On the Scalability of Snapshot Isolation*

Masoud Saeida Ardekani[1], Pierre Sutra[2], Marc Shapiro[3], and Nuno Preguiça[4]

[1] Université Pierre et Marie Curie, Paris, France
[2] University of Neuchâtel, Switzerland
[3] INRIA & Université Pierre et Marie Curie, Paris, France
[4] CITI, Universidade Nova de Lisboa, Lisbon, Portugal

Abstract. Many distributed applications require transactions. However, transactional protocols that require strong synchronization are costly in large scale environments. Two properties help with scalability of a transactional system: genuine partial replication (GPR), which leverages the intrinsic parallelism of a workload, and snapshot isolation (SI), which decreases the need for synchronization. We show that under standard assumptions (data store accesses are not known in advance, and transactions may access arbitrary objects in the data store), it is impossible to have both SI and GPR. Our impossibility result is based on a novel decomposition of SI which proves that, like serializability, SI is expressible on plain histories.

1 Introduction

Large scale transactional systems have conflicting requirements. On the one hand, strong transactional guarantees are fundamental to many applications. On the other, remote communication and synchronization are costly and should be avoided.[5]

To maintain strong consistency guarantees while alleviating the cost of synchronization, Snapshot Isolation (SI) is a popular approach in both distributed database replications [1–3], and software transactional memories [4, 5]. Under SI, a transaction accesses its own *consistent snapshot* of the data, which is unaffected by concurrent updates. A read-only transaction always commits unilaterally and without synchronization. An update transaction synchronizes on commit to ensure that no concurrent conflicting transaction has committed before it.

Our first contribution is to prove that SI is equivalent to the conjunction of the following properties: *(i)* no cascading aborts, *(ii)* strictly consistent snapshots, i.e., a transaction observes a snapshot that coincides with some point in (linear) time, *(iii)* two concurrent write-conflicting update transactions never

* This work is partially supported by FCT/MCT projects PEst-OE/EEI/UI0527/2011 and PTDC/EIA-EIA/108963/2008, and the European Commission's Seventh Framework Program (FP7) under grant agreement No. 318809 (LEADS).

[5] We address general-purpose transactions, i.e., we assume that a transaction may access any object in the system, and that its read- and write-sets are not known in advance.

both commit, and *(iv)* snapshots observed by transactions are monotonically ordered. Previous definitions of SI [6, 7] extend histories with abstract snapshot points. Our decomposition shows that in fact, like serializability, SI can be defined on plain histories [8].

Modern data stores replicate data for both performance and availability. Full replication does not scale, as every process must perform all updates. *Partial replication* (PR) aims to address this problem, by replicating only a subset of the data at each process. Thus, if transactions would communicate only over the minimal number of replicas, synchronization and computation overhead would be reduced. However, in the general case, the overlap of transactions cannot be predicted; therefore, many PR protocols perform system-wide global consensus [2, 3] or communication [9]. This negates the potential advantages of PR; hence, we require *genuine* partial replication [10] (GPR), in which a transaction communicates only with processes that replicate some object accessed in the transaction. With GPR, independent transactions do not interfere with each other, and the intrinsic parallelism of a workload can be thus exploited.

Our second contribution is to show that SI and GPR are incompatible. More precisely, we prove that an asynchronous message-passing system supporting GPR, even if it is failure-free, cannot compute monotonically ordered snapshots, nor strictly consistent ones.

This paper proceeds as follows. We introduce our system model in Section 2. Section 3 presents our decomposition of SI. Section 4 shows that GPR and SI are mutually incompatible. We discuss implications of this result and related work in Section 5. Section 6 closes this paper. Due to space constraints, some proofs are deferred to our companion technical report [11].

2 Model

This section defines the elements in our model and formalizes SI and GPR .

2.1 Objects & transactions

Let *Objects* be a set of objects, and $\mathcal{T}$ be a set of transaction identifiers. Given an object x and an identifier i, x_i denotes *version* i of x. A *transaction* $T_{i \in \mathcal{T}}$ is a finite permutation of read and write operations followed by a *terminating* operation, commit (c_i) or abort (a_i). We use $w_i(x_i)$ to denote transaction T_i writing version i of object x, and $r_i(x_j)$ to mean that T_i reads version j of object x. In a transaction, every write is preceded by a read on the same object, and every object is read or written at most once.[6] We note $ws(T_i)$ the write set of T_i, i.e., the set of objects written by transaction T_i. Similarly, $rs(T_i)$ denotes the read set of transaction T_i. The *snapshot* of T_i is the set of versions read by T_i. Two transactions *conflict* when they access the same object and one of them modifies it; they *write-conflict* when they both write to the same object.

[6] These restrictions ease the exposition of our results but do not change their validity.

2.2 Histories

A *complete history* h is a partially ordered set of operations such that (1) for every operation o_i appearing in h, transaction T_i terminates in h, (2) for every two operations o_i and o'_i appearing in h, if o_i precedes o'_i in T_i, then $o_i <_h o'_i$, (3) for every read $r_i(x_j)$ in h, there exists a write operation $w_j(x_j)$ such that $w_j(x_j) <_h r_i(x_j)$, and (4) any two write operations over the same objects are ordered by $<_h$. A *history* is a prefix of a complete history. For some history h, order $<_h$ is the *real-time order* induced by h. Transaction T_i is *pending* in history h if T_i does not commit, nor abort in h. We note $\ll_h$ the version order induced by h between different versions of an object, i.e., for every object x, and any two transactions T_i and T_j, $x_i \ll_h x_j = w_i(x_i) <_h w_j(x_j)$. Following Bernstein et al. [12], we depict a history as a graph. We illustrate this with history h_1 below in which transaction T_a reads the initial versions of objects x and y, while transaction T_1 (respectively T_2) updates x (resp. y).[7]

$$h_1 = r_a(x_0) \longrightarrow r_1(x_0).w_1(x_1).c_1$$
$$\searrow\ r_a(y_0).c_a \longrightarrow r_2(y_0).w_2(y_2).c_2$$

When order $<_h$ is total, we shall write a history as a permutation of operations, e.g., $h_2 = r_1(x_0).r_2(y_0).w_2(y_2).c_1.c_2$.

2.3 Snapshot Isolation

Snapshot isolation (SI) was introduced by Berenson et al. [8], then later generalized under the name GSI by Elnikety et al. [7]. In this paper, we make no distinction between SI and GSI.

Let us consider a function $\mathcal{S}$ which takes as input a history h, and returns an extended history h_s by adding a *snapshot point* to h for each transaction in h. Given a transaction T_i, the snapshot point of T_i in h_s, denoted s_i, precedes every operation of transaction T_i in h_s. A history h is in SI if, and only if, there exists a function $\mathcal{S}$ such that $h_s = \mathcal{S}(h)$ and h_s satisfies the following rules:

D1 (Read Rule)

$$\begin{array}{ll} \forall r_i(x_{j\neq i}), w_{k\neq j}(x_k), c_k \in h_s : & \\ \quad c_j \in h_s & (D1.1) \\ \wedge\ c_j <_{h_s} s_i & (D1.2) \\ \wedge\ (c_k <_{h_s} c_j \vee s_i <_{h_s} c_k) & (D1.3) \end{array}$$

D2 (Write Rule)

$$\begin{array}{l} \forall c_i, c_j \in h_s : \\ \quad ws(T_i) \cap ws(T_j) \neq \varnothing \\ \quad \Rightarrow (c_i <_{h_s} s_j \vee c_j <_{h_s} s_i) \end{array}$$

2.4 System

We consider a message-passing distributed system of n processes $\Pi = \{p_1, \ldots, p_n\}$. We shall define our synchrony assumptions later. Following Fischer et al. [13], an execution is a sequence of steps made by one or more processes. During an execution, processes may fail by crashing. A process that does not crash is said

[7] Throughout the paper, read-only transactions are specified with an alphabet subscript, and update transactions are shown with numeric subscript.

correct; otherwise it is *faulty*. We note $\mathfrak{F}$ the refinement mapping [14] from executions to histories, i.e., if ρ is an execution of the system, then $\mathfrak{F}(\rho)$ is the history produced by ρ. A history h is *acceptable* if there exists an execution ρ such that $h = \mathfrak{F}(\rho)$. We consider that given two sequences of steps U and V, if U precedes V in some execution ρ, then the operations implemented by U precedes (in the sense of $<_h$) the operations implemented by V in the history $\mathfrak{F}(\rho)$.[8]

2.5 Partial Replication

A data store $\mathcal{D}$ is a finite set of tuples (x, v, i) where x is an object (data item), v a value, and $i \in \mathcal{T}$ a version. Each process in Π holds a data store such that initially every object x has version x_0. For an object x, $\mathit{Replicas}(x)$ denotes the set of processes, or *replicas*, that hold a copy of x. By extension for some set of objects X, $\mathit{Replicas}(X)$ denotes the replicas of X; given a transaction T_i, $\mathit{Replicas}(T_i)$ equals $\mathit{Replicas}(rs(T_i) \cup ws(T_i))$.

We make no assumption about how objects are replicated. The coordinator of T_i, denoted $\mathit{coord}(T_i)$, is in charge of executing T_i on behalf of some client (not modeled). The coordinator does not know in advance the read set or the write set of T_i. To model this, we consider that every prefix of a transaction (followed by a terminating operation) is a transaction with the same id.

Genuine Partial Replication (GPR) aims to ensure that, when the workload is parallel, throughput scales linearly with the number of nodes [10]:

- **GPR.** For any transaction T_i, only processes that replicate objects accessed by T_i make steps to execute T_i.

2.6 Progress

The read rule of SI does not define what is the snapshot to be read. According to Adya [6], "transaction T_i's snapshot point needs not be chosen after the most recent commit when T_i started, but can be selected to be some (convenient) earlier point." To avoid that read-only transactions always observe outdated data, we add the following rule:

- **Non-trivial SI.** Consider an acceptable history h and a transaction T_i pending in h such that the next operation invoked by T_i is a read on some object x. Note x_j the latest committed version of x prior to the first operation of T_i in h. Let ρ be an execution satisfying $\mathfrak{F}(\rho) = h$. If $h.r_i(x_j)$ belongs to SI then *there exists* an execution ρ' extending ρ such that in history $\mathfrak{F}(\rho')$, transaction T_i reads at least (in the sense of $\ll_h$) version x_j of x.

In addition, we consider that the system provides the following progress guarantees on transactions:

[8] Notice that since steps to implement operations may interleave, $<_h$ is not necessarily a total order.

- **Obstruction-free Updates.** For every update transaction T_i, if $coord(T_i)$ is correct then T_i eventually terminates. Moreover, if T_i does not write-conflict with some concurrent transaction, then T_i eventually commits.

- **Wait-free Queries.** If $coord(T_i)$ is correct and T_i is a read-only transaction, then transaction T_i eventually commits.

3 Decomposing SI

This section defines four properties, whose conjunction is necessary and sufficient to attain SI. We later use these properties in Section 4 to derive our impossibility result.

3.1 Cascading Aborts

Intuitively, a read-only transaction must abort if it observes the effects of an uncommitted transaction that later aborts. By guaranteeing that every version read by a transaction is committed, rules D1.1 and D1.2 of SI prevent such a situation to occur. In other words, these rules *avoid cascading aborts*. We formalize this property below:

Definition 1 (Avoiding Cascading aborts). *History h avoids cascading aborts, if for every read $r_i(x_j)$ in h, c_j precedes $r_i(x_j)$ in h. ACA denotes the set of histories that avoid cascading aborts.*

3.2 Consistent and Strictly Consistent Snapshots

Consistent and strictly consistent snapshots are defined by refining causality into a dependency relation as follows:

Definition 2 (Dependency). *Consider a history h and two transactions T_i and T_j. We note $T_i \rhd T_j$ when $r_i(x_j)$ is in h. Transaction T_i depends on transaction T_j when $T_i \rhd^* T_j$ holds.*[9] *Transaction T_i and T_j are independent if neither $T_i \rhd^* T_j$, nor $T_j \rhd^* T_i$ hold.*

This means that a transaction T_i depends on a transaction T_j if T_i reads an object modified by T_j, or such a relation holds by transitive closure. To illustrate this definition, consider history $h_3 = r_1(x_0).w_1(x_1).c_1.r_a(x_1).c_a.r_b(y_0).c_b$. In h_3, transaction T_a depends on T_1. Notice that, even if T_1 causally precedes T_b, T_b does not depend on T_1 in h_3.

We now define consistent snapshots with the above dependency relation. A transaction sees a consistent snapshot iff it observes the effects of all transactions it depends on [15]. For example, consider the history $h_4 = r_1(x_0).w_1(x_1).c_1.r_2(x_1)$ $.r_2(y_0).w_2(y_2).c_2.r_a(y_2).r_a(x_0).c_a$ In this history, transaction T_a does not see a

[9] We note $\mathcal{R}^*$ the transitive closure of some binary relation $\mathcal{R}$.

consistent snapshot: T_a depends on T_2, and T_2 also depends on T_1, but T_a does not observe the effect of T_1 (i.e., x_1). Formally, consistent snapshots are defined as follows:

Definition 3 (Consistent snapshot). *A transaction T_i in a history h observes a consistent snapshot iff, for every object x, if (i) T_i reads version x_j, (ii) T_k writes version x_k, and (iii) T_i depends on T_k, then version x_k is followed by version x_j in the version order induced by h ($x_k \ll_h x_j$). We write $h \in CONS$ when all transactions in h observe a consistent snapshot.*

SI requires that a transaction observes the committed state of the data at some *point* in the past. This requirement is stronger than consistent snapshot. For some transaction T_i, it implies that (SCONSa) there exists a snapshot point for T_i , and (SCONSb) if transaction T_i observes the effects of transaction T_j, it must also observe the effects of all transactions that precede T_j in time. A history is called strictly consistent if both SCONSa and SCONSb hold.

To illustrate this, consider the following history: $h_5 = r_1(x_0).w_1(x_1).c_1.r_a(x_1).r_2(y_0).w_2(y_2).c_2.r_a(y_2).c_a$. Because $r_a(x_1)$ precedes c_2 in h_5, y_2 cannot be observed when T_a takes its snapshot. As a consequence, the snapshot of transaction T_a is not strictly consistent. This issue is disallowed by SCONSa. Now, consider history $h_6 = r_1(x_0).w_1(x_1).c_1.r_2(y_0).w_2(y_2).c_2.r_a(x_0).r_a(y_2).c_a$. Since c_1 precedes c_2 in h_6 and transaction T_a observes the effect of T_2 (i.e., y_2), it should also observe the effect of T_1 (i.e., x_1). SCONSb prevents history h_6 to occur.

Definition 4 (Strictly consistent snapshot). *Snapshots in history h are strictly consistent, when for any committed transactions T_i, T_j, $T_{k\neq j}$ and T_l, the following two properties hold:*

- $\forall r_i(x_j), r_i(y_l) \in h : r_i(x_j) \not<_h c_l$ (*SCONSa*)
- $\forall r_i(x_j), r_i(y_l), w_k(x_k) \in h :$
  $c_k <_h c_l \Rightarrow c_k <_h c_j$ (*SCONSb*)

We note SCONS the set of strictly consistent histories.

3.3 Snapshot Monotonicity

In addition, SI requires what we call monotonic snapshots. For instance, although history h_7 below satisfies SCONS, this history does not belong to SI. Indeed, since T_a reads $\{x_0, y_2\}$, and T_b reads $\{x_1, y_0\}$, there is no extended history that would guarantee the read rule of SI.

$$h_7 = r_a(x_0) \longrightarrow r_1(x_0).w_1(x_1).c_1 \longrightarrow r_b(x_1).c_b$$
$$r_b(y_0) \longrightarrow r_2(y_0).w_2(y_2).c_2 \longrightarrow r_a(y_2).c_a$$

SI requires monotonic snapshots. However, the underlying reason is intricate enough that some previous works [4, for instance] do not ensure this property, while claiming to be SI. Below, we introduce an ordering relation between snapshots to formalize snapshot monotonicity.

Definition 5 (Snapshot precedence). *Consider a history h and two distinct transactions T_i and T_j. The snapshot read by T_i precedes the snapshot read by T_j in history h, written $T_i \rightarrow T_j$, when $r_i(x_k)$ and $r_j(y_l)$ belong to h and either (i) $r_i(x_k) <_h c_l$ holds, or (ii) transaction T_l writes x and $c_k <_h c_l$ holds.*

For more illustration, consider histories $h_8 = r_1(x_0).w_1(x_1).c_1.r_2(y_0).w_2(y_2).r_a(x_1).c_2.r_b(y_2).c_a.c_b$ and $h_9 = r_1(x_0).w_1(x_1).c_1.r_a(x_1).c_a.r_2(x_1).r_2(y_0).w_2(x_2).w_2(y_2).c_2.r_b(y_2).c_b$. In history h_8, $T_a \rightarrow T_b$ holds because $r_a(x_1)$ precedes c_2 and T_b reads y_2. In h_9, c_1 precedes c_2 and both T_1 and T_2 modify object x. Thus, $T_a \rightarrow T_b$ also holds. We define snapshot monotonicity using snapshot precedence as follows:

Definition 6 (Snapshot monotonicity). *Given some history h, if the relation $\rightarrow^*$ induced by h is a partial order, the snapshots in h are* monotonic*. We note MON the set of histories that satisfy this property.*

According to this definition, both $T_a \rightarrow T_b$ and $T_b \rightarrow T_a$ hold in history h_7. Thus, history h_7 does not belong to MON.

Non-monotonic snapshots are observed under update serializability [16], that is when queries observe consistent state, but only updates are serializable.

3.4 Write-Conflict Freedom

Rule D2 of SI forbids two concurrent write-conflicting transactions from both committing. Since in our model we assume that every write is preceeded by a corresponding read on the same object, every update transaction depends on a previous update transaction (or on the initial transaction T_0). Therefore, under SI, concurrent conflicting transactions must be independent:

Definition 7 (Write-Conflict Freedom). *A history h is write-conflict free if two independent transactions never write to the same object. We denote by WCF the histories that satisfy this property.*

3.5 The decomposition

Theorem 1 below establishes that a history h is in SI iff (1) every transaction in h sees a committed state, (2) every transaction in h observes a strictly consistent snapshot, (3) snapshots are monotonic, and (4) h is write-conflict free. A detailed proof appears in our companion technical report [11].

Theorem 1. $SI = ACA \cap SCONS \cap MON \cap WCF$

To the best of our knowledge, this result is the first to prove that SI can be split into simpler properties. Theorem 1 also establishes that SI is definable on plain histories. This has two interesting consequences: (i) a transactional system does not have to explicitly implement snapshots to support SI, and (ii) one can compare SI to other consistency criterion without relying on a phenomena based characterization.[10]

[10] Contrary to, e.g., the work of Adya [6].

4 The impossibility of SI with GPR

This section leverages our previous decomposition result to show that SI is inherently non-scalable. In more details, we prove that none of MON, SCONSa or SCONSb is attainable in some asynchronous failure-free GPR system Π when updates are obstruction-free and queries are wait-free. To prove these results, we first characterize below histories acceptable by Π.

Lemma 1. *Let $h = \mathfrak{F}(\rho)$ be an acceptable history by* Π *such that a transaction T_i is pending in h. Note X the set of objects accessed by T_i in h. Only processes in $Replicas(X)$ make steps to execute T_i in ρ.*

Proof. (By contradiction.) Consider that a process $p \notin Replicas(X)$ makes steps to execute T_i in ρ. Since the prefix of a transaction is a transaction with the same id, we can consider an extension ρ' of ρ such that T_i does not execute any additional operation in ρ' and $coord(T_i)$ is correct in ρ'. The progress requirements satisfied by Π imply that T_i terminates in ρ'. However, process $p \notin Replicas(X)$ makes steps to execute T_i in ρ'. A contradiction to the fact that Π is GPR.

We now state that monotonic snapshots are not constructable in Π. Our proof holds because objects accessed by a transaction are not known in advance.

Theorem 2. *No asynchronous failure-free GPR system implements MON*

Proof. (By contradiction.) Let us consider (i) four objects x, y, z and u such that for any two objects in $\{x, y, z, u\}$, their replica sets do not intersect; (ii) four queries T_a, T_b, T_c and T_d accessing respectively $\{x, y\}$, $\{y, z\}$, $\{z, u\}$ and $\{u, x\}$; and (iii) four updates T_1, T_2, T_3 and T_4 modifying respectively x, y, z and u.

Obviously, history $r_b(y_0)$ is acceptable, and since updates are obstruction-free, $r_b(y_0).r_2(y_0).w_2(y_2).c_2$ is also acceptable. Applying that Π satisfies non-trivial SI, we obtain that history $r_b(y_0).r_2(y_0).w_2(y_2).c_2.r_a(x_0).r_a(y_2)$ is acceptable. Since T_a must be wait-free, $h = r_b(y_0).r_2(y_0).w_2(y_2).c_2.r_a(x_0).r_a(y_2).c_a$ is acceptable as well. Using a similar reasoning, history $h' = r_d(u_0).r_4(u_0).w_4(u_4).c_4$ $.r_c(z_0).r_c(u_4).c_c$ is also acceptable. We note ρ and ρ' respectively two sequences of steps such that $\mathfrak{F}(\rho) = h$ and $\mathfrak{F}(\rho') = h'$.

System Π is GPR. As a consequence, Lemma 1 tells us that only processes in $Replicas(x, y)$ make steps in ρ. Similarly, only processes in $Replicas(u, z)$ make steps in ρ'. By hypothesis, $Replicas(x, y)$ and $Replicas(u, z)$ are disjoint. Applying a classical indistinguishably argument [13, Lemma 1], both $\rho'.\rho$ and $\rho.\rho'$ are admissible by Π. Thus, histories $h'.h = \mathfrak{F}(\rho'.\rho)$ and $h.h' = \mathfrak{F}(\rho.\rho')$ are acceptable.

Since updates are obstruction-free, history $h'.h.r_3(z_0).w_3(z_3).c_3$ is acceptable. Note U the sequence of steps following $\rho'.\rho$ with $\mathfrak{F}(U) = r_3(z_0).w_3(z_3).c_3$. Observe that by Lemma 1 $\rho'.\rho.U$ is indistinguishable from $\rho'.U.\rho$. Then consider history $\mathfrak{F}(\rho'.U.\rho)$. In this history, T_b is pending and the latest version of object z is z_3, As a consequence, because Π satisfies non-trivial SI, there exists an extension of $\rho'.U.\rho$ in which transaction T_b reads z_3. From the fact that queries are

wait-free and since $\rho'.\rho.U$ is indistinguishable from $\rho'.U.\rho$, we obtain that history $h_1 = h'.h.r_3(z_0).w_3(z_3).c_3.r_b(z_3).c_b$ is acceptable. We note U_1 the sequence of steps following $\rho'.\rho$ such that $\mathfrak{F}(U_1)$ equals $r_3(z_0).w_3(z_3).c_3.r_b(z_3).c_b$.

With a similar reasoning, history $h_2 = h'.h.r_1(x_0).w_1(x_1).c_1.r_d(x_1).c_d$ is acceptable. Note U_2 the sequence satisfying $\mathfrak{F}(U_2) = r_1(x_0).w_1(x_1).c_1.r_d(x_1).c_d$.

Executions $\rho'.\rho.U_1$ and $\rho'.\rho.U_2$ are both admissible. Because Π is GPR, only processes in $\mathit{Replicas}(y, z)$ (resp. $\mathit{Replicas}(x, u)$) make steps in U_1 (resp. U_2). By hypothesis, these two replica sets are disjoint. Applying again an indistinguishably argument, $\rho'.\rho.U_1.U_2$ is an execution of Π. Therefore, the history $\hat{h} = \mathfrak{F}(\rho'.\rho.U_1.U_2)$ is acceptable. In this history, relation $T_a \to T_b \to T_c \to T_d \to T_a$ holds. Thus, $\hat{h}$ does not belong to MON. Contradiction.

Our next theorem states that SCONSb is not attainable. Similarly to Attiya et al. [17], our proof builds an infinite execution in which a query T_a on two objects never terminates. We first define a finite execution during which we interleave between any two consecutive steps to execute T_a, a transaction updating one of the objects read by T_a. We show that during such an execution, transaction T_a does not terminate successfully. Then, we prove that asynchrony allows us to continuously extend such an execution, contradicting the fact that queries are wait-free.

Definition 8 (Flippable execution). *Consider two distinct objects x and y, a query T_a over both objects, and a set of updates $T_{j\in[\![1,m]\!]}$ accessing x if j is odd, and y otherwise. An execution $\rho = U_1V_2U_2 \dots V_mU_m$ where,*

- *transaction T_a reads in history $h = \mathfrak{F}(\rho)$ at least version x_1 of x,*
- *for any j in $[\![1, m]\!]$, U_j is the execution of transaction T_j by processes Q_j,*
- *for any j in $[\![2, m]\!]$, V_j are steps to execute T_a by processes P_j, and*
- *both $(Q_j \cap P_j = \varnothing) \oplus (P_j \cap Q_{j+1} = \varnothing)$ and $Q_j \cap Q_{j+1} = \varnothing$ hold,*

is called flippable.

Lemma 2. *Let ρ be an execution admissible by Π. If ρ is flippable and histories accepted by Π satisfy SCONSb, query T_a does not terminate.*

Proof. Let h be the history $\mathfrak{F}(\rho)$. In history h transaction T_j precedes transaction T_{j+1}, it follows that h is of the form $h = w_1(x_1).c_1. * .w_2(y_2).c_2. * \dots$, where each symbol $*$ corresponds to either no operation, or to some read operation by T_a on object x or y.

Because ρ is flippable, transaction T_a reads at least version x_1 of object x in h. For some odd natural $j \geq 1$, let x_j denote the version of object x read by T_a. Similarly, for some even natural l, let y_l be the version of y read by T_a. Assume that $j < l$ holds. Therefore, h is of the form $h = \dots w_j(x_j) \dots w_l(y_l) \dots$.

Note k the value $l + 1$, and consider the sequence of steps V_k made by P_k right after U_l to execute T_a. Applying the definition of a flippable execution, we know that (F1) $(Q_l \cap P_k = \varnothing) \oplus (P_k \cap Q_k = \varnothing)$, and (F2) $Q_l \cap Q_k = \varnothing$. Consider now the following cases:

(Case $Q_l \cap P_k = \varnothing$.) It follows that ρ is indistinguishable from the execution $\rho'' = \dots U_j \dots V_kU_lU_k \dots$. Then from fact F2, ρ is indistinguishable from execution $\rho' = \dots U_j \dots V_kU_kU_l \dots$.

(Case $P_k \cap Q_k = \varnothing$) With a similar reasoning, we obtain that ρ is indistinguishable from $\rho' = \ldots U_j \ldots U_k U_l V_k \ldots$.
(Case $P_k \cap (Q_l \cup Q_k) = \varnothing$.) This case reduces to any of the two above cases.
Note h' the history $\mathfrak{F}(\rho')$. Observe that since ρ' is indistinguishable from ρ, history h' is acceptable. In history h', $c_k <_{h'} c_l$ holds. Moreover, $c_j <_{h'} c_k$ holds by the assumption $j < l$ and the fact that k equals $l + 1$. Besides, operations $r_i(x_j)$, $r_i(y_l)$ and $w_k(x_k)$ all belong to h'. According to the definition of SCONSb, transaction T_a does not commit in h'. (The case $j > l$ follows a symmetrical reasoning to the case $l > j$ we considered previously.)

Theorem 3. *No asynchronous failure-free GPR system implements SCONSb.*

Proof. (By contradiction.) Consider two objects x and y such that *Replicas*(x) and *Replicas*(y) are disjoint. Assume a read-only transaction T_a that reads successively x then y. Below, we exhibit an execution admissible by Π during which transaction T_a never terminates. We build this execution as follows:

[Construction.] Consider some empty execution ρ. Repeat for all $i >= 1$: Let T_i be an update of x, if i is odd, and y otherwise. Start the execution of transaction T_i. Since no concurrent transaction is write-conflicting with T_i in ρ and updates are obstruction-free, there must exist an extension $\rho.U_i$ of ρ during which T_i commits. Assign to ρ the value of $\rho.U_i$. Execution ρ is flippable. Hence, Lemma 2 tells us that transaction T_a does not terminate in this execution. Consider the two following cases: (Case $i = 1$) Because Π satisfies non-trivial SI, there exists an extension ρ' of ρ in which transaction T_a reads at least version x_1 of object x. Notice that execution ρ' is of the form $U_1.V_2.s.\ldots$ where *(i)* all steps in V_2 are made by processes in *Replicas*(x), and *(ii)* s is the first step such that $\mathfrak{F}(U_1.V_2.s.) = r_1(x_0).w_1(x_1).c_1.r_a(x_1)$. Assign $U_1.V_2$ to ρ . (Case $i > 2$) Consider any step V_{i+1} to terminate T_a and append it to ρ.

Execution ρ is admissible by Π. Hence $\mathfrak{F}(\rho)$ is acceptable. However, in this history transaction T_a does not terminate. This contradicts the fact that queries are wait-free.

SCONSa disallows some real time orderings between operations accessing different objects. Our last theorem shows that this property cannot be maintained under GPR.

Theorem 4. *No asynchronous failure-free GPR system implements SCONSa.*

Proof. (By contradiction.) Consider two distinct objects x and y such that *Replicas*(x) and *Replicas*(y) are disjoint. Let T_1 be an update accessing y, and T_a be a query reading both objects.

Obviously, history $h = r_a(x_0)$ is acceptable. Note U_a a sequence of steps satisfying $U = \mathfrak{F}(r_a(x_0))$. Because Π supports obstruction-free updates, we know the existence of an extension $U_a.U_1$ of U_a such that $\mathfrak{F}(U_1) = r_1(y_0).w_1(y_1).c_1$. By Lemma 1, we observe that $U_a.U_1$ is indistinguishable from $U_1.U_a$. Then, since Π satisfies non-trivial SI and read-only transactions are wait-free, there must exist an extension $U_1.U_a.V_a$ of $U_1.U_a$ admissible by Π and such that $\mathfrak{F}(V_a) = r_a(y_1).c_a$. Finally, since $U_a.U_1$ is indistinguishable from $U_1.U_a$ and $U_1.U_a.V_a$ is admissible, $U_a.U_1.V_a$ is admissible too. The history $\mathfrak{F}(U_a.U_1.V_a)$ is not in SCONSa. Contradiction.

As a consequence of the above, no asynchronous system, even if it is failure-free, can support both GPR and SI. In particular, even if the system is augmented with failure detectors [18], a common approach to model partial synchrony, SI cannot be implemented under GPR. This fact strongly hinders the usage of SI at large scale. In the following section, we further discuss implications of this result.

5 Discussion

A straightforward corollary of any of the theorems we proved in Section 4 is that neither strict serializability [19], nor opacity [20] is attainable under GPR. In the case of opacity, this answers negatively to a problem posed by Peluso et al. [21].

The classical (non-genuine) solution for building strictly consistent monotonic snapshots is to use total order broadcast (e.g., [2, 3]).

When a transaction declares objects it accesses in advance, a GPR system can install a snapshot just after the start of the transaction. As a consequence, such an assumption sidesteps our impossibility result.

A transactional system Π is *permissive* with respect to a consistency criterion $\mathcal{C}$ when every history $h \in \mathcal{C}$ is acceptable by Π. Permissiveness [22] measures the optimal amount of concurrency a system allows. If we consider again histories h_1 and h_2 in the proof of Theorem 2, we observe that both histories are serializable. Hence, every system permissive with respect to SER accepts both histories. By relying on the very same argument as the one we exhibit to close the proof of Theorem 2, we conclude that no transactional system is both GPR and permissive with respect to SER. For instance, none of the systems presented in [10, 23] accept history $h_{10} = r_1(x_0).w_1(x_1).c_1.r_2(x_0).r_2(y_0).w_2(y_2).c_2$.

Recent distributed transactional systems (e.g., [9, 24]) support weaker consistency criteria than SI or SER. In particular, Walter [9] supports Parallel Snapshot Isolation (PSI). PSI is weaker than SI, and allows snapshots to be non-monotonic. But, it still requires SCONSa to be ensured. Sovran et al. justify the use of PSI in Walter by the fact that SI is too expensive in a geographically distributed environment [9, page 4]. Our impossibility result establishes that, in order to scale, a transactional system needs supporting both non-monotonic *and* non-strictly consistent snapshots.

6 Conclusion

Partial replication and genuineness are two key factors of scalability in replicated systems. This paper shows that ensuring snapshot isolation (SI) in a genuine partial replication system is impossible. To state this impossibility result, we prove that SI is decomposable into a set of simpler properties. We show that two of these properties, namely snapshot monotonicity and strictly consistent snapshots cannot be ensured. As a corollary of our results, a GPR system cannot support neither strict serializability, nor opacity.

Bibliography

[1] K. Daudjee *et al.*, "Lazy database replication with snapshot isolation," in *VLDB*, Sep. 2006, pp. 715–726.

[2] D. Serrano *et al.*, "Boosting Database Replication Scalability through Partial Replication and 1-Copy-Snapshot-Isolation," in *PRDC*, Dec. 2007, pp. 290–297.

[3] J. E. Armendáriz-Iñigo *et al.*, "SIPRe: a partial database replication protocol with SI replicas," in *SAC*, Mar. 2008, pp. 2181–2185.

[4] A. Bieniusa *et al.*, "Consistency in hindsight: A fully decentralized stm algorithm," in *IPDPS*, Apr. 2010, pp. 1–12.

[5] T. Riegel *et al.*, "Snapshot isolation for software transactional memory," in *TRANSACT*, Jun. 2006.

[6] A. Adya, "Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions," Ph.D., MIT, Mar. 1999.

[7] S. Elnikety *et al.*, "Database Replication Using Generalized Snapshot Isolation," in *SRDS*, Oct. 2005, pp. 73–84.

[8] H. Berenson *et al.*, "A critique of ansi sql isolation levels," in *SIGMOD*, May 1995, pp. 1–10.

[9] Y. Sovran *et al.*, "Transactional storage for geo-replicated systems," in *SOSP*, Oct. 2011, pp. 385–400.

[10] N. Schiper *et al.*, "P-store: Genuine partial replication in wide area networks," in *SRDS*, Nov. 2010, pp. 214–224.

[11] M. Saeida Ardekani *et al.*, "Non-Monotonic Snapshot Isolation," INRIA, Tech. Rep. RR-7805, Oct. 2012.

[12] P. Bernstein *et al.*, *Concurrency Control and Recovery in Database Systems*. Addison Wesley Publishing Company, 1987.

[13] M. J. Fischer *et al.*, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985.

[14] M. Abadi *et al.*, "The existence of refinement mappings," *Theory Computer Science*, vol. 82, pp. 253–284, May 1991.

[15] A. Chan *et al.*, "Implementing Distributed Read-Only Transactions," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 2, pp. 205–212, Feb. 1985.

[16] H. Garcia-Molina *et al.*, "Read-only transactions in a distributed database," *ACM Trans. Database Syst.*, vol. 7, no. 2, pp. 209–234, Jun. 1982.

[17] H. Attiya *et al.*, "Inherent limitations on disjoint-access parallel implementations of transactional memory," in *SPAA*, 2009, pp. 69–78.

[18] T. D. Chandra *et al.*, "Unreliable failure detectors for reliable distributed systems," *Journal of the ACM*, vol. 43, no. 2, pp. 225–267, 1996.

[19] C. H. Papadimitriou, "The serializability of concurrent database updates," *Journal of the ACM*, vol. 26, no. 4, pp. 631–653, Oct. 1979.

[20] R. Guerraoui *et al.*, "On the correctness of transactional memory," in *PPoPP*, 2008, pp. 175–184.

[21] S. Peluso *et al.*, "Genuine replication, opacity and wait-free read transactions: can a stm get them all?" in *WTTM*, Madeira, Portugal, Jul. 2012.

[22] R. Guerraoui *et al.*, "Permissiveness in transactional memories," in *DISC*, Sep. 2008, pp. 305–319.

[23] D. Sciascia *et al.*, "Scalable deferred update replication," in *DSN*, Jun. 2012, pp. 1–12.

[24] S. Peluso *et al.*, "When scalability meets consistency: Genuine multiversion update-serializable partial data replication," in *ICDCS*, Jun. 2012, pp. 455–465.