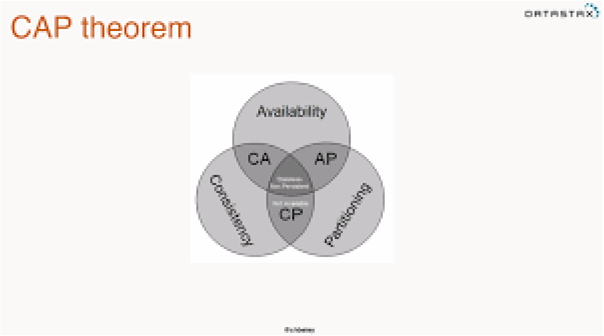
A More Consistent Understanding of Consistency


CAP theorem

Subhajit Sidhanta

IIT Bhilai/INESC ID Lisbon

Joint work with:

Ricardo Dias SUSE Linux GmbH

Prof. **Rodrigo Rodrigues** IST, ULisboa

Accepted at
Symposium on Reliable Distributed Systems (SRDS 2019), Lyon, France,
October 1-4, 2019

Consistency Models

- A *Consistency Model :* a contract (order among observed results) between the storage and the client (processor).
- Conventions we will use

$w^j(o, v_l)$ *: j*th *write on object o* with value v_l

$r^k(o)v_m$: k th *read on object o* that returned value v_m

Isolation Levels

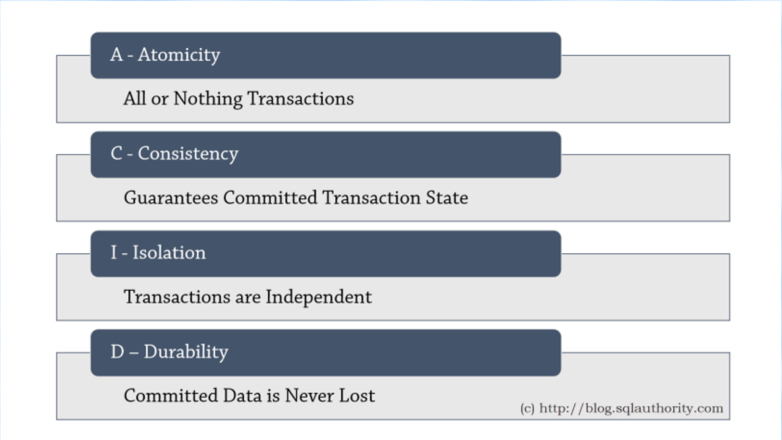
- An Isolation level: constraints the manner in which results of operations performed from a transaction is visible from other concurrent transactions.

- Conventions

$w^j_{tx}(o, v_l)$ *: j*th *write from transa*tx*n* on *object o* with value v_l

$r^k_{tx}(o)v_m$: k th *read from transa*tx*n* on object o returned value v_m

CONSISTENCY



A - Atomicity

All or Nothing Transactions

C - Consistency

Guarantees Committed Transaction State

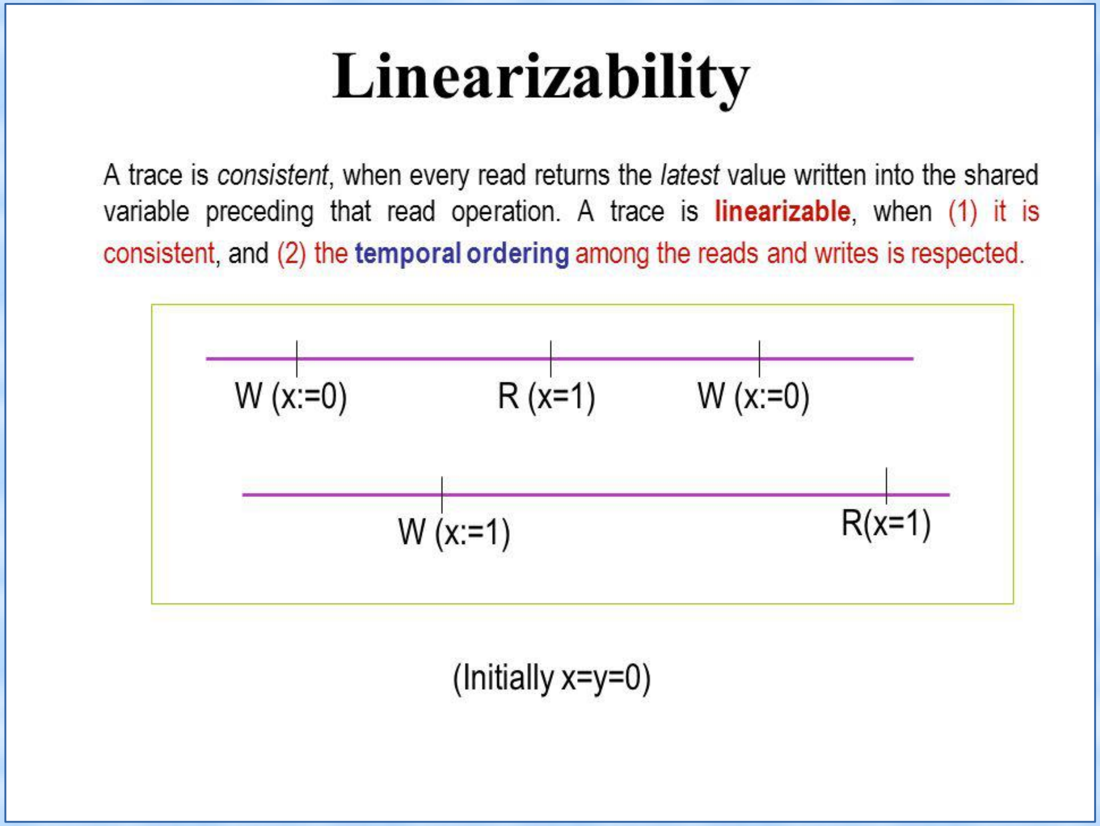
I - Isolation

Transactions are Independent

D – Durability

Committed Data is Never Lost

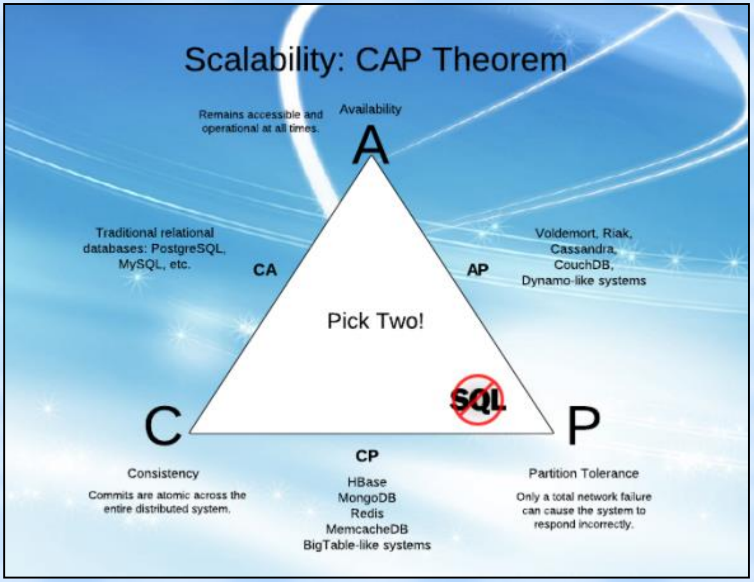
(c) http://blog.sqlauthority.com



Linearizability

A trace is *consistent*, when every read returns the *latest* value written into the shared variable preceding that read operation. A trace is **linearizable**, when (1) it is consistent, and (2) the **temporal ordering** among the reads and writes is respected.

(Initially x=y=0)



Scalability: CAP Theorem

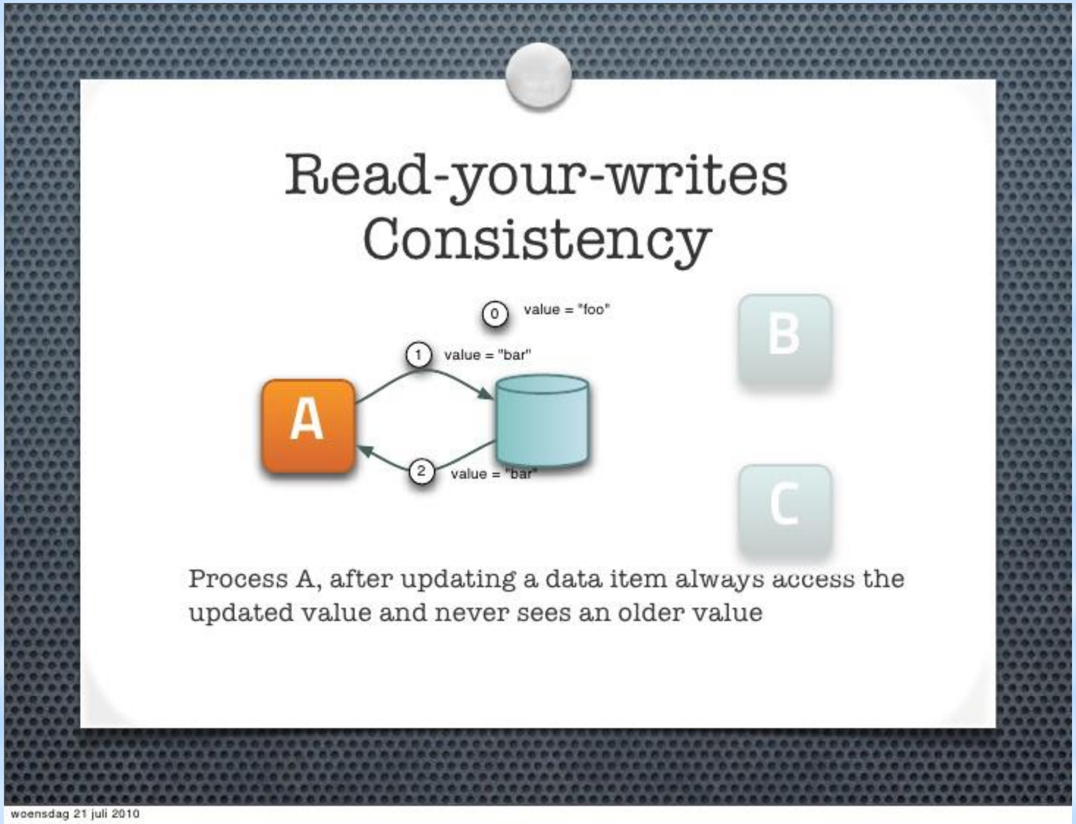
Distributed Systems

→ **Consistency Models**

Strongest (Linearizability)

weakest (Eventual)

WEAKER CONSISTENCY MODELS

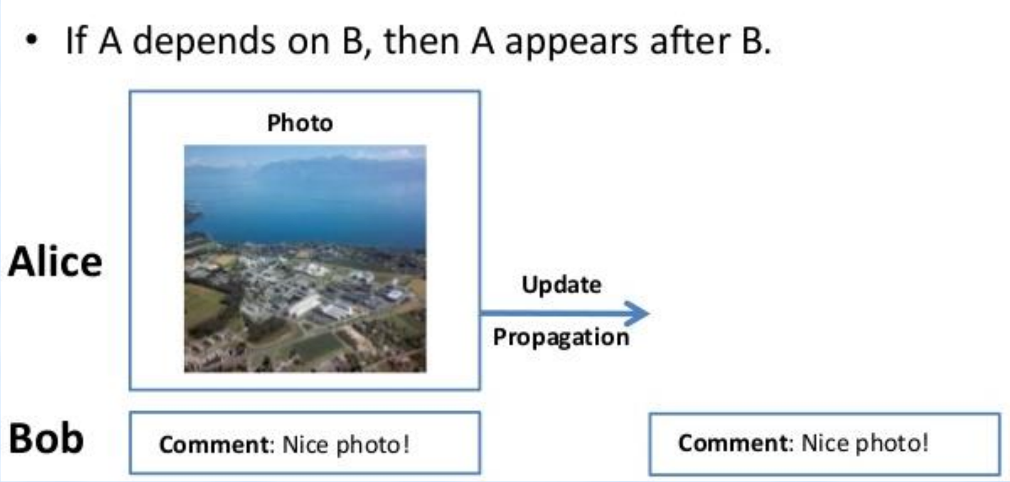


Read-your-writes Consistency

Process A, after updating a data item always access the updated value and never sees an older value

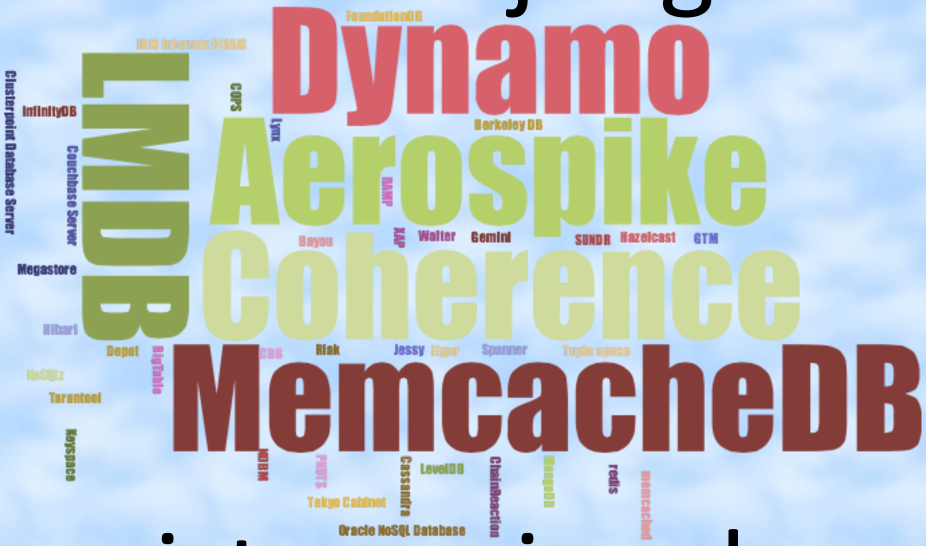
woensdag 21 juli 2010

Causal Consistency

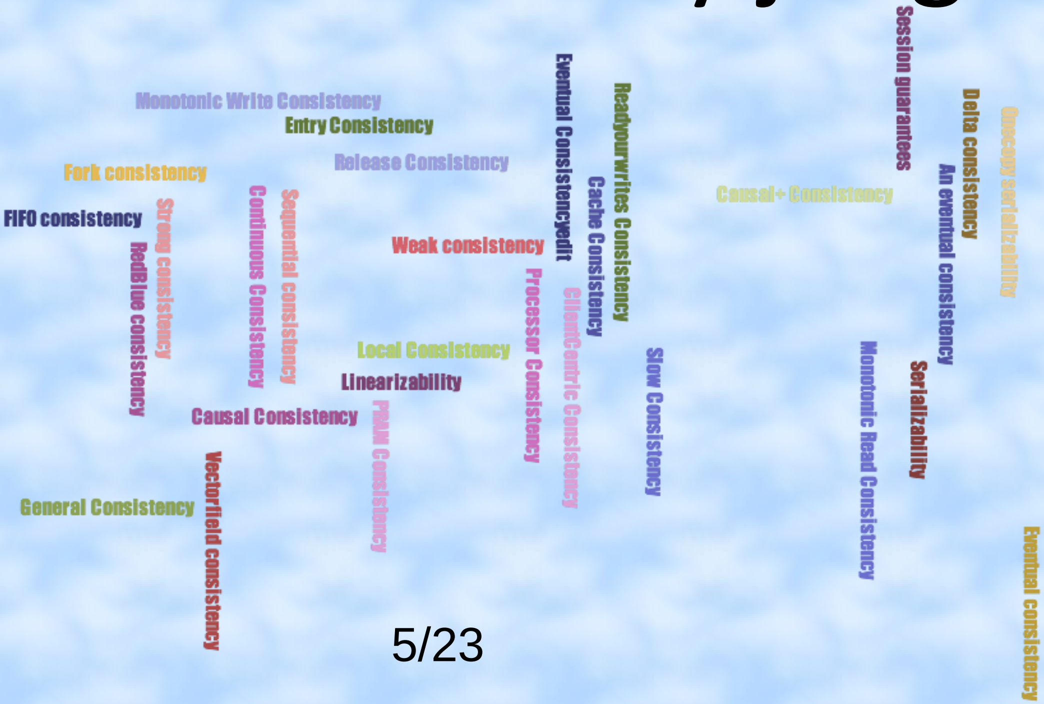


- If A depends on B, then A appears after B.

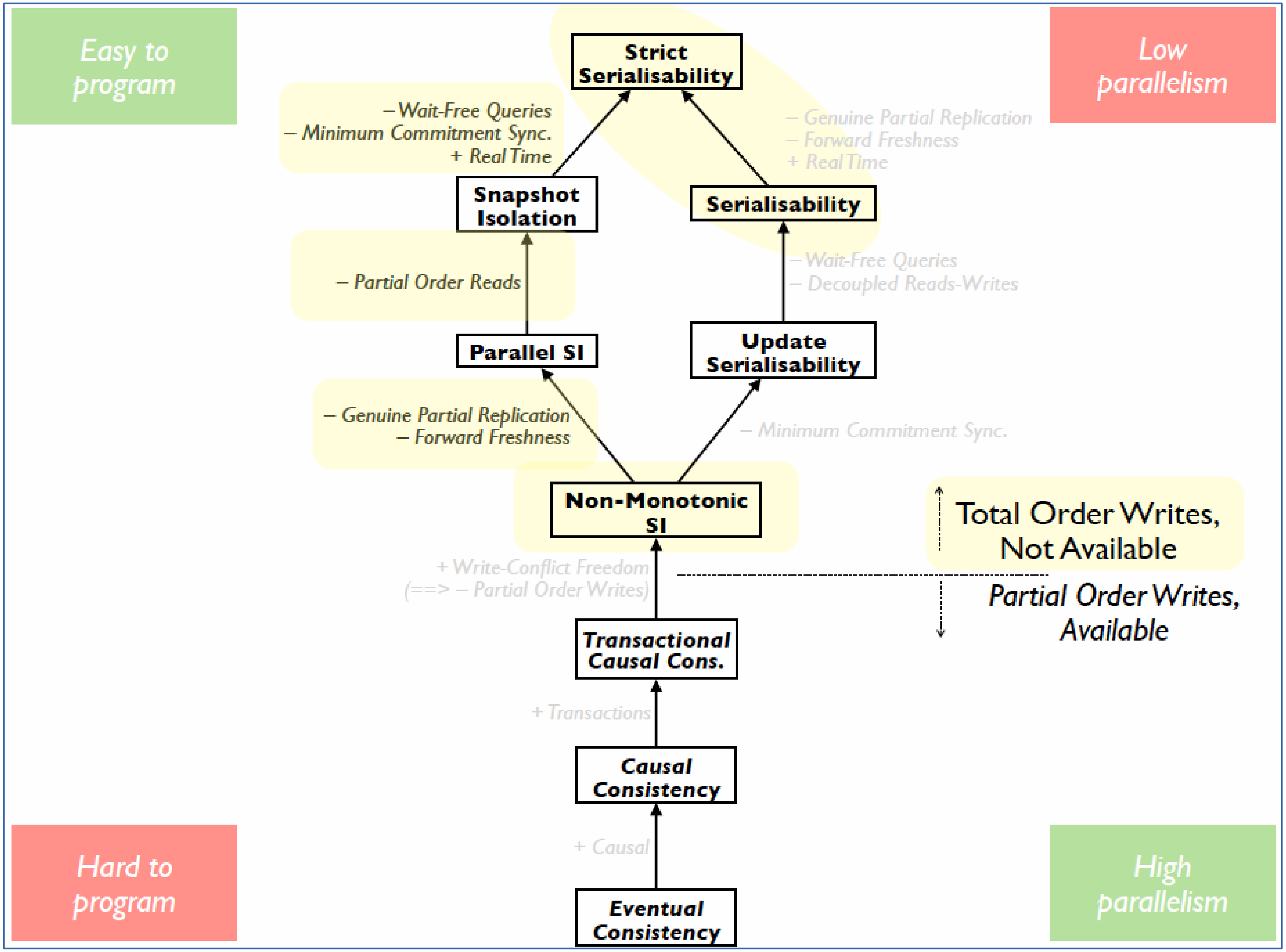
The “Database” jungle



The consistency jungle



JUNGLE OF CONSISTENCY MODELS



Easy to program

Low parallelism

Strict Serialisability

– Wait-Free Queries
– Minimum Commitment Sync.
+ Real Time

– Genuine Partial Replication
– Forward Freshness
+ Real Time

Snapshot Isolation

Serialisability

– Partial Order Reads

– Wait-Free Queries
– Decoupled Reads-Writes

Parallel SI

Update Serialisability

– Genuine Partial Replication
– Forward Freshness

– Minimum Commitment Sync.

Non-Monotonic SI

Total Order Writes, Not Available

+ Write-Conflict Freedom
(==> – Partial Order Writes)

Partial Order Writes, Available

Transactional Causal Cons.

+ Transactions

Causal Consistency

+ Causal

Eventual Consistency

Hard to program

High parallelism

The consistency jungle

- Large number of different models
- Defined using different formalisms
- Community-specific terms and definitions
- How do they compare?
  - “The causal+ consistency (…) falls between sequential and causal consistency” [COPS]
  - “FJC implies a number of (…) session guarantees” [Depot]

Towards an unifying specification syntax

- Goal: find a unified way to specify consistency ad isolation levels that is:
  - Simple and intuitive
  - Unifies consistency and isolation level definitions using a common syntax
  - Directly applicable to automated verification systems
  - Enables straightforward comparisons of levels
  - Allows for efficient verification of implementations

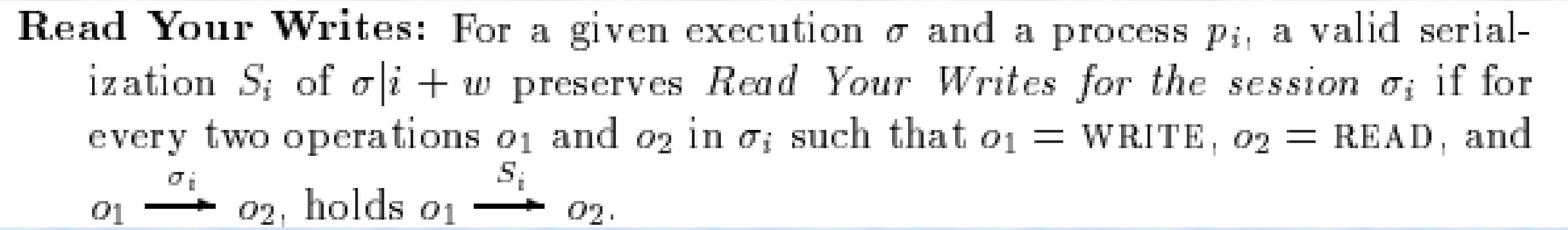
Some Common Terminology

- A **serialization (Ser)** is a sequence in which a group of storage operations are executed on a datastore.

- A serialization is said to be **legal** if every read operation returns the value written by the latest write operation preceding it in the serialization.

Adya et al.+ Chockler et al.

- Chockler's consistency definitions
    - descriptve (**informal**) specifications

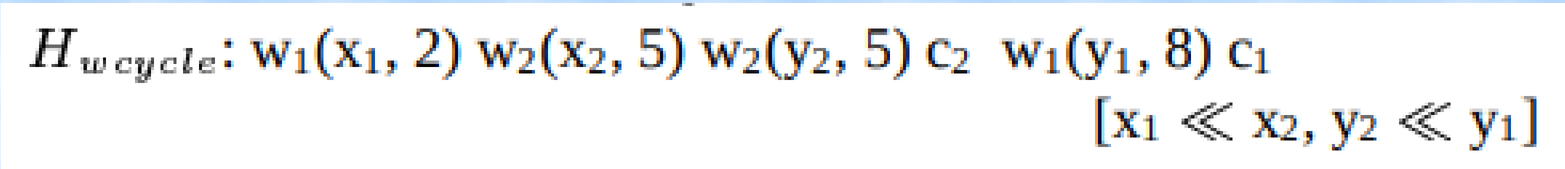


Read Your Writes: For a given execution σ and a process p_i, a valid serialization S_i of $\sigma|i + w$ preserves *Read Your Writes for the session* σ_i if for every two operations o_1 and o_2 in σ_i such that $o_1 = \text{WRITE}$, $o_2 = \text{READ}$, and $o_1 \xrightarrow{\sigma_i} o_2$, holds $o_1 \xrightarrow{S_i} o_2$.

-
    - equivalent legal serialization
- Adya's Generalized isolation levels
    - Similar goal applied to isolation levels
  - **Graphs** derived from trace of the execution
    - Nodes = transactions
    - Edges = order between transactions (ww/wr/rw)
      - **follow from version numbers**
  - Isolation levels defined by precluded **cycles**
    - Cycles represent "anomalies" (bad behaviors)

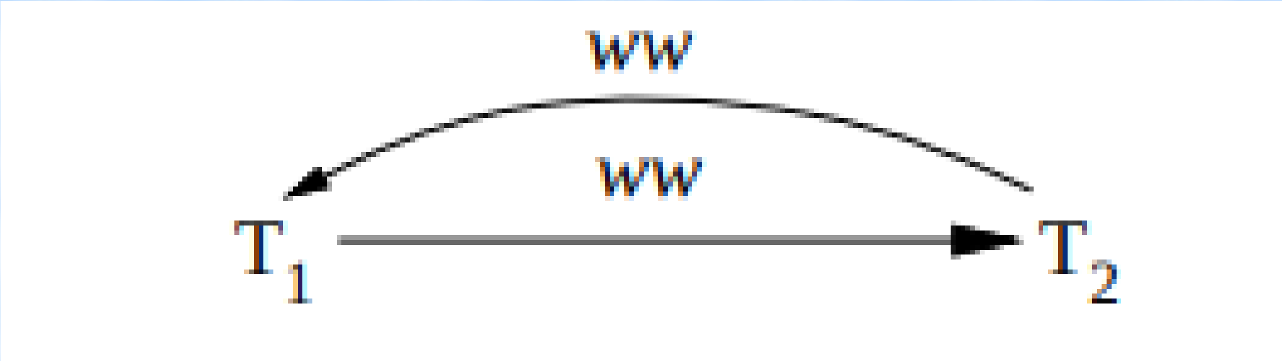
Adya: DSG based specifications

PL-1: updates of conflicting transactions are not interleaved



H_{wcycle}: $w_1(x_1, 2)\ w_2(x_2, 5)\ w_2(y_2, 5)\ c_2\ \ w_1(y_1, 8)\ c_1$

$[x_1 \ll x_2, y_2 \ll y_1]$



Example: Snapshot isolation

- Transaction t reads from a consistent snapshot, reflecting writes from transactions that committed before t began
- T can commit iff it does not have a write-write conflict with any concurrent transaction

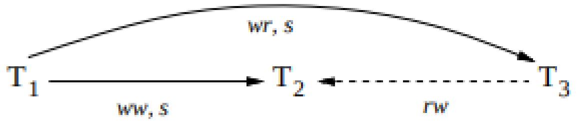


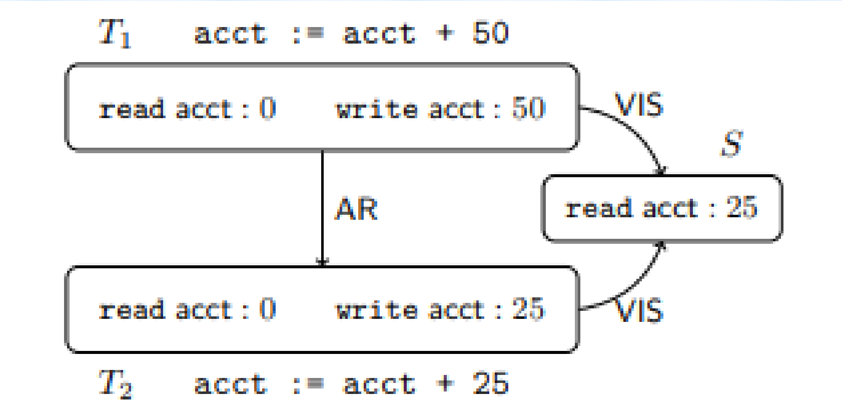
Figure 4-8: Start-ordered serialization graph for history H_{SI}

G-SIa: Interference. A history H exhibits phenomenon G-SIa if SSG(H) contains a read/write-dependency edge from Ti to Tj without there also being a start-dependency edge from Ti to Tj

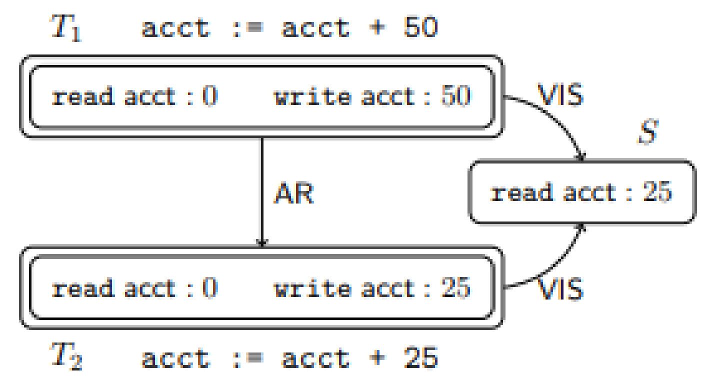
G-SIb: Missed Effects. A history H exhibits phenomenon G-SIb if SSG(H) contains a directed cycle with exactly one anti-dependency edge.

Cerone: Algebraic Rules based on Dependency Relations

Lost Update:



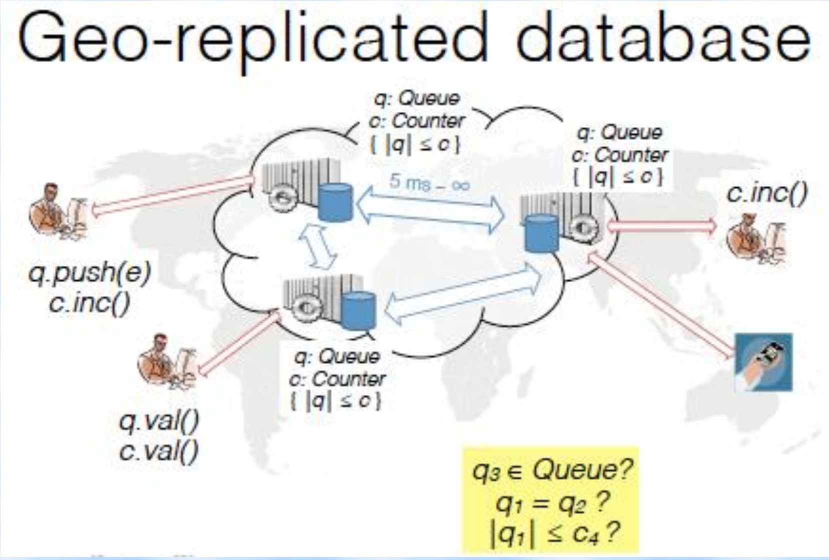
Serialisable Lost Update

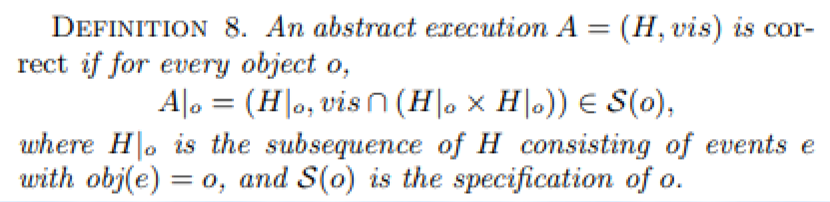


Any abstract execution $\mathcal{X} \in \mathsf{Executions}(\mathsf{SI})$ satisfies

$$(\mathsf{AR}_{\mathcal{X}} \,;\, \mathsf{VIS}_{\mathcal{X}}) \subseteq \mathsf{VIS}_{\mathcal{X}}{}^{3}.$$

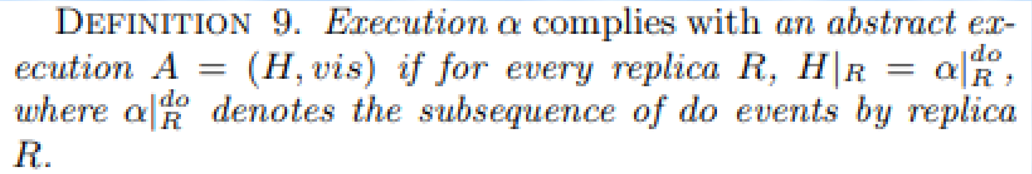
ATTIYA: CONSISTENCY SEMANTICS TIED TO TYPE OF STORAGE SYSTEM




DEFINITION 8. *An abstract execution* $A = (H, vis)$ *is correct if for every object* o,

$$A|_o = (H|_o, vis \cap (H|_o \times H|_o)) \in \mathcal{S}(o),$$

where $H|_o$ *is the subsequence of* H *consisting of events* e *with* $obj(e) = o$, *and* $\mathcal{S}(o)$ *is the specification of* o.


DEFINITION 9. *Execution* α complies with *an abstract execution* $A = (H, vis)$ *if for every replica* R, $H|_R = \alpha|_R^{do}$, *where* $\alpha|_R^{do}$ *denotes the subsequence of do events by replica* R.

ConSpec: Trading off detail for simplicity

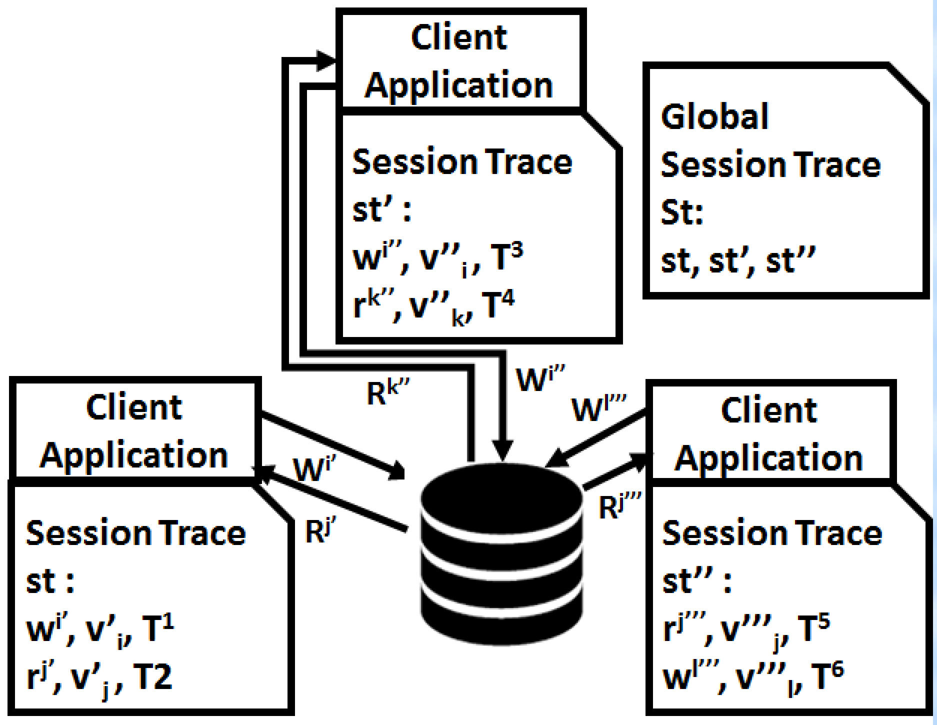
- Reasons for the shift to LTL:
  - Graph-based definitions specified in terms of Implicit and Explicit dependencies of various types
    - Requires prior understanding of meaning of each type of dependency
    - Implicit dependencies defined in terms of explicit dependencies
    - Difficult to make this derivation uniform across definitions
  - Much complicated representation

     complex graphs
  - Removing versioning from spec

     make the definitions truly implementation-independent

System Model

- Session trace st
  - client application
- Global Session trace St
  - Set of session traces



Why ConSpec

- Problems with earlier approaches:
  - Graph-based definitions specified in terms of Implicit and Explicit dependencies of various types
    - Requires prior understanding of meaning of each type of dependency
    - Implicit dependencies ⍰ defined in terms of explicit dependencies
    - Difficult to make this derivation uniform across definitions
  - Much complicated representation

    ⍰ complex graphs
  - Removing versioning from spec

    ⍰ make the definitions truly implementation-independent (already achieved by Ricardo)

Why LTL

- Consistency and Isolation

→ restrictions on temporal order in which results of operations can be observed

- Easier to understand and read
- Easier to build automated tools

→ a wide array of available automated verification tools

ConSpec Specification Format

$$\mathcal{O}_{St} = \bigcup_{st \in St} \{o \mid o \in st\}$$

A partial order $(\mathcal{O}_{St}, \preccurlyeq)$

1) for every operation o in $\mathcal{O}_{St}$, its output is equal to the one obtained by executing the sequential specification of an equivalent re-arrangement (i.e., permutation) of the operations preceding o in $\preccurlyeq$,

2) $(\mathcal{O}_{St}, \preccurlyeq)$ obeys E_C^S, which is an LTL expression restricting $(\mathcal{O}_{St}, \preccurlyeq)$

ConSpec Specifications

- RYW (Read Your Write)

$$E_C^S = \forall x \in \mathcal{X}, st \in \mathcal{S}_t, W_{st}^x, R_{st}^x \in st : (\Box (W_{st}^x \rightarrow \Diamond R_{st}^x) \rightarrow W_{st}^x \preccurlyeq_{st+w} R_{st}^x)$$

Violation Examples

st_1: $w(x,1), w'(x,2), r(x)2, r'(x)1$

- $\preccurlyeq_{st+w}^1 = W_{st}^x \preccurlyeq_{st+w}^1 R_{st}^{\prime x} \preccurlyeq_{st+w}^1 W_{st}^{\prime x} \preccurlyeq_{st+w}^1 R_{st}^x$
- $\preccurlyeq_{st+w}^2 = W_{st}^{\prime x} \preccurlyeq_{st+w}^2 R_{st}^x \preccurlyeq_{st+w}^2 W_{st}^x \preccurlyeq_{st+w}^2 R_{st}^{\prime x}$
- $\preccurlyeq_{st+w}^3 = R_{st}^x \preccurlyeq_{st+w}^3 W_{st}^{\prime x} \preccurlyeq_{st+w}^3 W_{st}^x \preccurlyeq_{st+w}^3 R_{st}^{\prime x}$

Satisfaction Examples

st_1': $w(x,1), w'(x,2), r(x)2, r'(x)2$

$\preccurlyeq_{st+w}^1 = W_{st}^x \preccurlyeq_{st+w}^1 W_{st}^{\prime x} \preccurlyeq_{st+w}^1 R_{st}^x \preccurlyeq_{st+w}^1 R_{st}^{\prime x}$

ConSpec Specifications: Contd

- Sequential Consistency

$$E_C^S = \forall x, y \in \mathcal{X}, st \in \mathcal{S}_t, O^x_{st}, O'^y_{st} \in st : \left(\Box\left(O^x_{st} \rightarrow \Diamond O'^y_{st}\right) \rightarrow O^x_{st} \preccurlyeq O'^y_{st} \wedge \forall O'', O''' \in \mathcal{S}_t : O'' \preccurlyeq O''' \vee O''' \preccurlyeq O''\right).$$

Violation Examples

$$st''_8 : w(x,1), w'(x,99), r(y)1$$
$$st''_8 : w(y,1), w'(y,99), r(x)1$$

Violation: 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.

Satisfaction Examples

Extending CAP Theorem

In an asynchronous system, it is possible to implement a consistency model E_C^S while simultaneously providing availability and partition tolerance if and only if for any global session invocation trace nd partition tolerance if and only if for any global E_C^S session invocation trace $\mathcal{S}_{it}$ and all of its partial orderings that are allowed by E_C^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., the following holds:

$$\textsc{RemoveAllExceptSession}(\preccurlyeq, o(x)) \equiv \preccurlyeq \setminus \{\langle o'(x), o(x)\rangle$$

$$\forall \mathcal{S}_{it}\ \forall \preccurlyeq \in \Pi(\mathcal{S}_{it}, E_C^S)\ \forall o \in max(\preccurlyeq)\ \ (\textsc{RemoveAllExceptSession}(\preccurlyeq, o) \in \Pi(\mathcal{S}_{it}, E_C^S))$$

We can see that both the causal and processor consistency definitions (plus the session guarantees – MR, MW, RYW, WFR) are only forcing constraints on the partial ordering across operations from the same session.

In contrast, SC requires that the visibility order $\preccurlyeq$ among operations from all the clients in the system forms a total order.

ConSpecChecker Tool

- Building of Automated Verification Tools
  - Spinroot based prototype
  - A global session trace is supplied to the tool as input
  - The Spin driver then runs the built-in model checker to check for counter-examples

The following snippet is taken from the PROMELA source file for the RYW consistency model

```
mtype = {r, w, x, y} ;
typedef Op {
mtype optype ;
mtype var ;
int val ; }
typedef PO {
Op st[max size ] ;
mtype status ; }
Op st[size] ;
Op po[po size ] ;
ltl cc { [] ( ¬ ( po[i].st[j].optype = w ⇒ ♦ po[i].st[j].optype = r)) }
```

ConSpecker Tool: Results

A) How long the tool takes to check the consistency of a session trace,
B) how this validation time varies depending on the length (or size) of the trace,
C) How it compares to checking traces expressed in conventional syntaxes.

We use two sets of traces:
1) generated by executing YCSB on top of a Cassandra cluster on Amazon aws,
2) obtained by executing the TPC-C benchmark on top of a MySQL database.

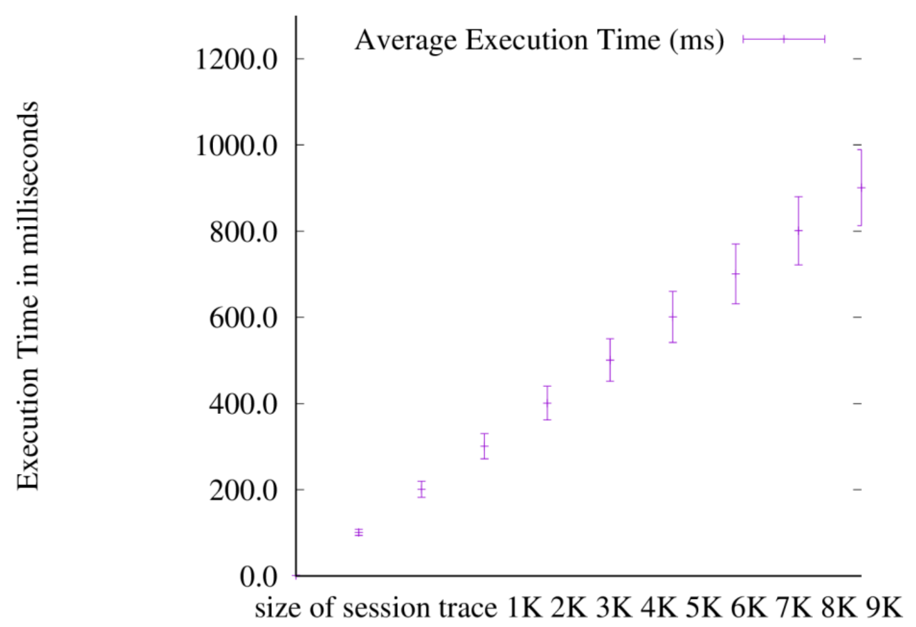
The tool was run over the above traces on an Apple MacBook Pro, with 8 GB 1600 MHz DDR3 RAM, 2.9 GHz Intel Core i7 processor, running MacOS Sierra v10.12.4

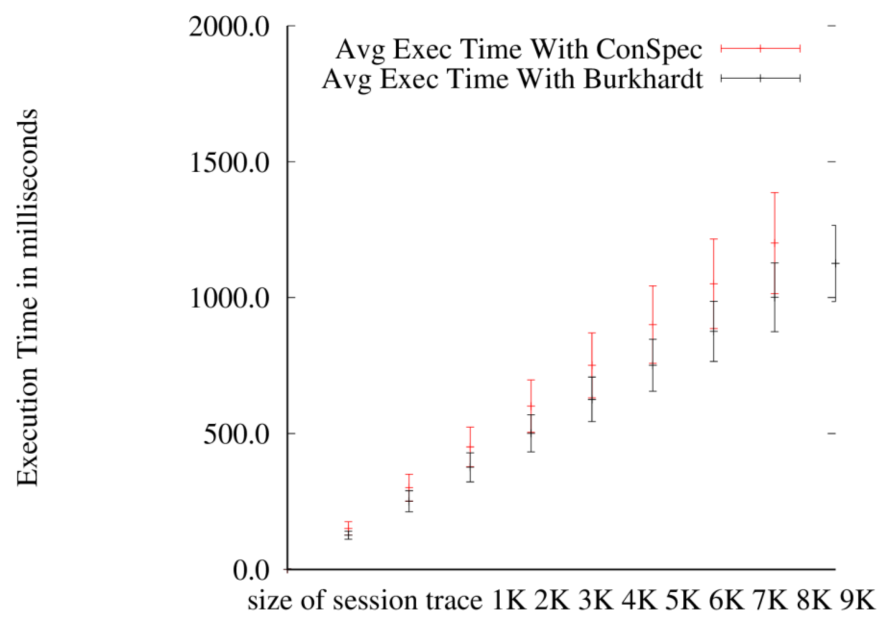
The partial order generator component of the tool was run on Java 1.8.0_121, and the PROMELA component were compiled and run on Spin v6.4.6.

ConSpecker Tool: Results

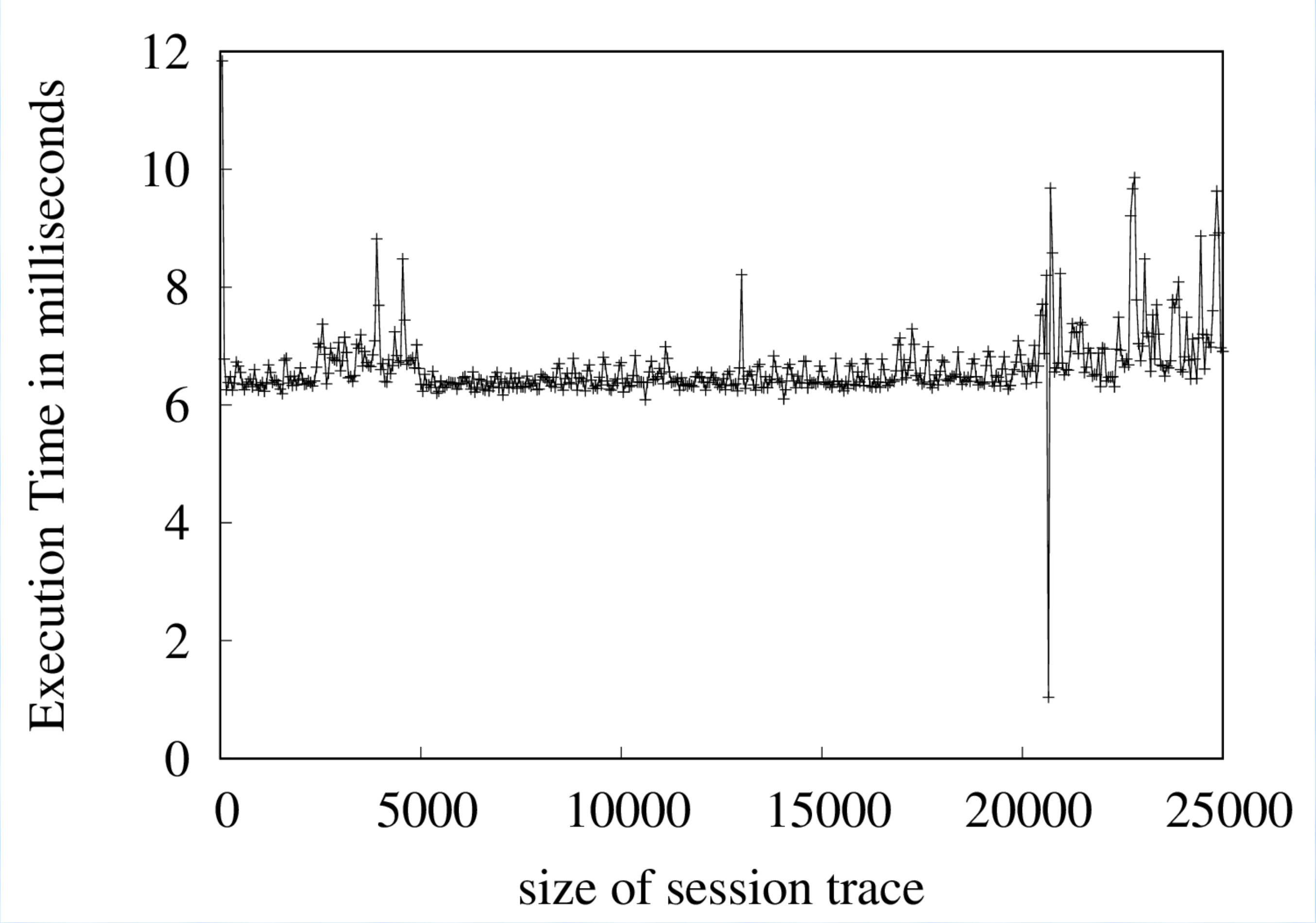
To generate a series of global session traces of different lengths, we are able to vary two configuration parameters of YCSB: the **thread count** and the **execution time**.
Using the thread count parameter, we simulated a number of concurrent YCSB **client threads** executing the given workload, where the number of clients corresponds to the value passed to this parameter.

Thus, each execution of the YCSB client with a given value of the thread parameter generates a global session trace consisting of multiple session traces, where each session trace comprises the entire sequence of operations performed from a specific client thread.





ConSpecker Tool: Results



Future Work (contd.)

- Explore combination of consistency and isolation
- Other advantages of LTL based definitions?
- Automated Verification Tool for verifying system **Code** against consistency and isolation specs
- Analyze the implications for system developers in terms of system design/development

ConSpec Specifications (Isolation Levels)

- PL-1: Proscribes directed cycles consisting entirely of write-dependency (ww) edges.

$$
\begin{aligned}
E^r = C = \forall St, st, tx, ty, x, y, W^i(x)'_{tx}, W^j(y)'_{tx},\\
W^k(x)'_{ty}, W^l(y)'_{ty} \in St \; \left(\left(\nexists R^m(x)'_{tx}, R^n(y)'_{ty} \in St \right) \right.\\
\left. \left(\left(\left(v'_n = v'_l \right) \wedge \left(v'_m = v'_i \right) \right) \vee \left(\left(v'_n = v'_j \right) \wedge \left(v'_m = v'_k \right) \right) \right) \right)
\end{aligned}
$$

Conclusions

- A unified, simple specification that formalizes consistency and isolation in an uniform syntax
- ConSpec seamlessly combines consistency and isolation using common syntax
- E.g., natural definition for transactions with consistency level X and isolation level Y
- Can leverage existing automated verification tools (Model Checkers/SAT solvers) to verify whether a storage system satisfies a claimed consistency model or isolation level
- Equivalence to previous definitions
- Extension of CAP

Thanks!

- Always Open to discussion/collaborations: subhajit@iitbhilai.ac.in
- Openings for t Assistants/Ir