On Reading Fresher Snapshots in Parallel Snapshot Isolation

Masoomeh Javidi Kishi
CSE, Lehigh University
Bethlehem, PA, USA
maj717@lehigh.edu

Roberto Palmieri
CSE, Lehigh University
Bethlehem, PA, USA
palmieri@lehigh.edu

***Abstract*—In this paper we briefly present FPSI, a distributed transactional in-memory key-value store whose primary goal is to enable transactions to read more up-to-date (fresher) versions of shared objects than existing implementations of the well-known Parallel Snapshot Isolation (PSI) correctness level, in the absence of a synchronized clock service among nodes. FPSI builds upon Walter, an implementation of PSI well suited for social applications. The novel concurrency control at the core of FPSI allows its abort-free read-only transactions to access the latest version of objects upon their first contact to a node.**

***Keywords*-Transactions; Consistency; Snapshot Isolation;**

I. Introduction

Snapshot Isolation (SI) [1] is a widely adopted consistency level often used as a practical alternative to Serializability [2], the gold standard criterion for concurrency control implementations, when building transactional systems [3], [4]. A concurrency control that provides SI significantly improves the level of concurrency and performance over serializable concurrency controls because it allows for executions where multiple transactions access the same set of shared objects, as long as their sets of written objects are disjoint.

One of the great advantages of SI is that a transaction should not abort even though the set of values read (so called *reading snapshot*), and not written, during its execution has been overwritten by a concurrent transaction [1]. By leveraging this property, along with a multi-versioned data repository, most SI concurrency controls [4], [5] simply define the reading snapshot as all the versions available at the time a transaction starts.

When deployed on a distributed systems where nodes do not share a synchronized clock and the communication among them is asynchronous, SI transactions cannot simply define a similar (up-to-dated) reading snapshot at the time they start because of the absence of a shared notion of time among nodes. In fact, in such a distributed system it is very challenging to deterministically identify whether a version is created before or after a certain point in time [4], [6], [5].

Walter [4] is a state-of-the-art distributed transactional system whose concurrency control implements a variant of SI, called Parallel Snapshot Isolation (or PSI), in which the transaction reading snapshot can be arbitrarily outdated in order to deal with the aforementioned absence of shared clocks among nodes. Walter logically assigns objects to so called *preferred node* so that if a transaction reads a version from the preferred nodes, its reading snapshot will be guaranteed to be up-to-date. Any access to a non-preferred node can result in outdated versions.

Walter attempts to patch the above issue related to accessing non-preferred node using asynchronous messages, sent outside the transaction critical path, aimed at periodically updating the logical clock of non-preferred nodes in the system. Without these messages, two negative side effects can occur. First, transactions attempting to write objects from non-preferred nodes will be repeatedly aborted until the above asynchronous messages are delivered. Second, transactions reading from non-preferred nodes can return arbitrarily old versions.

In this paper we shortly introduce *Fresher Parallel Snapshot Isolation* (or FPSI), a distributed transactional system that uses logical (vector) clocks [7] to implement an improved version of the original Walter's concurrency control with the goal of improving data freshness. Enabling transactions to access up-to-date snapshots allows for alleviating the undesirable behaviors that *i)* lead read-only transactions (i.e., transactions that do not write) to read outdated values, as in the previous example, and *ii)* lead update transactions to be continuously aborted (degrading overall progress) by implementing a technique called *visible reads* [8].

II. The Overview of FPSI Concurrency Control

A. The Challenge of Data Freshness in Walter

The major algorithmic challenge in achieving the data freshness is to deal with the technique used by Walter to update nodes' logical (vector) clocks upon transaction commits. In fact, since Walter's transaction reading snapshot can be arbitrarily old, vector clocks are updated without propagating causal dependency with concurrent transactions (which would instead be needed to preserve safety of subsequent read operations of a transaction). Walter can ignore that since its transactions do not read from concurrent transactions.

B. The Solution of FPSI

FPSI enriches Walter's vector clock with additional metadata, called *version-access-sets*, aimed at capturing dependency relations with concurrent transactions so that the latest version of an accessed object can be safely read at its first access. This goal is achieved by combining the version-access-sets, which traces transactions' write-after-read (anti-)dependency, with Walter's existing vector clocks [7] to ensure transactions' consistency.

More specifically, the version-access-set is associated to a version, and it contains identifiers of read-only transactions that read that specific version. During the commit phase of an update transaction, the set of identifiers of concurrent conflicting read-only transactions is collected. This set is propagated to the version-access-sets of the newly created versions of those update transactions.

In the following we overview the two policies used by FPSI to handle read operations of update and read-only transactions.

1) Read operations by Read-only Transactions: FPSI allows read-only transactions to always access the latest version of a shared object upon their first contact to a node (whether preferred or not) by advancing their reading snapshot. In other words, the reading snapshot of a read-only transaction T is not defined when T starts. Instead, it is established during its execution by means of attempting to include the newest versions that still preserve PSI.

From the set of versions included in the advanced reading snapshot of T, those versions whose version-access-set include T's identifier, should be excluded because that means T has already established an anti-dependency with the transactions that committed those versions. Among the remaining versions, the newest (meaning the freshest among them) is selected as the result of the read operation.

2) Read operations by Update Transactions: With respect to the update transactions, FPSI reduces the abort rate by deploying a similar technique to the one used for read-only transactions. However, since update transactions can abort during their execution, tracing (anti-)dependency using version-access-sets would not be practical since aborted transactions would then need an extra clean up phase that would significantly degrade performance. FPSI approaches this issue by deciding to read from a safe snapshot (inspired by [9]), which might be older than the freshest in case a previous read operation accessed a version committed by a concurrent transaction with an established anti-dependency with the executing update transaction.

More formally, an update transaction is guaranteed to read the latest version of an accessed object upon its first read. After that, the logical clock associated with the node handling the first read operation is used to derive whether subsequent read operations can or cannot access the latest consistent versions.

Note that FPSI by improving the reading snapshots of update transactions causes the update transactions do not repeatedly fail their validation due to the outdated reading snapshot which is more likely possible in Walter than FPSI.

III. Conclusion

In this paper we briefly introduce FPSI, a novel distributed concurrency control that provides an implementation of the PSI correctness level in which we improve the level of freshness of both update and read-only transactions. With FPSI, it is possible to prevent transactions from reading arbitrarily old versions, a significant drawback of current state-of-the-art solutions.

Acknowledgment

This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-17-1-0367 and by the National Science Foundation under Grant No. CNS-1814974.

References

[1] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil, "A critique of ansi sql isolation levels," in *ACM SIGMOD Record*, vol. 24, no. 2. ACM, 1995, pp. 1–10.

[2] P. A. Bernstein and N. Goodman, "Concurrency control in distributed database systems," *ACM Computing Surveys*, vol. 13, no. 2, pp. 185–221, 1981.

[3] M. J. Kishi, S. Peluso, H. F. Korth, and R. Palmieri, "SSS: scalable key-value store with external consistent and abort-free read-only transactions," in *ICDCS*, 2019, pp. 589–600.

[4] Y. Sovran, R. Power, M. K. Aguilera, and J. Li, "Transactional storage for geo-replicated systems," in *SOSP*, 2011, pp. 385–400.

[5] J. Du, S. Elnikety, and W. Zwaenepoel, "Clock-si: Snapshot isolation for partitioned data stores using loosely synchronized clocks," in *SRDS*, 2013, pp. 173–184.

[6] M. S. Ardekani, P. Sutra, and M. Shapiro, "Non-monotonic snapshot isolation: Scalable and strong consistency for geo-replicated transactional systems," in *SRDS*, 2013, pp. 163–172.

[7] F. Mattern *et al.*, *Virtual time and global states of distributed systems*. Citeseer, 1988.

[8] S. Peluso, R. Palmieri, P. Romano, B. Ravindran, and F. Quaglia, "Disjoint-access parallelism: Impossibility, possibility, and cost of transactional memory implementations," in *PODC*, 2015, pp. 217–226.

[9] D. R. Ports and K. Grittner, "Serializable snapshot isolation in postgresql," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1850–1861, 2012.