Quantitative Analysis of Consistency in NoSQL Key-Value Stores*

Si Liu[1], Jatin Ganhotra[2], Muntasir Raihan Rahman[3], Son Nguyen[4], Indranil Gupta[5], and José Meseguer[6]

1 **Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA**
siliu3@illinois.edu
2 **Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA**
jatin.ganhotra@gmail.com
3 **Microsoft, Redmond, WA, USA**
murahman@microsoft.com
4 **Addepar Inc., Sunnyvale, CA, USA**
son.nguyen@addepar.com
5 **Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA**
indy@illinois.edu
6 **Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA**
meseguer@illinois.edu

Abstract

The promise of high scalability and availability has prompted many companies to replace traditional relational database management systems (RDBMS) with NoSQL key-value stores. This comes at the cost of relaxed consistency guarantees: key-value stores only guarantee eventual consistency in principle. In practice, however, many key-value stores seem to offer stronger consistency. Quantifying how well consistency properties are met is a non-trivial problem. We address this problem by formally modeling key-value stores as probabilistic systems and quantitatively analyzing their consistency properties by both statistical model checking and implementation evaluation. We present for the first time a formal probabilistic model of Apache Cassandra, a popular NoSQL key-value store, and quantify how much Cassandra achieves various consistency guarantees under various conditions. To validate our model, we evaluate multiple consistency properties using two methods and compare them against each other. The two methods are: (1) an implementation-based evaluation of the source code; and (2) a statistical model checking analysis of our probabilistic model.

2012 ACM Subject Classification Cloud computing, Key-value stores, Model Checking, Rewrite systems
Keywords and Phrases NoSQL Key-value Store, Consistency, Statistical Model Checking, Rewriting Logic, Maude
Digital Object Identifier 10.4230/LITES-v004-i001-a003
Received 2015-12-25 **Accepted** 2016-12-04 **Published** 2016-12-23

Special Issue Editors Javier Campos, Martin Fränzle, and Boudewijn Haverkort
Special Issue Quantitative Evaluation of Systems

* This work was partially supported by NSF CNS 1319527, NSF 1409416, NSF CCF 0964471, and AFOSR/AFRL FA8750-11-2-0084.

1 Introduction

The promise of high scalability and availability has prompted many companies and organizations to replace traditional relational database management systems (RDBMS) with NoSQL key-value stores in order to store large data sets and support an increasing number of users. According to DB-Engines Ranking [16] by September 2016, three NoSQL datastores, namely MongoDB [30], Cassandra [12] and Redis [33], have advanced into the top 10 most popular database engines among 315 systems, highlighting the increasing popularity of NoSQL key-value stores. For example, Cassandra is currently being used at Facebook, Netflix, eBay, GitHub, Instagram, Comcast, and over 1500 more companies.

NoSQL key-value stores invariably replicate application data on multiple servers for greater availability in the presence of failures. Brewer's CAP theorem [11] implies that, under network partitions, a key-value store must choose between consistency (keeping all replicas in sync) and availability (latency). Many key-value stores prefer availability, and thus they provide a relaxed form of consistency guarantees (e.g., eventual consistency [39]). This means key-value store applications can be exposed to stale values. This can negatively impact key-value store user experience. Not surprisingly, in practice many key-value stores seem to offer stronger consistency than they promise. Therefore there is considerable interest in accurately predicting and quantifying what consistency properties a key-value store actually delivers, and in comparing in an objective, and quantifiable way how well properties of interest are met by different designs.

However, the task of accurately predicting such consistency properties is non-trivial. To begin with, building a large scale distributed key-value store is a challenging task. A key-value store usually embodies a large number of components (e.g., membership management, consistent hashing, and so on), and each component can be thought of as source code which embodies many complex design decisions. Today, if a developer wishes to improve the performance of a system (e.g., to improve consistency guarantees, or reduce operation latency) by implementing an alternative design choice for a component, then the only option currently available is to make changes to huge source code bases (e.g., Apache Cassandra [12] has about 345,000 lines of code). Not only does this require many man months, it also comes with a high risk of introducing new bugs, needs understanding in a huge code base before making changes, and is unfortunately not repeatable. Developers can only afford to explore very few design alternatives, which may in the end fail to lead to a better design.

In this paper we address these challenges by proposing a formally model-based methodology for designing and quantitatively analyzing key-value stores. We formally model key-value stores as probabilistic systems specified by *probabilistic rewrite rules* [1], and quantitatively analyze their properties by *statistical model checking* [34, 43]. We demonstrate the practical usefulness of our methodology by developing, to the best of our knowledge for the first time, a formal probabilistic model of Cassandra, as well as of an alternative Cassandra-like design, in Maude [13]. Our formal probabilistic model extends and improves a nondeterministic one we used in [26] to answer *qualitative* binary consistency queries[1] about Cassandra. It models the main components of Cassandra and its environment such as strategies for ordering multiple versions of data and message delay. We have also specified five consistency guarantees that are largely used in industry, *strong consistency* (the strongest consistency guarantee), *causal consistency* (the strongest consistency guarantee that is available in the presence of partitions), *read your writes*, *monotonic reads* and *consistency prefix* (popular intermediate consistency guarantees) [37, 38, 28, 4], in the

[1] A binary consistency query may ask, for example, whether a key-value store read operation is strongly (resp. weakly) consistent or not and other such Boolean-valued questions.

QuaTEx probabilistic temporal logic [1]. Using the PVeStA [3] statistical model checking tool, we have then quantified the satisfaction of such consistency properties in Cassandra under various conditions such as consistency level combination and operation issuing time latency. To illustrate the versatility and ease with which different design alternatives can be modeled and analyzed in our methodology, we have also modeled and analyzed the exact same properties for an alternative Cassandra-like design.

An important question is how much trust can be placed on such models and analysis. That is, how reliable is the predictive power of our proposed methodology? We have been able to answer this question for our case study as follows: (i) we have experimentally evaluated the same consistency properties for both Cassandra and the alternative Cassandra-like design[2]; and (ii) we have compared the results obtained from the formal probabilistic models and the statistical model checking with the experimentally-obtained results. Our analysis indicates that the model-based consistency predictions conform well to consistency evaluations derived experimentally from the real Cassandra deployment, with both showing that Cassandra in fact achieves much higher consistency (sometimes up to strong consistency) than the promised eventual consistency. They also show that the alternative design in most cases is not competitive in terms of the consistency guarantees considered. The purpose of the alternative design was two-fold. First, we wanted to see if the alternative approach would give better accuracy or not. Second, we also wanted to show that it was easier to try the alternative design with our formal model. This is one of the key benefits of building formal models, so that we can quickly try alternative approaches and check their accuracy and performance. Our entire Maude specification, including the alternative design, has less than 1000 lines of code, which further underlines the versatility and ease of use of our methodology at the software engineering level.

Our main contributions include:

- We present a formal methodology for the quantitative analysis of key-value store designs and develop, to the best of our knowledge for the first time, a *formal executable probabilistic model* for the Cassandra key-value store and for an alternative Cassandra-like design.
- We present, to the best of our knowledge for the first time, a statistical model checking analysis for quantifying five consistency guarantees in Cassandra and the alternative design.
- We demonstrate the good predictive power of our methodology by comparing the model-based consistency predictions with experimental evaluations from a real Cassandra deployment on a real cluster. Our results indicate similar consistency trends for the model and the deployment.

This paper extends the results presented in [24, 26] on two substantial ways. First, we explain various consistency models in greater detail including their formal definitions (Section 3) and specifications (Section 5.1). Second, we provide quantitative analysis of consistency using both statistical model checking and implementation-based estimation for three new consistency models (Section 5.2): monotonic reads, consistent pefix and causal consistency [37, 38, 28, 4].

The paper is organized as follows: Section 2 gives a brief overview of Cassandra, Maude, and statistical model checking. Section 3 explains replicated data consistency guarantess in NoSQL datastores. Section 4 presents our probabilistic model of Cassandra and an alternative Cassandra-like design. Section 5 shows the quantitative analysis of consistency by statistical model checking and the implementation-based estimation. Finally, Sections 6 and 7 discuss related work and concluding remarks, respectively.

[2] We implemented the alternative Cassandra-like design by modifying the source code of Apache Cassandra version 1.2.10.

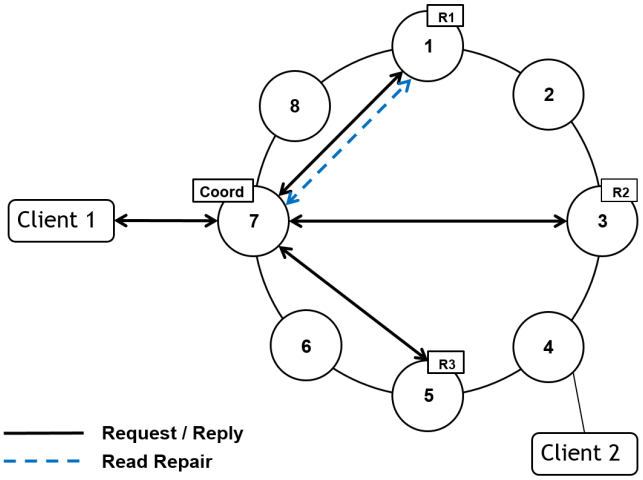

Figure 1 Cassandra deployed in a single cluster of 8 servers with replication factor 3.

2 Preliminaries

2.1 Cassandra Overview

Apache Cassandra [12] is a distributed, scalable, and highly available NoSQL database design. It is distributed over collaborative servers that appear as a single instance to the end client. Data items are dynamically assigned to several servers in the cluster (called the *ring*), and each server (called a *replica*) is responsible for different ranges of the data stored as key-value pairs. Each key-value pair is stored at multiple replicas for fault-tolerance. To place those replicas different strategies can be employed, e.g., the *Simple Strategy* places replicas clockwise in a single data center. For a private cloud Cassandra is typically deployed in a single data center with a single ring structure shared by all its servers.

In Cassandra a client can perform *read* or *write* operations to query or update data. When a client requests a read/write operation to a cluster, the server it is connected to will act as a *coordinator* to forward the request to all replicas that hold copies of the requested key-value pair. According to the specified *consistency level* in the operation, the coordinator will reply to the client with a value/ack after collecting *sufficient* responses from replicas. Cassandra supports tunable consistency levels, with `ONE`, `QUORUM` and `ALL` being the three major ones, meaning that the coordinator will respectively reply with the most recent value (namely, the value with the highest timestamp) to the client after hearing from *one* replica, a *majority* of replicas, or *all* replicas. Thus we call the above strategy of processing reads *Timestamp-based Strategy* (TB). Note that replicas may return different timestamped values to the coordinator. To ensure that all replicas agree on the most recent value, Cassandra uses in the background the *read repair* mechanism to update those replicas holding outdated values.

Figure 1 shows an example Cassandra system deployed in a single data center cluster of eight servers with three replicas and consistency level `QUORUM`. The read/write from client `1` is forwarded to all three replicas `1`, `3` and `5`. The coordinator `7` then replies to client `1` after receiving the first two responses, e.g., from `1` and `3`, to fulfill the request without waiting for the reply from `5`. For a read, upon retrieving all three possibly different versions of values, the coordinator `7` then issues a read repair write with the highest timestamped value to the outdated replica, `1`, in this example. Note that various clients may connect to various coordinators in the cluster, but requests from any client on the same key will be forwarded to the same replicas by those coordinators.

2.2 Rewriting Logic and Maude

Maude [13] is an expressive rewriting-logic-based formal specification language and high-performance simulation and model checking tool for concurrent, object-oriented systems. Maude specifications are executable, and the tool provides a variety of formal analysis methods, including simulation, reachability analysis, and linear temporal logic (LTL) model checking.

A Maude module specifies a *rewrite theory* $(\Sigma, E \cup A, R)$, where:

- Σ is an algebraic *signature*; that is, a set of *sorts*, *subsorts*, and *function symbols*.
- $(\Sigma, E \cup A)$ is a *membership equational logic theory* [13], with E a set of possibly conditional equations and membership axioms, and A a set of equational axioms such as associativity, commutativity, and identity, so that equational deduction is performed *modulo* the axioms A. The theory $(\Sigma, E \cup A)$ specifies the system's states as members of an algebraic data type.
- R is a collection of *labeled conditional rewrite rules* $[l] : t \longrightarrow t'$ **if** *cond*, specifying the system's local transitions.

We briefly summarize the syntax of Maude and refer to [13] for more details. Operators are introduced with the `op` keyword: `op` f `:` $s_1 \ldots s_n$ `->` s. They can have user-definable syntax, with underbars '`_`' marking the argument positions. Equations and rewrite rules are introduced with, respectively, keywords `eq`, or `ceq` for conditional equations, and `rl` and `crl`. The mathematical variables in such statements are declared with the keywords `var` and `vars`, or can be introduced on the fly, in which case they have the form $var{:}sort$. An equation $f(t_1, \ldots, t_n) = t$ with the `owise` ("otherwise") attribute can be applied to a subterm $f(\ldots)$ only if no other equation with left-hand side $f(u_1, \ldots, u_n)$ can be applied.

A *class* declaration `class` C `|` att_1 `:` s_1`, ...,` att_n `:` s_n declares a class C of objects with attributes att_1 to att_n of sorts s_1 to s_n. An *object instance* of class C is represented as a term `<` O `:` C `|` $att_1 : val_1, ..., att_n : val_n$ `>`, where O, of sort `Oid`, is the object's *identifier*, and where val_1 to val_n are the current values of the attributes att_1 to att_n. A *message* is a term of sort `Msg`. A system state is modeled as a term of the sort `Configuration`, and has the structure of a *multiset* made up of objects and messages. The dynamic behavior of a system is axiomatized by specifying each of its transition patterns by a rewrite rule. For example, the rule (with label `l`)

```
rl [l] :  m(O,w)
          < O : C | a1 : x, a2 : O', a3 : z >
       =>
          < O : C | a1 : x + w, a2 : O', a3 : z >
          m'(O',x) .
```

defines a family of transitions in which a message `m`, with parameters `O` and `w`, is read and consumed by an object `O` of class `C`, the attribute `a1` of the object `O` is changed to `x + w`, and a new message `m'(O',x)` is generated. Attributes whose values do not change and do not affect the next state, such as `a3` and `a2`, need not be mentioned in a rule. Note that in the above rule `O`, declared as either constant or variable of sort `Oid`, is the object's identifier that can also be a parameter of some function (e.g., the message function `m`).

2.3 Statistical Model Checking and PVeStA

Distributed systems are probabilistic in nature, e.g., network latency such as message delay may follow a certain probability distribution, plus some algorithms may be probabilistic. Systems of this kind can be modeled by *probabilistic rewrite theories* [1] with rules of the form:

$$[l] : t(\overrightarrow{x}) \to t'(\overrightarrow{x}, \overrightarrow{y}) \;\; if \;\; cond(\overrightarrow{x}) \;\; with \;\; probability \;\; \overrightarrow{y} := \pi(\overrightarrow{x})$$

where the term t' has additional new variables $\overrightarrow{y}$ disjoint from the variables $\overrightarrow{x}$ in the term t. Since for a given matching instance of the variables $\overrightarrow{x}$ there can be many (often infinite) ways to instantiate the extra variables $\overrightarrow{y}$, such a rule is *non-deterministic*. The probabilistic nature of the rule stems from the probability distribution $\pi(\overrightarrow{x})$, which depends on the matching instance of $\overrightarrow{x}$, and governs the probabilistic choice of the instance of $\overrightarrow{y}$ in the result $t'(\overrightarrow{x}, \overrightarrow{y})$ according to $\pi(\overrightarrow{x})$. In this paper we use the above PMaude [1] notation for probabilistic rewrite rules.

Statistical model checking [34, 43] is an attractive formal approach to analyzing probabilistic systems against temporal logic properties. Instead of offering a binary yes/no answer, it provides a quantitative real-valued answer and can verify a property up to a user-specified level of confidence by running Monte-Carlo simulations of the system model. For example, if we consider strong consistency in Cassandra, a statistical model-checking result may be "The Cassandra model satisfies strong consistency 86.87% of the times with 99% confidence". The quantitative answer, however, need not be a percentage or a probability: it may instead be a latency estimation, or a quantitative estimation of some other QoS property. Existing statistical verification techniques assume that the system model is purely probabilistic. Using the methodology in [1, 19] we can eliminate non-determinism in the choice of firing rules. We then use PVeStA [3], an extension and parallelization of the tool VeStA [35], to statistically model check purely probabilistic systems against properties expressed by QuaTEx probabilistic temporal logic [1]. The expected value of a QuaTEx expression is iteratively evaluated w.r.t. two parameters α and δ provided as input by sampling until the size of `(1-`α`)100%` confidence interval is bounded by δ. In this paper we will compute the expected probability of satisfying a property based on definitions of the form `p() = BExp ; eval E[# p()] ;`, where `#`, called "next", is a primitive temporal operator, `BExp` is a consistency-specific predicate (e.g., the predicate `sc?` in Section 5.1.1), and `p()` is a state predicate returning the probabilistic number between 1.0 and 0.0. Informally, the QuaTEx expression consists in a list of definitions of recursive temporal operators such as `p()`, followed by a query of the expected value of a path expression such as `eval E[# p()]` obtained combining the temporal operators.

3 Replicated Data Consistency

Distributed key-value stores usually sacrifice consistency for availability (Brewer's CAP theorem [11]), advocating the notion of weak consistency (e.g., Cassandra promises eventual consistency [12]). However, studies on benchmarking eventually consistent systems have shown that those platforms seem in practice to offer more consistency than they promise [40, 10]. Thus a natural question derived from those observations is "what consistency does your key-value store provide in practice?". We summarize below the prevailing consistency guarantees that have received considerable attention in recent research on distributed data stores [37, 38, 28, 4]. We will focus on five of them (*strong consistency*, *read your writes*, *monotonic reads*, *consistent prefix* and *causal consistency*) in the rest of this paper.

- Strong Consistency (SC) ensures that each read returns the value of the last write that occurred before that read.
- Read Your Writes (RYW) guarantees that the effects of all writes performed by a client are visible to her subsequent reads.
- Monotonic Reads (MR) ensures a client to observe a key-value store increasingly up to date over time.
- Consistent Prefix (CP) guarantees a client to observe an ordered sequence of writes starting with the first write to the system.

- (Time-) Bounded Staleness (BS) restricts the staleness of values returned by reads within a time period.
- Causal Consistency (CC) guarantees that the effects are observed only after their causes: reads will not see a write unless its dependencies are also seen.
- Eventual Consistency (EC) claims that if no new updates are made, eventually all reads will return the last updated value.

Note that SC and EC lie at the two ends of the consistency spectrum, while the other intermediate guarantees are not comparable in general [37].

In [26, 27] we investigated SC, RYW and EC from a *qualitative* perspective using standard model checking, where they were specified using *linear temporal logic* (LTL). The questions we asked and answered there were simply yes/no questions such as "Does Cassandra satisfy strong consistency?" and "In what scenarios does Cassandra violate read your writes?". We indeed showed by counterexamples that Cassandra violates SC and RYW under certain circumstances, e.g., successive write and read with the combinations of lower consistency levels. Regarding EC, the model checking results of our experiments with bounded number of clients, servers and messages conforms to the promise. We refer the reader to [26, 27] for details.

In this paper we look into the consistency issue for Cassandra in terms of SC, RYW, MR, CP and CC from a *quantitative*, statistical model checking perspective. To aid the specification of the five properties (Section 5.1), we now restate them more formally. As all operations from different clients can be totally ordered by their issuing times, we can first view, from a client's perspective, the behavior of a key-value store S as a history $H = o_1, o_2, \ldots, o_n$ of n read/write operations, where any operation o_i can be expressed as $o_i = (k_i, v_i, c_i, t_i)$, where t_i denotes the *global* time when o_i was issued by client c_i, and v_i is the value read from or written to on key k_i and where the sequence is time-increasing, i.e., $t_i \leq t_j$ if $i < j$. We can then define the consistency properties based on H:

- We say S satisfies SC if for any read $o_i = (k, v_i, c_i, t_i)$, provided there exists a write $o_j = (k, v_j, c_j, t_j)$ with $t_j < t_i$, and without any other write $o_h = (k, v_h, c_h, t_h)$ such that $t_j < t_h < t_i$, we have $v_i = v_j$. Note that c_h, c_i and c_j are not necessarily different;
- We say S satisfies RYW if either (1) S satisfies SC, or (2) for any read $o_i = (k, v_i, c_i, t_i)$, provided there exists a write $o_j = (k, v_j, c_j, t_j)$ with $c_i = c_j$ and $t_j < t_i$, and with any other write $o_h = (k, v_h, c_h, t_h)$ such that $c_i \neq c_h$ and $t_j < t_h < t_i$, we have $v_i = v_j$;
- We say S satisfies MR if for any two reads $o_{ri} = (k_{ri}, v_{ri}, c_{ri}, t_{ri})$ and $o_{rj} = (k_{rj}, v_{rj}, c_{rj}, t_{rj})$ in the sequence of reads $o_{r1}, o_{r2}, \ldots, o_{rn}$, provided there exists a sequence of writes $o_{w1}, o_{w2}, \ldots, o_{wm}$, if $v_{ri} = v_{wg}$ and $v_{rj} = v_{wh}$, we have $t_{wg} \leq t_{wh}$, provided $k_{ri} = k_{wg}$ and $k_{rj} = k_{wh}$. Note that c_{ri}, c_{rj}, c_{wg} and c_{wh} are not necessarily different;
- We say S satisfies CP if for a sequence of reads $o_{r1}, o_{r2}, \ldots, o_{rm}$, provided there exists a sequence of writes $o_{w1}, o_{w2}, \ldots, o_{wn}$, we have, if $n \geqslant m$, $v_{r1} = v_{w1}, v_{r2} = v_{w2}, \ldots, v_{rm} = v_{wm}$, provided that $k_{r1} = k_{w1}, k_{r2} = k_{w2}, \ldots, k_{rm} = k_{wm}$; otherwise, $v_{r1} = v_{w1}, v_{r2} = v_{w2}, \ldots, v_{rn} = v_{wn}, v_{rn+1} = v_{wn}, \ldots, v_{rm} = v_{wn}$, provided $k_{r1} = k_{w1}, k_{r2} = k_{w2}, \ldots, k_{rn} = k_{wn}, k_{rn+1} = k_{wn}, \ldots, k_{rm} = k_{wn}$. Note that c_{ri} and c_{wj} are not necessarily different, and the key pairs (e.g., k_{ri} and k_{rj}) are not necessarily the same;
- We introduce two notions, *causality order* and *serialization* [2, 38], before defining S satisfying CC. We say $o_i = (k_i, v_i, c_i, t_i)$ is *causally ordered* before $o_j = (k_j, v_j, c_j, t_j)$ if one of the following three cases holds: (i) $c_i = c_j$ and $t_i < t_j$; (ii) $v_j = v_i$, provided o_i is a write while o_j a read; (iii) there exists some other o_k such that o_i is causally ordered before o_k that is causally ordered before o_j (i.e., transitivity). We say L is a *serialization* of a history H if L is a linear sequence containing exactly the operations of H such that L satisfies SC. We say a serialization L *respects* a causality order if, for any two operations o_i and o_j in L, o_i is causally

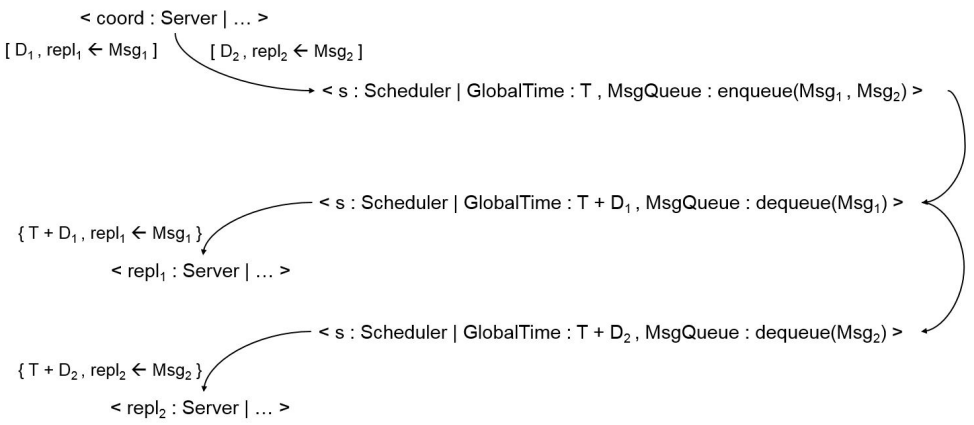

Figure 2 Visualization of rewrite rules for forwarding requests from a coordinator to the replicas.

ordered before o_j implies o_i precedes o_j in L. We then say S satisfies CC if H has a causality order such that for each client c_i there is a serialization of all operations from c_i and all writes (probably from different clients) in H that respects that causality order.

4 Probabilistic Modeling of Cassandra Designs

This section describes a formal probabilistic model of Cassandra as well as an alternative Cassandra-like design. Section 4.1 shows the underlying communication model of Cassandra components, and how the associated non-deterministic rewrite rules are transformed into purely probabilistic ones in Maude. Section 4.2 presents an alternative Cassandra-like design in terms of read processing strategy. The entire executable Maude specifications are available at `https://sites.google.com/site/siliunobi/lites-cassandra`.

4.1 Formalizing Probabilistic Communication in Cassandra

In [26] we built a formal executable model of Cassandra summarized in Section 2.1. Specifically, we modeled the ring structure, clients and servers, messages, and Cassandra's dynamics. Moreover, we also introduced a *scheduler* object to schedule messages by maintaining a global clock `GlobalTime`[3] and a queue of inactive/scheduled messages `MsgQueue`. By activating those messages, it provides a deterministic total ordering of messages[4] and allows synchronization of all clients and servers, aiding formal analysis of consistency properties (Section 5.1).

To illustrate the underlying communication model, Figure 2 visualizes a segment of the system transitions showing how messages flow between a coordinator and the replicas through the scheduler in terms of rewrite rules. The delayed messages (of the form `[...]`) `[D1, repl1 <- Msg1]` and `[D2, repl2 <- Msg2]`, targeting replicas `repl1` and `repl2`, are produced by the coordinator at global time `T` with the respective message delays `D1` and `D2`. The scheduler then enqueues both messages for scheduling. As the global time advances, messages eventually become active (of the

[3] Although in reality synchronization can never be exactly reached due to clock skew [23], cloud system providers use NTP or even expensive GPS devices to keep all clocks synchronized (e.g., Google Spanner [15]). Thus our abstraction of a global clock is reasonable.

[4] It is possible to have two active messages with the same delivery time. In that case messages will be ordered alphabetically.

form {...}), and are appropriately delivered to the replicas. For example, the scheduler first dequeues `Msg1` and then `Msg2` at global time `T + D1` and `T + D2` respectively, assuming `D1 < D2`. Note that messages can be consumed by the targets only when they are active.

As mentioned in Section 2.3, we need to eliminate nondeterminism in our previous Cassandra model prior to statistical model checking. This can be done by transforming nondeterministic rewrite rules to purely probabilistic ones. Below we show an example transformation, where both rules illustrate how the coordinator reacts upon receiving a read reply `ReadReplySS` from a replica, with `KV` the returned key-value pair of the form `(key,value,timestamp)`, `ID` and `A` the read and client's identifiers, and `CL` the read's consistency level, respectively. The coordinator `S` adds `KV` to its local buffer, and returns to `A` the highest timestamped value determined by `tb` via the message `ReadReplyCS`, provided it has collected the consistency-level number of responses determined by `cl?`.

In the nondeterministic version `[...-nondet]`, the outgoing message is equipped with a delay `D` nondeterministically selected from the delay set `delays` where `DS` defines the rest delays of the set. We keep the set unchanged so that standard model checking will explore all possible choices of delays each time the rule is fired. For example, if `delays` are : `(2.0,4.0)`, two read replies will be generated nondeterministically with the delays `2.0` and `4.0` time units respectively, each of which will lead to an execution path during the state space exploration. Note that `AS` refers to the rest of the attributes of server `S`.

```
crl [on-rec-rrep-coord-nondet] :
    {T, S <- ReadReplySS(ID,KV,CL,A)}
    < S : Server | buffer: BF, delays: (D,DS), AS >
 =>
    < S : Server | buffer: BF', delays: (D,DS), AS >
    (if cl?(CL,BF') then
     [D, A <- ReadReplyCS(ID,tb(BF'))]
     else none fi)
 if BF' := add(ID,KV,BF) .
```

We transform the above rule to the probabilistic version `[...-prob]`, where the delay `D` is distributed according to the parameterized probability distribution function `distr(...)`. Once the rule fires, only one read reply will be generated with a probabilistic real-valued message delay.

```
crl [on-rec-rrep-coord-prob] :
    {T, S <- ReadReplySS(ID,KV,CL,A)}
    < S : Server | buffer: BF, AS >
 =>
    < S : Server | buffer: BF', AS >
    (if cl?(CL,BF') then
     [D, A <- ReadReplyCS(ID,tb(BF'))]
     else none fi)
 if BF' := add(ID,KV,BF) with probability D := distr(...) .
```

Likewise, all nondeterministic rules in our previous model can be transformed to purely probabilistic rewrite rules. Furthermore, as explained in [1, 19], the use of continuous time and the actor-like nature of the specification ensure that *at most one probabilistic rule will be enabled at each time instant*, thus eliminating any remaining nondeterminism from the firing of rules.

4.2 Alternative Strategy Design

Two major advantages of our model-based approach are: (1) the ease of exploring different design alternatives (e.g., designing an alternative strategy to TB described in Section 2.1) in early design stages, and (2) the ability to predict their effects before implementation. Here we illustrates the first part by presenting as an alternative design the *Timestamp-agnostic Strategy* (TA). The key idea is that, instead of using timestamps to decide which value will be returned to the client as TB does (Section 2.1), TA uses the values themselves to decide which replica has the latest value. For example, if the replication factor is 3, then for a `QUORUM` read, the coordinator checks whether the values returned by the first two replicas are identical: if they are, the coordinator returns that value; otherwise it waits for the third replica to return a value. If the third value matches one of the first two values, the coordinator returns the third value. So for a `QUORUM` read TA guarantees that the coordinator will reply with the value that has been stored at a majority of replicas. For an `ALL` read, the coordinator compares all three values; if they are all the same, it returns that value. Notice that TA and TB agree on processing `ONE` reads.

To formalize TA (or other alternative strategies) we only need to specify the corresponding functions of the returned values from the replicas buffered at the coordinator, as we defined `tb` for TB, and for deciding if enough responses have been fetched by the coordinator, as we defined `cl?` for TB, without redefining the underlying model. Note that our component-based model also makes it possible to dynamically choose the optimal strategy in favor of consistency guarantees. More precisely, once the system has been specified endowed with a family of strategies having respective strengths in consistency guarantees (which can be measured by statistical model checking), the coordinator can invoke the corresponding strategy-specific function based on the client's specified preference. For example, given strategies S1/S2 offering consistency properties C1/C2, if a client issues two consecutive reads with desired consistency C1, C2, respectively, the coordinator will generate, e.g., the C1-consistent value for the preceding read, by calling the strategy function for S1.

On the other hand, the key change for implementing TA is to implement our own `Resolve()` function for class `RowDataResolver`. We rewrote Cassandra's `resolveSupersetNew()` function for the TA approach. This took some code modification, and it required a complete understanding of the Cassandra internal code details. In terms of Cassandra code changes, 150 lines of code were added and 50 were removed. We observe that formalizing TA in Maude was much easier and faster than implementing it in Cassandra.

5 Quantitative Analysis of Consistency in Cassandra

How well do our Cassandra model and its TA alternative design satisfy different consistency guarantees? Does TA provide better consistency than TB based on our model? Are those results in agreement with actual implementations? We propose to investigate these questions by statistical model checking and by implementation-based evaluation of those consistency properties in terms of the two strategies.

5.1 Formalization of Consistency Properties

5.1.1 Strong Consistency

Scenario. SC ensures that each read returns the value of the last write that occurred before that read. Thus we design a scenario with one read and two writes to see if the read will return the

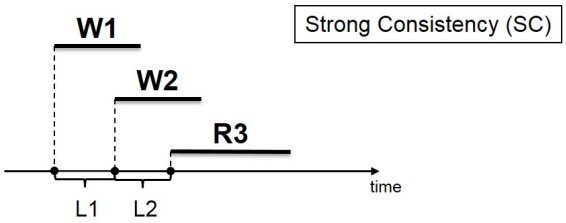

Figure 3 Experimental Scenario of Statistical Model Checking of SC.

last write[5]. Figure 3 shows the experimental scenario for SC, with each parallel line denoting one session of one client. The scenario consists of three consecutive operations, W1, W2 and R3, issued by three different clients, respectively, where `L1` and `L2` are the issuing latencies between them. We choose to experiment with consistency level `ONE` for both W1 and W2 to evaluate different consistency levels for R3. Thus we name each subscenario (TB/TA-O/Q/A) depending on the target strategy and R3's consistency level, e.g., (TB-Q) refers to the case checking SC for TB with R3 of `QUORUM`.

Formal Specification of SC. Based on the consistency definition (Section 3) and the above scenario, SC is satisfied if R3 reads the value of W2. Thus we define a parameterized predicate `sc?(A,A',C)` that holds if we can match the value returned by the subsequent read `O` (R3 in this case) from client `A` with that in the preceding write `O'` (W2 in this case) from client `A'`. Note that the attribute `store` *lists* the associated information of each operation issued by the client, e.g., operation `O` was issued at global time `T` on key `K` with returned/written value `V` for a read/write. Note that in the rest of this paper `REST` refers to the rest objects of the entire configuration (Section 2.2).

```
op sc? : Address Address Config -> Bool .

eq sc?(A,A',< A  : Client | store : ((O, K,V,T ), ...), ... >
            < A' : Client | store : ((O',K,V,T'), ...), ... > REST) = true .

eq sc?(A,A',C) = false [owise] .
```

5.1.2 Read Your Writes

Scenario. RYW guarantees that the effects of all writes performed by a client are visible to her subsequent reads. Thus we design a scenario with two writes and one read from the same client to see if the read will return the last write; we also add another write from a different client since returning the latest write from a different client will also satisfy RYW. Figure 4 shows the experimental scenario for RYW. The scenario consists of four operations, where W1, W2 and R3 are issued by one client and *strictly ordered* (a subsequent operation will be blocked until the preceding one finishes) while W4 is from the other client[6]. The issuing latency `L` is tunable, which

[5] Note that we need to minimize all the experimental scenarios for the statistical model checking in this paper for two reasons: (1) statistical model checking can only support a limited number of operations to avoid state space explosion; and (2) we observed that in our experiments adding a few more operations would not affect the results much but would aggravate the statistical model checking.

[6] Section 3 describes two disjoint cases for RYW, which we mimic with tuneable `L`: if `t4 < t2`, only W2 is RYW-consistent; otherwise both W2 and W4 are RYW-consistent.

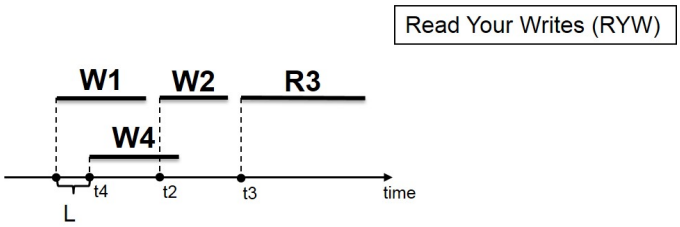

Figure 4 Experimental Scenario of Statistical Model Checking of RYW.

can vary the issuing time of W4. Thus we can derive the corresponding cases in RYW's definition (Section 3), and specify and analyze the property accordingly. We choose to experiment with consistency level `ONE` for both W1 and W4 to evaluate different combinations of consistency levels for W2 and R3. The only possible cases violating RYW are, if we forget W4 for the moment, (R3,W2) = (O,O)/(O,Q)/(Q,O) due to the fact that a read is guaranteed to see its preceding write from the same client, if `R + W > RF` with `R` and `W` the respective consistency levels and `RF` the replication factor. For example, if the replication factor `RF` is 3 (thus `ONE/QUORUM/ALL` is 1/2/3), a `ONE` read `R` and a `QUORUM` write `W` cannot ensure RYW.

Formal Specification of RYW. We define a parameterized predicate `ryw?(A,A',C)` that holds if the subsequent read `O2` (R3 in this case) returns the appropriate value required for RYW to hold. This can happen in two different ways: (1) `O2` returns the value `V` by the preceding write `O1` (W2 in this case), or (2) `O2` returns the value `V'` by a more recent write `O3` (W4 in this case if issued after W2, which is determined by `T3 >= T1`).

```
op ryw? : Address Address Config -> Bool .

eq ryw?(A,A',< A  : Client | store : (..., (O1,K,V,T1), (O2,K,V,T2), ...),
     ... > REST) = true .   --- case (1)

ceq ryw?(A,A',< A  : Client | store : (..., (O1,K,V,T1), (O2,K,V',T2), ...), ... >
        < A' : Client | store : ((O3,K,V',T3), ...), ... > REST) = true
     if T3 >= T1 .   --- case (2)

eq ryw?(A,A',C) = false [owise] .
```

5.1.3 Monotonic Reads

Scenario. MR ensures that a client will observe a key-value store increasingly up to date over time. Thus we design a scenario with two writes followed by two consecutive reads to see if the two reads will return the two writes in order. Figure 5 shows the experimental scenario for MR. We consider a scenario with four operations, where two writes, W1 and W2, are issued by two different clients, and a third client issues two strictly ordered reads, R3 and R4. `L1` and `L2` are the issuing latencies between W1 and W2, and W2 and R3, respectively. We choose to experiment with the same consistency level for R3 and R4, and with consistency level `ONE` for W1 to evaluate different combinations of consistency levels for W2 and R3/R4.

Formal Specification of MR. We define a parameterized predicate `mr?(A1,A2,A3,C)` that holds if the subsequent read `O4` (R4 in this case) from client `A3` returns the appropriate value required for MR to hold. This can happen in three different ways: (1) if the preceding read `O3` (R3 in this case) gets the default value *dft*, any value returned by `O4` is MR-consistent; (2) if `O3` reads the

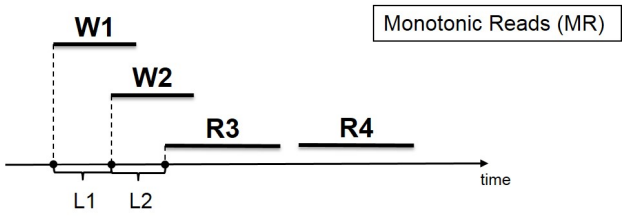

Figure 5 Experimental Scenario of Statistical Model Checking of MR.

value V1 from client A1's write O1 (W1 in this case), O4 has to return the value V1 or V2 (from client A2's write O2) to satisfy MR; and (3) if O3 returns V2, O4 needs to match it to guarantee MR.

```
op mr? : Address Address Address Config -> Bool .

eq mr?(A1,A2,A3,< A3 : Client | store: ((O3,K,dft,T3), (O4,K,V4,T4), ...),
     ... > REST) = true .    --- case (1)

eq mr?(A1,A2,A3,< A1 : Client | store: ((O1,K,V1,T1), ...), ... >
                < A3 : Client | store: ((O3,K,V1,T3), (O4,K,V1,T4), ...),
     ... > REST) = true .
eq mr?(A1,A2,A3,< A1 : Client | store: ((O1,K,V1,T1), ...), ... >
                < A2 : Client | store: ((O2,K,V2,T2), ...), ... >
                < A3 : Client | store: ((O3,K,V1,T3), (O4,K,V2,T4), ...),
     ... > REST) = true .    --- case (2)

eq mr?(A1,A2,A3,< A2 : Client | store: ((O2,K,V2,T2), ...), ... >
                < A3 : Client | store: ((O3,K,V2,T3), (O4,K,V2,T4), ...),
     ... > REST) = true .    --- case (3)

eq mr?(A1,A2,A3,C) = false [owise] .
```

5.1.4 Consistent Prefix

Scenario. CP guarantees that a client will observe an ordered sequence of writes starting with the first write to the system. Thus we design a scenario with six operations on two different keys[7], where four strictly ordered writes, W1, W2, W3 and W4, are interleaved on two keys, K1 and K2, and issued by a single client, and two strictly ordered reads, R1 (on K1) and R2 (on K2), by the other client. Figure 6 shows the experimental scenario for CP. The issuing latency L between W1 and R1 is tunable, which can vary the issuing time of R1. We choose to experiment with the same consistency level for the writes/reads to evaluate different combinations of consistency levels.

Formal Specification of CP. We define a parameterized predicate cp?(A1,A2,C) that holds if the successive reads O5 (R5 in this case) and O6 (R6 in this case) from client A2 return an ordered sequence of writes. This can happen in three different ways: (1) if O5 gets the default value *dft1* for K1, O6 can only read the default value *dft2* for K2 to satisfy CP; (2) if O5 returns the value V1 from client A1's write O1 (W1 in this case), O6 has to return the value V2 or *dft2* (either case

[7] For reads on a single key in a system like Cassandra where writes completely overwrite previous values of a key, eventual consistency reads can guarantee consistent prefix. So we consider reads on multiple keys [37].

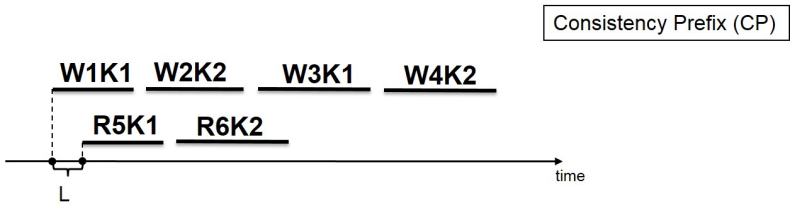

Figure 6 Experimental Scenario of Statistical Model Checking of CP.

guarantees an ordered sequence of writes); and (3) if O5 reads the value V3 from the write O3 (W3 in this case), both V2 and V4 are CP-consistent.

```
op cp? : Address Address Config -> Bool .

eq cp?(A1,A2,< A2 : Client | store: ((O5,K1,dft1,T5),
     (O6,K2,dft2,T6), ...), ... > REST) = true .    --- case (1)

eq cp?(A1,A2,< A1 : Client | store: ((O1,K1,V1,T1), ...), ... >
             < A2 : Client | store: ((O5,K1,V1,T5), (O6,K2,dft2,T6),
     ...), ... > REST) = true .
eq cp?(A1,A2,< A1 : Client | store: ((O1,K1,V1,T1), (O2,K2,V2,T2), ...), ... >
             < A2 : Client | store: ((O5,K1,V1,T5), (O6,K2,V2,T6),
     ...), ... > REST) = true .    --- case (2)

eq cp?(A1,A2,< A1 : Client | store: (..., (O2,K2,V2,T2), (O3,K1,V3,T3), ...), ... >
             < A2 : Client | store: ((O5,K1,V3,T5), (O6,K2,V2,T6),
     ...), ... > REST) = true .
eq cp?(A1,A2,< A1 : Client | store: (..., (O3,K1,V3,T3), (O4,K2,V4,T4), ...), ... >
             < A2 : Client | store: ((O5,K1,V3,T5), (O6,K2,V4,T6),
     ...), ... > REST) = true .    --- case (3)

eq cp?(A1,A2,C) = false [owise] .
```

5.1.5 Causal Consistency

Scenario. CC guarantees that the effects are observed only after their causes: reads will not see a write unless its dependencies are also seen. Thus we design a scenario with a write and a read from two different clients to bridge them by causality. Figure 7 shows the experimental scenario for CC. We consider a scenario with five operations on two different keys, where client C1 issues two strictly ordered writes, W1 and W2, on the keys, K1 and K2, respectively, and the other client C2 first issues a read on K2, and then issues two consecutive operations W4 and R5 on K1. Note that, although operations from the same session of a client are causally ordered (e.g., W1 and W2), we check in this scenario causality that arises when the first read R3 returns the value of W2, establishing the causality between W2 and R3, and thus W1 and W4 (i.e., corresponding to the definition of CC (Section 3), we check from client C2's perspective if there is a serialization of all five operations that respects the causality order). The issuing latency L between W1 and R3 is tunable, which can vary the issuing time of R3. We choose to experiment with the same consistency level ONE for W1, W2 and R3 to evaluate different combinations of consistency levels for W4 and R5.

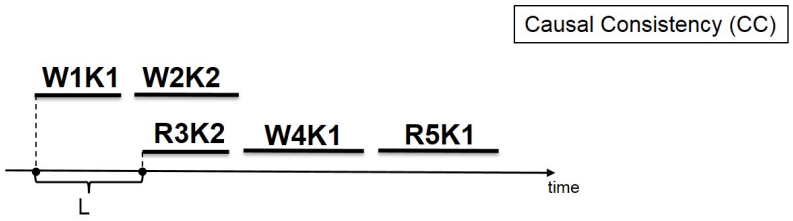

Figure 7 Experimental Scenario of Statistical Model Checking of CC.

Formal Specification of CC. We define for CC a parameterized predicate `cc?(A1,A2,C)` that holds if `O5` (R5 in this case) returns the value `V4` from client `A2`'s write `O4` (W4 in this case), provided `O3` (R3 in this case) reads the value `V2` from client `A1`'s write `O2` (W2 in this case); otherwise, any value returned by `O5` is CC-consistent, since no causality arises.

```
op cc? : Address Address Config -> Bool .

eq cc?(A1,A2,< A1 : Client | store: ((O1,K1,V1,T1), (O2,K2,V2,T2), ...), ... >
             < A2 : Client | store: ((O3,K2,V2,T3), (O4,K1,V4,T4), (O5,K1,V4,T5),
     ...), ... > REST) = true .   --- causality arises

eq cc?(A1,A2,< A1 : Client | store: ((O1,K1,V1,T1), (O2,K2,V2,T2), ...), ... >
             < A2 : Client | store: ((O3,K2,dft2,T3), (O4,K1,V4,T4), (O5,K1,V5,T5),
     ...), ... > REST) = true .   --- no causality

eq cc?(A1,A2,C) = false [owise] .
```

5.2 Analysis Results for Consistency Guarantees

We are now ready to present the analysis results for all those consistency models on both statistical model checking and implementation-based evaluation. We compare the two strategies, TA and TB, from two aspects: (1) if both show the similar trends as the latency increases; and (2) if one provides better consistency than the other.

First, we define the following setting for our experimental scenarios of statistical model checking:

- We consider a single cluster of 4 servers, and the replication factor of 3.
- All replicas are initialized with default key-value pairs.
- Each read/write can have consistency level `ONE`, `QUORUM` or `ALL`.
- We consider the lognormal distribution for message delay with the mean μ `= 0.0` and standard deviation σ `= 1.0` [9].
- All consistency probabilities are computed with a 99% confidence level of size at most 0.01 (Section 2.3).

Note that for the rest of this paper we name each subscenario (TB/TA-OO/OQ/QO/...) depending on the target strategy and the combination of consistency levels for the experimental scenarios regarding the consistency models excluding SC. For simplicity, we let an operation occur immediately upon its preceding operation from the same session (or client) finishes.

Regarding the experimental setup on the other hand, we deploy Cassandra on a single cluster of 4 Emulab [41] servers within the same rack.[8] The servers used were Dell r710 2U servers with

[8] For a private cloud, Cassandra is typically deployed in a single data center with a single ring structure shared by all its servers. The network latency effects will not impact our experiment results as all servers are in

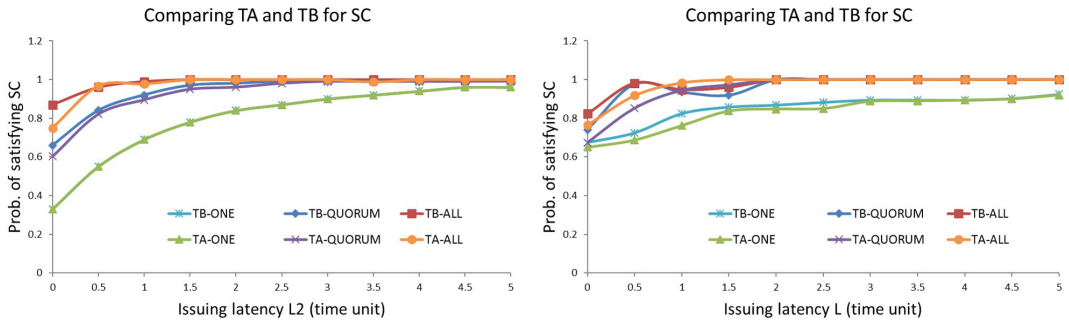

Figure 8 Probabilities of Satisfying SC by Statistical Model Checking (Left) and by Real Deployment Run (Right).

the following configuration: one 2.4 GHz 64-bit Quad Core Xeon E5530 Nehalem processor, 5.86 GT/s bus speed, 8 MB L3 cache, VT (VT-x, EPT and VT-d) support and 12 GB 1066 MHz DDR2 RAM; the OS used was 64-bit Ubuntu 14.04 LTS, 3.13.0 kernel and Cassandra version was 1.2.10. We use YCSB [14] to inject read/write workloads. For RYW, MR, CP and CC tests, we use two separate YCSB clients.[9] Our test workloads are read-heavy (which are representative of many real-world workloads such as Facebook's photo storage [8]) with 90% reads, and we vary consistency levels between `ONE`, `QUORUM`, and `ALL`. We run Cassandra and YCSB clients for fixed time intervals (each client performs 20,000 operations) and log the results. Based on the logs generated, we calculate the percentage of reads that satisfy various consistency models considered in this paper. We perform binning on all the results to generate the issuing latency intervals. The reason is that in our statistical model checking experiments we can specify the exact latency interval between operations, but the same is impossible for a real deployment. From our real deployment experiments, we get the probabilities for a continuous range of latency values, instead of a discrete set as in the statistical model checking experiments. Thus we divide all the results and perform binning to generate the issuing latency intervals. Note that other experimental configurations such as the replication factor, consistency levels and message delay distribution follow our setup for statistical model checking.

5.2.1 Analysis Results for SC

The left plot in Figure 8 shows the resulting probability of satisfying SC by statistical model checking, where the probability (of R3 reading the value of W2) is plotted against the issuing latency (`L2`) between them. Regarding TB, from the results (and intuitively), given the same issuing latency, increasing the consistency level provides higher consistency; given the same consistency level, higher issuing latency results in higher consistency (as the replicas converge, a sufficiently later read (R3) will return the consistent value up to 100%). Surprisingly, `QUORUM` and `ALL` reads start to achieve SC within a very short latency around `0.5` and `1.5` time units respectively (with `5` time units for even `ONE` reads).

On the other hand, all observations for TB apply to TA in general. In fact, for `QUORUM` and `ALL` reads, the two strategies perform almost the same, except that: (1) for `ALL` reads, TB provides

the same rack and the time difference between each pair of requests would be in the same range. While showing scalability is not the goal of this paper, we wish to do larger scale experiments in the future. There are resource challenges related to scaling the model-checking to larger scales (e.g., parallelizing it the right way), and we hope to solve this in our future work.

[9] Even if the tests allow a large number of YCSB clients, we set it to 2 to match our statistical model checking experimental setting.

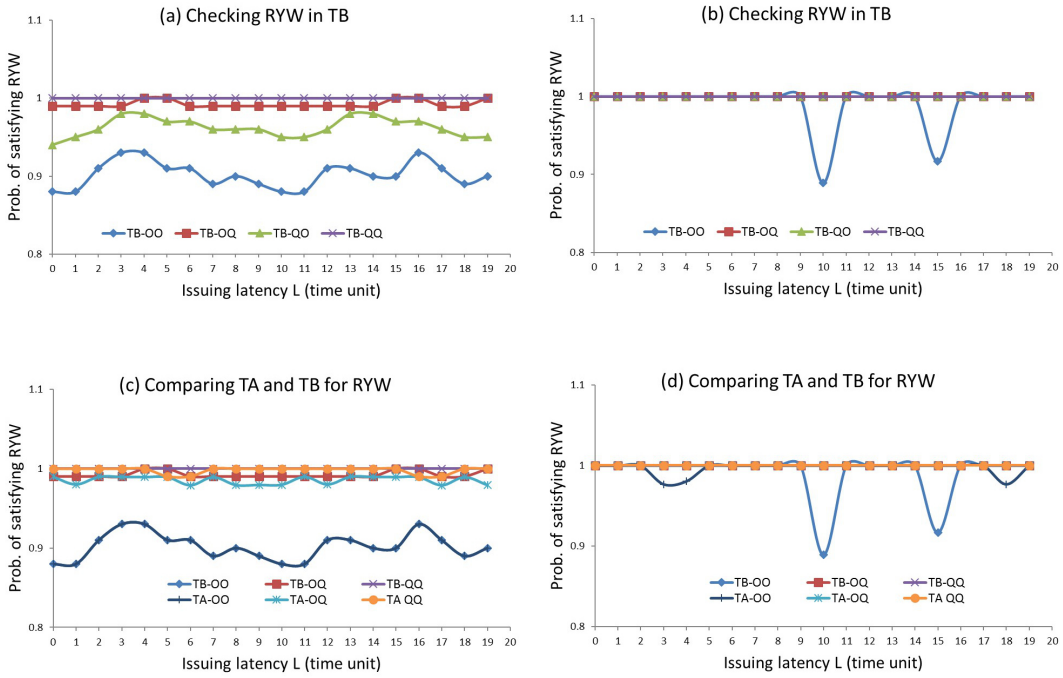

Figure 9 Probabilities of Satisfying RYW by Statistical Model Checking (Left) and by Real Deployment Run (Right).

noticeably more consistency than TA within an extremely short latency of `0.5` time units; and (2) for `QUORUM` reads, TB offers slightly more consistency than TA within `2.5` time units.

Based on these results it seems fair to say that both TB and TA provide high SC, especially with `QUORUM` and `ALL` reads. The consistency difference between the two strategies results from the overlap of R3 and W2. More precisely, since the subsequent read has higher chance to read multiple versions of the key-value pair with lower issuing latency, TA, only relying on the version itself, will return the matched value that is probably stale.

Regarding the implementation-based evaluation (the right plot in Figure 8), we show the resulting, experimentally computed probability of strongly consistent reads against `L2` (Figure 3) for deployment runs regarding the two strategies (only for `QUORUM` and `ALL` reads). Overall, the results indicate similar trends for the model predictions and real deployment runs (both plots in Figure 8): for example, for both model predictions and deployment runs, the probability is higher for `ALL` reads than for `QUORUM` reads regarding both strategies, especially when `L2` is low; consistency does not vary much with different strategies. Note that we observe the dips that break the monotonic behavior of TB-ALL and TB-QUORUM at a lower value of `L` because it is possible that the updated value from W2 has not reached all the replicas. With a higher value of `L`, we do not observe any break from the monotonic behavior.

5.2.2 Analysis Results for RYW

Figure 9(a) shows the resulting probability of satisfying RYW, where the probability (of R3 reading the value of W2 or a more recent value) is plotted against the issuing latency (`L`) between W1 and W4. From these results it is straightforward to see that scenarios (TB-OA/QQ/AO/AA) guarantee RYW due to the fact "`R3 + W2 > RF`". Since we have already seen that the Cassandra model satisfied SC quite well, it is also reasonable that all combinations of consistency levels

provide high RYW consistency, even with the lowest combination (O,O) that already achieves a probability around 90%. Surprisingly, it appears that a QUORUM read offers RYW consistency nearly 100%, even after a preceding write with its consistency level down to ONE (scenario (TB-OQ)). Another observation is that, in spite of the concurrent write from the other client, the probability of satisfying RYW stays fairly stable.

Figure 9(c) shows the comparison of TA and TB regarding RYW, where for simplicity we only list three combinations of consistency levels from R3's perspective with W2's consistency level fixed to ONE (in fact, with W2's consistency level increases, the corresponding scenarios will provide even higher consistency). In general, all observations for TB apply to TA, and it seems fair to say that both TA and TB offer high RYW consistency. The two strategies agree on the combination (O,O). However, TA cannot offer higher consistency than TB in any other scenario, with TA providing slightly lower consistency for some points, even though TA's overall performance is close to TB's over issuing latency. One reason is that TA does not respect the fact "R + W > RF" in general (e.g., two strictly ordered Quorum write and read cannot guarantee RYW).

Regarding the implementation-based evaluation (Figure 9(b) and (d)), we show the resulting probability of RYW-consistent reads against L (Figure 4) for deployment runs regarding two strategies. Both the model predictions and deployment runs show very high probability of satisfying RYW. This is expected since for each client the operations are mostly ordered, and for any read operation from a client, we expect any previous write from the same client to be committed to all replicas. For the deployment runs, we observe that we get 100% RYW consistency, except for scenario (TB-OO), which matches expectations, since ONE is the lowest consistency level and does not guarantee anything more than EC. This also matches our model predictions in Figure 9, where we see that the probability of satisfying RYW for scenario (TB-OO) is lower compared to other cases. In general the results indicate similar trends for the model predictions and real deployment runs, however, we observe some dips for TA/TB-OO as ONE is the lowest consistency level and does not provide a strict guarantee. Even though there are dips for certain values of L, the dips still have at least the high probability value of 90%.

5.2.3 Analysis Results for MR

Figure 10(a) shows the resulting probability of satisfying MR, where the probability is plotted against the issuing latency (L2) between W2 and R3. Again, it seems reasonable that all combinations of consistency levels offer high MR consistency, even with the lowest combination (O,O) that almost achieves a probability of 100%, and QUORUM reads gain more MR consistency than ONE reads. Surprisingly, it appears that the probability only relies on the consistency level of the reads, regardless of that of the concurrent write (W2): (O,O)/(O,Q) and (Q,O)/(Q,Q) with ONE/QUORUM reads have nearly the same probability trend. Note that we do not plot ALL reads because QUORUM reads already provide fairly stable probability of 100%.

Figure 10(c) shows the comparison of TB and TA regarding MR, where for simplicity we only list three lower combinations of consistency levels (with W2's consistency level increases, scenarios (TB/TA-QQ) will actually offer even higher consistency). Generally, all observations for TB apply to TA, and it seems fair to say that both strategies provide high MR consistency. TA's overall performance resembles TB's over issuing latency, but TA provides slightly higher consistency for some points.

Regarding the implementation-based evaluation (Figure 10(b) and (d)), we show the resulting probability of MR-consistent reads against L2 (Figure 5) for deployment runs regarding two strategies. Both the model predictions and deployment runs show high probability of satisfying MR. For the deployment runs, we observe that we get 100% MR consistency as the issuing latency

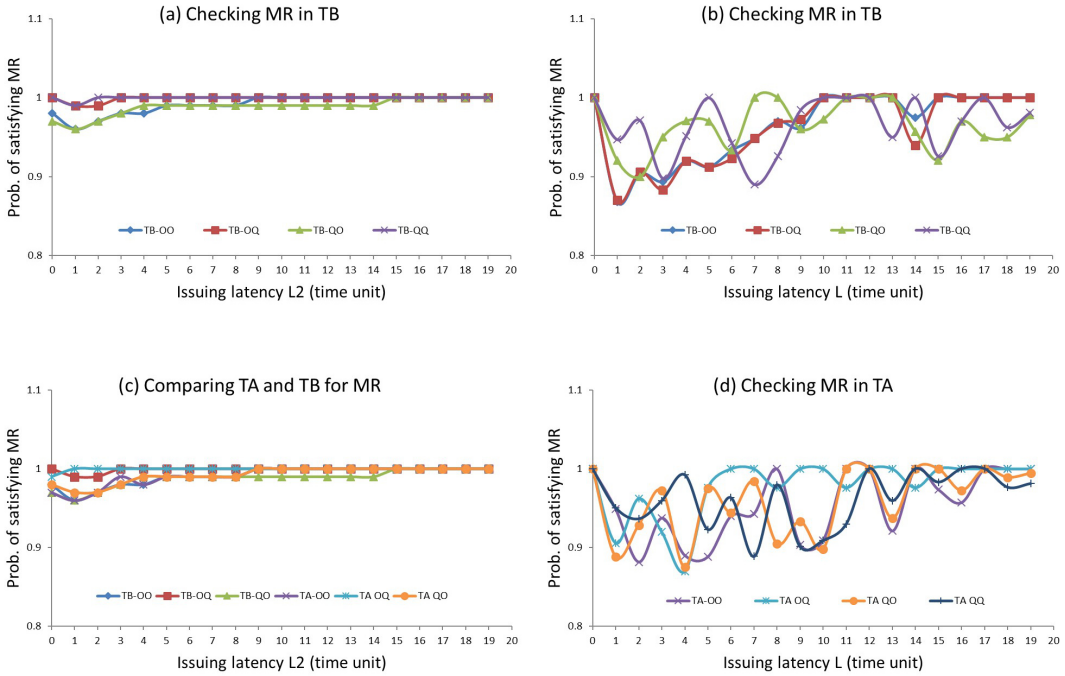

Figure 10 Probabilities of Satisfying MR by Statistical Model Checking (Left) and by Real Deployment Run (Right).

increases. For TA, the MR consistency fluctuates between 86% and 100%, but converges to 100% as the issuing latency increases. Note that the scenarios where the MR consistency is less than 100% are (OO) and (QO) for both strategies, and their MR consistency percentages are less compared to scenarios (QQ) and (OQ), which is expected.

5.2.4 Analysis Results for CP

Figure 11(a),(c),(e) show the resulting probability of satisfying CP by statistical model checking, where the probability (of R5 and R6 observing an ordered sequence of writes) is plotted against the issuing latency (`L`) between W1 and R5. By fixing the consistency level of writes, we can see how the consistency level of reads affect CP consistency over issuing latency: surprisingly, it appears that lower reads can achieve higher CP consistency. Specifically, `ONE` reads curve above `QUORUM` reads that curve above `ALL` reads in terms of each strategy. The reason is that reads with lower consistency level have lower latency (the interval between issuing time and finishing time for a read request), giving rise to higher chance to observe consecutive writes. Instead, higher latency reads allows more writes to reach the replicas during that interval, resulting in an inconsecutive observation (e.g., one anomaly occurs when R5 returns the value of W1 while R6 reads the value of W4).

We can compare the behaviors of TA and TB w.r.t. the choice of the three consistency levels: compared to TA, TB (1) conforms for `ONE` reads, (2) provides strictly higher CP consistency for `QUORUM` reads before some issuing latency, and, surprisingly, (3) gains more CP consistency only before some issuing latency for `ALL` reads. The reason is that CP aims at guaranteeing an ordered view of writes to the system instead of data freshness (e.g., R5 and R6 fetching W1 and W4 would be a CP-anomaly, though W4 is more recent than W2). Thus, after more and more writes have taken effect at the replicas (as `L` increases), TA with `ALL` reads is more likely to return the *convergent* value that respects the order.

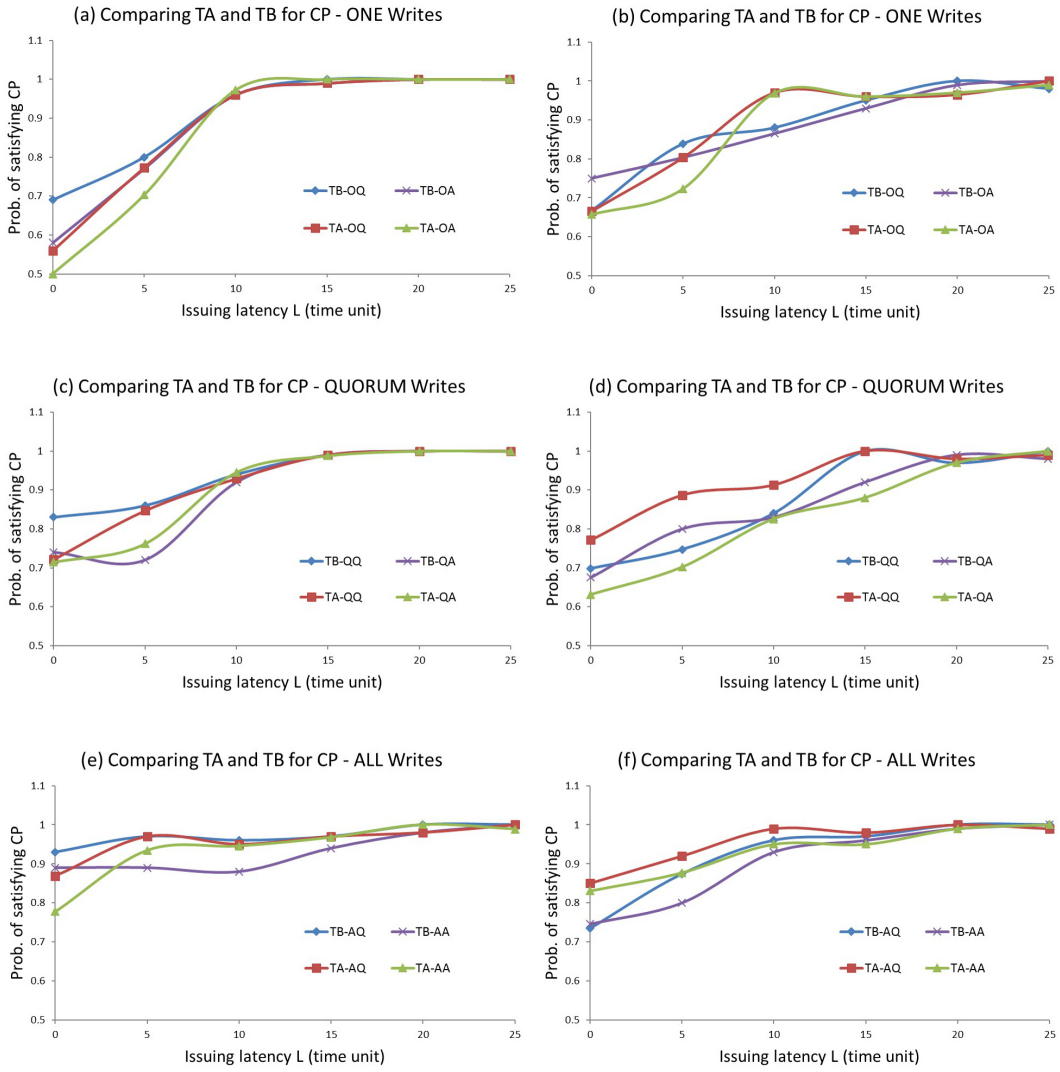

Figure 11 Probabilities of Satisfying CP by Statistical Model Checking (Left) and by Real Deployment Run (Right).

Note that all scenarios start with lower probability of satisfying CP (e.g., with `ONE` writes TB can only provide a probability down to 60%) because, when reads and writes are highly concurrent, reads would probably return the values of unordered writes.

Regarding the implementation-based evaluation (Figure 11(b),(d),(f)), we show the resulting probability of CP-consistent reads against `L` (Figure 6) for deployment runs regarding two strategies. The results indicate similar trends for the model predictions and real deployment runs. Both the model predictions and deployment runs show high probability of satisfying CP as the issuing latency `L` increases. For lower values of `L`, we observe CP consistency values between 60% and 80%, and, as `L` increases, the CP consistency value also increases to 70%–90%, gradually converging to 100%.

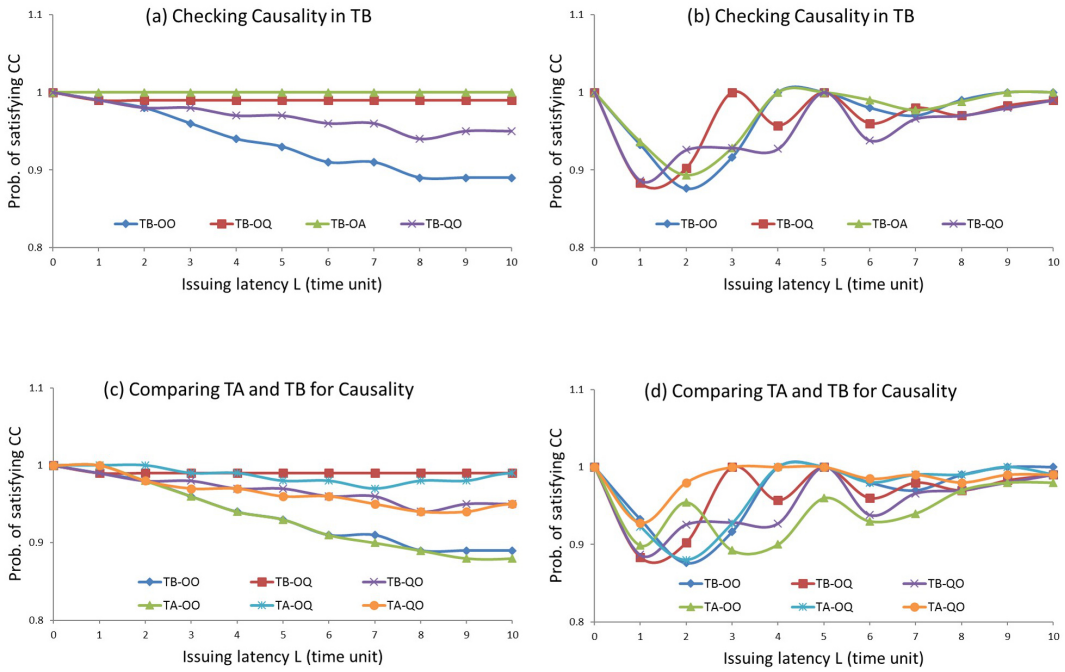

Figure 12 Probabilities of Satisfying CC by Statistical Model Checking (Left) and by Real Deployment Run (Right).

5.2.5 Analysis Results for CC

Figure 12(a) shows the resulting probability of satisfying CC, where the probability (of R5 returning the value of W4 provided R3 reads the value of W2) is plotted against the issuing latency (`L`) between W1 and R3. It is reasonable that all combinations of consistency levels achieve high CC consistency, even with the lowest combination `(O,O)` that provides a probability around 90%, and reads with higher consistency level gain more CC consistency (e.g., scenario (TB-OA) curves above scenario (TB-OQ) that curves above scenario (TB-QO/OO)). It is also straightforward to see that, with the consistency level of the preceding write (W4) from the same client increases, more consistency will be achieved (e.g., scenario (TB-QO) curves above scenario (TB-OO)). Note that all scenarios except (TB-OA) start with a probability of 100%, and gradually drop down and stabilize at a lower probability. The reason is that with a lower issuing latency there are fewer chances for R3 to read the value of W2 (if R3 does not return the value of W2, no causality will arise between these two operations) and therefore any value returned by R5 is considered to be CC-consistent; with an increasing issuing latency, R3 will be more likely to fetch the value of W2, which leaves the combination of consistency levels of W4 and R5 to be the only factor to affect the consistency (that also explains why the probability of satisfying CC stays fairly stable after some point).

Figure 12(c) shows the comparison of TB and TA in terms of CC, where for simplicity we only list three lower combinations of consistency levels due to the fact "`R5 + W4 > RF`". Both strategies achieve high CC consistency, and overlap for the combinations `(O,O)` and `(O,Q)`. Regarding `(O,Q)`, TA's performance resembles TB's over issuing latency, and both surpass each other slightly for some points.

Regarding the implementation-based evaluation (Figure 12(b) and (d)), we show the resulting probability of CC-consistent reads against `L` (Figure 7) for deployment runs regarding two strategies.

Both the model predictions and deployment runs show high probability of satisfying CC. For the model predictions, we observe that (TA-OO) and (TB-OO) have the lowest CC-consistency values around 90%, which is expected and also matches with our observations from the real deployment runs. In general, we observe that we get 90% - 100% CC-consistent reads, independent of the issuing latency.

5.2.6 Summary and Comparison

Our Cassandra model actually achieves much higher consistency (up to SC) than the promised EC, with `QUORUM` reads sufficient to provide up to 100% consistency in almost all scenarios. Comparing TA and TB, it seems fair to say that TA is not a competitive design alternative to TB in terms of the consistency models considered in this paper except CP, even though TA behaves close to TB in most cases; regarding CP, TA surpasses TB with `ALL` reads during a certain interval of issuing latency.

Our model, including the alternative design, is less than 1000 lines of code and the time to compute the probabilities for the consistency guarantees is 15 minutes (worst-case) on a 2.9 GHz Intel 4-Core i7-3520M CPU with 3.7 GB memory. The upper bound for model runtime depends on the confidence level of our statistical model checker (99% confidence level for all our experiments).

Both the model predictions and implementation-based evaluations reach the same conclusion: Cassandra provides much higher consistency than the promised EC, and TA does not improve consistency compared to TB in terms of the consistency models considered in this paper except for CP where TA provides higher consistency than TB with `ALL` reads during a certain interval of issuing latency. Note that the actual probability values from both sides might differ due to factors like hard-to-match experimental configurations, the inherent difference between statistical model checking and implementation-based evaluation[10], and processing delay at client/server side that our model does not include. However, the important observation is that the resulting *trends* from both sides are similar, leading to the same conclusions w.r.t. consistency measurements and strategy comparison.

Note that our experiments did not focus on the performance aspects of the TA and TB approaches in Cassandra. Our goal in this paper is to show that results derived from model-checking (using our model) agree reasonably well with experimental reality, e.g., see the comparisons and matches between our implementation results and the model results. The paper shows that this agreement holds true as far as various consistency models are concerned, and even if one changes how timestamps are used in responding to queries (comparing TA and TB).

6 Related Work

6.1 Model-based Performance Analysis of NoSQL stores

The work in [31] presents a queueing Petri net model of Cassandra parameterized by benchmarking only one server. The model is scaled to represent the characteristics of read workloads for different replication strategies and cluster sizes. Regarding performance, only response times and throughput are considered. The work in [20] benchmarks three NoSQL databases, namely Cassandra, MongoDB and HBase, by throughput and operation latency. Two simple high-level

[10] In general, implementation-based evaluation is based on a single trace of tens of thousands of operations (e.g., each YCSB client in our experiments performs 20,000 operations), while statistical model checking is based on sampling tens of thousands of Monte-Carlo simulations of several operations (that can be considered as a segment of the trace) up to a certain statistical confidence (e.g., our statistical model checking runs 40,000 Monte-Carlo simulations for the experimental scenario of CP which has only six operations).

queuing network models are presented that are able to capture those performance characteristics. Compared to both, our probabilistic model embodies the major components and features of Cassandra, and serves as the basis of statistical analysis of consistency with multiple clients and servers. Our model is also shown to be able to measure and predict new strategy designs by both statistical model checking and by checking the conformance of the model checking results with the code-based evaluation. Other recent work on model-based performance analysis includes [7], which applies multi-formalism modeling approach to the Apache Hive query language for NoSQL databases.

6.2 Experimental Consistency Benchmarking in NoSQL stores

The work in [32, 40, 10] proposes active and passive consistency benchmarking approaches for distributed NoSQL key-value stores, where operation logs are analyzed to find consistency violations. While these papers focus on consistency benchmarking using synthetic traces (e.g., generated using YCSB), the authors in [29] perform the first consistency benchmarking study of a large-scale production system (Facebook TAO system). YCSB+T [17] benchmarks isolation guarantees for NoSQL transactional databases, while the authors in [6] developed a benchmarking suite for interactive social networking applications running on top of NoSQL databases. The work in [5] proposes probabilistic notions of consistency to predict the data staleness, and uses Monte-Carlo simulations to explore the trade-off between latency and consistency in Dynamo-style partial quorum systems. Their focus is more on developing the theory of consistency models. However, we focus on building a probabilistic model for a key-value store like Cassandra itself, and our objective is to compare the consistency benchmarking results with the model-based predictions from our statistical model checking.

6.3 Rewriting Logic-based Analysis of Cloud Computing Systems

The work in [26] presents a formal model of the popular distributed key-value store Cassandra, and formally analyzes different consistency properties using Maude from a *qualitative* perspective. The work in [24] looks into the consistency issue for Cassandra only in terms of SC and RYW from a *quantitative*, statistical model checking perspective. This paper extends the previous results by providing quantitative analysis of three new consistency models: monotonic reads, consistent prefix and causal consistency. There is other recent work on rewriting logic-based analysis of *cloud computing* systems, including, e.g., [25], which formalizes RAMP transactions and their extensions and optimizations in rewriting logic and performs model checking verification of key properties using the Maude tool; [21, 22], which uses Maude and Real-Time Maude to define a formal model of Google's widely-replicated data store Megastore (a hybrid between a NoSQL store and a relational database) and to develop an extension of Megastore. These models were simulated for QoS estimation and model checked for functional correctness; [36], which formally models and analyzes availability properties of a ZooKeeper-based group key management service; [18], which proposes and analyzes DoS resilience mechanisms for cloud-based systems; [42], which gives formal semantics to the KLAIM language and uses it to specify and analyze cloud-based architectures;

7 Conclusion

Our main focus in this paper has been twofold: (i) to predict what consistency properties Cassandra can provide in actual practice by using statistical model checking; and (ii) to demonstrate the predictive power of our model-based approach in key-value store design by comparing statistical

model checking predictions with implementation-based evaluations. Our analysis is based on a formal probabilistic model of Cassandra. To the best of our knowledge, we are the first to develop such a formal model. The purpose of the alternative design was two-fold. First, we wanted to see if the alternative approach i.e., the Timestamp Agnostic (TA) approach would give better accuracy or not. Second, we also wanted to show that it was easier to try the alternative design with our formal model. This is one of the key benefits of building formal models so that we can quickly try alternative approaches and check their accuracy/performance. Our goal is not to do a systematic performance analysis of TA and TB approaches in Cassandra that would require larger setups not currently feasible for the statistical model checking analysis. Our goal instead is to show that results derived from model-checking (using our model) agree reasonably well with experimental reality, e.g., see the comparisons and matches between our implementation results and the model results. The paper shows that this agreement holds true as far as various consistency models are concerned, and even if one changes how timestamps are used in responding to queries (comparing TA and TB).

In this paper we have only investigated consistency guarantees in a quantitative manner. A natural next step would be to specify and quantify other performance metrics. Depending on the perspective (key-value store providers, users, or application developers), different metrics (e.g., throughput and operation latency) can be used to measure key-value store performance. We also plan to refine our model in order to quantify those metrics. While showing scalability is not the goal of this paper, we wish to do larger scale experiments in the future. There are various resource challenges related to scaling the model checking to larger system sizes (e.g., parallelizing it in the proper way) that we plan to address and solve this in our future work. More broadly, our long-term goal is to develop a library of formally specified executable components embodying the key functionalities of NoSQL key-value stores (not just Cassandra), as well as of distributed transaction systems [25]. We plan to use such components and the formal analysis of their performance to facilitate efficient exploration of the design space for such systems and their compositions with modest levels of effort and increased flexibility.

References

1 Gul A. Agha, José Meseguer, and Koushik Sen. Pmaude: Rewrite-based specification language for probabilistic object systems. *Electr. Notes Theor. Comput. Sci.*, 153(2):213–239, 2006. doi:10.1016/j.entcs.2005.10.040.

2 Mustaque Ahamad, Gil Neiger, James E. Burns, Prince Kohli, and Phillip W. Hutto. Causal memory: Definitions, implementation, and programming. *Distributed Computing*, 9(1):37–49, 1995. doi:10.1007/BF01784241.

3 Musab AlTurki and José Meseguer. Pvesta: A parallel statistical model checking and quantitative analysis tool. In Andrea Corradini, Bartek Klin, and Corina Cîrstea, editors, *Algebra and Coalgebra in Computer Science – 4th International Conference, CALCO 2011, Winchester, UK, August 30 – September 2, 2011. Proceedings*, volume 6859 of *Lecture Notes in Computer Science*, pages 386–392. Springer, 2011. doi:10.1007/978-3-642-22944-2_28.

4 Peter Bailis, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. The potential dangers of causal consistency and an explicit solution. In Michael J. Carey and Steven Hand, editors, *ACM Symposium on Cloud Computing, SOCC'12, San Jose, CA, USA, October 14-17, 2012*, page 22. ACM, 2012. doi:10.1145/2391229.2391251.

5 Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica. Probabilistically bounded staleness for practical partial quorums. *PVLDB*, 5(8):776–787, 2012. URL: http://vldb.org/pvldb/vol5/p776_peterbailis_vldb2012.pdf.

6 Sumita Barahmand and Shahram Ghandeharizadeh. BG: A benchmark to evaluate interactive social networking actions. In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*. www.cidrdb.org, 2013. URL: http://www.cidrdb.org/cidr2013/Papers/CIDR13_Paper93.pdf.

7 Enrico Barbierato, Marco Gribaudo, and Mauro Iacono. Performance evaluation of nosql big-data applications using multi-formalism models. *Future Generation Comp. Syst.*, 37:345–353, 2014. doi:10.1016/j.future.2013.12.036.

8 Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, and Peter Vajgel. Finding a needle in haystack: Facebook's photo storage. In Remzi H. Arpaci-Dusseau and Brad Chen, editors, *9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, BC, Canada, Proceedings*, pages 47–60. USENIX Association,

2010. URL: http://www.usenix.org/events/osdi10/tech/full_papers/Beaver.pdf.

9 Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *IMC*, pages 267–280, 2010.

10 David Bermbach and Stefan Tai. Eventual consistency: How soon is eventual? an evaluation of amazon s3's consistency behavior. In Karl M. Göschka, Schahram Dustdar, and Vladimir Tosic, editors, *Proceedings of the 6th Workshop on Middleware for Service Oriented Computing, MW4SOC 2011, Lisbon, Portugal, December 12-16, 2011*, page 1. ACM, 2011. doi:10.1145/2093185.2093186.

11 Eric A. Brewer. Towards robust distributed systems (abstract). In Gil Neiger, editor, *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, July 16-19, 2000, Portland, Oregon, USA.*, page 7. ACM, 2000. doi:10.1145/343477.343502.

12 Cassandra, 2016. http://cassandra.apache.org.

13 Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott, editors. *All About Maude – A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

14 Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with YCSB. In Joseph M. Hellerstein, Surajit Chaudhuri, and Mendel Rosenblum, editors, *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, Indianapolis, Indiana, USA, June 10-11, 2010*, pages 143–154. ACM, 2010. doi:10.1145/1807128.1807152.

15 James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson C. Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's globally-distributed database. In Chandu Thekkath and Amin Vahdat, editors, *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*, pages 261–264. USENIX Association, 2012. URL: https://www.usenix.org/conference/osdi12/technical-sessions/presentation/corbett.

16 DB-Engines, 2016. http://db-engines.com/en/ranking.

17 Akon Dey, Alan Fekete, Raghunath Nambiar, and Uwe Röhm. YCSB+T: benchmarking web-scale transactional databases. In *Workshops Proceedings of the 30th International Conference on Data Engineering Workshops, ICDE 2014, Chicago, IL, USA, March 31 – April 4, 2014*, pages 223–230. IEEE Computer Society, 2014. doi:10.1109/ICDEW.2014.6818330.

18 Jonas Eckhardt, Tobias Mühlbauer, Musab AlTurki, José Meseguer, and Martin Wirsing. Stable availability under denial of service attacks through formal patterns. In Juan de Lara and Andrea Zisman, editors, *Fundamental Approaches to Software Engineering – 15th International Conference, FASE 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 – April 1, 2012. Proceedings*, volume 7212 of *Lecture Notes in Computer Science*, pages 78–93. Springer, 2012. doi:10.1007/978-3-642-28872-2_6.

19 Jonas Eckhardt, Tobias Mühlbauer, José Meseguer, and Martin Wirsing. Statistical model checking for composite actor systems. In Narciso Martí-Oliet and Miguel Palomino, editors, *Recent Trends in Algebraic Development Techniques, 21st International Workshop, WADT 2012, Salamanca, Spain, June 7-10, 2012, Revised Selected Papers*, volume 7841 of *Lecture Notes in Computer Science*, pages 143–160. Springer, 2012. doi:10.1007/978-3-642-37635-1_9.

20 Andrea Gandini, Marco Gribaudo, William J. Knottenbelt, Rasha Osman, and Pietro Piazzolla. Performance evaluation of nosql databases. In András Horváth and Katinka Wolter, editors, *Computer Performance Engineering – 11th European Workshop, EPEW 2014, Florence, Italy, September 11-12, 2014. Proceedings*, volume 8721 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2014. doi:10.1007/978-3-319-10885-8_2.

21 Jon Grov and Peter Csaba Ölveczky. Formal modeling and analysis of google's megastore in real-time maude. In Shusaku Iida, José Meseguer, and Kazuhiro Ogata, editors, *Specification, Algebra, and Software – Essays Dedicated to Kokichi Futatsugi*, volume 8373 of *Lecture Notes in Computer Science*, pages 494–519. Springer, 2014. doi:10.1007/978-3-642-54624-2_25.

22 Jon Grov and Peter Csaba Ölveczky. Increasing consistency in multi-site data stores: Megastore-cgc and its formal analysis. In Dimitra Giannakopoulou and Gwen Salaün, editors, *Software Engineering and Formal Methods – 12th International Conference, SEFM 2014, Grenoble, France, September 1-5, 2014. Proceedings*, volume 8702 of *Lecture Notes in Computer Science*, pages 159–174. Springer, 2014. doi:10.1007/978-3-319-10431-7_12.

23 Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978. doi:10.1145/359545.359563.

24 Si Liu, Son Nguyen, Jatin Ganhotra, Muntasir Raihan Rahman, Indranil Gupta, and José Meseguer. Quantitative analysis of consistency in nosql key-value stores. In Javier Campos and Boudewijn R. Haverkort, editors, *Quantitative Evaluation of Systems, 12th International Conference, QEST 2015, Madrid, Spain, September 1-3, 2015, Proceedings*, volume 9259 of *Lecture Notes in Computer Science*, pages 228–243. Springer, 2015. doi:10.1007/978-3-319-22264-6_15.

25 Si Liu, Peter Csaba Ölveczky, Muntasir Raihan Rahman, Jatin Ganhotra, Indranil Gupta, and José Meseguer. Formal modeling and analysis of RAMP transaction systems. In Sascha Ossowski,

editor, *Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, April 4-8, 2016*, pages 1700–1707. ACM, 2016. doi:10.1145/2851613.2851838.

26 Si Liu, Muntasir Raihan Rahman, Stephen Skeirik, Indranil Gupta, and José Meseguer. Formal modeling and analysis of cassandra in maude. In Stephan Merz and Jun Pang, editors, *Formal Methods and Software Engineering – 16th International Conference on Formal Engineering Methods, ICFEM 2014, Luxembourg, Luxembourg, November 3-5, 2014. Proceedings*, volume 8829 of *Lecture Notes in Computer Science*, pages 332–347. Springer, 2014. doi:10.1007/978-3-319-11737-9_22.

27 Si Liu, Muntasir Raihan Rahman, Stephen Skeirik, Indranil Gupta, and José Meseguer. Formal modeling and analysis of Cassandra in Maude. Technical Report. Manuscript, 2014. URL: https://sites.google.com/site/siliunobi/icfem-cassandra.

28 Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. In Ted Wobber and Peter Druschel, editors, *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October 23-26, 2011*, pages 401–416. ACM, 2011. doi:10.1145/2043556.2043593.

29 Haonan Lu, Kaushik Veeraraghavan, Philippe Ajoux, Jim Hunt, Yee Jiun Song, Wendy Tobagus, Sanjeev Kumar, and Wyatt Lloyd. Existential consistency: measuring and understanding consistency at facebook. In Ethan L. Miller and Steven Hand, editors, *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP 2015, Monterey, CA, USA, October 4-7, 2015*, pages 295–310. ACM, 2015. doi:10.1145/2815400.2815426.

30 MongoDB, 2016. http://www.mongodb.org.

31 Rasha Osman and Pietro Piazzolla. Modelling replication in nosql datastores. In Gethin Norman and William H. Sanders, editors, *Quantitative Evaluation of Systems – 11th International Conference, QEST 2014, Florence, Italy, September 8-10, 2014. Proceedings*, volume 8657 of *Lecture Notes in Computer Science*, pages 194–209. Springer, 2014. doi:10.1007/978-3-319-10696-0_16.

32 Muntasir Raihan Rahman, Wojciech M. Golab, Alvin AuYoung, Kimberly Keeton, and Jay J. Wylie. Toward a principled framework for benchmarking consistency. In Michael J. Freedman and Neeraj Suri, editors, *Proceedings of the Eighth Workshop on Hot Topics in System Dependability, HotDep 2012, Hollywood, CA, USA, October 7, 2012*. USENIX Association, 2012. URL: https://www.usenix.org/conference/hotdep12/workshop-program/presentation/rahman.

33 Redis, 2016. http://redis.io.

34 Koushik Sen, Mahesh Viswanathan, and Gul Agha. On statistical model checking of stochastic systems. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, volume 3576 of *Lecture Notes in Computer Science*, pages 266–280. Springer, 2005. doi:10.1007/11513988_26.

35 Koushik Sen, Mahesh Viswanathan, and Gul A. Agha. VESTA: A statistical model-checker and analyzer for probabilistic systems. In *Second International Conference on the Quantitative Evaluaiton of Systems (QEST 2005), 19-22 September 2005, Torino, Italy*, pages 251–252. IEEE Computer Society, 2005. doi:10.1109/QEST.2005.42.

36 Stephen Skeirik, Rakesh B. Bobba, and José Meseguer. Formal analysis of fault-tolerant group key management using zookeeper. In *13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2013, Delft, Netherlands, May 13-16, 2013*, pages 636–641. IEEE Computer Society, 2013. doi:10.1109/CCGrid.2013.98.

37 Doug Terry. Replicated data consistency explained through baseball. *Commun. ACM*, 56(12):82–89, 2013. doi:10.1145/2500500.

38 Douglas B. Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos K. Aguilera, and Hussam Abu-Libdeh. Consistency-based service level agreements for cloud storage. In Michael Kaminsky and Mike Dahlin, editors, *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP'13, Farmington, PA, USA, November 3-6, 2013*, pages 309–324. ACM, 2013. doi:10.1145/2517349.2522731.

39 Werner Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, 2009. doi:10.1145/1435417.1435432.

40 Hiroshi Wada, Alan Fekete, Liang Zhao, Kevin Lee, and Anna Liu. Data consistency properties and the trade-offs in commercial cloud storage: the consumers' perspective. In *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings*, pages 134–143. www.cidrdb.org, 2011. URL: http://www.cidrdb.org/cidr2011/Papers/CIDR11_Paper15.pdf.

41 Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In David E. Culler and Peter Druschel, editors, *5th Symposium on Operating System Design and Implementation (OSDI 2002), Boston, Massachusetts, USA, December 9-11, 2002*. USENIX Association, 2002. URL: http://www.usenix.org/events/osdi02/tech/white.html.

42 Martin Wirsing, Jonas Eckhardt, Tobias Mühlbauer, and José Meseguer. Design and analysis of cloud-based architectures with KLAIM and maude. In Francisco Durán, editor, *Rewriting Logic and Its Applications – 9th International Workshop, WRLA 2012, Held as a Satellite Event of ETAPS, Tallinn, Estonia, March 24-25, 2012, Revised Selected Papers*, volume 7571 of *Lecture Notes in Computer Science*, pages 54–82. Springer, 2012. doi:10.1007/978-3-642-34005-5_4.

43 Håkan L. S. Younes and Reid G. Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Inf. Comput.*, 204(9):1368–1409, 2006. doi:10.1016/j.ic.2006.05.002.