A Consistent Understanding of Consistency

Subhajit Sidhanta
INESC-ID/IST

Ricardo Dias
SUSE Linux GmbH

Rodrigo Rodrigues
INESC-ID/IST

ABSTRACT

We propose a specification language named *ConSpec*, which enables the formalization of different consistency semantics that a storage system may provide, using a simple and uniform syntax, which is independent of the design and implementation of the target storage system. ConSpec addresses the recent profusion of consistency definitions, which are often accompanied by either informal definitions, or definitions that are tied to implementation-level details. To enable a simple and uniform description of various existing proposals, ConSpec builds on recent proposals that view weak consistency definitions as partial orderings between operations forming a visibility graph, which reduces the consistency definition to a set of restrictions on those partial orders, described using Linear Temporal Logic (LTL). We use ConSpec to revisit several existing models in light of a common way to define and compare them. Furthermore, the use of LTL enabled us to leverage existing automatic checkers to build a tool for validiting whether a given trace of the execution of a system meets a certain consistency semantics. Finally, using our generic definitions and the insights from analyzing different models, we restate the CAP theorem to precisely define the class of consistency definitions that can and cannot be implemented in a highly-available, partition-tolerant way, in contrast with the original definition which only considered linearizability.

1. INTRODUCTION

The development of modern Internet-based applications and services requires application developers to take into account the semantics of the underlying storage system, to ensure correctness and acceptable performance. In fact, there is evidence that the lack of a good comprehension of these semantics can lead programmers to unintendedly break the application semantics [2]. However, the task of understanding and comparing the semantics provided by storage systems, normally encapsulated in a consistency definition (or an isolation definition, in case of a transactional interface), is made difficult by several factors. First, the current set of possible consistency models is not only large but also expanding, as new storage systems often coin new terms for the consistency semantics that they offer [7, 15, 13]. Second, existing consistency semantics have definitions that are often imprecise and/or are tied to implementation specifics (such as versioning [1] or the existence of replicas at different sites [14]). Third, even in the cases for which precise definitions exist, it is difficult to compare different definitions, since they are written using different formalisms and using community-specific terms or notations.

To address these issues, we propose a specification language called *ConSpec*, a generic definition for consistency models, which can be easily parameterized to obtain precise definitions of a variety of semantics. ConSpec builds on the observations made by recent proposals that it is possible to define weak consistency semantics in terms of partial orders over the set of operations that were executed in the system, which comprise a visibility graph [14, 9, 14]. This allows us to reduce the configurable part of the consistency definition to a set of restrictions, written in Linear Temporal Logic (LTL), to a generic partial order. Intuitively, this allows us to express a set of rules that specify the allowed order in which results of operations can be visible to a client application.

With this framework in place, we were able to express several existing consistency definitions in a common language, and compare them in a precise way, thereby defining a hierarchy of consistency models. In addition, our variants of the definitions for these consistency models were linked to their original definitions by proving their equivalence.

Another contribution of this paper is that we built a tool, which is available online, to check whether a given trace violates a certain consistency model. This tool leverages our LTL based syntax, which allowed us to reuse existing automatic checkers and adapt them to our constructs in a direct way.

Our final contribution is that we were able to restate in much broader terms the CAP theorem [4, 8], whose original proof used linearizability as synonym for the strong consistency captured by the "C" property. In our new formulation of this theorem, we are able to define necessary and sufficient conditions for a given consistency model to be *CAP-strong*, i.e., bound by the impossibility of being implemented in a highly available, partition-tolerant way.

The remainder of this paper is organized as follows. Section 2 presents the system model, and the basic definitions and terminology used in the paper. Section 3 presents the syntax of ConSpec, and the general format of ConSpec specifications. Section 4 lists down the specifications of various consistency models and isolation levels, expressed as ConSpec expressions. Section 5 discusses how the lattice of consistency models can be hierarchically organized. Section 6 presents a restatement of the CAP theorem, and proves the same. Section 7 analyzes the lattice of consistency models in terms of the restated CAP theorem. Section 8 proves the equivalence of the ConSpec specifications with the state-of-the-art

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

definitions, by deriving the ConSpec specification for each of the consistency models and isolation levels. Section 9 briefly presents the design of a prototype automated verification tool that can validate a session trace with respect to a ConSpec specification. Finally Section 10 illustrates the equivalence between the ConSpec expressions and the dependency graphs used by the state-of-the-art definitions. We have also include a discussion of examples of session traces that cause violations of ConSpec specifications in the appendix.

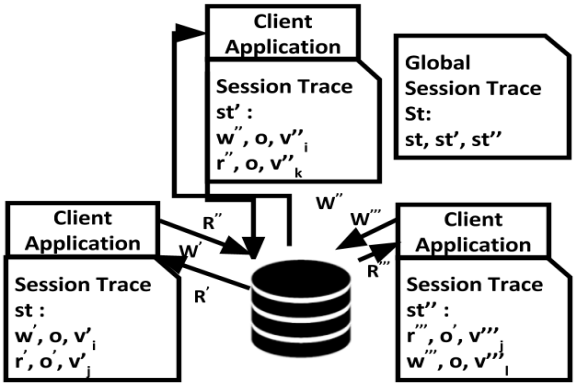

Figure 1: System Model

2. BASIC DEFINITIONS

We assume a set of clients that access the storage system by invoking operations. A very large class of consistency definitions (namely those describing the behaviour of loads and stores on computer hardware) assume that these operations are partitioned into two classes, namely read operations that do not affect the result of subsequent operations and write operations that do. Given that these consistency definitions are based on the existence of these two classes of operations, we will assume that the space of possible operations is partitioned in this way, but we develop our definitions such that they can be generalized in a straightforward way to other models. (For example, databases make a similar distinction between queries and updates, and the state machine replication model distinguishes between read-only and read-write requests.)

Similarly, many consistency definitions reason about an interface that exposes the existence of multiple objects (e.g., different memory addresses seen by a CPU or different keys in a key-value store). As such, we assume that the interface allows the programmer to specify objects that are associated with each operation, where we define object o_j to comprise: 1) a unique name, given by the string o_j, and 2) a value v_j, which can be of any datatype, such as float, double, string, etc.

A *session trace* st is a sequence of operations executed by the same client, ordered by the real-time instant when they were invoked. In this paper we assume that a client only invokes an operation after the preceding operation has returned its response value.

Under the above assumption that the set of possible operations is partitioned into two classes, we define that a write operation op, which writes a value v to an object o, is denoted as $w(o, v)$. Similarly, a read operation op on object o that outputs a value v is denoted $r(o)v$. This allows us to define a session trace st as a sequence of tuples $\langle w, v^{'} \rangle$ or $\langle r, v^{'} \rangle$.

The *global session trace* St denotes the set of all session traces in a given execution of the system. Figure 1 illustrates the above system model.

Our notation defines special purpose LTL operators as shorthands to longer expressions. These correspond to scope operators that specify the context within which the expression is restricted. For example, we define a special-purpose operator X^o, which is derived from the LTL modal operator "next", i.e., X. The superscript o in the above operator is used to specify that the operator defines the LTL relationship "next" among only the operations performed on the object o. E.g., an expression wX^or denotes that among operations performed on the object o, the execution of the operation $r(o)v_m$ immediately follows the execution operation $w(o, v_l)$. Finally, we assume the system has a sequential specification, which is a correctness condition, corresponding to the output of the operations in centralized system that executes operation in sequence, one at a time. In the case of the sequential specification of a system whose interface are read and write operations, the read operation to object o must output the value associated with the most recent write operation to the same object o.

3. CONSPEC

In this section we present our generic ConSpec definition, which can be parameterized to obtain specifications for commonly used consistency models and isolation levels.

ConSpec builds on recent proposals for consistency definitions that treat a subset of the operations in a given trace differently (e.g., operations labeled as being "strongly consistent"), since they only enforce visibility among those specific operations [14, 9, 14]. We can similarly see different consistency definitions as enforcing different visibility relationships only among a subset of the system operations, often depending on their types (e.g., reads versus writes).

As such, the generic definition of ConSpec requires the existence of a partial order that intuitively forms a "visibility graph", i.e., the output of each operation must reflect the effects of the operations that precede it according to that partial order. This allows us to see different consistency or isolation models as imposing different restrictive conditions on this precedence. Such a restrictive condition is then expressed as an LTL expression E^s.

DEFINITION 1. *Generalized form of ConSpec: Given a global session trace St, we say that St satisfies a consistency model $\mathcal{C}$ if there exists a partial order $(\mathcal{O}, \preccurlyeq)$ over the set $\mathcal{O}$ comprising operations present in all session traces in St, where $\mathcal{O} = \bigcup_{st \in St} \{o^i \mid o^i \in st\}$, such that 1) for every operation op^i in St, its output is equal to the one obtain by executing the sequential specification of a linear extension of the operations preceding op^i in $\preccurlyeq$, and 2) $(\mathcal{O}, \preccurlyeq)$ obeys E^s, which is an LTL expression restricting $(\mathcal{O}, \preccurlyeq)$.*

Condition 1, when applied to a system whose interface consists only of read and write, translates to a requirement that every read operation in St must return the value of the most recent write according to $\preccurlyeq$. Condition 2 can be expressed as $E^s \models (\mathcal{O}, \preccurlyeq)$, where E^s is the ConSpec parameterizattion for each consistency model $\mathcal{C}$, and $\models$ is the satisfies operator.

4. SPECIFYING EXISTING MODELS

In this section, we present the ConSpec specifications for several common consistency models and isolation levels documented in the literature [6, 18, 5]. Subsequently, in Section 8, we prove the equivalence between these definitions and their original counterparts.

4.1 Specifications for Consistency Models

We start by specifying a series of session guarantees, which were originally proposed by Terry et al. in the context of a mobile com-

puting storage system called Bayou [18]. These are four guarantees that apply to individual sessions, allowing applications to see a view of the storage system that is consistent with their previous operations. Their original definition is implementation-dependent, since it is stated, for instance, in terms of the order in which writes are applied at various server replicas. Subsequently there were authors who wrote implementation-independent definition for these session guarantees [6, 5].

We start with the Read Your Writes (RYW) session guarantee, which informally precludes a read operation r from reading a value

stated as follows. Let us consider that a client application performs a write operation followed by a read operation on an object o, observed in any given session trace, obtained by executing the above client application. We use serialization of a partial execution S_p, specified in Definition 12.1, to denote a legal serialization of all operations comprising a given client application, as well as all write operations executed from all other concurrent client applications. Following the format of Definition 1, the above RYW definitions are reduced to LTL-like expression, as follows.

$$\mathcal{E}^s = \forall st \in St, W^{i'}, R^{j'} \in st\big(\Big(W^{i'} F R^{j'} \in st\Big) \rightarrow \Big(W^{i'} \preccurlyeq R^{j'}\Big)\big), \tag{1}$$

where the partial order $(\mathcal{O}, \preccurlyeq)$ restricts the order among conflicts pair of writes and reads in an equivalent serialization of a partial execution S_p, defined in Definition 12.1.

Read/Session Monotonic (MR) specifies: read operations on a storage system object invoked from a client application or a session must always return results in an increasing order of recency as per their mutual order in the session trace. MR is formally represented by the following expression.

$$\mathcal{E}^s = \forall st \in St, R^{i'}, R^{j'} \in st\big(R^{i'} F R^{j'} \in st \rightarrow \Big(R^{i'} \preccurlyeq R^{j'}\Big)\big), \tag{2}$$

where $(\mathcal{O}, \preccurlyeq)$ constrains the order among conflicting reads in an equivalent S_p.

Causal Consistency specifies: results of operations (in a given client application) that are bound by a causal relationship [3] must be visible in the session trace according to their mutual execution order with respect to an observed session trace. The ConSpec expression for Causal Consistency is given below.

$$\begin{aligned} E^s = \forall st \in St, Op^{i'}, Op^{j'} \in st \\ \big(\Big(Op^{i'} F Op^{j'} \in st\Big) \vee \big(\Big(Op^{i'} = W^{i'}\Big) \wedge \\ \Big(Op^{j'} = R^{j'}\Big) \wedge (v_i = v_j)\big) \rightarrow \Big(Op^{i'} \preccurlyeq Op^{j'}\Big)\big), \end{aligned} \tag{3}$$

where the partial order $(\mathcal{O}, \preccurlyeq)$ enforces order among conflicting operations in an equivalent S_p to follow the causal order.

Strict Serializability [11] is a consistency model (not to be confused with the isolation level which goes by the same name) adds the restriction of linearizability to serializability. In addition to the existential quantification over an equivalent legal serialization over the global session trace, it further specifies: any session trace must observe latest result of all operations in a client application. Using the notion of serialization of a client, given in Equation 12.1, Strict Serializability is expressed as follows.

$$E^s = \forall st \in St, Op^{i'}, Op^{j'} \in st\big(Op^i X^o Op^{j'} \in st \rightarrow \Big(Op^{i'} \preccurlyeq Op^{j'}\Big)\big), \tag{4}$$

where the partial order $(\mathcal{O}, \preccurlyeq)$ restricts an equivalent serialization of a client S_c, defined in Definition 12.1, to respect the order of sucessive operations in a given session trace.

Write Follows Read (WFR) specifies: a write operation, that follows a read operation on a common storage system object, must happen while the object contains a value that is at least as old as the one that was read by the preceding read operation. WFR is expressed as follow.

$$E^s = \forall st \in St, R^{i'}, W^{j'} \in st\big(\Big(R^{i'} F W^{j'} \in st\Big) \rightarrow \Big(R^{i'} \preccurlyeq W^{j'}\Big)\big), \tag{5}$$

where the partial order $(\mathcal{O}, \preccurlyeq)$ constrains the order among conflicting pairs of reads and writes in an equivalent S_p.

Monotonic Writes (MW) model is specified as: successive write operations invoked from a client application on a storage system object must occur according to their mutual order in the session trace. MW is expressed as follows, in terms of a legal serialization for the given execution, specified in Definition .

$$E^s = \forall st \in St, W^{i'}, W^{j'} \in st\big(W^{i'} F W^{j'} \in st \rightarrow \Big(W^{i'} \preccurlyeq W^{j'}\Big)\big), \tag{6}$$

where the partial order $(\mathcal{O}, \preccurlyeq)$ restricts the order among conflicting writes in an equivalent S_p.

Owen et al. [17] specifies Total Store Order (TSO) as: read operations performed on the same storage system object by client applications can not return a subsequent (or newer) value unless the result of a given write is observed by all client applications, i.e., observed in all session traces. A session trace for a client application can not view results of a sequence of write operations in an order which differs from the invocation order. Following Definition 1, TSO is expressed as follows.
$E^s = \forall x, st \in St \ \nexists op^i(x), op^j(x) \in \mathcal{O}$
$\Big(Op^{i'}(x) F Op^{j'}(x) \in st\Big) \rightarrow \Big(Op^{j''}(x) F Op^{i''}(x) \in st\Big)$,
which can be rewritten as follows.

$$\forall x, \Big(Op^{i'}(x) F Op^{j'}(x) \in St\Big) \rightarrow \Big(Op^{i'}(x) \preccurlyeq (x) Op^{j'}(x)\Big), \tag{7}$$

where $\preccurlyeq (x)$ is a partial order over a given object x with respect to a given global session trace St.

Processor Consistency specifies: operations in a client application must execute according to the invocation order with respect to a given processor (client); i.e., result of a preceding operation invoked from a given clientr must be observed in all processors before result of a succeeding operation can be observed in any processor (or client application). Reads against write operations should view the results of the writes according to the invocation order of these writes in a client application. Thus, Processor Consistency can be expressed as
$E^s = \forall x, y, st, st' \in St \ \nexists op^i(x), op^j(y) \in \mathcal{O}$
$\Big(Op^{i'}(x) F Op^{j'}(y) \in st\Big) \rightarrow \Big(Op^{j''}(y) F Op^{i''}(x) \in st'\Big)$,

which can be rewritten as follows.

$$\forall x, y, st \in St \left(Op^{i^{'}}(x) F Op^{j^{'}}(y) \in st \right) \rightarrow \left(Op^{i^{'}}(x) \preccurlyeq Op^{j^{'}}(y) \right), \tag{8}$$

where $\preccurlyeq$ is a partial order over operations across all data objects with respect to a given global session trace St.

4.2 Specification of Isolation Levels

For specifying isolation levels, we consider only the isolation levels defined by Adya et al. [1]. We show the equivalence between the graphical representation of the dependency relations used by Adya et al. and the LTL specifications of ConSpec in Section 10. The isolation level PL-1 mandates that write operations in a transaction must be executed such that write-write dependency cycles among transactions are proscribed. Following Definition 1, PL-1 is specified as follow.

$$\begin{aligned} E^s = \forall st \in St, tx, ty \in St, x, y, \\ W^i(x)^{'}_{tx}, W^j(y)^{'}_{tx}, W^k(x)^{'}_{ty}, W^l(y)^{'}_{ty} \in St \\ \left(\left(\nexists R^m(x)^{'}_{tx}, R^n(y)^{'}_{ty} \in St \right) \right. \\ \left. \left(\left(v_n^{'} = v_l^{'} \right) \wedge \left(v_m^{'} = v_i^{'} \right) \right) \vee \left(\left(v_n^{'} = v_j^{'} \right) \wedge \left(v_m^{'} = v_k^{'} \right) \right) \right) \end{aligned} \tag{9}$$

where tx and ty are consecutive transactions.

The isolation level PL-2 states that operations from a transaction can not read a value that was not the final value that was committed by a transaction. Further, a transaction cannot read values written by aborted transactions. Additionally, operations in a transaction must be executed in an order that proscribes write-write and write-read dependency cycles [1] among transactions. PL-2 is given as

$$\begin{aligned} E^r = \forall st, st^{'} \in St, tx, ty \in St \\ \left(\left(\left(\nexists a_{tx} \in st \left(W^i(x)^{'}_{tx} F R^j(x)^{'}_{ty} \in st \right) \right. \right. \right. \\ \left. \left. \left. \wedge \nexists a_{ty} \in st \left(W^k(y)^{'}_{ty} F R^l(y)^{'}_{tx} \in st \right) \right) \wedge \left(\nexists W^i(x)^{'}_{tx}, W^j(x)^{'}_{tx}, \right. \right. \right. \\ \left. \left. \left. R^k(x)^{'}_{ty} \in st \right) \left(\left(W^i(x)^{'}_{tx} F W^j(x)^{'}_{tx} F c_{tx} \in st \right) \right. \right. \right. \\ \left. \left. \left. \wedge \left(v_k^{'} = v_i^{'} \right) \right) \wedge \nexists \left(W^i(x)^{'}_{ty}, W^j(x)^{'}_{ty}, R^k(x)^{'}_{tx} \in st \right) \right. \\ \left. \left(\left(W^i(x)^{'}_{ty} F W^j(x)^{'}_{ty} F c_{ty} \in st \right) \wedge \left(v_k^{'} = v_i^{'} \right) \right) \wedge \right. \\ \left(\nexists W^i(x)^{'}_{tx}, W^j(y)^{'}_{tx}, W^k(x)^{'}_{ty}, W^l(y)^{'}_{ty}, R^m(x)^{'}_{tx}, R^n(y)^{'}_{ty} \in St \right. \\ \left. \left(\left(\left(v_n^{'} = v_l^{'} \right) \wedge \left(v_m^{'} = v_i^{'} \right) \right) \vee \left(\left(v_n^{'} = v_j^{'} \right) \wedge \left(v_m^{'} = v_k^{'} \right) \right) \right) \right) \end{aligned} \tag{10}$$

where c_{tx} and c_{ty} are the commit statements for the transactions tx and ty, respectively. F is the traditional LTL next operator.

The isolation level PL-3 specifies that, on top of PL-2, read-write dependency cycles [1] must also be proscribed among transactions. PL-3 is given as

$$\begin{aligned} E^r = C = \forall st, st^{'} \in St, tx, ty \in St \\ \left(\left(\left(\nexists a_{tx} \in st \left(W^i(x)^{'}_{tx} F R^j(x)^{'}_{ty} \in st \right) \right. \right. \right. \\ \left. \left. \left. \wedge \nexists a_{ty} \in st \left(W^k(y)^{'}_{ty} F R^l(y)^{'}_{tx} \in st \right) \right) \wedge \left(\nexists W^i(x)^{'}_{tx}, W^j(x)^{'}_{tx}, \right. \right. \right. \\ \left. \left. \left. R^k(x)^{'}_{ty} \in st \right) \left(\left(W^i(x)^{'}_{tx} F W^j(x)^{'}_{tx} F c_{tx} \in st \right) \right. \right. \right. \\ \left. \left. \left. \wedge \left(v_k^{'} = v_i^{'} \right) \right) \wedge \nexists \left(W^i(x)^{'}_{ty}, W^j(x)^{'}_{ty}, R^k(x)^{'}_{tx} \in st \right) \right. \\ \left. \left(\left(W^i(x)^{'}_{ty} F W^j(x)^{'}_{ty} F c_{ty} \in st \right) \wedge \left(v_k^{'} = v_i^{'} \right) \right) \wedge \right. \\ \left(\nexists W^i(x)^{'}_{tx}, W^j(y)^{'}_{tx}, W^k(x)^{'}_{ty}, W^l(y)^{'}_{ty}, R^m(x)^{'}_{tx}, R^n(y)^{'}_{ty} \in St \right. \\ \left. \left(\left(\left(v_n^{'} = v_l^{'} \right) \wedge \left(v_m^{'} = v_i^{'} \right) \right) \vee \left(\left(v_n^{'} = v_j^{'} \right) \wedge \left(v_m^{'} = v_k^{'} \right) \right) \right) \wedge \right. \\ \left(\nexists R^i(x)^{'}_{tx}, W^j(x)^{'}_{ty}, R^k(x)^{'}_{ty}, R^l(y)^{'}_{ty}, W^m(y)^{'}_{tx}, R^n(y)^{'}_{tx} \in St \right) \\ \left(\left(R^i(x)^{'}_{tx} F W^j(x)^{'}_{ty} F R^k(x)^{'}_{ty} \in St \right) \wedge \right. \\ \left. \left(R^l(y)^{'}_{ty} F W^m(y)^{'}_{tx} F R^n(y)^{'}_{tx} \in St \right) \right) \\ \left(\left(v_k^{'} = v_j^{'} \right) \wedge \left(v_n^{'} = v_m^{'} \right) \right) \left. \right) \left. \right) \end{aligned} \tag{11}$$

The isolation level PL-2.99 specifies that, on top of PL-2, read-write item-dependency cycles must be proscribed among transactions. PL-2.99 is given as

$$\begin{aligned} E^r = C = \forall st, st^{'} \in St, tx, ty \in St \\ \left(\left(\left(\nexists a_{tx} \in st \left(W^i(x)^{'}_{tx} F R^j(x)^{'}_{ty} \in st \right) \right. \right. \right. \\ \left. \left. \left. \wedge \nexists a_{ty} \in st \left(W^k(y)^{'}_{ty} F R^l(y)^{'}_{tx} \in st \right) \right) \wedge \left(\nexists W^i(x)^{'}_{tx}, W^j(x)^{'}_{tx}, \right. \right. \right. \\ \left. \left. \left. R^k(x)^{'}_{ty} \in st \right) \left(\left(W^i(x)^{'}_{tx} F W^j(x)^{'}_{tx} F c_{tx} \in st \right) \right. \right. \right. \\ \left. \left. \left. \wedge \left(v_k^{'} = v_i^{'} \right) \right) \wedge \nexists \left(W^i(x)^{'}_{ty}, W^j(x)^{'}_{ty}, R^k(x)^{'}_{tx} \in st \right) \right. \\ \left. \left(\left(W^i(x)^{'}_{ty} F W^j(x)^{'}_{ty} F c_{ty} \in st \right) \wedge \left(v_k^{'} = v_i^{'} \right) \right) \wedge \right. \\ \left(\nexists W^i(x)^{'}_{tx}, W^j(y)^{'}_{tx}, W^k(x)^{'}_{ty}, W^l(y)^{'}_{ty}, R^m(x)^{'}_{tx}, R^n(y)^{'}_{ty} \in St \right. \\ \left. \left(\left(\left(v_n^{'} = v_l^{'} \right) \wedge \left(v_m^{'} = v_i^{'} \right) \right) \vee \left(\left(v_n^{'} = v_j^{'} \right) \wedge \left(v_m^{'} = v_k^{'} \right) \right) \right) \wedge \right. \\ \left(\nexists R^i(x)^{'}_{tx}, W^j(x)^{'}_{ty}, R^k(x)^{'}_{ty}, R^l(y)^{'}_{ty}, W^m(y)^{'}_{tx}, R^n(y)^{'}_{tx} \in St \right) \\ \left(\left(R^i(x)^{'}_{tx} F W^j(x)^{'}_{ty} F R^k(x)^{'}_{ty} \in St \right) \wedge \right. \\ \left. \left(R^l(y)^{'}_{ty} F W^m(y)^{'}_{tx} F R^n(y)^{'}_{tx} \in St \right) \right) \\ \left(\left(v_k^{'} = v_j^{'} \right) \wedge \left(v_n^{'} = v_m^{'} \right) \right) \left. \right) \left. \right) \end{aligned} \tag{12}$$

5. A HIERARCHY OF CONSPEC SPECIFICATIONS

Definitions of consistency models are specified using ConSpec in the form of LTL properties. Hence, ConSpec specifications can be classified using the same basic principles by which LTL properties are categorised [16]. There are four broad categories of LTL properties: safety, guarantee, recurrence, and persistence. Consider that ϕ is a given LTL property, which may be a conjunction or disjunction of multiple LTL properties. A safety property is expressed in a generic format $\forall \phi$, whereas a guarantee property is expressed as $\exists \phi$; recurrence and persistence properties are expressed as $R\phi$ and $P\phi$, respectively. The specification $\mathcal{E}^s$ for each of the consistency models RYW, MR, Causal, WFR, and MW can be expressed

in the form $\forall p \rightarrow \exists q$, which can be rewritten as $\forall Fp \rightarrow Fq$. This form of LTL specification is called the conditional guarantee form of LTL property. The specifications for the TSO and PC consistency models can be expressed in the form $\forall p_1 \circ p_2 \circ^{'} ... \circ^{''} p_k$, where $\circ, \circ^{'}, ..., \circ^{''}$ are conjunction or disjunction operators of first order logic over LTL constraints $p_1, p_2, ..., p_k$. The above expression can be transformed into the form $\forall p$, which is called the safety form of LTL property. Thus, the ConSpec specifications can be broadly classified as conditional guarantee and safety properties.

Now, let us further analyze the consistency models belonging to the conditional guarantee form. The specifications $\mathcal{E}^s$ of RYW, MR, Causal, WFR, and MW can be expressed in the form $\mathcal{E}^s = A \rightarrow B$ (refer to Defintion 1), where the precondition A is specified as $A = Op^{i^{'}} F Op^{j^{'}} \in st$. The precondition A in the specification $\mathcal{E}^s$ for Causal consistency (refer to Equation 3) imposes an LTL constraint on any pair of operations op^i and op^j comprised in a given session trace. Since the above operations op^i and op^j can be both read and write operations, precondition A for Causal consistency is more restrictive than others. Thus, in turn, the set of session traces that satisfies the precondition A for Causal consistency is smaller in size than the set that satisfies RYW, MR, WFR, and MW. However, the postcondition B for RYW, MR, Causal WFR, MW uniformly defines an existential quantification on the serialization of a partial execution for a given client. The strength (degree of the strength) of a consistency model can be quantified by the size of the set of session traces that consistency model; smaller size implies more restrictive condition, which, in turn, implies stronger the consistency model. Hence, Causal consistency is stronger than the rest of the above consistency models, namely RYW, MR, Causal WFR, and MW. The precondition A for RYW (refer to Equation 4.1) comprises an LTL constraint $W^{i^{'}} F R^{j^{'}}$ on successive write and read operations $w^{i^{'}}$ and $r^{j^{'}}$. On the other hand, WFR (refer to Equation 5) comprises an LTL constraint $R^{i^{'}} F W^{j^{'}}$ on successive read and write operations $r^{i^{'}}$ and $w^{j^{'}}$. However the writes $w^{j^{'}}$ have no meaning in the observed session traces, unless viewed in company of a succeeding read $r^{k^{'}}$. Thus, the precondition A for WFR can be expressed as $W^{i^{'}} F R^{j^{'}} F R^{k^{'}}$, making it more restrictive; thus the set of traces which satisfy WFR is smaller than that of RYW and MR. Hence, WFR is stronger than the precondition $A = R^{i^{'}} F R^{j^{'}}$ for MR, as well as the precondition $A = W^{i^{'}} F R^{j^{'}}$. Similarly, the precondition $A = W^{i^{'}} F W^{j^{'}}$ for MW can be expressed as $A = W^{i^{'}} F W^{j^{'}} F R^{k^{'}}$, making it also stronger than RYW and MR. Strict serializability model, on the other hand, specifies the precondition A as an LTL constraint on any pair of operations; particularly, it specifies A as $A = Op^{i^{'}} X Op^{j^{'}}$. The LTL operator next X implies that the set of session traces satisfies the above precondition is even smaller than the set that satisfies Causal consistency. Hence Strict Serializability is stronger than the rest of the consistency models of the conditional guarantee form.

6. REWRITING CAP THEOREM IN TERMS OF CONSPEC

CAP theorem [4, 8] states that it is impossible to simultaneously achieve atomic consistency (linearizability), availability (i.e., the property that there must be a response against every request), and partition tolerance (i.e., tolerance of network partitions) in a distributed system. However, the original statement of CAP is restricted to linearizability, and does not address the wide array of consistency models supported by modern storage systems. Indeed modern storage systems relax consistency guarantees to favour availability and partition tolerance according to the use case and system requirements. The various degrees of relaxation gives rise to a lattice of consistency models, ranging from RYW to Strict Serializability. In this section, we rewrite the CAP theorem to take into account this wide array of consistency models. Here, we analyze the specifications of consistency models, and derive the condition that makes a particular consistency model CAP-strong, i.e., strong enough to be bounded by the constraints of CAP theorem, as follows. First, we state and prove the condition for a consistency model to be CAP-strong, and then we state and prove the opposite case, i.e., we prove the condition for a consistency model to be CAP-weak.

THEOREM 1 (EXTENDED CAP THEOREM). *It is impossible for a storage system to satisfy a consistency model that enforces restrictions on order among the results of operations performed by different clients executing on the system, while simultaneously providing high availability and partition tolerance.*

To elaborate, if the specification E^s (following the generic specification format given in Definition 1) of a consistency model imposes a partial order $\preccurlyeq$ among results observed by pairs of operations performed by different clients executing on underlying storage system (i.e., among operations from different clients comprised in a global execution represented by a global session trace St), that consistency model is regarded as a CAP-strong consistency model. On the other hand, a consistency model that imposes a partial order $\preccurlyeq$ only among results observed by pairs of operations performed by the same client comprised in a global execution represented by a global session trace St, is regarded as CAP-weak. It directly follows from Theorem 6 that the CAP-weak condition, in turn, implies that the specified partial order $\preccurlyeq$ as per Definition 1 must be a partial order on only operations from the same client , comprised in St. In other words, for a CAP-weak consistency model, Condition 1 in Definition 1 can be rewritten as the following condition. Thus, the condition $\forall Op^{i^{'}}, Op^{j^{'}} \in St : Op^{i^{'}} \preccurlyeq Op^{j^{'}} \vee Op^{j^{'}} \preccurlyeq Op^{i^{'}}$ must hold for operations $op^i \in st$ and $op^j \in st^{'}$ comprised in separate session traces st and $st^{'}$ comprising results observed by operations performed by different clients Cl_1 and Cl_2 .

Proof: Following the proof style of Glibert et al. [8], we prove the above theorem by contradiction as follows. Let us consider a pair of concurrent clients, namely Cl_1 and Cl_2 executing on two storage server nodes N_1 and N_2. Let us consider that the network between N_1 and N_2 is partitioned at a particular instant T_i of execution of the clients, i.e., all communication between Cl_1 and Cl_2 are lost. Further let us assume that Cl_1 and Cl_2 perform operations on a common data object x. The clients perform a sequence of write operations α onto x until a network partition occurs, and they are placed in two disjoint components of the network G_1, G_2. Since the system is still available, both servers accept client requests, and apply the corresponding writes onto local copies of x. Because of this, Cl_1 (located in N_1) applies a new sequence of writes β onto x while Cl_2 (located in N_2) applies a different sequence of writes γ to x. Since these components are disconnected, none of these writes are propagated among the servers N_1 and N_2. Therefore, the global session trace St, as observed by the client Cl_1, is a sequence α, β for the conflicting object x; whereas St, as observed by Cl_2, is a sequence α, γ. Let operations op_i and op_j be the prefix of the executions β, and γ observed by clients Cl_1 and Cl_2, respectively. Consider that the storage system enforces a consistency model which imposes a partial order $\preccurlyeq$ among results

of operations from different concurrent clients executing on the underlying storage system. Since $\preccurlyeq$ enforces a partial order among results of operations performed by different concurrent clients, the results observed by clients Cl_1 and Cl_2 must be restricted to a common partial order $\preccurlyeq$. This implies that results of operations op_i and op_j in each of the global session traces observed by the clients Cl_1 and Cl_2 must either satisfy $op_i \preccurlyeq op_j$ or $op_j \preccurlyeq op_i$, respectively. Since $\beta \neq \gamma$ in St, hence $op_i \preccurlyeq op_j$ holds in β, while $op_j \preccurlyeq op_i$ holds in γ; hence the above condition of partial order among operations from different clients is violated in the global session traces α, β and α, γ. This contradicts the requirement of a common partial order $\preccurlyeq$ on the operations performed by different clients comprised in St. Hence, such a consistency model can not achieve high availability and partition tolerance at the same time.

THEOREM 2. *It is possible for a storage system to satisfy a consistency model that enforces a partial order among results observed by operations performed by the same client executing on the system, while simultaneously providing high availability and partition tolerance.*

Following the approach of the proof to Theorem 6, the above theorem is proved as follows. We consider two servers N_1 and N_2 that serve requests from clients Cl_1 and Cl_2. We consider that a partition occurs at a particular instant between the servers N_1 and N_2. Let us consider that the underlying storage system satisfies a consistency model $\mathcal{C}$ that imposes a partial order $\preccurlyeq$ on a global execution comprising operations performed by all clients (represented by a global session trace St). The global session traces observed by clients Cl_1 and Cl_2 comprise sequences of operations, given as α, β and α, γ, respectively. Because of the lack of communication of updates between the servers N_1 and N_2 serving Cl_1 and Cl_2 in presence of a partition, the global session trace observed by Cl_1 satisfies the partial order $\preccurlyeq = \alpha \preccurlyeq \beta$ or $\beta \preccurlyeq \alpha$; whereas the global session trace observed by Cl_2 satisfies a different partial order $\preccurlyeq^{'} = \alpha \preccurlyeq \gamma$ or $\gamma \preccurlyeq \alpha$. Since $\beta \neq \gamma$, each of the above session traces satisfies a partial order $\preccurlyeq$ and $\preccurlyeq^{'}$. Hence, in presence of partitions, the above system remains available, while satisfying a partial order on the global session trace St. Thus, such a consistency model can be implemented in a manner which simultaneously achieves high availability and partition tolerance.

7. ANALYSIS OF CONSISTENCY MODELS WITH RESPECT TO CAP

For RYW, MR, Causal consistency, WFR, and MW, E^s imposes restrictive conditions on operations comprised in an equivalent legal serialization of a partial execution S_p, composed of operations performed by the given client, along with only writes from other concurrent clients. Since S_p does not comprise read operations from all clients in St, the above consistency models impose only a strict partial order with respect to operations in St. Thus, writes from concurrent clients are allowed to overwrite results of writes from a given client as long as there exists an equivalent legal serialization comprising the combination of operations from the given client and the concurrent writes. It only restricts that the partial order, i.e., precedence, among the operations from each client is maintained. Similarly, Processor consistency (PC) restricts that the order among operations in a given client must be preserved in all executions, i.e., no client may observe the operations from a given client in a different order than the invocation order. Hence, PC does not enforce a partial order across operations from different concurrent clients, it only imposes a strict partial order on all operations from a given client. Thus, these comprise a weaker class of consistency models that can be satisfied without synchronization among clients, i.e., these consistency models are not dependent among reliable communication among concurrent clients (like broadcasting of results on common data objects, or blocking of conflicting operations). Using the expression for S_c from Definition 12.1 in Equation 4, the specification for Strict Serializability reduces to $E^s = \forall i, j, st\big(Op^i X^o Op^{j^{'}} \in st \rightarrow \exists \Big(Op^{i^{'}}, Op^{j^{'}} \in Ser \wedge Op^{i^{'}}, Op^{j^{'}} \in st\Big) \Big(Op^{i^{'}} X^o Op^{j^{'}} \in S_c\Big)\big)$. Since, $S_c \subset Ser$, $Op^{i^{'}} X^o Op^{j^{'}} \in S_c$ implies $Op^{i^{'}} X^o Op^{j^{'}} \in Ser$ Hence, moving the existential quantifier out of the enclosing scope of the universal quantifier, the above expression can be rewritten as $E^s = \exists \Big(Op^{i^{'}}, Op^{j^{'}} \in Ser\Big) \forall i, j, st\big(Op^i X^o Op^{j^{'}} \in st \rightarrow \Big(Op^{i^{'}} X^o Op^{j^{'}} \in Ser\Big)\big)$. Thus, Strict Serializability restricts that the mutual execution order among operations performed by each client in a global execution must be preserved in an equivalent legal serialization Ser, comprised of all operations in a global execution. Thus, Strict serializability imposes a partial order among operations from all clients comprised in St. TSO imposes restrictions on the execution order and observed results among all operations performed by different concurrent clients observed in a global session trace St. Hence, it directly follows from the definitions that TSO imposes a partial order on St, where each operation performed. by a given client in St is bounded by a partial order relation with an operation in St performed by another concurrent clienr. Hence, these consistency models require strict synchronization among operations from different clients, such that operations from each client must be executed in isolation from other concurrent clients.

8. DERIVING CONSPEC SPECIFICATIONS FROM THE STATE-OF-THE-ART DEFINITIONS

Here we show how the ConSpec specifications can be directly derived from state-of-the-art specifications of consistency and isolation level [6, 18]. As already discussed, state-of-the-art consistency specifications define axiomatic rules for consistency models using first order logic expressions. Consistency definitions are specified by Chockler et al. [6] as follows. They use the following formalization syntax in their definitions. Consider two operations op^1 and op^2 invoked from a client application, such that op^1 is a write operation that updates a storage system object o, and op^2 is a read operation that reads o. The symbol $\rightarrow$ denotes precedence relationship [3] between two operations. Chockler et al. used the notation σ to denote an execution of a sequence of read and write operations performed by a client application (referred to as a process by Chockler). They also use a specialised form of the precedence operator, $\xrightarrow{\sigma}$, where the superscript σ is used to indicate that the precedence relationship is defined specifically with respect to an execution σ by a client application. Thus, an expression $op^1 \xrightarrow{\sigma} op^2$ indicates that an execution of the operation op^1 precedes, i.e., happens before, an execution of the operation op^2 in an execution sequence σ. Chockler et al. denote a partial execution comprising all operations performed by a process (or a client application) and all write operations from from other processes as $\sigma|i + w$. They denote a legal serialization for the above partial execution as S_i; we denote the above legal serialization as S_p. For means and purposes of this paper, an execution sequence σ according to the terminology of Chockler et al. is equivalent to a

session trace st in ConSpec. Hence, we assign the notation st to an execution sequence σ in Chockler's definitions. We can rewrite the precedence relation $op^1 \xrightarrow{\sigma} op^2$, in terms of Linear Temporal Logic (i.e., LTL) $op^1 F op^2 \in st$, where we replace the precedence operator $\rightarrow$ with the equivalent LTL operator F that denotes "eventually". The definitions by Chockler et al. express a precedence relation $op^1 \rightarrow op^2$ in a serialization S_i as $op^1 \xrightarrow{S_i} op^2$, where we specify the scope of the precedence operator $\rightarrow$ with the serialization S_i. We can rewrite the above precedence relation as $op^1 \xrightarrow{S_i} op^2$, in terms of Linear Temporal Logic (i.e., LTL) $op^1 F op^2 \in S$, where we assign the notation S to a serialization S_i, and replace the precedence operator with the equivalent LTL operator F that denotes "eventually". Also, Chockler et al. bases their definitions on a serialization of an execution sequence comprising all operations performed by a client application (or a process), along with writes by all other clients. In ConSpec, we denote such a serialization as S_p. We can express the precedence relation for such a serialization in terms of LTL as $op^1 F op^2 \in S_p$.

Let w^i and r^j be signatures of read and write operations op^1 and op^2, respectively. Chockler et al. states the RYW consistency model as: if the condition represented by the expression $op^1 \xrightarrow{\sigma} op^2$ holds for a given execution sequence σ (let us call this Condition 1) comprising the above write and read operations, there must exist a serialization S_i,comprising the operations op^1 and op^2, for which the condition given by the expression $op^1 \xrightarrow{S} op^2$ must hold (let us call this Condition 2). The above precedence relationships among operations op^1 and op^2 in Condition 1 can be directly expressed in terms of LTL as follows. Let Op^1 and Op^2 be propositional logic variables that indicate whether operations op^1 and op^2 on an object o have returned (indicated by the value of the variables being TRUE) or not (indicated by the value of the variables being FALSE), respectively. Condition 1, which expresses the precondition for RYW with respect to a given execution order σ, can be expressed in terms of the LTL expression $Op^1 F Op^2 \in st$, where a session trace st is equivalent to σ for the means and purposes of this paper. The above LTL expression implies: if the propositional variable Op^1 is true at one instant, Op^2 is eventually true in some later instant for the same session trace st. In other words, the operation op^2 on object o follows the operation op^1 in the given session trace st. The expression $op^1 \xrightarrow{Si} op^2$ in Condition 2 can be expressed in terms of LTL as $Op^1 F Op^2 \in S_p$, where S_p is equivalent to S_i, i.e., both notations represent a legal serialization comprising all operations performed from a given client application along with writes from all other clients. Further, since RYW talks about a sequence of write operations followed by a read, the propositional variables Op^1 and Op^2 in both Condition 1 and Condition 2 can safely be replaced by a new propositional variable W^i and R^j. Since $W^i F R^j \in S_p$ denotes a partial order among W^i and R^j, we can rewrite the above expression as $W^i \preccurlyeq R^j$. Thus, Chockler's definition reduces into the specification given in Equation 4.1.

Session Monotonic or Monotonic Read (also referred to as Session Causality or MR) consistency model is another popular consistency model [6, 18]. According to Chockler et al., MR is expressed as: if the condition $op^1 \xrightarrow{\sigma} op^2$ holds for a given execution sequence σ (Condition 1), where both op^1 and op^2 must be read operations, there must exist a serialization S_i,comprising the operations op^1 and op^2, for which the condition $op^1 \xrightarrow{S} op^2$ must hold (Condition 2). As in the case of RYW, the expressions $op^1 \xrightarrow{\sigma} op^2$, and $op^1 \xrightarrow{S} op^2$ can be rewritten as LTL expressions $R^i F R^j \in st$ and $R^i F R^j \in S_p$, respectively. Again, since $R^i F R^j \in S_p$ denotes a partial order among R^i and R^j, we can rewrite the above expression as $R^i \preccurlyeq R^j$, thus reducing the above specification into Equation 2. Thus, the ConSpec specification for MR is directly derived from the definition by Chockler et al.

Causal consistency is specified by Chockler et al. as follows. Chockler et al. defines a *direct precedence relation* (which we denote as $\overset{\sigma}{\Rightarrow}$) between operations op^i andop^k in the execution order σ as follows. $op^i \overset{\sigma}{\Rightarrow} op^j$ implies that either op^j reads values written by operation op^i, or a precedence relationship exists in the execution order σ between op^i and op^j, i.e., $op^i \xrightarrow{\sigma} op^j$ exists. The former part of the precondition can be expressed in the form $Op^{i'} F Op^{j'}$, where the LTL-like operator F is equivalent to the precedence operator. The latter part of the precondition implies read operation Op^i reads the value written by Op^j, i.e., $\left(Op^{i'} = W^{i'}\right) \wedge \left(Op^{j'} = R^{j'}\right) \wedge (v_i = v_j)$. The precondition of causal consistency specifies that a transitive closure must exist for a direct precedence relation $\overset{\sigma}{\Rightarrow}$, i.e., $\overset{\star}{\Rightarrow}$ must exist, i.,e., the condition $Op^{i'} F Op^{j'} \vee \left(\left(Op^{i'} = W^{i'}\right) \wedge \left(Op^{j'} = R^{j'}\right) \wedge (v_i = v_j)\right)$ must hold. Chockler et al. specifies that for causal consistency, if the above precondition holds, the condition $op^i \xrightarrow{S} op^j$ must hold, i.e., an occurrence of op^i must be followed by an occurrence of op^j in any legal serialization S_i. The above postcondition can be expressed in terms of LTL as $\exists S_p \left(\forall Op^i F Op^j \in S_p\right)$. The operation op^i precedes op^j such that the condition $Op^{i'} F Op^{j'} \in st$ is satisfied. Since $Op^{i'} F Op^{j'} \in S_p$ denotes a partial order among $Op^{i'}$ and $Op^{j'}$, we can rewrite the above expression as $Op^{i'} \preccurlyeq Op^{j'}$. Thus, Chockler's definition reduces into the specification given in Equation 3.

Chockler et al. states strict serializability as: the order of execution among operations comprised in any execution sequence of a given client application must match the precedence order in a legal serialization for the given execution sequence. The order of execution among operations in an execution sequence σ can be given in terms of a series of LTL expression that express the precedence of successive operations in the sequence. The precedence relation between any two successive operations op^i and op^j can be expressed as $op^i X op^j$. Thus, the above condition can be expressed as: any two operations in an execution sequence must execute in an order that matches the precedence relation among these operations in a legal serialization of the above execution sequence. In other words, any two operations must occur in a session trace in an order that matches the precedence relation of these operations in a legal serialization of those operations. Hence, this condition can be expressed in terms of LTL as $Op^i X Op^j \in st \rightarrow \exists S_c \left(Op^i X Op^j \in S_c\right)$. Since $Op^{i'} X Op^{j'} \in S_c$ denotes a partial order among $Op^{i'}$ and $Op^{j'}$, we can rewrite the above expression as $Op^{i'} \preccurlyeq Op^{j'}$. Thus, Chockler's definition reduces into the specification given in Equation 4.

Chockler et al. states the WFR consistency model as: if the condition represented by the expression $op^1 \xrightarrow{\sigma} op^2$ holds for a given execution sequence σ of read operation followed by write operation (let us call this Condition 1), there must exist a serialization S_i,comprising the operations op^1 and op^2, for which the condition given by the expression $op^1 \xrightarrow{S} op^2$ must hold (let us call this Condition 2). The above precedence relationships among operations op^1 and op^2 in Condition 1 can be directly expressed in terms of an LTL expression $R^i F W^j \in st$, where a session trace st is equivalent to σ for the means and purposes of this paper. The expression $op^1 \xrightarrow{S} op^2$ in Condition 2 can be expressed in terms of LTL as

$R^iFW^j \in S_p$, where S_p is equivalent to S_i. Since $R^iFW^i \in S_p$ denotes a partial order among R^i and W^i, we can rewrite the above expression as $R^i \preccurlyeq W^i$. Thus, Chockler's definition reduces into the specification given in Equation 5.

Chockler et al. states the MW consistency model as: if the condition represented by the expression $op^1 \xrightarrow{\sigma} op^2$ holds for a given execution sequence σ of write operations (let us call this Condition 1), there must exist a serialization S_i,comprising the operations op^1 and op^2, for which the condition given by the expression $op^1 \xrightarrow{S} op^2$ must hold (let us call this Condition 2). The above precedence relationships among operations op^1 and op^2 in Condition 1 can be directly expressed in terms of an LTL expression $Op^1FOp^2 \in st$, where a session trace st is equivalent to σ for the means and purposes of this paper. The expression $op^1 \xrightarrow{Si} op^2$ in Condition 2 can be expressed in terms of LTL as $Op^1FOp^2 \in S_p$, where S_p is equivalent to S_i. Further, since MW talks about a sequence of write operations, the propositional variable Op^1 and Op^2 in both Condition 1 and Condition 2 can safely be replaced by new propositional variables W^i and W^j. Since $W^iFW^i \in S_p$ denotes a partial order among W^i and W^i, we can rewrite the above expression as $W^i \preccurlyeq W^i$. Thus, Chockler's definition reduces into the specification given in Equation 6.

Total Store Order (TSO or Total Order) is specified by Owen et al. [17] as follows. In any two executions of a sequence of operations by two processors (or client applications), a pair of operations on a common storage system object x must be executed in the same precedence order. The above condition implies that any session trace st must comprise write-read sequences $op^i(x), op^j(x)$ on an object x in the temporal order specified in the global session trace St. Thus, $E^s = \forall x, st \in St \ \nexists op^i(x), op^j(x) \in \mathcal{O}$ $\left(Op^{i^{'}}(x)FOp^{j^{'}}(x) \in St\right) \rightarrow \left(Op^{j^{''}}(x)FOp^{i^{''}}(x) \in st\right)$ The above expression can be further rewritten in terms of a partial order $\preccurlyeq(x)$ over a given object x as $\forall x, \left(Op^{i^{'}}(x)FOp^{j^{'}}(x) \in St\right) \rightarrow \left(Op^{i^{'}}(x) \preccurlyeq (x) Op^{j^{'}}(x)\right)$, where $\preccurlyeq(x)$ is a partial order over a given object x with respect to a given global session trace St. Similarly, the specification for Processor consistency (PC) model can be directly derived using the above line of reasoning; PC talks about preserving the order of operations executed by individual processors (or client applications). Processor Consistency specifies: operations in a client application must execute according to the invocation order with respect to a given processor (client); i.e., result of a preceding operation invoked from a given client must be observed in all processors before result of a succeeding operation can be observed in any processor (or client application). Reads against write operations should view the results of the writes according to the invocation order of these writes in a client application. Consider that a given client observes operations $op^i(x), op^j(y)$ in a temporal order $Op^{i^{'}}(x)FOp^{j^{'}}(y)$ observed in a session trace st. Then, no other session trace $st^{'}$ observed by any other concurrent client may observe the operations in an order which violates the temporal order in st. Thus, Processor Consistency can be expressed as $E^s = \forall x, y, st, st^{'} \in St \ \nexists op^i(x), op^j(y) \in \mathcal{O}$ $\left(Op^{i^{'}}(x)FOp^{j^{'}}(y) \in st\right) \rightarrow \left(Op^{j^{''}}(y)FOp^{i^{''}}(x) \in st^{'}\right)$, which can be rewritten as follows. $\forall x, y, st \in St$ $\left(Op^{i^{'}}(x)FOp^{j^{'}}(y) \in st\right) \rightarrow$ $\left(Op^{i^{'}}(x) \preccurlyeq Op^{j^{'}}(y)\right)$, where $\preccurlyeq$ is a partial order over operations across all data objects with respect to a given global session trace St.

The ConSpec specifications for the isolation levels are derived from the definitions by Adya et al. [1]. Here, we show that the isolation specifications in ConSpec, given in Equations 9, 10, 11, and 12, follow directly from the definitions of isolation levels PL-1, PL-2, PL-3, and PL-2.99 [1]. We analyze the ConSpec Equations, and demonstrate their equivalence to the definitions of [1], as follows.

The PL-1 specification states that the anomaly G0 must be proscribed in any session trace collected for the execution of a client application. G0 specifies that there can not be any directed cycle in the dependency graph corresponding to an execution of a pair of transactions, comprising entirely of write dependency edges. Let us consider a pair of transactions tx and ty. Let us consider that write operations $w^i(x)_{tx}$ and $w^j(y)_{tx}$ write to objects x and y from transaction tx, and $w^k(x)_{ty}$ and $w^l(y)_{ty}$ write to objects x and y from transaction ty. According to PL-1 there can not exist a pair of read operations $r^m(x)_{tx}$ and $r^n(y)_{ty}$ invoked from transactions tx and ty such that the following conditions are simultaneously satisfied: 1) $r^m(x)_{tx}$ reads the result of the write operation $w^i(x)_{tx}$ causing a ww dependency from tx to ty, i.e., $v_m^{'} = v_i^{'}$ must not hold, and 2) $r^n(y)_{ty}$ reads the result of the write operation $w^l(y)_{ty}$ causing a ww dependency between ty to tx, i.e., $v_n^{'} = v_l^{'}$ must not hold. Simultaneous satisfaction of conditions 1 and 2 results in a cycle comprising ww dependencies between tx and ty. Thus, we can express the specification of PL-1 as $C = \forall St, st, tx, ty, x, y, W^i(x)_{tx}^{'}, W^j(y)_{tx}^{'}, W^k(x)_{ty}^{'}, W^l(y)_{ty}^{'} \in St$ $\left(\nexists R^m(x)_{tx}^{'}, R^n(y)_{ty}^{'} \in St\right)\left(\left(\left(v_n^{'} = v_l^{'}\right) \wedge \left(v_m^{'} = v_i^{'}\right)\right) \vee \left(\left(v_n^{'} = v_j^{'}\right) \wedge \left(v_m^{'} = v_k^{'}\right)\right)\right)$ which is identical to Equation 9.

The isolation level PL-2 proscribes the anomalies G1-a, G1-b, and G1-c [1]. We show that the ConSpec specification for PL-2, given in Equation 10, disallows the above anomalies as follows. **G1-a**: According to G1-a, a transaction can not observe a value written by an aborted transaction. In other words, if either transaction tx or transaction ty aborts, i.e., either of the conditions $a_{tx} \in st$ or $a_{ty} \in st$ hold, then there can not exist a read operation that reads the value written by an aborted transaction, i.e., the conditions $W^i(x)_{tx}^{'}FR^j(x)_{ty}^{'} \in st$ or $W^k(y)_{ty}^{'}FR^l(y)_{tx}^{'} \in st$ can not hold. Thus, the G1-a condition ca be directly translated to the expression $\nexists a_{tx} \in st \ \left(W^i(x)_{tx}^{'}FR^j(x)_{ty}^{'} \in st\right) \wedge$ $\nexists a_{ty} \in st \ \left(W^k(y)_{ty}^{'}FR^l(y)_{tx}^{'} \in st\right)$ in Equation 10. **G1-b**: According to G1-b, a transaction must always read the final committed version of an object. Let us consider consecutive write operations $w^i(x)_{tx}$ and $w^j(x)_{tx}$ invoked from transaction tx followed by the commit statement c_{tx}, and a read operation $r^k(x)_{ty}$ invoked from transaction ty, i.e., $W^i(x)_{tx}^{'}FW^j(x)_{tx}^{'}Fc_{tx}FR^k(x)_{ty}^{'} \in st$. According to G1-b, ty can never read a value written by an operation in tx that was not finally committed by tx. Thus, $r^k(x)_{ty}$ cannot return the result of $w^i(x)_{tx}$ since it was overwritten by another write $w^j(x)_{tx}$ before the commit c_{tx}. The above condition can be expressed as $\nexists\left(\left(W^i(x)_{tx}^{'}FW^j(x)_{tx}^{'}Fc_{tx}FR^k(x)_{ty}^{'} \in st\right) \wedge \left(v_k^{'} = v_i^{'}\right)\right) \wedge \nexists\left(\left(W^i(x)_{ty}^{'}FW^j(x)_{ty}^{'}Fc_{ty}FR^k(x)_{tx}^{'} \in st\right) \wedge \left(v_k^{'} = v_i^{'}\right)\right)$. **G1-c**: G1-c specifies that there can be no direct cycle comprising dependency edges, i.e., wr and ww dependencies, between transactions in a given execution. The condition $\nexists\left(W^i(x)_{tx}^{'}, R^j(y)_{tx}^{'}, W^k(y)_{ty}^{'}, R^l(x)_{ty}^{'} \in St\right)\left(\left(v_l^{'} = v_i^{'}\right) \wedge \left(v_k^{'} = v_j^{'}\right)\right)$ proscribes wr dependency cycles. The expression $\left(\nexists W^i(x)_{tx}^{'}, W^j(y)_{tx}^{'}, W^k(x)_{ty}^{'}, W^l(y)_{ty}^{'}, R^m(x)_{tx}^{'}, R^n(y)_{ty}^{'}\right.$

$\in St)\big(\big(\big(v_n^{'}=v_l^{'}\big)\wedge\big(v_m^{'}=v_i^{'}\big)\big)\vee\big(\big(v_n^{'}=v_j^{'}\big)\wedge\big(v_m^{'}=v_k^{'}\big)\big)\big)$ proscribes ww cycles. Thus, the ConSpec specification of PL-2 in Equation 10 proscribes G1-a, G1-b, and G1-c; thus, Equation 10 is equivalent to PL-2 specifications by Adya et al.

The isolation level PL-3 proscribes the anomaly G-2 [1]. According to the ConSpec specification for PL-3 (refer to Equation 11), apart from proscribing G1 anomalies, G2 anomalies are also disallowed. According to G2, their can be effectively no cycle comprising anti-dependency edges between transactions. The expression
$\big(\nexists\big(R^i(x)_{tx}^{'}FW^j(x)_{ty}^{'}FR^k(x)_{ty}^{'}\in St\big)\wedge$
$\big(R^l(x)_{ty}^{'}FW^m(x)_{tx}^{'}FR^n(x)_{tx}^{'}\in St\big)\big)\big(\big(v_k^{'}=v_j^{'}\big)\wedge\big(v_n^{'}=v_m^{'}\big)\big)\big)$
proscribes anti-dependency cycles. Thus, the ConSpec expression disallows G2, and effectively follows PL-3. Following the same line of reasoning, we can prove that the ConSpec expression in Equation 12 effectively specifies PL-2.99.

9. IMPLEMENTATION

We provide an open source automated verification tool built using Spin, an open source software verification tool [12]. Our tool can verify whether a given session trace satisfies a given consistency model or a given isolation level, specified in terms of a ConSpec expression. The source code and instructions for running the ConSpec tool is provided in an open source github repository https://github.com/ssidhanta/ConSpecTool. The tool is run from the shell command line with a session trace as input argument. It accepts specifications of a given ConSpec definition in the form of an LTL expression, provided as an input to the command line or as a file input. First, it generates a set of equivalent legal serializations for a given session trace. Then, it runs the Spin model checker over the session trace, and the serialization list, and determines whether the session trace satisfies the specified consistency model or not.

Using the PROMELA meta language, the Spin tool captures the semantics of a given storage system. It parses a given session trace and the set of equivalent serializations generated for a given storage system. The PROMELA specifications are translated into C code. Spin acts as a driver that verifies the generated C code against the ConSpec definitions expressed in terms of LTL formulas. The LTL model checker embedded within Spin verifies whether a given session trace satisfies a given ConSpec specifications. The github repository comprises a PROMELA specification for each of the consistency models and isolation levels, the Spin-based main source file, and the code for generating the list of equivalent serializations.

10. EQUIVALENCE BETWEEN DEPENDENCY RELATIONS AND LTL RELATIONS OF CONSPEC

State-of-the-art definitions specify consistency levels in terms of dependency relations, namely $w \xrightarrow{wr} r$, $w \xrightarrow{ww} w^{'}$, and $w \xrightarrow{wr} r$ dependency, respectively [10]. The $w \xrightarrow{wr} r$ dependency can be translated to the form $\exists\big(\big(W^i\in st\big)\wedge\big(R^j\in St\big)\big)$ $\big(v^{i^{'}}=v^{j^{''}}\big)$, where we represent the write and read operations w and r involved in the dependency relation with the propositional variable W^i and R^j, respectively. Denoting w, and $w^{'}$ with propositional variables W^i and W^j, the dependency relation $w \xrightarrow{ww} w^{'}$ is translated into ConSpec form as follows.

$$\exists\big(\big(W^{i^{'}}\in st\big)\wedge\big(W^{j^{'}}\in St\big)\big)\big(\exists\big(R^{l^{''}}\in St\big)\big(v_l{}^{''}=v_j{}^{''}\big)\wedge\big(\big(v_l{}^{'}=v_i{}^{'}\big)\vee\big(\big(W^{i^{''}}FR^{l^{''}}\in St\big)\wedge\big(\nexists W^{i^{''}}FW^{m^{''}}FR^{l^{''}}\in St\big)\big)\big)\big) \tag{13}$$

Similarly, denoting r, and w with propositional variables R^i and W^j, the $r \xrightarrow{rw} w$ dependency is translated into ConSpec form as follows.

$$\exists\big(\big(R^{i^{'}}\in st\big)\wedge\big(W^{j^{'}}\in St\big)\wedge\big(\exists W^{k^{''}}\in st\big)\big(v_i{}^{'}=v_k{}^{'}\big)\wedge\big(\exists R^{l^{''}}\in St\big)\big(v_l{}^{'}=v_j{}^{'}\big)\big)\big(\big(v_k{}^{'}=v_l{}^{'}\big)\vee\big(\big(W^{k^{''}}FR^{l^{''}}\in St\big)\wedge\big(\nexists W^{k^{''}}FW^{m^{''}}FR^{l^{''}}\in St\big)\big)\big) \tag{14}$$

Adya et al. define isolation levels in terms of $w \xrightarrow{wr} r$, $w \xrightarrow{ww} w^{'}$, and $w \xrightarrow{wr} r$ dependencies among transactions [1]. The wr dependency relation of the dependency graph can be translated into ConSpec form
$\exists\big(\big(W^i_{tx}\in st\big)\wedge\big(R^j_{ty}\in St\big)\big)\big(v^{i^{'}}=v^{j^{''}}\big)$. The ww dependency relation between transactions tx and ty can be translated to

$$\exists\big(\big(W^{i^{'}}_{tx}\in st\big)\wedge\big(W^{j^{'}}_{yx}\in St\big)\big)\big(\exists\big(R^{l^{''}}ty\in St\big)\big(v_l{}^{''}=v_j{}^{''}\big)\wedge\big(\big(v_l{}^{'}=v_i{}^{'}\big)\vee\big(\big(W^{i^{''}}FR^{l^{''}}_{ty}\in St\big)\wedge\big(\big(\nexists W^{i^{''}}_{tx}FW^{m^{''}}_{tx}FR^{l^{''}}_{ty}\in St\big)\wedge\big(\nexists W^{i^{''}}_{tx}FW^{m^{''}}_{ty}FR^{l^{''}}_{ty}\in St\big)\big)\big)\big)\big) \tag{15}$$

The rw dependency relation between transactions tx and ty can be translated into ConSpec form

$$\exists\big(\big(R^{i^{'}}_{tx}\in st\big)\wedge\big(W^{j^{'}}_{ty}\in St\big)\wedge\big(\exists W^{k^{''}}_{tx}\in st\big)\big(v_i{}^{'}=v_k{}^{'}\big)\wedge\big(\exists R^{l^{''}}_{ty}\in St\big)\big(v_l{}^{'}=v_j{}^{'}\big)\big)\big(\big(v_k{}^{'}=v_l{}^{'}\big)\vee\big(\big(W^{k^{''}}_{tx}FR^{l^{''}}_{ty}\in st^{''}\big)\wedge\big(\big(\nexists W^{k^{''}}_{tx}FW^{m^{''}}_{tx}FR^{l^{''}}_{ty}\in St\big)\wedge\big(\nexists W^{k^{''}}_{tx}FW^{m^{''}}_{ty}FR^{l^{''}}_{ty}\in St\big)\big)\big)\big) \tag{16}$$

11. REFERENCES

[1] A. Adya, B. Liskov, and P. E. O'Neil. Generalized isolation level definitions. In *ICDE*, pages 67–78, 2000.

[2] P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Feral concurrency control: An empirical investigation of modern application integrity. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1327–1342, New York, NY, USA, 2015. ACM.

[3] P. Bailis, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Bolt-on causal consistency. In *SIGMOD '13*.

[4] E. A. Brewer. Towards robust distributed systems (Invited Talk). In *Proc. of the 19th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 2000.

[5] S. Burckhardt. Principles of eventual consistency. *Found. Trends Program. Lang.*, 1(1-2):1–150, Oct. 2014.

[6] G. Chockler, R. Friedman, and R. Vitenberg. *Consistency Conditions for a CORBA Caching Service*, pages 374–388. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[7] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, 1(2):1277–1288, Aug. 2008.

[8] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, June 2002.

[9] A. Gotsman, H. Yang, C. Ferreira, M. Najafzadeh, and M. Shapiro. 'cause i'm strong enough: Reasoning about consistency choices in distributed systems. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '16, pages 371–384, New York, NY, USA, 2016. ACM.

[10] J. L. Hennessy and D. A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2011.

[11] M. P. Herlihy and J. M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, July 1990.

[12] G. Holzmann. *Spin Model Checker, the: Primer and Reference Manual*. Addison-Wesley Professional, first edition, 2003.

[13] A. Lakshman and P. Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, Apr. 2010.

[14] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues. Making geo-replicated systems fast as possible, consistent when necessary. In *Proc. of the 10th USENIX conference on Operating Systems Design and Implementation*, OSDI'12, pages 265–278, Berkeley, CA, USA, 2012. USENIX Association.

[15] 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, New York, NY, USA, 2011. ACM.

[16] Z. Manna and A. Pnueli. A hierarchy of temporal properties (invited paper, 1989). In *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, PODC '90, pages 377–410, New York, NY, USA, 1990. ACM.

[17] S. Owens, S. Sarkar, and P. Sewell. A better x86 memory model: X86-tso. In *Proceedings of the 22Nd International Conference on Theorem Proving in Higher Order Logics*, TPHOLs '09, pages 391–407, Berlin, Heidelberg, 2009. Springer-Verlag.

[18] D. B. Terry, A. J. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. W. Welch. Session guarantees for weakly consistent replicated data. In *Proceedings of the Third International Conference on Parallel and Distributed Information Systems*, PDIS '94, pages 140–149, Washington, DC, USA, 1994. IEEE Computer Society.

12. APPENDIX

12.1 Serialization Definitions

A *serialization* is a linear sequence comprising all operations comprising a global execution on a storage system. A serialization is said to be *legal* if every read operation returns the value written by the latest write operation preceding it in the serialization. Following Chockler et al. [6], ConSpec defines consistency levels in terms of valid legal serialization orders equivalent to a given execution. Similar to Chockler et al., ConSpec defines different consistency models in terms of different specialized forms of legal serialization, which have been defined below. ConSpec defines those consistency models, which can not be expressed in terms of legal serialization, in terms of constraints on the global execution order, represented by the global session trace St. Let the propositional variable $Op^{i'}$ denote an execution of the i'th operation comprised in the set $mathcalO$ of operations executed from a given client application in a session trace.

DEFINITION 2. *(Legal Serialization) We represent a global execution, comprising concurrent operations executed from multiple clients on a system, with a global session trace St, which is a set of operations from all clients. A legal serialization Ser of a given global execution is represented by a sequence of propositional variables $Op^{i'}$, each variable Op^{i} denoting an execution of an operation op^{i}, comprised in the set of operations $\mathcal{O}$ executed by a given client, observed a global session trace St, as follows.* $Ser = \{Op^{i'} \mid Op^{i'} \in st\}$, *such that* $\forall op^{i'}, op^{j'} \in \mathcal{O}$ $\big(Op^{i'} = W^{i'} \wedge Op^{j'} = R^{j'} \wedge \exists st \in St\ \big(\big(Op^{i'} F Op^{j'} \in st \wedge \nexists Op^{k'} = W^{k'}\ \big(Op^{i'} F Op^{k'} F Op^{j'} \in St\big)\big) \rightarrow \big(v_j{}' = v_i{}'\big)\big)\big)$.

DEFINITION 3. *(Serialization of a Client) We represent a local execution of a given client observed at a client machine, with a local session trace st, comprising results observed for operations executed by a given client application, comprised in the set of operations $\mathcal{O}$. We formally define* serialization of a client, *S_c, as the restriction of a legal serialization of a global execution to the operations invoked by a particular client application, which in turn, corresponds to operations observed in a particular session trace st.* $S_c = \{op^{i'}, op^{j'} \in \mathcal{O} \wedge Op^{i'} \mid Op^{i'} \in Ser \wedge Op^{i'} \in st\}$.

DEFINITION 4. *(Serialization of a Partial Execution) The notation S_p is used to denote a legal serialization of a partial execution comprising all operations from the set of operations $\mathcal{O}$ performed by a given client application, represented by a local session trace st, and all write operations executed from all other concurrent client applications, comprised in the global session trace St. The above partial execution is denoted as $\sigma|i + w$ by Chockler et al. Thus, we formally denote S_p as*
$S_p = \{Op^{i'} \mid \big(Op^{i'} \in st\big) \vee \big(Op^{i'} = W^{i'} \wedge Op^{i'} \notin st\big)\}$, *such that* $\forall op^{i'}, op^{j'} \in \mathcal{O}\big(Op^{i'} = W^{i'} \wedge Op^{j'} = R^{j'} \wedge$
$\exists st \in St\ \big(\big(Op^{i'} F Op^{j'} \in st \wedge \nexists Op^{k'} = W^{k'}$
$\big(Op^{i'} F Op^{k'} F Op^{j'} \in St\big)\big) \rightarrow \big(v_j{}' = v_i{}'\big)\big)\big)$.

The above legal serialization does not comprise read operations from other clients, hence its a serialization of a partial execution.

12.2 Violation Examples: Verified Against ConSpec Specifications

Using example session traces, we illustrate situations where a given consistency model is violated. Let us consider a session trace st_1: $w^1(x,1), w^2(x,2), r^1(x)2, r^2(x)1$ to demonstrate how violation of RYW consistency can occur. From the session trace st_1, the following LTL expression $W^1FW^2FR^1FR^2$ follows directly from the temporal order of execution as per the session trace. In st_1, the read operation $r^1(x)2$ follows the write operation $w^1(x,1)$ can be expressed by the LTL expression W^1FR^1; which satisfies the condition given in LHS of the shorter specification E^s in Equation 2 if $W^i = W^1$, $R^j = R^1$. The session trace st_1 can be transformed into the following partial legal serializations (that also comprises all write operations from other clients: $Si_p^1 = W^1FR^2FW^2FR^1$, $Si_p^2 = W^2FR^1FW^1FR^2$, and $Si_p^3 = R^1FW^2FW^1FR^2$. Only for Si_p^1, the condition $W^1FR^2 \in Si_p^1$ holds, which does not hold for Si_p^2 and Si_p^3. thus the condition corresponding to the RHS of the implies operator $\rightarrow$ in E^s , given in Equation 4.1, holds. Next, we consider the sequence in the session trace where the write operation $w^1(x,1)$ is followed by the read operation $r^2(x)1$, expressed by the LTL expression W^1FR^2. As discussed before a partial serialization Si_p^1 exists for the session trace such that $W^1FR^2 \in Si_p^1$, satisfying the first condition in the RHS of Equation 4.1. Next, we consider the sequence write operation W^2 followed by R^1 in the session trace, given as W^2FR^1. This implies that the condition $W^iF^xR^j$ in the LHS of the E^s condition in Equation 4.1 holds, where $W^i = W^2$ and $R^j = R^1$. Again a legal partial serialization Si_p^1 exists such that $W^2FR^1 \in Si_p^1$. Next, we consider the sequence write operation W^2 followed by R^2 in the session trace, given as W^2FR^2. This implies that the condition $W^iF^xR^j$ in the LHS of E^s in Equation 4.1 holds, where $W^i = W^2$ and $R^j = R^2$. Si_p^1 is the only legal partial serialization that satisfies the condition $W^iFR^j \in Si_p$ in the RHS of Equation 4.1 for all the above write-read operation sequences. However, Si_p does not satisfy $W^iFR^j \in Si_p$ for the write-read sequence W^2FR^2. For anomaly condition C in Equation 4.1 to hold, their cannot exist in the global session trace a read operation r^k preceding the read operation r^j (which is $r^2(x)1$) and a write operation w^l preceding w^i (which is $w^2(x,2)$ in this case) such that the LTL condition $\left(v^{k'} = v^{i'}\right) \wedge \left(\left(v^{j'} = v^{l'}\right) \vee \left(\left(W^{l'}FR^{j'} \in St\right) \wedge \left(\nexists W^{l'}FW^{m'}FR^{j'} \in St\right)\right)\right)$ hold. However, in the given session trace st_1, there does exist a read $r^1(x)2$ and a write $w^1(x,1)$, such that conditions $\left(v^{k'} = v^{i'}\right)$ and $\left(v^{j'} = v^{l'}\right)$ hold, thus, in turn, violating the anomaly condition. Hence, the RYW condition is violated for the given session trace.

Let us consider a session trace st_2: $w^1(x,1)$, st_2': $w^2(x,2)$, and st_2'': $r^1(x)1, r^2(x)2, r^3(x)1$ to illustrate how violation of MR consistency can occur. Since the above session traces comprise write operations followed by read on the same object, as well as multiple read operations on same object x, they satisfy the conditions given in LHS of Equation 2. First, let us consider the sequence of read operations $r^1(x)1$ and $r^2(x)2$ in st^2, given by the LTL expression $R^1F^xR^2$. If and only if $R^i = R^1$ and $R^j = R^2$, the above sequence of reads can satisfy the LHS of the condition E^s in Equation 2. The above traces can be transformed into legal partial serialization order $Si_p = W^1FR^1FR^3FW^2FR^2$, such that $R^1FR^2 \in Si_p$ holds, thus satisfying the first condition the RHS of the implies operator $\rightarrow$ in E^s in Equation 2. Next, we consider the sequence $r^1(x)1$ followed by $r^3(x)1$ in the above traces, given as the LTL expression R^1FR^3. The above condition satisfies the LHS of E^s Equation 2, given that $R^i = R^1$ and $R^j = R^3$, which, in turn, implies $v_i = v_1 = 1$ and $v_j = v_2 = 1$. As discussed above, a legal partial serialization Si_p exists for the above trace, such that $R^1FR^3 \in Si_p$ holds; the condition E^s in the RHS of Equation 2 holds for the above sequence of reads. Next, we consider the sequence $r^2(x)2$ followed by $r^3(x)1$ in the above traces, given as R^2FR^3. However, the above sequence of reads does not match the order among these reads in the serialization Si_p, i.e., $R^2FR^3 \in Si_p$ holds; thus it does not satisfy the first condition in the RHS of Equation 2. Further, for the LHS of Equation 2 to hold for the above sequence of reads, $R^i = R^2$ and $R^j = R^3$, which, in turn, implies $v_i = v_2 = 2$ and $v_j = v_3 = 1$. For the anomaly condition C in Equation 2 to hold for the sequence of read operations $r^1(x)1$ followed by $r^3(x)1$, there cannot exist write operations w^m and w^n corresponding to the reads r^i and r^j such that the LTL condition $\left(v^{i'} = v^{n''}\right) \wedge \left(\left(v^{j'} = v^{m''}\right) \vee \left(\left(W^{m'}FR^{j'} \in St\right) \wedge \left(\nexists W^{m'}FW^{p'}FR^{j'} \in St\right)\right)\right)$ holds. Here, $W^m = W^1$ and $W^n = W^2$, since W^m and W^n are the only write operations in the given session history. Hence, $v_m = v_1 = 1$ and $v_n = v_2 = 2$. Also, it is given that $v_i = v_1 = 1$ and $v_j = v_2 = 2$. Hence, the conditions $\left(v^i = v^n\right)$ and $\left(v^j = v^m\right)$ is satisfied; thus, in turn, the above LTL condition is also satisfied. Thus, the violation condition C in Equation 2 holds. Hence, the above traces violate the MR condition.

Let us consider a session trace st_3: $w^1(x,1), r^1(x)2$, st_3': $r^2(x)1w^2(x,2), r^3(x)1$, and st_3'': $r^4(x)2, r^5(x)1$ to demonstrate a violation of Causal consistency. Since, there are successive operations with direct precedence relation on the object x, st_3 satisfies LHS of the E^s condition in Equation 3. We can transform the above session traces into a partial serialization $Si_p = W^1FR^2FR^3FR^5FW^2FR^1FR^4$. First, let us consider the sequence of operations $w^1(x,1)$ and $r^1(x)2$ in the above traces, given by the LTL expression W^1FR^1. The above expression satisfies the condition $Op^iF^xOp^j$, where $Op^i = W^1$ and $Op^j = R^1$; thus satisfying the LHS of the implies operator $\rightarrow$ in E^s in Equation 3. The above sequence of writes and reads satisfies the condition $W^1FR^1 \in Si_p$, thus satisfying the first condition the RHS of the implies operator $\rightarrow$ in E^s in Equation 3. Following the same line of argument, the sequence of operations $w^1(x,1)$ and $r^2(x)1$, $w^1(x,1)$ and $r^3(x)1$, $w^1(x,1)$ and $r^4(x)2$, $w^1(x,1)$ and $r^5(x)1$ and $w^1(x,1)$ and $w^2(x,2)$ satisfy the LHS and RHS of E^s in the Equation 3. Next, let us consider the sequence of operations $w^2(x,2)$ and $r^3(x)1$, which is expressed using LTL condition $W^2F^xR^3$. Thus, the above sequence satisfies LHS of E^s in the Equation 3. The above sequence of writes and reads do not satisfies the condition $W^2FR^3 \in Si_p$, violating the RHS of E^s in Equation 4.1. Let us consider the anomaly condition C in Equation 3 for the write and read operations $w^2(x,2)$ (which is w^i in this case) and $r^3(x)1$ (which is r^j in this case). Since $w^2(x,2)$ is followed by $r^3(x)1$ in the session trace, they satisfy the LHS of the condition C. For the RHS of C to hold, there cannot a exist write operations w^m preceding the write $w^2(x,2)$ such that the LTL condition $\left(v^{j'} = v^{m''}\right) \vee \left(\left(W^{m'}FOp^{j'} \in st\right) \wedge \left(\nexists W^{m'}FW^{n'}FOp^{j'} \in st\right)\right)$ holds. We can assign $W^m = W^1$ since $w^1(x,1)$ precedes $w^2(x,2)$. Subsequently the condition $\left(v^{j'} = v^{m''}\right)$ holds (since $R^j = R^3$ and $v_j = 1$ in this case). Hence, the RHS of C is satisfied for Equation 3 for the above sequence of operations. It results in violation of Causal consistency condition.

All the above session traces violate the specification of Strict Serializability, given in Equation 4. Consider the session traces

st_3, $st_3^{'}$, and $st_3^{''}$, which can be transformed into the legal serialization Si, which is, in this case, equivalent to Si_p in absence of writes from other client applications in the given session history (see above). The sequence of operations $w^1(x,1)$ and $r^1(x)2$ in the above session trace satisfies the LHS of the condition E^s (i.e., the condition Op^iXOp^j). Since $v_i \neq v_j$, the above sequence of write and read operations do not satisfy the condition in the RHS of E^s, i.e., it violates the condition $W^1XR^1 \in Si$. For the anomaly condition C to hold for the above sequence of operations, let us consider the sequence of operations $w^2(x,2)$ and $r^3(x)1$. For the RHS of C to hold, there cannot a exist write operations r^k following the read $r^3(x)1$ such that the LTL condition $(\left(v^{k^{''}} = v^{i^{'}}\right) \wedge \left(v^{j^{'}} \neq v^{i^{'}}\right)) \wedge \not\exists(\left(Op^{i^{'}} = W^{i^{'}} \oplus R^{i^{'}}\right) \wedge \left(Op^{i^{'}}FW^{j^{'}}FR^{k^{'}} \in st\right) \wedge \left(Op^{i^{'}}FW^{j^{''}}FR^{k^{'}}FR^{l^{'}} \in St\right))$ $(\left(v^{l^{''}} = v^{j^{'}}\right) \wedge \left(v^{k^{''}} = v^{i^{'}}\right))$ holds. We can assign $r^k = r^4$ since $r^4(x)2$ follows $r^3(x)1$. Subsequently the conditions $\left(v^{k^{''}} = v^{i^{''}}\right)$ and $\left(v^{j^{'}} \neq v^{i^{'}}\right)$ holds, since r^j yields v_j = 1, and r^k yields v_k = 2. Thus, it also violates the anomaly condition in Equation 4, violating strict serializability.

We consider the session traces
st_4: $w^1(x,1)$, $st_4^{'}$: $r^1(x)1, w^2(x,2)$, and $st_4^{''}$: $r^2(x)2, r^3(x)1$ to show how WFR consistency can be violated. The above session traces can be transformed into a legal serialization $Si_p = W^1FR^1FR^2FW^2$. The write operation $w^1(x,1)$ follows the read operation $r^1(x)1$, and can be expressed as $W^1F^xR^1$. The above expression satisfies the condition W^iFR^j, where $W^i = W^1$ and $R^j = R^1$. Hence, the above traces satisfy the condition in the LHS of Equation 5. Further, by observation, the above sequence of write-read operations satisfies the condition $W^1FR^1 \in Si_p$. Thus, the condition E^s in the RHS of Equation 5 holds. According to the anomaly condition C in Equation 5, there must not exist a read operation $r^n(x)$ such that the conditions $\left(R^iF^xW^jF^xR^n \in st\right) \wedge \left(W^jF^xR^n \in st\right)$ and $v^n = v^i$ are satisfied. Since $r^2(x)1$ succeeds $w^2(x,2)$, we can assign $R^n = R^2$. But, the read operation $r^2(x)1$ returns a value 1 which was written by $w^1(x,1)$; thus the condition $v^n = v^i$ holds. Thus, st_4 violates the second condition in RHS of Equation 5; hence, it violates WFR consistency.

Let us consider the session traces
st_5: $w^1(x,1), r^1(x)2, w^2(x,2)$, $st_5^{'}$: $r^2(x)2$, and
$st_5^{''}$: $r^3(x)1, r^4(x)2$, and demonstrate violation of MW consistency with it. The above session traces can be transformed in a legal serialization $Si = W^1FW^2FR^1FR^2$. Since the above session traces comprise a sequence of successive writes $w^1(x,1), w^2(x,2)$ on same object x, it can be expressed as W^iFW^j, thus satisfying the LHS of E^s in Equation 6, where $W^i = W^1$ and $W^j = W^2$. Also, the above sequence of write operations satisfies the first condition of the RHS of Equation 6, i.e., $W^iFW^j \in Si$. The above traces can be transformed into a legal serialization order $Si = w^1(x,1), r^2(x)1, w^2(x,2), r^1(x)2$, thus satisfying the first condition in the RHS of Equation 5. For the anomaly condition C to hold, there cannot exist a sequence of write operations $w^1(x,1)$ and $w^2(x,2)$ followed by a sequence of read operations r^k and r^l such that the following condition holds: $\left(\not\exists R^{k^{'}}FR^{l^{'}} \in St\right) \wedge \left(v_k^{'} = v_j^{'}\right) \wedge \left(v_l^{'} = v_i^{'}\right)$. For the above session trace, we assign the operations $r^2(x)2$ and $r^3(x)1$ to the read operation r^k and r^l. However, the above read operations r^k yields v_k = 2, and r^l yields v_l = 1, such that the condition $\left(v_k^{'} = v_j^{'}\right) \wedge \left(v_l^{'} = v_i^{'}\right)$ holds.

Consider a group of 2 clients (or processors), namely Cl_1 and Cl^2, respectively. Let us consider that operations are invoked from the above 2 clients, and a session trace st_6 is collected as follows. st_6: $w^1(x,1), r^1(y)1$, $st_6^{'}$: $w^2(y,1), r^2(x)2$. For the above session trace to satisfy TSO consistency, the condition $\left(W^{i^{'}}(x)FR^{j^{''}}(x) \in St\right) \wedge \left(v_j^{''} \neq v_i^{'}\right) \wedge \left(R^{k^{''}}(x)FR^{j^{''}}(x) \in St\right) \wedge \left(v_k^{''} \neq v_i^{'}\right) \wedge \left(R^{k^{'}}FR^{j^{'}} \in st\right) \wedge \left(W^{l^{'}}FW^{i^{'}} \in St\right) \wedge \left(v^{k^{'}} = v^{i^{'}}\right) \wedge (\left(v^{j^{'}} = v^{l^{'}}\right) \vee (\left(W^{l^{'}}FR^{j^{'}} \in St\right) \wedge \not\exists W^{m^{'}}\left(W^{l^{'}}FW^{m^{'}}FR^{j^{'}} \in St\right)))$ must hold. In the above session traces, let us assign the operation $w^1(x,1)$ as w^i. $r^2(x)2$ occurs in another session trace, and thus invoked was from a different client. Further, $r^2(x)2$ follows $w^1(x,1)$ in the global session order. Hence we can assign the operation $r^2(x)2$ as r^j. Since $r^2(x)2$ returns 2 instead of 1, the condition $v_j^{''} \neq v_i^{'}$ does not hold. Also, there doesn't exist an operation r^k preceding $r^2(x)2$ that satisfies the condition $v_k^{''} \neq v_i^{'}$. Thus, the given session trace violates Equation 7.

Consider two client applications (or processors), namely Cl_1 and Cl^2, respectively. Consider a session trace collected by executing operations on the above client applications as follows. st_7: $w^1(x,1), w^2(y,1)$, $st_7^{'}$: $r^1(x)1$, $st_7^{''}$: $r^2(y)1, r(x)1$. To satisfy the PC consistency, any sequence of write operations w^i and w^j invoked by a client must satisfy the condition $(\not\exists\left(W^{i^{'}}FW^{j^{'}} \in st\right) \wedge \left(st^{'} \in St\right) \wedge \left(R^{k^{'}}FR^{l^{'}} \in st^{'}\right))(\left(v_k^{''} = v_j^{'}\right) \wedge \left(v_l^{'} = v_i^{'}\right))$. Thus, there must not exists a sequence of reads r^k and r^l such that the condition $\left(v_k^{''} = v_j^{'}\right) \wedge \left(v_l^{'} = v_i^{'}\right)$ holds. In the above session traces, the write operations are executed according to the precedence order $w^1(x,1)$ followed by $w^2(y,1)$, hence we assign $w^1(x,1)$ and $w^2(y,1)$ as w^i and w^j. Also, let us assign the sequence of read operations $r(x)1$ and $r^2(y)1$ as r^k and r^l, respectively. According to the above condition, reads against the above writes should view the results of the above write operations according to the invocation order of these writes. However, in the given session trace, the client application Cl_2 observes the above writes out of order. Since $r^2(y)1$ executes before ,$r(x)1$ in the above session traces, Cl_2 observes the result of $w^2(y,1)$ before observing result of $w^1(x,1)$. Thus, the above traces violate Equation 8.

Consider a session trace st_8: $w_{tx}^1(x,2), w_{ty}^2(x,5), w_{ty}^1(y,5)$, $c_{ty}, w_{tx}^2(y,8), r_{ty}^1(x)5, r_{ty}^2(y)8, c_{tx}$. According to the specification of PL-1 isolation level (refer to Equation 9), there cannot exist read operations r_{ty}^m and r_{ty}^n such that they satisfy the condition $\left(\left(v_n^{'} = v_l^{'}\right) \wedge \left(v_m^{'} = v_i^{'}\right)\right) \vee \left(\left(v_n^{'} = v_j^{'}\right) \wedge \left(v_m^{'} = v_k^{'}\right)\right)$. In the session trace st_8, the write operations $w_{tx}^1(x,2), w_{tx}^2(y,8), w_{ty}^2(x,5)$, and $w_{ty}^1(y,5)$ are W_{tx}^i, W_{tx}^j, W_{ty}^k, and W_{ty}^l, respectively. Also, we can assign the read operations $r_{ty}^1(x)5$ and $r_{ty}^2(y)8$ as r_{ty}^m and r_{ty}^n, respectively. Under the above assignment, we have v_m = 8 and v_n = 5, which in turn, satisfies the conditions $\left(v_n^{'} = v_j^{'}\right) \wedge \left(v_m^{'} = v_k^{'}\right)$. This, in turn, satisfies the anomaly condition for PL-1. Thus, st_8 violates PL-1.

Let us consider a session trace
st_9: $w_{tx}^1(x,1), r_{ty}^1(x)1, w_{tx}^2(x,2), c_{ty}, c_{tx}$. According to PL-2

(refer to Equation 10), a transaction never must read a value from an operation if the transaction executing above operation has not yet committed. In other words there must not exist a read operation $r^k(x)_{ty}$ such that $r^k(x)_{ty}$ reads a value written by a write operation $w^i(x)_{tx}$ that does not write the final (committed) version of the object x for the transaction tx, i.e., the condition
$\nexists\big(\big(W^i(x)'_{tx}FW^j(x)'_{tx}Fc_{tx} \in st\big) \wedge \Big(v'_k = v'_i\Big)\big) \wedge$
$\nexists\big(\big(W^i(x)'_{ty}FW^j(x)'_{ty}Fc_{ty} \in st\big) \wedge \Big(v'_k = v'_i\Big)\big)$ does not hold.
In the session trace st_9, the read operation $r^1_{ty}(x)1$ returns a value 1 written by the operation $w^1_{tx}(x,1)$ in transaction tx. However, the write operation $w^2_{tx}(x,2)$, which follows $w^1_{tx}(x,1)$ in st_9, writes the committed version (i.e., the final version) of x, which is 2. Thus, the above condition does not hold for st_9, resulting in violation of PL-2.

Consider a session trace
st_{10}: $r^1_{tx}(tx)50, r^1_{ty}(x){-}50, r^2_{ty}(y)100, w^1_{ty}(x,100), w^2_{ty}(y,50)$, $c_{ty}, w^1_{tx}(y,-50), r^2_{ty}(x)100, r^2_{tx}(y){-}50, c_{tx}$. In st_{10}, the read operation $r^1_{tx}(x)50$ on object x in tx is followed by write operation $w^1_{ty}(x,100)$. The read operation $r^2_{ty}(x)100$ following the commit c_{ty} returns the value 100, indicating a rw dependency between transactions tx and ty. Similarly, the read operation $r^2_{ty}(y)100$ on object y in ty is followed by write operation $w^1_{tx}(y,50)$. The read operation $r^2_{tx}(y){-}50$ following the commit c_{ty} returns the value -50, implying a read-write dependency between ty and tx. Thus, there is a read-write dependency cycle in S_{11}, violating PL-3 specifications. Let us consider a session trace st_{11}: $r^1_{tx}(tx)50, r^1_{ty}(x){-}50$, $r^2_{ty}(x)100, w^1_{ty}(x,100), w^2_{ty}(y,50), c_{ty}, w^1_{tx}(x,-50), r^2_{ty}(x)100$, $r^2_{tx}(x){-}50, c_{tx}$ Following similar line of reasoning, we can observe that the session trace st_{11} violates the isolation level PL-2.99, given in Equation 12.