UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

A formal proof for Speculative Snapshot Isolation

Dissertation presented by
Sébastien D'OREYE DE LANTREMANGE

for obtaining the Master's degree in
Computer Science

Supervisor(s)
Peter VAN ROY

Reader(s)
Zhongmiao LI, Guillaume MAUDOUX , Xavier GILLARD

Academic year 2016-2017

Abstract

This work presents a formal proof for a speculative consistency model called Speculative Snapshot Isolation (SPSI). It states the SPSI properties in a formal way, proves SPSI correctness by showing that any speculative execution can be mapped to a correct, non-speculative one and shows that this mapping does not violate any SI property.

Contents

Chapter 1

Introduction

Snapshot Isolation (SI) is a (weak) consistency model widely used in commercial applications [1], that has been analyzed and adapted multiple times. The goal of this work is to prove the correctness of a new extension of SI called SPeculative Snapshot Isolation (SPSI). This extension is introduced by Zhongmiao Li, Paolo Romano and Peter Van Roy in their paper *Speculative Transaction Processing in Geo-Replicated Data Stores* while defining a protocol called STR [2]. They give 4 properties for SPSI, as well as 2 properties for SI, in an informal (textual) way. The first sections of this paper are dedicated to formalize those properties, and try to express them in a logical and mathematical way, and the further sections are establishing a formal proof that this extension is correct, based on the properties formalized earlier.

The proof presented in this work is semantic and not practical. The correctness computed is theoretical, and to work, an algorithm implementing SPSI must respect its semantics. We assume that a speculative system *will* eventually know whether or not a transaction will abort. This prediction seems hard to make instantly, but protocols like STR are given a *coordinator* protocol, that computes this information across the system. This computation is not involved in the proof formulated in this work.

STR is presented as "a geo-distributed, partially replicated transactional data store, which leverages on novel speculative techniques to mask the inter-replica synchronization latency" [2]. Georeplication allows the system to remain available even if it partially fails, and reduces access time for clients by reducing the distance between them and the datacenters. The downsides of this practice is the large amount of communication required between the datacenters to maintain interesting properties, such as ACID transactions (see chapter 2). Speculation is is a mechanism that has bee designed to improve performance by reducing the lock holding time for transactions. It defines speculative reads, that allow transactions to observe data produced by transactions that have not yet been certified across the whole system, and speculative commits, that allow transactions to expose their work to other transactions on the same node before global certification. Those speculative techniques are adapted by STR to make them fit for georeplicated systems, handling subtle anomalies and maintain robustness.

In addition to that, STR comes with a self-tuning mechanism [2] that allows it to adjust the speculation level to deal with high workloads. This allows STR to increase throughput by up to 6 times, and reduce latency by up to 100 times.

The proof made in this paper is based on two assumptions. The first assumption is that the algorithms defined for STR do not violate any of the SPSI properties [2]. This correctness is not proven in this work, but a sketchy proof of those properties has been given in [2]. The second assumption is that SI-compliant executions are correct [3]. The method that is used

to prove the correctness of SPSI is to show that speculative executions can be mapped to non-speculative executions, without violating any of the semantic properties of either protocol. This requires to define what an execution is, what a speculative execution is, and a strategy to get rid of the new elements introduced by speculations and the anomalies that can arise from it [2].

The proof in itself consists of two parts. The first part explains why a speculative execution can be mapped to a non-speculative one with the concept of termination. The second part of the proof shows that this mapping does not violate any property of SI executions, and thus that it is correct. It rebuilds those properties from the properties of speculative executions.

Chapter 2

Notations, definitions and other useful information

Many of the definitions and formal notations used in this paper are introduced by Andrea Cerone and Alexey Gotsman in their paper *Analyzing Snapshot Isolation* [3]. Here is a list of useful definitions as well as notations that are used in this paper.

1. Semantics

Semantics is the study of *meaning*. It is used in several domains of analysis, ranging from philosophy to formal logic. In the domain of programming languages, semantics is defined as the way to understand what the syntax says. The semantic analysis of a program is then focused on what happens inside the memory unit of the computer whilst executing that program, regardless of the language in which the program was written, or its performance.

2. Transaction

In distributed systems, transactions are used to carry operations on the database in a way that complies with the ACID properties. ACID is an acronym that states for Atomic, Consistent, Isolated and Durable. Atomic means that either all of the transaction's operations are carried out, or none are. No state in which part of a transaction's events have been executed is observable. Consistent means that a transaction's work has taken the system from a valid state to another valid state, in compliance with database constraints and rules. Transactions do not bring the system into error. Isolated transactions are transactions that carry out actions that bring the system into a state that could be observed if those transactions had been executed sequentially. Isolation is a difficult property to maintain, and is the reason why concurrency control mechanisms, like SPSI, are put in place. Finally, Durability means that a transaction's work should be persistent through system shutdowns, even if the shutdown is the result of a crash.

Formally, a *transaction* is a pair (E, po) where E is a nonempty set of *events* (which all include an *operation*, either a `read` or a `write`) and po is the *program order* which is total on the events in E. An event's operation can be known using the $op(e)$ function.

$\mathcal{T}$ is the finite set of committed transactions with disjoint sets of events. Those committed transactions are the ones that are semantically visible, as opposed to aborted transactions who completely disappear from the semantics of the execution.

3. Snapshot

The *snapshot* of a transaction can be seen as a picture of the database, taken by the trans-

action when it starts. This picture includes the work of all final-committed transactions as well as the work of local-committed transactions that occur on the same node as the starting transaction. In this work, the set of all transactions included in a transaction T's snapshot is denoted $VIS^{-1}(T)$, and the set of transactions included in a speculative transaction S's snapshot is denoted $VISe^{-1}(S)$.

In STR, when a transaction starts, it records the time at which it started in a *Read Snapshot* timestamp, denoted RS. This timestamp is equivalent to the physical time of the node from which the transaction originates.[2]

4. Snapshot Isolation

Snapshot Isolation is a transactional consistency model. In Snapshot Isolation, transactions execute on a local snapshot ("picture") of the database, that is guaranteed to contain the updates done by terminated previous transactions. If two transactions, anywhere in the system, have a part of their execution that occurs at the same time, those transactions are said to be concurrent. In opposition, serializability only allows sequential transactions.

A good example to illustrate how this works is to consider a popular bank, having agencies and ATM's all around the world. Every agency or ATM has its own computer, accessing the database. Imagine a customer having an account with a balance of 100 euros. He decides to withdraw this money to buy something. The ATM at which he stands then starts a transaction, asking for his current balance, substracting 100 from it and committing the result to the database. At the same time, at the other end of the world, an old aunt of his decides to give him 100 euros for his birthday. The agency at which she stands deposits the money on his account, starting another transaction, asking for his current balance - which has not yet been updated by his own withdrawal. If the two transactions serialize, one will have to commit before the other starts, and the value of the customer's account will be correct. In Snapshot Isolation, the two transactions are concurrent, and they both execute on a local snapshot. This means that the transactions do *not* take the other transaction's update into account. Of course, this is an anomaly and mechanisms can be set up to prevent it.

5. Commit

When a transaction has executed all its events, it must commit the work it has done to the system. To ensure atomicity, no effect can be committed until certified across the whole system.

In SI executions, a transaction must pass a *write conflict* (see further) detection test, and is then assigned a *commit timestamp*, which corresponds to the time from which the modifications done by the transactions are available *on all nodes*.

6. Speculative commit

In SPSI executions, committing comes in two stages. As described in STR [2], transactions first undergo a *local certification* phase, and are assigned a *local commit timestamp*, here denoted LC. From this point on, the transaction's modifications are available *on the local node*. This means that other transactions occurring on the same node can use those modifications for their own computations, despite those modifications not yet being certified across the whole system. This is why this protocol is called *speculative*. Finally, after being globally certified, transactions are attributed a *final commit timestamp*, denoted FC, and their modifications become available on all nodes.

7. Abort

In a transactional model, transactions that do not commit (and are not stuck in an infinite loop) are said to abort. Aborts are the result of a failure in the transaction's certification across the system : if the transaction violates a property of the system, it removes itself from the execution to prevent damage. Semantically, aborted transactions have strictly no effect : all the work they were trying to commit has been undone.

8. Write-write conflict

A *write-write conflict* is said to occur when two concurrent transactions are trying to update the same value, unaware of eachother's presence. The snapshot of one of those conflicting transactions will not contain the work of the other, resulting in the loss of information.

When a transaction starts, its snapshot contains a "picture" of the database. In this snapshot, all the transactions that have previously committed are *visible* to this transaction. Imagine a transaction T1 trying to update a value V. T1 takes its snapshot and starts working. *After* T1.RS, another transaction, T2, also takes a snapshot of the database and tries to update V too. T1 and T2 are *concurrent*, but unaware that their work might be overwritten by another transaction.

9. Commit order

CO and COe are *commit orders*. Commit orders are total.

CO orders *non speculative* transactions by commit time : T comes before S in CO if T commits before S. We write $T \xrightarrow{CO} S$.

COe has more cases to handle. COe orders *speculative* transactions by commit timestamps. Speculative transactions have more than one commit timestamp : their *local commit* timestamp and their *final commit* timestamp. It is worth noting that transactions that have not yet final-committed have no FC, and thus order might change once globlal certification completes. It is possible to observe transactions T and S with T.LC < S.FC, thus T comes before S in COe, but when T final commits, S.FC < T.FC. S now comes before T in COe.

Thus, COe orders transactions as follows :

If T local-commits before S local-commits, then $T \xrightarrow{COe} S$
If T final-commits before S local-commits, then $T \xrightarrow{COe} S$
If T local-commits before S final-commits, then $T \xrightarrow{COe} S$
If T final-commits before S final-commits, then $T \xrightarrow{COe} S$

10. Visibility

If two distinct transactions update the same data item, x, it is important that at least one of those transactions includes in its snapshot the work of the other. This necessity is captured by the visibility order, which we called VIS. In a snapshot, two write-conflicting transactions must be ordered, in one way or the other, by this order.

11. Execution

In Snapshot Isolation (SI), an *execution* is a tuple $\mathcal{X} = (\mathcal{T}, SO, VIS, CO)$, where $(\mathcal{T}, SO)$ is the *history* of the transactions of a session and their order SO. VIS and CO are the *visibility* and *commit order*. CO is a total order, and we have the property $VIS \subseteq CO$. In

[3], SO is defined as the *session order*. Sessions allow *transaction chopping*, on which we do not focus. We leave sessions on the side for this work, as they do not affect correctness, leaving us with the definition $\mathcal{X} = (\mathcal{T}, VIS, CO)$. When considering *speculative* executions, this definition changes, mainly because transactions have more possible states than in a standard SI execution.

12. Speculative execution

An execution in SI has a *transaction set* which includes all committed transactions. In an execution allowing speculation, new kinds of transactions appear. We distinguish the *locally committed* transactions from the *finally committed* transactions, and in the *locally committed* transactions set we distinguish the transactions that will eventually *final commit* from the transactions that will abort. The system does not know the difference *a priori*, and has to perform *certification* algorithms [2] to decide whether to final-commit or abort a transaction. In this work, we focus our approach on a semantic basis, meaning that we have a "God's eye" view of those transactions, knowing which transactions will commit or abort.

We define a *speculative execution* as follows :

$\mathcal{X}_s = (\mathcal{T}_s = (\mathcal{T}_{fc} \cup \mathcal{T}_{lc \to f} \cup \mathcal{T}_{lc \to a}), VISe, COe)$

$\mathcal{T}_s$ is the set of transactions in a speculative execution. In this set, we distinguish three subsets of transactions with different properties : $\mathcal{T}_{fc}$ corresponds to the transactions that have already *final committed*, $\mathcal{T}_{lc \to f}$ corresponds to the set of transactions that have local committed and will final commit, and $\mathcal{T}_{lc \to a}$ corresponds to the set of transaction that have local committed but will abort during global certification.

13. $WriteTx_x$ and $ReadTx_x$ sets

$WriteTx_x$ and $ReadTx_x$ are the sets of transactions that contain respectively at least one `write` or `read` event involving the item x. They are not to be mixed with the writeset or readset of a transaction, which are the sets of all the values written to or read by the transaction.

14. Order notations

Any order relation between two transactions T and S can be written in two equivalent ways : $T \xrightarrow{VIS} S$ is equilvalent to $(T, S) \in VIS$.

15. Termination

The execution of a distributed system can be seen as an endless sequence of states, altered by *events*. When there are no more computations to perform, the last state, called *final*, is repeated indefinitely and the execution is considered complete.

Chapter 3

Properties of SI

Non-speculative Snapshot Isolation executions obey to two properties called SI-1 and SI-2 [2]. The SI-1 and SI-2 properties can be defined using axioms from [3].

3.1 SI-1

"*All operations read the most recent committed version as of when the transaction began.*"

This property has two implications : on one hand, it ensures that, within a transaction $T = (E, po)$, a read event $e \in T.E$ on a data item x will return the same value as the last read or write event on x preceding e within $T.E$. This means that no event not belonging to $T.E$ can interfere with x. This property is called the *internal consistency* axiom [3] :

$\forall (E, po) \in \mathcal{T} \cdot \forall e \in E \cdot \forall (x, n) \cdot op(e) = \texttt{read}(x, n) \wedge \{f \mid op(f) = \_(x, \_) \wedge f \xrightarrow{po} e\} \Longrightarrow op(max_{po}\{f \mid op(f) = \_(x, \_) \wedge f \xrightarrow{po} e\}) = \_(x, n)$

On the other hand, SI-1 ensures that if the read event $e \in T.E$ is the first event of the transaction according to $T.po$, the value read by e is the value set or read by the last committed transaction according to CO. This means that the value read is determined by transactions that are *included* in T's snapshot. This property is called the *external consistency* axiom [3]:

$\forall T \in \mathcal{T} \cdot \forall x, n \cdot T \vdash \texttt{read}(x, n) \Longrightarrow max_{CO}(VIS^{-1}(T) \cap WriteTx_x) \vdash \texttt{write}(x, n)$

3.2 SI-2

"*The write-sets of any committed concurrent transaction must be disjoint.* "

This property ensures that no transaction will ever try to write to a value that is already written by another concurrent transaction, resulting in the loss of an update [3].

This anomaly can be prevented by introducing the *NoConflict* axiom [3] :

$\forall T, S \in \mathcal{T} \cdot \forall x \cdot (T, S \in WriteTx_x \wedge T \neq S) \Longrightarrow (T \xrightarrow{VIS} S \vee S \xrightarrow{VIS} T)$

$WriteTx_x$ is the set of transactions that perform a `write` operation on x. This property implies that when two different transactions try to write to the same data item x, one of them

has to be aware of the other, and thus wait until it has committed before reading and updating x.

Chapter 4

Speculative Snapshot Isolation

To extend SI in order to allow *speculative* transactions to occur, we need to extend some of the definitions.

As described in STR [2], speculation comes in two forms : *specuative reads* and *speculative commits*. Speculative reads occur when a transactions read an update that has been done by a transaction that has not yet passed the global certification, and a speculative commit allows transactions to expose their updates to other local transactions before global certification.

A *speculative* transaction is a transaction whose snapshot may include *locally committed* transactions. The first notion to introduce is thus the notion of *local commit*. We distinguish the *local* commits from the *final* commits as follows [2] : Transactions are first executed in the node where they were originated. When they request to commit, they undergo a local certification phase, which checks for conflicts with concurrent transactions, originated either locally or remotely. If the local certification phase succeeds, we say that transactions local commit and are attributed a local commit timestamp, LC. Next, they execute a global certification phase that detects conflicts with transactions originated at any other node in the system. Transactions that pass the global certification phase are said to final commit and are attributed a final commit timestamp, FC. A transaction thus has new information that is important to consider : its *originating node, T.node*.

The different commit stages occur at different times, and it is important to keep track of when those events occurred. We thus add to the transaction its *Read Snapshot, T.RS*, corresponding to the time at which the transaction took its snapshot of the dataset, its *local commit timestamp, T.LC* and its *final commit timestamp, T.FC*.

We have seen that local commits can be used by locally issued transactions to perform operations. We thus need to extend the notion of *commit order* and *visibility* in order to include local commits in them.

The commit order, CO, is a total order on $\mathcal{T}$ that orders all transactions that have committed in SI. In SPSI, transactions have more than one commit timestamp, and thus this order mus be adapted. We thus define COe, the extended commit order :

$$\forall T, S \in \mathcal{T}_s \cdot T \xrightarrow{COe} S \implies T.FC \leq S.RS \vee (T.LC \leq S.RS \wedge T.node == S.node)$$

Using the extended commit order, COe, we can build $VISe$, the extended visibility order, that extends the VIS order by including speculative commits in transactions' snapshots on a local node. If a transaction T local-commits on a node right before a transaction S starts on

the same node, then T's updates should be included in S's snapshot. We say that T is *visible* to S :

$T \xrightarrow{VISe} S \iff T \xrightarrow{COe} S \wedge (\exists x \cdot ((T \in WriteTx_x \wedge S \in ReadTx_x) \vee (T \in WriteTx_x \wedge S \in WriteTx_x)))$

Chapter 5

Properties of SPSI

This section presents the four properties of SPSI as defined in [2]. Those properties can then be expressed in a formal way using the extended commit and visibility orders defined earlier.

5.1 SPSI-1

"*Speculative Snapshot Read* "

A transaction T originated from node N at time t must observe the most recent version created by transactions that :

- i) final commit with timestamp $FC \leq t$
- ii) local commit at node N with timestamp $LC \leq t$

5.2 SPSI-2

"*No Write-Write conflict among Final Committed Transactions* "

The write-sets of any final committed transaction must be disjoint.

5.3 SPSI-3

"*No Write-Write conflict among Transactions in a Speculative Snapshot* "

Let S be the set of transactions included in a snapshot. The write-sets of any concurrent transactions in S must be disjoint.

5.4 SPSI-4

"*No Dependencies from Uncommitted Transactions* "

A transaction can only be final committed if it does not data or flow depend on any local-committed or aborted transaction.

5.5 Expressing the SPSI properties using COe and VISe

This section formalizes the 4 properties stated above. They are written in a way that is deliberately similar to the properties of SI, to make them easier to understand but also to help see

what conditions must be met in order to be able to map SPSI properties to SI properties.

5.5.1 SPSI-1

Once we consider the commit order to include local commits, the SPSI-1 property remains very similar to the SI-1 property. Internal consistency does not change at all, for it ensures internal consistency. Transactions are still atomic.

$$\forall (E, po) \in (\mathcal{T}_s \setminus \mathcal{T}_{lc \to a}) \cdot \forall e \in E \cdot \forall (x, n) \cdot op(e) = \texttt{read}(x, n) \wedge \{f \mid op(f) = \_(x, \_) \wedge f \xrightarrow{po} e\} \Longrightarrow op(max_{op}\{f \mid op(f) = \_(x, \_) \wedge f \xrightarrow{po} e\}) = \_(x, n)$$

External consistency is a bit different from the axiom defined for non-speculative SI. The value read by a transaction is not necessarily the result of a final committed transaction : it can be the result of a local-committed transaction, given that it occurred on the same node. This is why we extended the commit order : it now handles this new possibility. The axiom can thus be written as :

$$\forall T \in (\mathcal{T}_s \setminus \mathcal{T}_{lc \to a}) \cdot \forall x, n \cdot T \vdash \texttt{read}(x, n) \Longrightarrow max_{COe}(VISe^{-1}(T) \cap WriteTx_x) \vdash \texttt{write}(x, n)$$

This allows transactions from a same node to include local commits in their snapshots.

5.5.2 SPSI-2

SPSI-2 states that no *write-write conflict* can be observed among *final* committed transactions, hence the restriction to $\mathcal{T}_{fc}$. *Write-write conflicts* are defined in the same fashion as for SI executions, using visibility : if two distinct final-committed transactions happen to write to the same item, x, one of them has to be visible to the other. In other words, the update done by one transaction must be included in the other transaction's snapshot. We thus write :

$$\forall T, S \in \mathcal{T}_{fc} \cdot \exists x \cdot (T, S \in WriteTx_x \wedge T \neq S) \Longrightarrow (T \xrightarrow{VISe} S \vee S \xrightarrow{VISe} T)$$

5.5.3 SPSI-3

SPSI-3 is focused on preventing *write-write conflicts* within a *speculative* transaction's snapshot, denoted $VISe^{-1}(...)$. This property completes SPSI-2 in the role of preventing write-write conflicts in the system : within a snapshot, two distinct write-conflicting transactions cannot be concurrent. One of them has to be visible to the other, which is again captured by $VISe$.

$$\forall U \in \mathcal{T}_s \cdot \forall T, S \in VISe^{-1}(U) \cdot \exists x \cdot (T, S \in WriteTx_x \wedge T \neq S) \Longrightarrow (T \xrightarrow{VISe} S \vee S \xrightarrow{VISe} T)$$

5.5.4 SPSI-4

SPSI-4 introduces the concepts of *data dependency* and *flow dependency*.

A transaction T is *data dependent* on a transaction S if T needs to read items that are written to by S.

In [2], data dependencies are said to arise when *"A local committed transaction T exposes its state to other transactions via the* speculative read mechanism *"*. This mechanism is not very clearly defined in the paper, but it can be guessed that it is the mechanism allowing a transaction to read an item committed by T after its local certification, but before its global certification.

This means that data dependency can be solved using $VISe$: T comes before its dependent transactions in COe, and the data they have in common (x) is written to by T and read by dependent transactions (S in $VISe$ definition). Being ordered by $VISe$ then solves the data dependency problem.

Flow dependency was irrelevant in SI since the transactions were non-speculative. Flow dependency is a dependency that matters only to the user, and is completely exogenous to the system. Flow-dependent transactions must be ordered because one of them has to commit first, but this ordering does not depend on the data inside the system.

As an example, imagine someone that is planning a trip, and let the two transactions he needs to make be the booking of a plane ticket and the booking of a hotel room. The first step he takes is to book the plane, and the next step is booking the hotel room. But since there is no point in booking a room in a hotel where the client cannot go, the plane booking must not fail in order to complete the hotel booking. If the booking of the plane fails, then the booking of the hotel must abort (since it *flow-depends* on an aborted transaction). This is a form of flow-dependency. However, the booking of a plane seat or hotel room are completely valid as standalone transactions, and the dependency comes solely from the client's point of view.

To tackle this problem, we introduce a new partial order, FO, that orders final-commit timestamps of transactions that are flow-dependent. This order is considered as given by the programmer and is not defined within the system :

$$T \xrightarrow{FO} S \implies T.FC > S.FC$$

We can now define a new order, called *Dependency Order*, DO, as $VISe \cup FO$. DO orders all transactions that are either data- or flow-dependent.

Chapter 6

SPSI satisfies SI

In this section we will prove that introducing speculation in Snapshot Isolation does not alter the final results of executions. To achieve that goal, we assume that executions that comply to the SI-1 and SI-2 properties are *correct* [3].

6.1 Speculative execution termination

In this section, we prove that a speculative execution can be mapped to a non-speculative execution, according to their respective definitions and the concept of termination. In the following sections, we reconstruct the properties of the non-speculative execution that we have mapped to the speculative execution, proving that no property is violated in the process.

Proposition 1 : Speculative executions can be mapped to non-speculative executions.

As defined in section two, speculative executions can observe 3 types of transactions : *final-committed* transactions, *local-committed* transactions that will final-commit, and *local-committed* transactions that will abort. The main property of this definition is that if the number of local committed transactions is zero, then all transactions have final committed and this definition is equivalent to the definition of non-speculative execution in SI. Assuming a system for which all final committed transactions respect SPSI-1 to 4, the speculative execution will eventually reach a stage in which speculation disappears, and the execution "terminates" (the final state is repeated). In this final state, both $\mathcal{T}_{lc \to f}$ and $\mathcal{T}_{lc \to a}$ are empty, since all the work transactions needed to make has been done and certified across the whole system. Conflicting transactions have been aborted, and from a semantic point of view, it is as if they had never existed; and non-conflicting transactions have been certified and have final committed. The two "local committed" transactions sets have thus been emptied.

The set of speculative transactions is composed of the final committed transactions, the local committed transactions that will final commit and the local committed transactions that will abort .
$\mathcal{T}_s = \mathcal{T}_{fc} \cup \mathcal{T}_{lc \to f} \cup \mathcal{T}_{lc \to a}$

We know that transactions within $\mathcal{T}_{lc \to a}$ have aborted, and those within $\mathcal{T}_{lc \to f}$ have final committed. Those sets are thus empty :
$\mathcal{T}_{lc \to f} \cup \mathcal{T}_{lc \to a} = \varnothing$

We now have :
$\mathcal{T}_s = \mathcal{T}_{fc} = \mathcal{T}$

Thus, we can write $\mathcal{X}_s = (\mathcal{T}, VISe, COe)$

The commit order COe orders transactions according to their commit timestamps. Since there are no more local commits, all remaining transactions have a final commit timestamp to order them. Those transactions have been certified across the whole system and are thus equivalent to non speculative transactions. COe is then equivalent to CO. And since $VISe$ uses COe in its definition, we can say for the same reason that $VISe$ is equivalent to VIS. We thus have :

$\mathcal{X}_s = (\mathcal{T}, VIS, CO)$

And finally, we have matched the definition of a speculative execution to the definition of a non-speculative execution :

$\mathcal{X}_s = (\mathcal{T}, VIS, CO) = \mathcal{X}$

6.2 Correctness

We have shown that a speculative execution can have the same definition as a non-speculative execution if it terminates correctly. But so far, there is no guarantee that this non-speculative execution that we have built is correct, i.e. complies to SI-1 and SI-2. The goal of this section is to construct those two properties from what we know of speculative executions.

6.2.1 Rebuilding SI-1

Proposition 2 : SI-1 can be retrieved from the properties of SPSI.

The SI-1 property comes in two parts. The first part is called *internal consistency axiom*, and the second is called *external consistency axiom*.

Internal consistency is unaffected by speculation. As stated before, internal consistency ensures isolation of the transactions : within the events sequence of a particular transaction, no effect other than that of the running transaction can be observed. We have shown that in a terminated speculative execution, the transactions set is equivalent to the non-speculative transactions set :

$\forall (E, po) \in (\mathcal{T}_s \setminus \mathcal{T}_{lc \to a}) \cdot \forall e \in E \cdot \forall (x, n) \cdot op(e) = \texttt{read}(x, n) \wedge \{f \mid op(f) = \_(x, \_) \wedge f \xrightarrow{po} e\} \Longrightarrow op(max_{op}\{f \mid op(f) = \_(x, \_) \wedge f \xrightarrow{po} e\}) = \_(x, n)$

Thus

$\forall (E, po) \in \mathcal{T} \cdot \forall e \in E \cdot \forall (x, n) \cdot op(e) = \texttt{read}(x, n) \wedge \{f \mid op(f) = \_(x, \_) \wedge f \xrightarrow{po} e\} \Longrightarrow op(max_{op}\{f \mid op(f) = \_(x, \_) \wedge f \xrightarrow{po} e\}) = \_(x, n)$

Internal consistency states that within the transaction, every event reads or writes to data that has been read or written to by the previous event. But this property says nothing about what the *first* event reads. This is why it needs to be augmented with *external consistency*. External consistency states that the first event of a transaction, if it is a read, will read a value that has been written to by the previous transaction, according to COe :

$$\forall T \in (\mathcal{T}_s \setminus \mathcal{T}_{lc \to a}) \cdot \forall x, n \cdot T \vdash \texttt{read}(x, n) \implies max_{COe}(VISe^{-1}(T) \cap WriteTx_x) \vdash \texttt{write}(x, n)$$

Since no more speculation is occurring, we observe transactions that are semantically equivalent to non-speculative transactions. COe orders them in the same way as CO, and $VISe$ becomes equivalent to VIS by definition. Thus, the external consistency axiom becomes similar to the axiom defined in SI-1 :

$$\forall T \in \mathcal{T} \cdot \forall x, n \cdot T \vdash \texttt{read}(x, n) \implies max_{CO}(VIS^{-1}(T) \cap WriteTx_x) \vdash \texttt{write}(x, n)$$

We have thus reconstructed the SI-1 property from a speculative execution that complies to the SPSI-1 property.

6.2.2 Rebuilding SI-2

Proposition 3 : SI-2 can be retrieved from the properties of SPSI.

The SI-2 property ensures the isolation of the snapshots : it guarantees that if two distinct transactions, T and S, try to update the same value, then they are ordered by VIS, which means that the update of one of them is included in the snapshot of the other :

$$\forall T, S \in \mathcal{T} \cdot \forall x \cdot (T, S \in WriteTx_x \wedge T \neq S) \implies (T \xrightarrow{VIS} S \vee S \xrightarrow{VIS} T)$$

SPSI-2 gives a similar guarantee to transactions that have final-committed : it states that between two final committed transactions, no write-write conflict can be observed : if two final-committed transactions T and S both update the same value, one of them is visible to the other.

$$\forall T, S \in \mathcal{T}_{fc} \cdot \exists x \cdot (T, S \in WriteTx_x \wedge T \neq S) \implies (T \xrightarrow{VIS} S \vee S \xrightarrow{VISe} T)$$

Having shown that, after execution, COe becomes equivalent to CO and $VISe$ becomes equivalent to VIS in a similar fashion, we can write :

$$\forall T, S \in \mathcal{T}_{fc} \cdot \exists x \cdot (T, S \in WriteTx_x \wedge T \neq S) \implies (T \xrightarrow{VIS} S \vee S \xrightarrow{VIS} T)$$

But SPSI-2 only involves final-committed transactions. To complete the definition, we add property SPSI-3, that ensures that in the snapshot of a speculative transaction, no write-write conflict can be observed either :

$$\forall U \in \mathcal{T}_s \cdot \forall T, S \in VISe^{-1}(U) \cdot \exists x \cdot (T, S \in WriteTx_x \wedge T \neq S) \implies (T \xrightarrow{VISe} S \vee S \xrightarrow{VISe} T)$$

Again, mapping $VISe$ to VIS, we have :

$$\forall U \in \mathcal{T}_s \cdot \forall T, S \in VIS^{-1}(U) \cdot \exists x \cdot (T, S \in WriteTx_x \wedge T \neq S) \implies (T \xrightarrow{VIS} S \vee S \xrightarrow{VIS} T)$$

This property completes SPSI-2 by extending its action to transactions *included in a snapshot*. The property is now extended to *all* transactions : transactions that have final-committed in SPSI-2 and all kinds of transactions in the snapshot of U. By unifying those sets together, we have a property on all transactions, that has become SI-2 with the equivalence between $\mathcal{T}_s$ and $\mathcal{T}$.

$$\forall T, S \in \mathcal{T} \cdot \exists x \cdot (T, S \in WriteTx_x \wedge T \neq S) \implies (T \xrightarrow{VIS} S \vee S \xrightarrow{VIS} T)$$

As a reminder, we assumed SI executions to be correct. If a speculative execution can be mapped to such a correct execution, without violating any of its properties, then the speculative execution is considered correct as well. Additionally, the properties of SPSI have been built in a fashion that offers strong guarantees on transactions *during* the speculative execution.

We now have retrieved the two properties of an SI execution. The reader might have noticed that we have not used SPSI-4 doing this. That is because SPSI-4 does not directly involve speculation mechanisms, but rather give an order to prevent speculation mechanisms from losing ordering information that might not be detectable by the system, such as flow dependency. After mapping a speculative execution to a non speculative execution , SPSI-4 still holds for that execution : it ensures that, when speculating, a transaction does not overshoot another if it needs that other transaction to final commit first, according to DO.

Conclusion

In this work, we have formalized the properties of SI and SPSI, and used those properties to show that the state resulting from a speculative execution is indistinguishable from the state resulting from a non-speculative execution, and that the properties of SPSI allow to reconstruct the properties of SI. In addition to that, the properties of SPSI offer strong guarantees, even on transactions that *will* eventually abort.

Bibliography

[1] ADYA Atul (1999). *Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions*

[2] LI Zhongmiao, ROMANO Paolo, VAN ROY Peter (2017). *Speculative Transaction Processing in Geo-Replicated Data Stores*

[3] CERONE Andrea, GOTSMAN Alexey (2016). *Analysing Snapshot Isolation*

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**