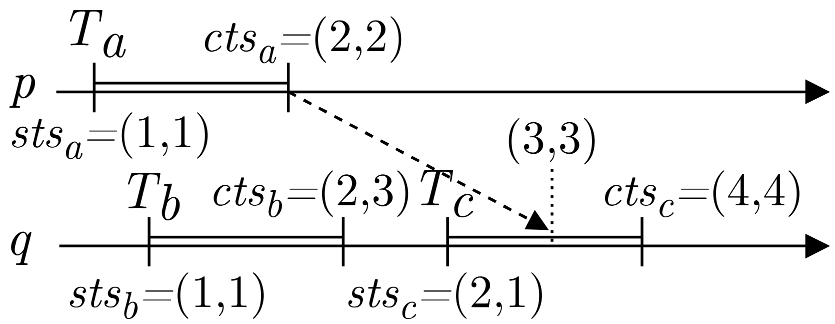
- **Transaction oracle (TxO):**
  - Central component;
  - Processes transactions' certifications and commits;
  - Flushes data based on commit order.

- **Transaction middleware (TxMW):**
  - Embed in every site;
  - Processes transactions' reads and writes; applies them to its local cache.

- ***Global_t:***
  - Monotonically assigned to a transaction when it commits, by the TxO;
  - Mainly used for transaction certification;
  - Unique across all sites.

- ***Local_t:***
  - Assigned to a transaction when it is applied, by the TxMW;
  - Mainly used for snapshot computation;
  - Unique in each site.

- $(local\_t, global\_t)$ pair assigned to each transaction/object.



T_a : $w(x)$
T_b : $w(y)$
T_c : $w(y), w(z)$

- Reading (k):
  - Look in the transaction's (T) cache for the record (r) with key k;
  - If not found, look in the site's cache for the most recent k where $r.local\_t \leq T.local\_t \wedge T.global\_t \leq r.global\_t$;
  - If not found, look in the KVS for k for the most recent record where $\mathrm{r}.global\_t \leq T.global\_t$.

- Writing (k, v):
  - Add k, v, and the $global\_t$ of the current record in the snapshot with key k to T's cache.

- Certification (T):
  - For each element e in T's write-set, evaluate if there is any record r with the same key where $r.global\_t > e.global\_t$;
  - If none is found, T can commit ($T.global\_t = next(global\_t)$).

- Apply (T):
  - For each element e in T's write-set, set $e.global\_t = T.global\_t$, $e.local\_t = next(local\_t)$, and add e to the TxMW cache;
  - Increment the TxMW's current $local\_t$.

PROBLEM

BACKGROUND

TOPSI OVERVIEW

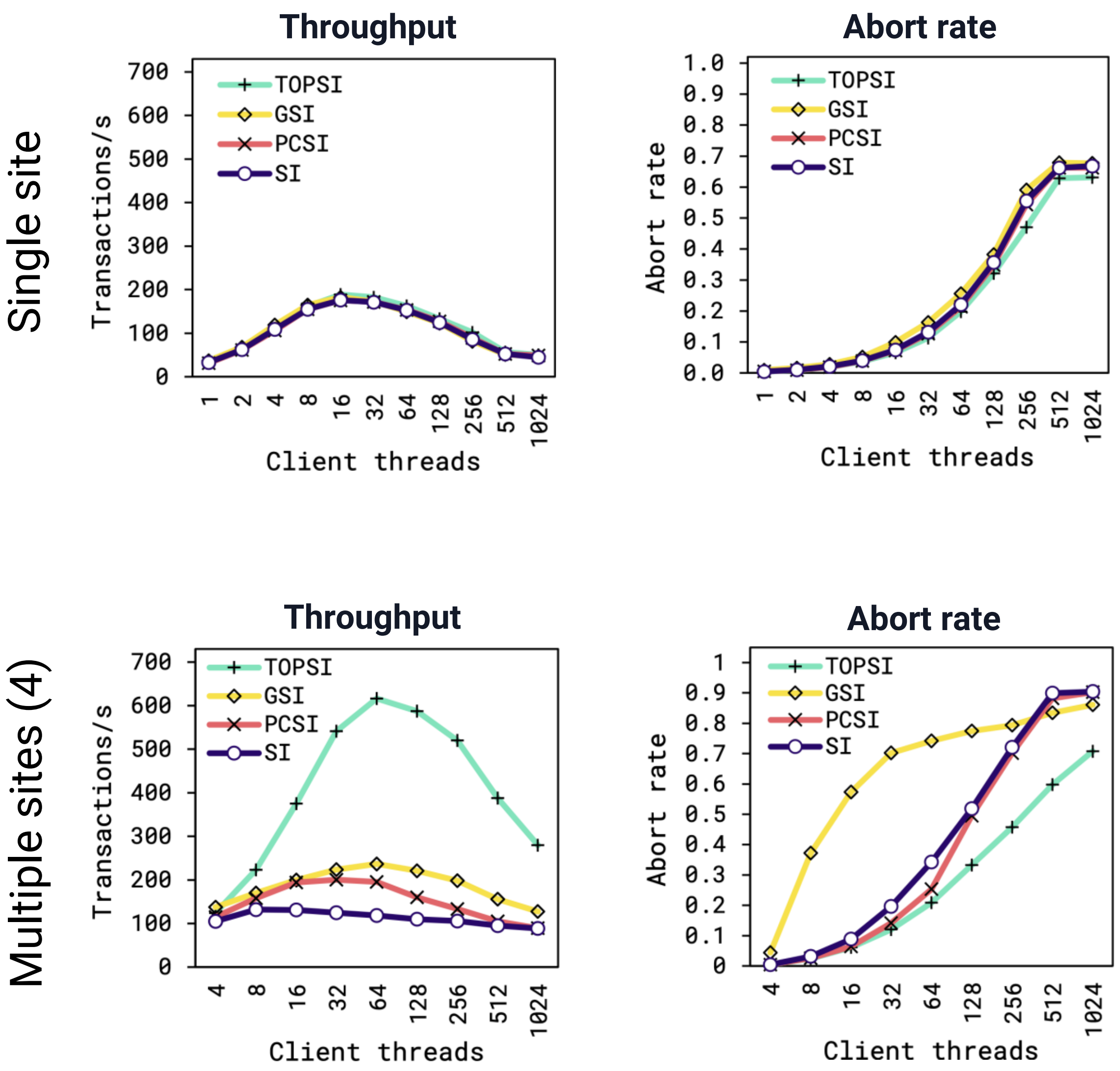
SYSTEM MODEL

TIME MANAGEMENT

TRANSACTION EXECUTION

PRELIMINARY RESULTS

CONCLUSIONS AND FUTURE WORK



- TOPSI presents better scalability than GSI, PCSI and SI;
- The two-timestamp solution ensures low storage and computation overheads;
- Having different sites share the same transaction history means that TOPSI is better suited for the class of systems that disaggregate computation from storage;
- For future work, we aim to create an architecture built with TOPSI that takes full advantage of parallelism to achieve optimal performance.

TOPSI: Totally-Ordered Prefix Parallel Snapshot Isolation

Nuno Faria & José Pereira

INESCTEC and University of Minho

Portugal

Partially funded by project AIDA -- Adaptive, Intelligent and Distributed Assurance Platform (POCI-01-0247-FEDER-045907) co-financed by the European Regional Development Fund (ERDF) through the Operacional Program for Competitiveness and Internationalisation (COMPETE 2020) and by the Portuguese Foundation for Science and Technology (FCT) under CMU Portugal.