PhysiCS-NMSI: efficient consistent snapshots for scalable snapshot isolation

Alejandro Tomsic, Tyler Crain, Marc Shapiro

PhysiCS-NMSI: efficient consistent snapshots for scalable snapshot isolation

Alejandro Z. Tomsic
Sorbonne-Universités
UPMC-LIP6 &
Inria Paris, France
alejandro.tomsic@lip6.fr

Tyler Crain
Sorbonne-Universités
UPMC-LIP6 &
Inria Paris, France
tyler.crain@lip6.fr

Marc Shapiro
Sorbonne-Universités
UPMC-LIP6 &
Inria Paris, France
marc.shapiro@acm.org

ABSTRACT

Non-Monotonic Snapshot Isolation (NMSI), a variant of the widely deployed Snapshot Isolation (SI), aims at improving scalability by relaxing snapshots. In contrast to SI, NMSI snapshots are causally consistent, which allows for more parallelism and a reduced abort rate.

This work documents the design of PhysiCS-NMSI, a transactional protocol implementing NMSI in a partitioned data store. It is the first protocol to rely on a single scalar taken from a physical clock for tracking causal dependencies and building causally consistent snapshots. Its commit protocol ensures atomicity and the absence of write-write conflicts.

We argue that PhysiCS-NMSI approach increases concurrency and reduces abort rate and metadata overhead as compared to state-of-art systems.

1. INTRODUCTION

Snapshot Isolation (SI) [6] is a popular isolation level provided by commercial databases such as Oracle and SQL Server. Under SI, when a transaction starts, it takes a snapshot, the view of the database that it can read from. When an update transaction commits, its effects are applied atomically, creating a new snapshot. SI forbids write-write conflicts; if two transactions attempt to update the same item(s) concurrently, one must abort. In contrast to serialisability, it ignores read-write conflicts.

SI requires *strictly consistent snapshots (SCS)*, which are statically defined at a transaction's starting point. SCSs disallow reading the effects of updates committed past that point, which increases the probability of write-write conflicts and, hence, of aborts [5]. Moreover, SCSs must include all updates committed up to that point, which may cause reads wait to include currently committing updates [8].

Non-Monotonic Snapshot Isolation (NMSI) is a variant of SI that still provides a strong model for programmers by preventing write conflicts. It allows more scalable implementations by weakening snapshots [4]. Under NMSI, a transaction reads from a *causally-consistent snapshot (CCS)* [12], which can include versions committed after its starting point. Specifically, a transaction can read any version that is *causally compatible* with its previously-read items. This flexibility provides more implementation freedom, as reduces the amount of blocking reads and aborts compared to SI. However, existing implementations of NMSI incur large metadata for tracking causal dependencies [4]. This translates into storage, communication and processing overhead proportional to the number of servers in the system.

In this paper, we present PhysiCS-NMSI, a transactional system ensuring NMSI over a partitioned data store. It is the first protocol to rely on loosely synchronised physical clocks for encoding causality metadata in a single scalar. It provides causally consistent snapshots where the snapshot can be moved forward throughout its duration, and a two-phase commit (2PC) protocol for ensuring that a transaction atomically commits at multiple partitions with no write-write conflicts.

The contributions of this paper are:

- A novel approach to implement NMSI based on loosely synchronised physical clocks, relying on single-scalar metadata, that tackles the scalability limitations of existing implementations.
- A comparison and discussion of the implications of the proposed approach when compared to state-of-art solutions.

The rest of the paper is organised as follows. Section 2 presents an overview and the transactional protocol of PhysiCS-NMSI. Section 3 discusses its potential advantages when compared to the state of the art. We conclude our work and point future work directions in section 4.

2. PhysiCS-NMSI

2.1 Overview

We consider a single site, partitioned data store. The system is comprised of a number of servers, each storing a discrete subset of data items, called a database partition. Servers are equipped with a physical clock, used to timestamp events. Clocks are loosely synchronized by a time synchronization protocol such as NTP [1]. The precision of such protocol does not affect our system's correctness. Large clock skew between servers might impact performance.

A data item is assigned to a partition based on the hash of its object identifier (*oid*). The data store keeps multiple versions of each object, identified by a scalar timestamp. A

PaPOC'16, April 18-21, 2016, London, United Kingdom

DOI: http://dx.doi.org/10.1145/2911151.2911166

version contains the value of the object and a timestamp that encodes its causal dependencies. Both the commit and dependency timestamps are assigned by the commit protocol.

PhysiCS-NMSI provides a transactional interface. A transaction consists of a sequence of reads and writes, surrounded by a begin and a commit marker.

2.2 Protocol

Each partition runs a *data node (DN)*, a sequential process responsible for handling operations issued to its stored objects. When a client starts a transaction, the server receiving the request creates a *transaction coordinator (TC)* process, that handles client operations and communicates with DNs. A TC executes client operations sequentially. We introduce the protocol in terms of steps executed by a TC and DN(s) to execute a transaction.

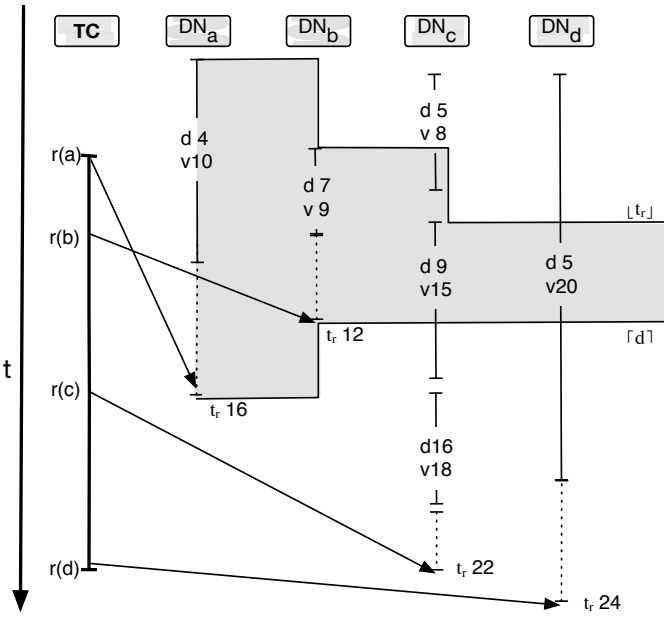
Reading from a CCS. requires all item versions a transaction reads to be causally compatible among each other. Namely, versions X_i and Y_j of items x and y are causally compatible if there does not exist X_k, another version of x, such that X_k is created after X_i, and Y_j causally depends on X_k [9, 12], where a causal dependency is defined by the happens before relation [2, 11].

A TC keeps two variables to determine the causally compatible versions that a transaction can read from:

- $\lceil d \rceil$: the upper bound of the dependency timestamp of a version a transaction can read. It is used to prevent reading a version that causally depends on an overwritten version of an object already in the snapshot. After the transaction's first read, $\lceil d \rceil$ is set to the read timestamp of the returned version. The read timestamp of a given version is the latest time at which a given version has not been overwritten. As new objects are read, the value of $\lceil d \rceil$ is updated to the minimum read timestamp observed.
- $\lfloor t_r \rfloor$: the lower bound of the read timestamp of a version a transaction can read. It is used to prevent reading a value that could be later overwritten with a version timestamp bigger than a previously seen dependency time, thus violating the causal snapshot. It is initialized to the dependency time of the first object read by the transaction. As new objects are read, the value of $\lfloor t_r \rfloor$ is updated to the maximum dependency timestamp observed.

When performing a read request, a TC sends to a DN the *oid* of the requested object along with the transaction's $\lceil d \rceil$ and $\lfloor t_r \rfloor$. A DN receiving such request uses this information to retrieve the most recent causally compatible version stored. In PhysiCS-NMSI, a given version is causally compatible if (*i*) it has *dependency timestamp not higher than* $\lceil d \rceil$ and (*ii*) *read timestamp no-smaller than* $\lfloor t_r \rfloor$. The DN replies to the TC including the version's value along with its version, dependency and read timestamps. A TC uses this information to update, if necessary, the transaction's $\lceil d \rceil$ and $\lfloor t_r \rfloor$.

Figure 1 shows a TC executing a transaction that reads four objects (identified by their *oid*s a, b, c and d), each stored at a different DN. We depict object versions as segments delimited by their version and dependency timestamps. The grey area shows how $\lceil d \rceil$ and $\lfloor t_r \rfloor$ evolve over



A TC builds a CCS by reading elements a, b, c and d from four different DNs. d, v and t_r represent an object's dependency, version and read timestamp, respectively.

Figure 1: Building a CCS in PhysiCS-NMSI

the lifetime of the transaction. Graphically, a version can be included in the snapshot when a segment taken from its dependency timestamp to the read timestamp (dotted lines) intersects with the grey area.

The transaction reads version 10 of object a, and initializes $\lceil d \rceil = 16$, a's read timestamp, and $\lfloor t_r \rfloor = 4$, the dependency timestamp of that version. After reading b, it further shrinks the area to $\lceil d \rceil = 12$ due to a clock skew between DN_a and DN_b, and $\lfloor t_r \rfloor = 7$. When reading c, multiple versions (8 and 15) are compatible with the snapshot. In this case, DN_c returns the newest of them, and the TC increases and $\lfloor t_r \rfloor = 9$. Even when the DN hosting C receives the request from the TC when its clock value is 22 (t_r), it can not return this read timestamp because there exists a newer version (18) valid at that point. The DN is allowed to return the maximum timestamp when the version returned was valid (in this case, 17). Finally, when reading d, version 20 can be safely included in the snapshot.

When a compatible version is being committed by the 2PC protocol, a DN can choose to wait for it to finish committing and include it the snapshot, or to return an older compatible version. The latter choice av*oid*s blocking, which might improve response time and parallelism. Nevertheless, if the pending commit succeeds and the transaction later updates the object, it will abort due to a conflict. Another special case presents when a DN receives a read request and its read time is smaller than $\lfloor t_r \rfloor$. Here, the server advances its local clock to $\lfloor t_r \rfloor$ and returns the latest available version. This av*oid*s blocking and reducing the transaction's grey area. If this situation arises simultaneously with a a version committing with timestamp smaller than the DN's new clock, the DN must wait until it commits and include

it in the snapshot.

As a transaction progresses, the grey area becomes smaller. This can continue until $\lceil d \rceil = \lfloor t_r \rfloor$. This case still allows for freshness (reading versions committed after the transaction's starting point), as long as new read values have a version timestamp bigger and dependency timestamp smaller than $\lceil d \rceil = \lfloor t_r \rfloor$. Even in this case, there is always a version compatible with the snapshot.

Updating an item. When a client issues a write operation to a given object, a TC buffers the update locally. If the transaction did not read the object up to that point, the coordinator issues a special read request to the appropriate DN, which returns its current version and dependency timestamp. The update is buffered along with the version timestamp the TC is aware of. This is used at commit time for certifying the absence of write-write conflicts.

Committing a transaction. PhysiCS-NMSI commits read-only transactions without further checks, even when they read from multiple partitions. When an update transaction starts to commit, if it updates data items stored at a single DN, the new version timestamp is assigned the value of the clock at its server, where it commits in a single round-trip to the TC after locally certifying the absence of write-write conflicts. The certification is successful at a DN if, for every updated object, no transaction has created a newer version than the version timestamp read by the transaction.

When a transaction updates multiple DNs, the TC coordinates a two-phase commit protocol (2PC) that decides on the outcome (commit or abort) of the transaction and defines the version (commit) timestamp of its updated objects. In both DN cases, the new dependency timestamp is set to the maximum version timestamp of the items accessed by the transaction.

2PC protocol. When a transaction updated multiple partitions, the TC sends a prepare message to each updated DN that includes its updates and their read versions. Upon receiving a prepare message, a DN certifies its updates do not write-conflict with other concurrently committing transactions. A successful DN replies to the TC with its current clock timestamp. A DN detecting a conflict sends an abort message to the TC, which aborts the transaction at other DNs. After receiving all prepare responses, the TC computes the transaction's commit time as the maximum timestamp received from a DN. It sends back a commit message to DNs including it. Upon receiving such message, a DN applies its updates along with their commit and dependency timestamps, making them visible to further transactions.

3. COMPARISON TO OTHER SYSTEMS

SCS and physical clocks. Existing systems that rely on loosely synchronised physical clocks to determine database snapshots implement the stronger SCS [8], even when they implement causal consistency [3, 7, 9]. Such systems have to wait in two cases: (i)a committing version that needs to be included in the snapshot, and (ii) clocks to catch up with the snapshot timestamp in the presence of clock skew between servers [8]. In contrast, PhysiCS-NMSI must only wait in case of a clock be behind by a significant amount, i.e., behind $\lfloor t_r \rfloor$ set by the dependency timestamp of an object previously read by the transaction, occurring simultaneously with a pending commit with version timestamp expected between its current clock and $\lfloor t_r \rfloor$. We believe this situation is rare in a partitioned data store expected to be deployed at a single DC, where the clock skew allowed by NTP is small. Indeed, a recent experiment at Facebook showed 99.9th percentile clock skew across all web servers in a data centre was 35 ms [13].

Clock-SI is a transactional protocol that implements SI relying on physical clocks to totally order snapshots [8]. Under Clock-SI, a snapshot is frozen at the transaction's start point. When committing an update transaction, the commit protocol certifies conflicts against that point. Under PhysiCS-NMSI, a transaction is allowed to read values committed after it started. The commit protocol checks each updated object for conflicts against its own read version. This may reduce its abort rate.

CCS in strong consistency. Strong consistency models that are compatible with CCS include NMSI, Parallel Snapshot Isolation (PSI) [15] and update serialisability (US) [10]. Existing implementations unnecessarily limit a snapshot's freshness; they freeze a snapshot globally at the beginning of a transaction [15], or at a partition when reading from it [4, 14]. This can cause reading old versions even when newer causally compatible ones are available. In contrast, PhysiCS-NMSI allows a snapshot to include the most recent causally compatible version of an object, which may improve freshness and reduce aborts under certain workloads.

Causal dependency metadata. CCS implementations often rely on vector clocks sized with the number of servers [4, 14, 15], or metadata that grows with the number of objects [12]. This incurs storage, communication, and processing overhead that increases as these systems scale. PhysiCS-NMSI reduces this overhead by compacting causal dependencies in a single scalar. The drawback of this approach is that it generates false dependencies, which may lead to read versions conservatively for not violating causality, and unnecessarily aborting transactions.

Multi-versioning. A CCS allows a transaction to choose between multiple versions of the same object. When compared to a SCS (where there is a single compatible version of any given object), this may (i) reduce the processing cost of retrieving a compatible version, as potentially fewer versions would need to be checked for compatibility, (ii) reduce the storage overhead of maintaining old versions, as a transaction can read versions committed past its starting point, and (iii) weaken the properties of the replication mechanism [16], as an outdated replica may still be able to provide causally compatible versions to a transaction. To the best of our knowledge, SCS and CCS have not been compared in terms of these three aspects.

4. CONCLUSIONS AND FUTURE WORK

We have presented the outline of PhysiCS-NMSI, a scalable approach to implementing strong consistency with causally consistent snapshots based on loosely synchronised physical clocks to encode causal dependencies in a single-scalar. PhysiCS-NMSI's design is aimed at addressing the scalability issues of state of the art implementations of SI and NMSI, and causally consistent snapshots in general.

Our next step is to support our assumptions through experimental results. Future research includes adapting our protocols to support geo-replication.

5. REFERENCES

[1] NTP: The network time protocol. https://www.ntp.org/, retrieved August 2015.

[2] M. Ahamad, J. E. Burns, P. W. Hutto, and G. Neiger. Causal memory. In *Proc. 5th Int. Workshop on Distributed Algorithms*, pages 9–30, Delphi, Greece, Oct. 1991.

[3] D. D. Akkoorath, A. Z. Tomsic, M. Bravo, Z. Li, T. Crain, A. Bieniusa, N. Preguiça, and M. Shapiro. Cure: Strong semantics meets high availability and low latency. In *IEEE 36th International Conference on Distributed Computing Systems*, ICDCS '16, Nara, Japan, June 2016. IEEE Computer Society.

[4] M. S. Ardekani, P. Sutra, and M. Shapiro. Non-monotonic snapshot isolation: Scalable and strong consistency for geo-replicated transactional systems. In *Proceedings of the 2013 IEEE 32Nd International Symposium on Reliable Distributed Systems*, SRDS '13, pages 163–172, 2013.

[5] M. S. Ardekani, P. Sutra, M. Shapiro, and N. Preguiça. On the scalability of snapshot isolation. In *Proceedings of the 19th International Conference on Parallel Processing*, Euro-Par'13, pages 369–381, 2013.

[6] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ANSI SQL isolation levels. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, SIGMOD '95, pages 1–10, 1995.

[7] J. Du, S. Elnikety, A. Roy, and W. Zwaenepoel. Orbe: Scalable causal consistency using dependency matrices and physical clocks. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 11:1–11:14, New York, NY, USA, 2013. ACM.

[8] J. Du, S. Elnikety, and W. Zwaenepoel. Clock-SI: Snapshot isolation for partitioned data stores using loosely synchronized clocks. In *Proceedings of the 2013 IEEE 32Nd International Symposium on Reliable Distributed Systems*, SRDS '13, pages 173–184, Washington, DC, USA, 2013. IEEE Computer Society.

[9] J. Du, C. Iorgulescu, A. Roy, and W. Zwaenepoel. GentleRain: Cheap and scalable causal consistency with physical clocks. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, pages 4:1–4:13, New York, NY, USA, 2014. ACM.

[10] R. C. Hansdah and L. M. Patnaik. Update serializability in locking. In *Proceedings on International Conference on Database Theory*, pages 171–185, New York, NY, USA, 1986. Springer-Verlag New York, Inc.

[11] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

[12] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Don't settle for eventual: Scalable causal consistency for wide-area storage with COPS. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 401–416, 2011.

[13] H. Lu, K. Veeraraghavan, P. Ajoux, J. Hunt, Y. J. Song, W. Tobagus, S. Kumar, and W. Lloyd. Existential consistency: Measuring and understanding consistency at facebook. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 295–310, New York, NY, USA, 2015. ACM.

[14] S. Peluso, P. Ruivo, P. Romano, F. Quaglia, and L. Rodrigues. When scalability meets consistency: Genuine multiversion update-serializable partial data replication. In *Proceedings of the 2012 IEEE 32Nd International Conference on Distributed Computing Systems*, ICDCS '12, pages 455–465, Washington, DC, USA, 2012. IEEE Computer Society.

[15] Y. Sovran, R. Power, M. K. Aguilera, and J. Li. Transactional storage for geo-replicated systems. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 385–400, 2011.

[16] I. Zhang, N. K. Sharma, A. Szekeres, A. Krishnamurthy, and D. R. K. Ports. Building consistent transactions with inconsistent replication. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 263–278, New York, NY, USA, 2015. ACM.