A Consistent Understanding of Consistency
Regular Submission

Subhajit Sidhanta, Affiliation: INESC-ID
Email: ssidhanta@gsd.inesc-id.pt
Address: INESC-ID, R. Alves Redol 9, 1000-029 Lisboa
Phone: +351 912413257

Ricardo J. Dias, Affiliation: NOVA LINCS, Universidade NOVA de Lisboa & SUSE Linux GmbH

Rodrigo Rodrigues, Affiliation: INESC-ID/IST (U. Lisboa)

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 validating 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.

A Consistent Understanding of Consistency

SUBHAJIT SIDHANTA*, INESC-ID
RICARDO J. DIAS, NOVA LINCS, Universidade NOVA de Lisboa & SUSE Linux GmbH
RODRIGO RODRIGUES, INESC-ID/IST (U. Lisboa)

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 validating 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.

ACM Reference format:
Subhajit Sidhanta, Ricardo J. Dias, and Rodrigo Rodrigues. 2016. A Consistent Understanding of Consistency. 1, 1, Article 1 (January 2016), 13 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

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 [6]. However, the task of understanding and comparing the semantics provided by storage systems, normally encapsulated in a consistency definition, 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

*The corresponding author

DOI: 10.1145/nnnnnnn.nnnnnnn

offer [10, 17, 21]. 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 [19]). 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 [13, 19, 19]. 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 [7, 12], 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 discusses relevant related work. Section 3 presents the assumed system model, and basic definitions and terminology used in the paper. Section 4 presents a general format of specifications based on ConSpec. Section 5 lists the specifications of various consistency models, expressed using ConSpec. Section 6 presents a reformulation of the CAP theorem, and formally proves the same. Section 7 analyzes the lattice of consistency models in terms of the restated CAP theorem. Section 8 presents the design of a prototype automated verification tool that can validate a session trace with respect to a ConSpec specification.

2 RELATED WORK

Consistency definitions restrict the set of valid traces for the execution of a given system with a storage-like interface. In broad terms, the gold standard of consistency definitions are so-called "strong" consistency levels, which have the characteristic of approximating the behavior that is obtained when interacting with a system whose implementation has a single, centralized server that executes operations one at a time. There are several examples of such consistency models [3, 5, 14, 18, 20, 24, 26], most of which include precise definitions of their intended semantics.

Strong consistency is often forfeited by the algorithms that implement these storage systems, in order to achieve better performance (e.g., when processors cache possibly stale data in a multiprocessor) and/or

better availability (e.g., when multiple replicas of the data exist and operations proceed while contacting only a subset of them) [11, 17, 23, 25, 27, 28].

Often, the definitions of these consistency models is vague and/or underspecified. For instance, the strong consistency option of Cassandra, a widely used NoSQL storage system, is stated in terms of the size of the quorums that are used for read and write operations, leaving unspecified what happens as the system reconfigures and the set of replicas of a data item changes [16]. Even when the specification is more precise, it may suffer from being tied to implementation details that may not be widely applicable. For instance, some definitions assume the existence of a centralized server that keeps monotonically increasing version number associated with the data [1]; others explicitly define consistency in terms of the state maintained by different replicas of the data [19]; finally, Terry et al. defined a set of session guarantees that strengthen the consistency offered by eventually consistent systems, but these are defined operationally in terms of the replicas that are accessed and the operations that these replica process and the respective order in which they are processed [30].

With the goal of obtaining more precise and less operational definitions, Chockler et al. [9] defined these session guarantees [and a few other definitions, namely processor consistency, sequential consistency, and causal sonsistency] using first-order logic expressions. Even more broadly, Burckhardt et al. [8] present the definitions of a wide variety of consistency models using properties like visibility and happens before order as building blocks. Our definitions build on recent proposals for new consistency models that are weakly consistent by default but distinguish a subset of the operations, and enforce visibility restrictions among those operations [13, 19, 19]. In contrast, our goal is not to propose a new consistency model, but to gain a deeper understanding of existing ones.

Compared to these prior approaches, ConSpec provides a generic way to describe consistency specifications, where each consistency level corresponds to a different parameterization of our generic definition. Furthermore, we use our generic framework to generalize the CAP theorem as proved by Gilbert and Lynch [12]. Finally, we provide a software artifact to check whether traces meet a certain consistency level.

Other authors have explored the CAP theorem beyond its original formulation and first proof. Mahajan et al. [22] defined Real Time Causal Consistency, a stronger variant of causal consistency, and proved that it is the strongest consistency model that can be provided in a highly available and eventually consistent implementation. Recently, Attiya et al. [4] provided a formal specification of systems that implement causal consistency, and proved that Observable Causal Consistency, a stronger variant of causal consistency, is the strongest consistency model that can be provided in a highly available, partition tolerant manner. In addition, their definitions are tied to implementation-level concepts, namely a set of replicas connected by a network. In contrast, we generalize CAP by defining necessary and sufficient conditions for consistency models to be implemented in an available and partition-tolerant way, which are applied to precisely characterize the CAP line in a hierarchy of consistency models.

3 BASIC DEFINITIONS

We assume a set of processes (or clients) that interface the storage system by invoking operations. An operation comprises an invocation and the respective response. A very large class of consistency definitions (namely those describing the behavior 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 many of the consistency levels we describe require this interface, we will also incorporate this division into our definitions. Note, however, that our definitions 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, whenever required by the consistency definition, we allow for the possibility that the interface allows the programmer to specify an object o associated with reads and writes.

Our execution models each process as a deterministic state machine, whose transitions can be triggered either by an external input (i.e., an operation invocation) or by receiving a message from the network, and where the transition can trigger sending messages and/or issuing outputs (i.e., an operation response). A *session trace* st is a sequence of operations $o \in O$ executed by the same client, ordered by the time when they were invoked. In this paper we assume that clients are well-formed, i.e., a client only invokes an operation after the preceding operation has returned its response value. As such, session traces can be modelled as sequences of pairs $o = \langle invocation, response \rangle$. Invocations (resp. responses) belong to a generic set of possible invocations $\mathcal{I}$ (resp. responses $\mathcal{R}$). We define a session invocation trace sit as the sequence of invocations that are obtained from transforming each element in a session trace using the projection operator to obtain only the invocations. We define a session invocation trace to be compatible with a session trace if the projection of the invocations in the session trace matches the session invocation trace (denoted $st \bowtie sit$).

In our notation, we denote an invocation of a write operation that writes a value v to an object x, as $w(x, v)$ (with an empty response). Conversely, a read operation on object x that outputs a value v' is denoted $r(x)v'$. This allows us to define a shorthand notation for a session trace $st \in \mathcal{S}_t$ as a sequence of tuples $\langle w, v \rangle$ or $\langle r, v' \rangle$. In our formulas, when a quantifiers restricts an operation to a given type (read or write), we simply use the letters R and W, i.e., we write $\forall R \in O$ as a shorthand for $\forall r(x)v \in O$.

The *global session trace* $\mathcal{S}_t$ (resp. global session invocation trace $\mathcal{S}_{it}$) denotes the set of all session traces (resp. session invocation traces) in a given execution of the system.

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 F_{st} to restrict the LTL operator "eventually" to operations comprised in the session trace st. E.g., an expression $w\, F_{st}\, r$ denotes that the operation $w(x, v)$ is followed by the operation $r(x)v$ in a session trace st.

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 x must output the value associated with the most recent write operation to the same object x. For other types of interfaces (e.g., in state machine replication), that specification may change, and in some cases is specific to the service interface (e.g., the state machine being replicated).

4 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. Note that ConSpec focuses only on safety conditions. Defining liveness conditions is left as future work.

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 [13, 19, 19]. 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 models as imposing different restrictive conditions on this precedence. Such a restrictive condition is then expressed as an LTL expression E^s.

Definition 4.1. Generalized form of ConSpec: Given a global session trace $\mathcal{S}_t$, we say that $\mathcal{S}_t$ satisfies a consistency model C if there exists a partial order $(O_{St}, \preccurlyeq)$ over the set O_{St} comprising operations present in all session traces in $\mathcal{S}_t$, i.e., $O_{St} = \bigcup_{st \in St}\{o \mid o \in st\}$, such that 1) for every operation o in $\mathcal{S}_t$, its output is equal to the one obtain by executing the sequential specification of a linear extension of the operations preceding o in $\preccurlyeq$, and 2) $(O_{St}, \preccurlyeq)$ obeys E^s, which is an LTL expression restricting $(O_{St}, \preccurlyeq)$.

Condition 1, when applied to a system whose interface consists only of reads and writes, translates to a requirement that every read operation in $\mathcal{S}_t$ must return the value of the most recent write according to $\preccurlyeq$. (In the case of two or more concurrent preceding writes, the system must arbitrate an order for them, e.g., use the "last writer wins" policy [31].) Condition 2 can be expressed as $E^s \models (O_{St}, \preccurlyeq)$, where E^s is the ConSpec parameterization for each consistency model C, and $\models$ is the satisfies operator.

5 SPECIFYING EXISTING MODELS

In this section, we present the ConSpec specifications for several common consistency models documented in the literature [8, 9, 29]. Subsequently, in the Appendix, we discuss how these definitions correspond to their original counterparts. We start by specifying a series of session guarantees, which were originally proposed by Terry et al. in the context of a mobile computing storage system called Bayou [29]. 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 tied to some implementation-level concepts, 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 formal definitions for these session guarantees [8, 9].

We start with the Read Your Writes (RYW) session guarantee, which informally precludes a read operation r from reading a value for object x that precedes a value the same client previously wrote to the same object in the same session. Following the format of Definition 4.1, RYW can be defined by the following restrictions on $\preccurlyeq$.

$$\forall st \in \mathcal{S}_t \quad E^s = \forall\, W', R' \in st :\ W'\, F_{st}\, R' \Rightarrow W' \preccurlyeq_{st+w} R', \tag{1}$$

where $st \in \mathcal{S}_t$ is the session from the standpoint of which the session guarantees are being upheld, and $\preccurlyeq_{st+w}$ denotes the restriction of $\preccurlyeq$ to the elements of st and the write operations of all other clients.

The Read (or Session) Monotonic, also called Monotonic Reads guarantee (MR) specifies intuitively that read operations on a given object invoked from the same session must always return results in an increasing order of recency. MR is formally represented by the following expression constraining the partial order defined by ConSpec.

$$\forall st \in \mathcal{S}_t \quad E^s = \forall\, R', R'' \in st :\; R'\, F_{st}\, R'' \Rightarrow R' \preccurlyeq_{st+w} R'', \tag{2}$$

with the same meaning for st and $\preccurlyeq_{st+w}$ as in the previous definition.

Write Follows Read (WFR) specifies the following: a write operation that follows a read operation in the same session must be ordered after the write operation seen by that read. WFR is expressed in ConSpec as follows.

$$\forall st \in \mathcal{S}_t \quad E^s = \forall\, R', W' \in st :\; R'\, F_{st}\, W' \Rightarrow R' \preccurlyeq_{st+w} W', \tag{3}$$

with the same meaning for st and $\preccurlyeq_{st+w}$ as in RYW.

Finally, the last session guarantee is the Monotonic Writes (MW) model, which is specified as follows: successive write operations invoked from the same session must be applied in the order in which they appear in the session trace. MW is expressed in ConSpec as follows.

$$\forall st \in \mathcal{S}_t \quad E^s = \forall\, W', W'' \in st :\; W'\, F_{st}\, W'' \Rightarrow W' \preccurlyeq_{st+w} W'', \tag{4}$$

with the same meaning for st and $\preccurlyeq_{st+w}$ as in RYW.

Our generic formulation allows for an interesting perspective on the definition of causal consistency [2], which can be expressed as the union of the previous session guarantees for all sessions. As such, any pair of operations in the same session need to be constrained by the partial order of the ConSpec definition:

$$E^s = \forall\; st \in \mathcal{S}_t,\; Op', Op' \in st :\; Op'\, F_{st}\, Op'' \Rightarrow Op' \preccurlyeq Op'', \tag{5}$$

with the subtle difference that this is a global property, instead of being specific to a given session trace $st \in \mathcal{S}_t$ and the respective subset of the relation $\preccurlyeq$.

For Processor Consistency [2], we consider the more well known Goodman's definition of Processor Consistency (PC) over the alternate definition used implemented in the DASH system. Goodman's PC specifies: write operations performed from each client application must be observed by all clients (i.e., must occur in all session traces) according to their invocation orders. Thus, PC can be expressed as follows.

$$\begin{aligned} E^s = &\forall st \in \mathcal{S}_t,\; R'(x), R''(y), W'(x), W''(y) \in st : \\ &W'(x)\, F_{st}\, W''(y) \Rightarrow W'(x) \preccurlyeq W''(y) \wedge R'(x)\, F_{st}\, R''(y) \Rightarrow R'(x) \preccurlyeq R''(y). \end{aligned} \tag{6}$$

where $\preccurlyeq$ is a partial order over the global session trace $\mathcal{S}_t$.

Sequential consistency (SC) is a consistency model which requires that a global execution comprising operations executed from one or more clients must be equivalent to the result of executing the operations in a sequential order, such that the mutual order among operations from each client in the sequence

respects the invocation order of the operations defined in the client. Using ConSpec, SC is expressed as follows.

$$E^s = \forall st \in \mathcal{S}_t,\ Op, Op' \in st,\ Op'', Op''' \in O_{St} : \\ Op\, F_{st}\, Op' \Rightarrow Op \preccurlyeq Op' \quad \wedge \quad (Op'' \preccurlyeq Op''' \vee Op''' \preccurlyeq Op'') \tag{7}$$

Figure 1 shows an example execution where the global session trace contains two sessions, each with three operations (two writes followed by a read). This execution meets all the consistency levels we defined with the exception of sequential consistency. The right hand side of the figure shows a partial order that can be used to support the execution, and that meets the constraints specified by all the consistency levels except those specified by sequential consistency. For sequential consistency, the constraints would require the partial order to be a total order, which is impossible to achieve while also obeying the session order and explaining the results that are observed according to the sequential specification of a read/write interface. This is because one would have to serialize both reads before the respective writes of value 99, but that would be impossible to achieve in a total order that respects the session orders.

Session 1	Session 2
w(x,1)	w(y,1)
w(x,99)	w(y,99)
r(y)1	r(x)1

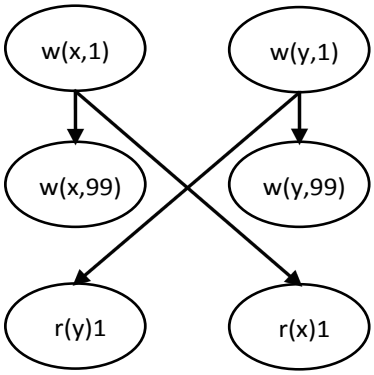

Fig. 1. System Model

6 REWRITING THE CAP THEOREM IN TERMS OF CONSPEC

The CAP conjecture was initially stated informally as the impossibility of simultaneously achieving strong Consistency, Availability, and Partition tolerance in a replicated system [7]. When this theorem was subsequently proven by Gilbert and Lynch [12], these three properties were stated precisely, and, in this context, strong consistency was defined as atomicity (or linearizability [14]).

The fact that the original proof of CAP is restricted to linearizability raises the question of whether CAP holds using other definitions from the wide array of consistency models supported by modern storage systems. In this section, we rewrite the CAP theorem in terms of ConSpec to precisely define the class of consistency models that can and cannot be implemented in a highly-available, partition-tolerant way.

To begin with, we need a helper definition to enumerate all admissible partial orders for a given restriction condition E^s and set of operations in a global session invocation trace $\mathcal{S}_{it}$.

Definition 6.1 (Partial order enumeration). Given a global session invocation trace $\mathcal{S}_{it}$ and a restrictive condition E^s for a ConSpec definition, we define the partial order enumeration of this session invocation trace and condition, $\Pi(\mathcal{S}_{it}, E^s)$ as the set of partial orders over the elements of any compatible session trace $\mathcal{S}_{it}$ that are valid under E^s, i.e.:
$\Pi(\mathcal{S}_{it}, E^s) \equiv \{\preccurlyeq: \exists \mathcal{S}_t \ (E^s \ \vDash \ (\mathcal{S}_t, \preccurlyeq) \wedge \mathcal{S}_t \bowtie \mathcal{S}_{it})\}$

This allows us to define the following necessary and sufficient condition for a consistency model to have an available and partition tolerant implementation.

THEOREM 6.2 (EXTENDED CAP THEOREM). *In an asynchronous system, it is possible to implement a consistency model E^s while simultaneously providing availability and partition tolerance if and only if for any global session invocation trace $\mathcal{S}_{it}$ and all of its partial orderings that are allowed by E^s, when you consider the set of maxima of each partial order, it is always possible to make them depend only on the previous operation in the same session and still obtain a valid partial order, i.e.:*
$\forall \mathcal{S}_{it} \ \forall \preccurlyeq \in \Pi(\mathcal{S}_{it}, E^s) \ \forall o \in max(\mathcal{S}_{it}, \preccurlyeq)$ (RemoveAllExceptSession$(\preccurlyeq, o) \in \Pi(\mathcal{S}_{it}, E^s)$)
where we define REMOVEALLEXCEPTSESSION *as a partial order where the maximum o is only directly ordered after prior operations in the same session, i.e.:*
REMOVEALLEXCEPTSESSION$(\preccurlyeq, o) \equiv \preccurlyeq \setminus \{\langle o', o\rangle$*, where $\langle o', o\rangle$ belongs to the transitive reduction of $\preccurlyeq$, and o' does not belong to the same session as o*$\}$.

Proof: We start by proving the implication in the direction ($\Rightarrow$). Following the proof style of Gilbert et al. [12], we prove this by contradiction as follows. Let us assume, by contradiction, that consistency model E^s is implemented by an algorithm that is highly available during partitions but does not meet the condition at the end of the Theorem. Let us consider initially that there are only two clients, c_1 and c_2, with sessions $s1$ and $s2$, respectively. The fact that E^s does not meet this condition means precisely that there must exist a global session trace $\mathcal{S}_t$ with a valid partial order $\preccurlyeq$ that has a maximum element o_{s1} such that $o_{s2} \preccurlyeq o_{s1}$ (where o_{s1}, o_{s2} belong to $s1$ and $s2$), and where it is not admissible to have a partial order where $o_{s2} \not\preccurlyeq o_{s1}$.

Now let us construct the following execution. First, we run the system under the exact same conditions that produced $\mathcal{S}_t$ until client c_1 is about to execute o_{s1} and client c_2 is about to execute o_{s2}. At this point, a partition occurs that separates c_1 and c_2, which persists until the end of the execution. By the availability and partition-tolerance properties, the operations o_{s1} and o_{s2} will eventually complete and, by our initial

assumption in the previous paragraph, the former operation must see the effects of the latter, i.e., the partial order that supports that execution must be such that $o_{s2} \preccurlyeq o_{s1}$. Now run the exact same execution, but where the client c_2 crashes right before invoking o_{s2}. This execution is indistinguishable from the previous one from the standpoint of c_1. Thus c_1 will follow the same sequence of states and produce the same outputs as in the previous execution. This would mean that the algorithm would not meet its ConSpec specification, since o_{s1} would reflect the execution of an operation that was not part of the global session trace $\mathcal{S}_t$, namely o_{s2}. This contradicts the fact that the algorithm that was used meets that specification and the CAP properties.

The assumption about there being only two clients does not lose generality because, with more clients, a pair of clients c_1, c_2 under the conditions above must also exist. Then the proof generalizes beyond two clients by the partitioning the clients into two sets, one containing c_1 and another containing c_2, and crashing all the clients in the same side of the partition as c_2.

Next we focus on the implication in the direction ($\Leftarrow$). Here, we need to prove that if a consistency model E^s meets the condition:

$\forall \mathcal{S}_{it}\ \forall \preccurlyeq \in \Pi(\mathcal{S}_{it}, E^s)\ \forall o \in max(\mathcal{S}_{it}, \preccurlyeq)$ (REMOVEALLEXCEPTSESSION$(\preccurlyeq, o) \in \Pi(\mathcal{S}_{it}, E^s)$)

then it has an available and partition-tolerant implementation.

Given any global session invocation trace $\mathcal{S}_{it}$, we prove this by induction on the length of the execution that produced $\mathcal{S}_{it}$. The base case with an empty execution is vacuously true, since an empty trace meets any consistency condition (no safety properties are ever violated by an empty trace). For the induction step, we need to prove that, given an execution for which an available and partition-tolerant implementation produced a trace that conforms to E^s, it is possible for a client to invoke a new operation and produce an output that is also consistent. This is true because, even in the case that the client that invokes the new operation is partitioned from the remaining of the clients, it is always legal to produce an operation that depends on a prior operation from the same session and all the operations that transitively precede it according to $\preccurlyeq$. Furthermore, the valid output of this operation can be determined by using only information that is local to the session, by running the sequential specification of the system on the graph of preceding operations.

7 ANALYSIS OF CONSISTENCY MODELS WITH RESPECT TO CAP

The previous section defined necessary and sufficient conditions for a consistency model C to have an available and partition-tolerant implementation. Now, we analyze the E^s-expressions for the consistency models that we studied in Section 5, to determine how they fare with respect to Theorem 6.

We can see that the session gurantees (MR, MW, RYW, WFR) and both the causal and processor consistency definitions are only forcing constraints on the partial ordering across operations from the same session. This implies that these constraints are compatible with the conditions on the right hand side of the equivalence of Theorem 6. In particular, it is the case that it is always legal to remove orderings between operations across sessions, since these are never constrained by the implications in the various different E^s expressions. Therefore, we conclude that Causal Consistency, Processor Consistency, and all four session guarantees are not affected by CAP, i.e., can have highly available and partition-tolerant implementations.

In contrast, SC requires that the visibility order $\preccurlyeq$ among operations from all the clients in the system forms a total order. This implies that if an operation is related by the transitive reduction of $\preccurlyeq$ to a

previous operation from another session, it is not possible to remove this element of the partial order and still obtain a valid partial order, since it would violate the condition in the definition of SC that any two operations need to be ordered with respect to each other. Thus, this does not meet the necessary and sufficient condition for a partition-tolerant, highly available implementation.

8 IMPLEMENTATION

We provide an open source automated verification tool built using Spin [15], an open source software verification framework. The source code of the ConSpec tool and instructions for running it is made publicly available in a github repository https://github.com/ssidhanta/ConSpecTool. A given session trace is supplied to the tool as an input. The expression E^s for a particular consistency model C is specified as a Spin LTL formula. E^s acts as a safety property, and the system behaviour is modelled from a given session trace using the PROMELA meta language. The PROMELA source file provides an abstract model of interaction of the system with clients. The tool systematically attempts to verify whether a given session trace is valid under a given consistency model C, specified in terms of E^s. Internally, Spin translates the PROMELA source file into C code. The Spin driver then runs the built-in model checker to check for counter-examples for the above generated C code against the Spin formula extracted from E^s. The Spin model checker validates the generated C code with the Spin formula as the invariant, with the set of equivalent legal serializations generated from the given session trace supplied as an input to the model checkers.

9 CONCLUSIONS

In this paper, we presented a generic framework called ConSpec for defining consistency. ConSpec enables definitions that are precise, follow a generic structure, and are independent of implementation details. We used ConSpec to derive several concrete definitions of existing consistency levels. Furthermore, ConSpec also enabled a generic version of the CAP theorem, where the "C" property is not longer tied to a specific strong consistency definition. Instead, we define necessary and sufficient conditions for a consistency level to be within the scope of CAP, i.e., for the existence or not of a partition tolerant and available implementation. Furthermore, we developed an automated tool for verifying whether a given session trace satisfies a consistency model.

ConSpec opens several interesting avenues for future work. First, we intend to apply ConSpec to a wider range of consistency models. Second, we intend to extend it to support isolation levels of transactional systems, where the visibility of individual operations within a transaction must be constrained. Finally, we intend to further develop our automatic verification tools and promote their adoption by the developer community.

REFERENCES

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

[2] M. Ahamad, R. A. Bazzi, R. John, P. Kohli, and G. Neiger. The power of processor consistency. In *Proceedings of the Fifth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '93, pages 251–260, New York, NY, USA, 1993. ACM.

[3] M. Ahamad, G. Neiger, J. E. Burns, P. Kohli, and P. Hutto. Causal memory: Definitions, implementation and programming, 1994.

[4] H. Attiya, F. Ellen, and A. Morrison. Limitations of highly-available eventually-consistent data stores. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC '15, pages 385–394, New York, NY, USA, 2015. ACM.

[5] H. Attiya and J. L. Welch. Sequential consistency versus linearizability. *ACM Trans. Comput. Syst.*, 12(2):91–122, May 1994.

[6] 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.

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

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

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

[10] 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.

[11] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, Oct. 2007.

[12] 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.

[13] 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.

[14] 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.

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

[16] D. Inc. Configuring data consistency. http://docs.datastax.com/en/archived/cassandra/2.0/cassandra/dml/dml_config_consistency_c.html, 2017.

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

[18] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Comput.*, 28(9):690–691, Sept. 1979.

[19] 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.

[20] R. Lipton and J. S. Sandberg. PRAM : a scalable shared memory. Technical Report CS-TR-180-88, Princeton University (NJ US), 1988.

[21] 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.

[22] P. Mahajan, L. Alvisi, and M. Dahlin. Consistency, availability, convergence. Technical Report TR-11-22, Computer Science Department, University of Texas at Austin, May 2011.

[23] C. Meiklejohn. Riak PG: Distributed process groups on dynamo-style distributed storage. In *Erlang '13*.

[24] M. Mizuno, M. Raynal, and J. Z. Zhou. *Sequential consistency in distributed systems*, pages 224–241. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

[25] E. Plugge, T. Hawkins, and P. Membrey. *The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing*. Apress, Berkely, CA, USA, 1st edition, 2010.

[26] M. Raynal and A. Schiper. *From causal consistency to sequential consistency in shared memory systems*, pages 180–194. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

[27] T. Schütt, F. Schintke, and A. Reinefeld. Scalaris: Reliable transactional P2P key/value store. In *Proceedings of the 7th ACM SIGPLAN Workshop on ERLANG*, ERLANG '08, pages 41–48, New York, NY, USA, 2008. ACM.

[28] R. Sumbaly, J. Kreps, L. Gao, A. Feinberg, C. Soman, and S. Shah. Serving large-scale batch computed data with project Voldemort. In *Proc. of the 10th USENIX Conference on File and Storage Technologies (FAST)*, 2012.

[29] 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.

[30] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *SOSP '95*.

[31] R. H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Syst.*, 4(2):180–209, June 1979.