Paper review:

High-performance Transaction Processing in SAP HANA

Erfan Zamanian

Feb. 2015

Background

- SAP HANA:
  - Main-memory database
  - Supports both analytical and transactional workloads
  - Allows for column and row store
  - Uses MVCC with distributed SI and locking scheme
  - The deadlock detection is handled centrally.

Main Contribution

- A number of optimizations for:
  - Distributed Snapshot Isolation (SI)
    - Exploiting locality of transactions
    - Local transactions pay for local coordination
    - Global transactions pay for global coordination
  - Two-phase commit protocol
    - Minimizing synchronized logging
    - Less communication cost

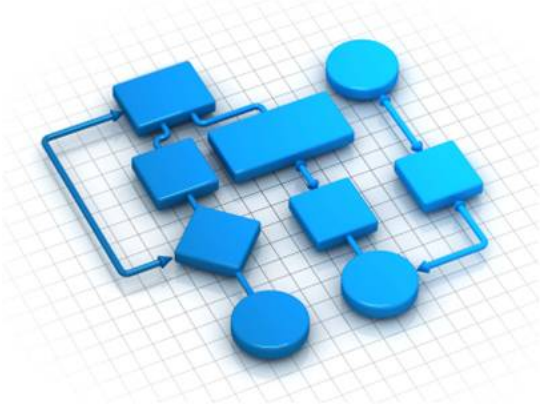
Category of the Paper

Improvement over existing work

Weaknesses

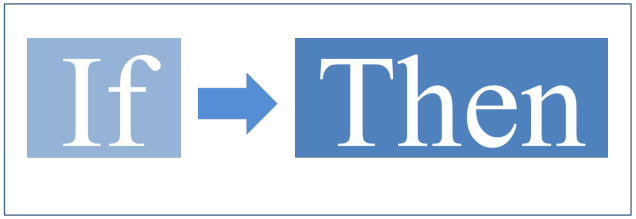
- Lacks scientific methodology
- Contributions do not support all claims
- Incompatibility of text and figure
- Leaves some questions unanswered

Methodology



- For the distributed snapshot isolation:
  - Introduced the notion of transaction token
  - A little bit verbose
- For 2PC optimization:
  - Ad hoc in nature (early commit ack, skipping writes, group 2PC)
  - More of an engineering effort than research
- No experiments, numbers, graphs, …
  - Particularly essential for this type of papers

Claims vs. Results



- The paper's main claim:
  - High throughput for OLTP while allowing OLAP
- But only discusses optimization for OLTP

- For example:
  - Impacts of the delta buffer on OLAP?
  - How frequently/when merge delta with with main store?
  - Index on these tables?

- Bottom line: this paper does not discuss OLAP

Incompatibility of text and figure

2.1 General Concurrency Control Mechanism

Isolation of concurrent transactions is enforced by a central transaction manager maintaining information about all write transactions and the *consistent view manager* deciding on visibility of records per table. A so-called *transaction token* is generated by the transaction manager for each transaction and encodes what transactions are open and committed at the point in time the transaction starts. The transaction token contains all information needed to construct the consistent view for a transaction or a statement. It is passed as an additional context information to all operations and engines that are involved in the execution of a statement.

For token generation, the transaction manager keeps track of the following information for all write transaction: (i) unique transaction IDs (TID), (ii) the state of each transaction, i.e., *open*, *aborted*, or *committed*, and (iii) once the transaction is committed, a commit ID (CID) (see Figure 1). While CIDs define the global commit order, TIDs identify a single transaction globally without any order. However, there are reasons to avoid storing the TID and CID per modified record. In the compressed column store, for example, the write sets of the transactions are directly stored in the modifiable delta buffer of the table. To achieve fine granular transaction control and high performance it becomes necessary to translate the information associated with the global CID into an representation that only uses TIDs, to avoid revisiting all modified records during the CID association at the end of the commit. The data consolidation is achieved using the transaction token. It contains the global reference to the last committed transaction at this point in time and the necessary information to translate this state into a set of predicates based on TIDs. The transaction token contains the following fields: `maxCID`, `minWriteTID`, `closedTIDs`, and the `TID`. Now, we will explain how the state information of the `maxCID` is translated into a set of `TID` filter predicates. The `minWriteTID` attribute of the token identifies *all* records that are visible to *all* transactions that are currently running hereby exploiting the assumption that the number of concurrently running transactions is limited and the order of the TIDs is similar to the order of the CIDs. This means that for all transactions T_j all records R with $TID_R < minWriteTID$ are visible. Since there can be committed transactions between `minWriteTID` and `TID` the attribute `closedTIDs`

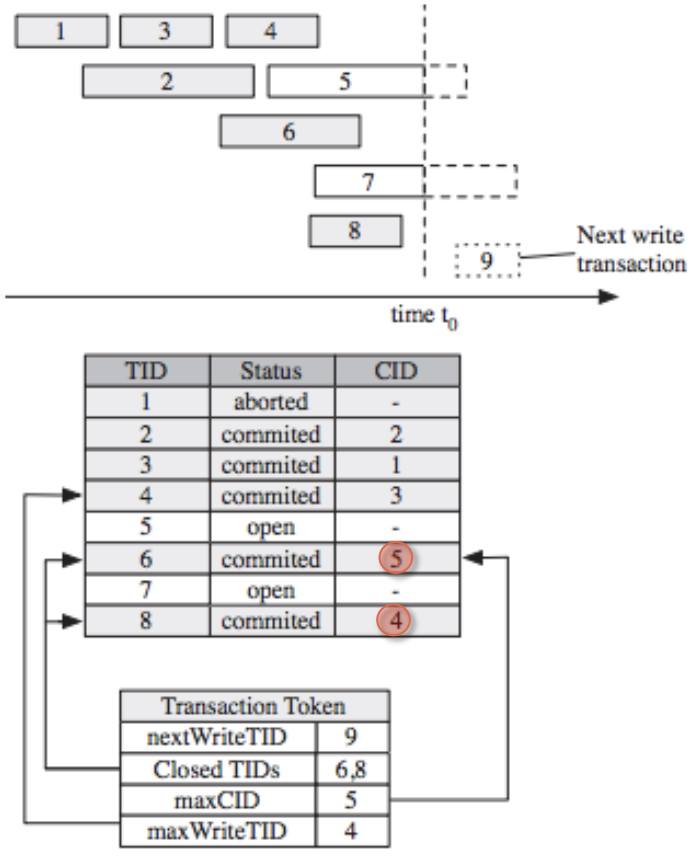


Figure 1: Data Consolidation with Transaction Token

Unanswered Questions

- In section 2.2
  - Each record has local_TID and global_TID
  - A local write transaction sees local_TID
  - A global write transaction sees global_TID and local_TID

Record_ID	Local_TID	Global_TID
1	5	6
2	7	8
3	9	2
4	5	6

- Algorithm correctness?

Unanswered Questions

- Local transactions always commit

- Paper's assumptions:
  - Short-running transactions are local-only
  - Long-running transactions are multi-node

- SI follows "first committer wins"

Unaddressed question:

How is starvation handled?

Suggestions

- In HANA:
  - Coordinator assigns a range of TIDs to each node
  - Eliminates the communication cost for TID request
  - Coordinator discards the unused TIDs periodically
- Suggestion: further reduce the communication cost by:
  - Assigning fixed TID numbering scheme (e.g. mod, hash)

Conclusion

- The paper presents some novel optimizations
- Not quite ready for publication in such a venue
- Benefits from
  - revising some sections
  - improving the flow of the paper
  - following a systematic approach
  - adding experiments

And now, let's welcome the authors